

WebSphere® eXtreme Scale versión 7.1

Guía de programación

IBM

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

© Copyright IBM Corporation 2009, 2010.

Contenido

Figuras v

Tablas vii

Acerca de la *Guía de programación* . . . ix

Capítulo 1. Cómo empezar con WebSphere eXtreme Scale. 1

Capítulo 2. Preparación para desarrollar aplicaciones 7

Interfaces de programación de WebSphere eXtreme Scale 7
Consideraciones del cargador de clases y la classpath 8
Configuración de un entorno de desarrollo 8
Ejecución de una aplicación de servidor o cliente WebSphere eXtreme Scale con Apache Tomcat en Rational Application Developer. 10
Ejecución de una aplicación de servidor o cliente integrada con WebSphere Application Server en Rational Application Developer. 12

Capítulo 3. Acceso a los datos con aplicaciones cliente 15

Interfaz ObjectGrid 15
Interfaz BackingMap 18
Conexión a un ObjectGrid distribuido 22
Interacción con un ObjectGrid utilizando ObjectGridManager. 23
 Métodos createObjectGrid 23
 Métodos de getObjectGrid 28
 Métodos removeObjectGrid 28
 Control del ciclo de vida de un ObjectGrid 29
 Acceso al fragmento de ObjectGrid 31
Acceso a los datos en WebSphere eXtreme Scale 31
 Procedimientos recomendados de CopyMode 35
 Correlaciones de matriz de bytes 40
Utilización de sesiones para acceder a los datos de la cuadrícula 42
 SessionHandle para el direccionamiento 47
 Integración de SessionHandle 47
Almacenamiento en memoria caché de objetos sin relaciones implicadas (API ObjectMap) 50
 Introducción a ObjectMap 51
 Correlaciones dinámicas 54
 ObjectMap y JavaMap 58
 Correlaciones como colas FIFO 58
Almacenamiento en memoria caché de objetos y sus relaciones (API EntityManager). 61
 Definición de un esquema de entidad 62
 EntityManager en un entorno distribuido 73
 Interacción con EntityManager 76
 Soporte de planes de captación de EntityManager 83

Impacto en el rendimiento de la interfaz EntityManager 87
Colas de consulta de entidades 90
Interfaz EntityManager Query 94
Guía de aprendizaje del gestor de entidades: visión general 95
Recuperación de entidades y objetos (API de consulta) 95
 Consulta de datos en varios husos horarios. 99
 Inserción de datos para husos horarios diferentes. 101
 Utilización de la API ObjectQuery 101
 API EntityManager Query 106
 Referencia para consultas de eXtreme Scale 110
 Ajuste del rendimiento de consultas. 121
 Utilización de objetos que no sean claves para encontrar las particiones (interfaz PartitionableKey) 133
Programación de transacciones 134
 Visión general del proceso de transacciones 134
 Manejo de bloqueos 150
 Aislamiento de transacciones 167
 Excepción de colisión optimista 169
Configuración de clientes con WebSphere eXtreme Scale 170
 Seguimiento de las actualizaciones de correlación de una aplicación 175
 Habilitación de la réplica en el cliente 178
Ejemplo de la API de DataGrid 179

Capítulo 4. Acceso a los datos con el servicio de datos REST 185

Operaciones con el servicio de datos REST 185
Protocolos de solicitud para el servicio de datos REST 187
 Solicitudes de recuperación con el servicio de datos REST 187
 Recuperación de elementos que no sean entidades con los servicios de datos REST. 194
 Solicitudes de inserción con los servicios de datos REST 199
 Solicitudes de actualización con los servicios de datos REST 203
 Solicitudes de supresión con los servicios de datos REST 207
Simultaneidad optimista. 209

Capítulo 5. programación con sistema plug-ins y API del sistema 211

Introducción a los plug-ins 211
Plug-ins para desalojar los objetos de memoria caché 213
 Desalojador TimeToLive (TTL). 216
 Conexión de un desalojador conectable. 219
 Cómo escribir un desalojador personalizado 222

Procedimientos recomendados para optimizar el rendimiento del desalojador de plug-in	226
Plug-ins para transformar los objetos almacenados en memoria caché	228
Desarrollo de árbitros personalizados para la réplica con varios maestros	229
Plug-in ObjectTransformer	230
Plug-ins para el mantenimiento de versiones y la comparación de objetos de memoria caché	237
Plug-ins para proporcionar escuchas de sucesos	241
Plug-in MapEventListener	242
Plug-in ObjectGridEventListener	243
Plug-ins para la indexación personalizada de los objetos de memoria caché	246
Índice compuesto HashIndex	248
Uso de índices en el acceso a los datos que no son clave	251
Plug-ins para la comunicación con almacenes persistentes	255
Escribir un cargador	257
Consideraciones de programación del cargador JPA	261
Plug-in JPAEntityLoader	263
Uso de un cargador con correlaciones de entidad y tuples	265
Escribir un cargador con un controlador de precarga de réplica	270
Plug-ins para gestionar los sucesos del ciclo de vida de transacciones	275
Visión general del proceso de transacciones	279
Introducción a las ranuras de plug-in	279
Gestores de transacciones externas	282
Plug-in WebSphereTransactionCallback	284
Capítulo 6. Programación de las tareas administrativas	287
API de servidor incorporado	287
Uso de la API de servidor incorporado	289
Supervisión con la API de estadísticas	292
Supervisión con PMI de WebSphere Application Server	294
Habilitación de PMI	295
Recuperar estadísticas de PMI	297
Módulos PMI	299
Acceso a MBeans mediante la herramienta wsadmin	306
Capítulo 7. Programación de la integración JPA	309
Cargadores JPA	309
Visión general del programa de utilidad de precarga JPA basada en cliente	311

Programación del programa de utilidad de precarga JPA basada en cliente	313
Actualizador de datos basado en la hora de JPA	320
Inicio del actualizador basado en la hora de JPA	322

Capítulo 8. Programación de la integración de Spring 327

Visión general de integración en la infraestructura de Spring	327
Transacciones gestionadas de Spring	328
Beans de ampliación gestionados de Spring	331
Beans de ampliación de Spring y soporte de espacio de nombres	332

Capítulo 9. Programación de la seguridad 337

API de seguridad	337
Programación de la autenticación de cliente	338
Programación de autorización de cliente	356
Autenticación de cuadrícula	364
Seguridad local	364

Capítulo 10. Consideraciones sobre el rendimiento para desarrolladores de aplicaciones 371

Ajuste de la JVM	371
Procedimientos recomendados de CopyMode	373
Correlaciones de matriz de bytes	378
Procedimientos recomendados para optimizar el rendimiento del desalojador de plug-in	380
Procedimientos recomendados para optimizar el rendimiento de bloqueos	383
Rendimiento de serialización	384
Procedimientos recomendados para la interfaz ObjectTransformer	385

Capítulo 11. Resolución de problemas 389

Registros y rastreo	389
Opciones de rastreo	391
IBM Support Assistant para WebSphere eXtreme Scale	393
Mensajes	394
Notas del release	394

Avisos 397

Marcas registradas 399

Índice. 401

Figuras

1.	La interacción de la consulta con las correlaciones de objeto ObjectGrid y cómo se define un esquema para las clases y se asocia a una correlación de ObjectGrid	102
2.	La interacción de la consulta con las correlaciones de objeto ObjectGrid y cómo se define y asocia el esquema de entidad con una correlación de ObjectGrid.	107
3.	Cargador	255
4.	Estructura del módulo ObjectGridModule	299
5.	Ejemplo de estructura del módulo ObjectGridModule	300
6.	Estructura de mapModule	301
7.	Ejemplo de la estructura del módulo mapModule	301
8.	Estructura del módulo hashIndexModule	303
9.	Ejemplo de estructura del módulo hashIndexModule	303
10.	Estructura de agentManagerModule	304
11.	Ejemplo de la estructura de agentManagerModule.	305
12.	Estructura de queryModule.	306
13.	Ejemplo de la estructura de queryModule de QueryStats.jpg	306
14.	Arquitectura de cargador JPA	310
15.	El cargador de cliente que utiliza la implementación JPA para cargar el ObjectGrid	312
16.	Renovación periódica	321
17.	Flujo de autenticación y autorización de cliente	337

Tablas

1. Interfaz ObjectGrid	15	10. Escenario de punto muerto de varias llaves ordenadas, continuación	158
2. Otros métodos	98	11. Escenario de fuera de servicio con bloqueo U	159
3. Clave para el resumen de BNF.	119	12. Modalidades del cargador de cliente.	312
4. Matriz de compatibilidad de modalidad de bloqueo	152	13. Lista de métodos y MapPermission requeridos	357
5. Escenario de puntos muertos de llave única	156	14. Lista de métodos y ObjectGridPermission requeridos	358
6. Puntos muertos de llave única, continuación	156	15. Permisos para un ObjectMap alojado en un servidor	359
7. Puntos muertos de llave única, continuación	156		
8. Puntos muertos de llave única, continuación	157		
9. Escenario de punto muerto de varias llaves ordenadas.	158		

Acerca de la *Guía de programación*

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 está especialmente indicada para los desarrolladores de aplicaciones.

Estructura de esta publicación

La publicación contiene información sobre los siguientes temas principales:

- **Capítulo 1** incluye información sobre cómo empezar con WebSphere eXtreme Scale.
- **Capítulo 2** incluye información sobre cómo programar WebSphere eXtreme Scale.
- **Capítulo 3** incluye información sobre cómo acceder a datos.
- **Capítulo 4** incluye información sobre las API y los plug-ins del sistema.
- **Capítulo 5** incluye información sobre cómo integrarse con la infraestructura Spring.
- **Capítulo 6** incluye información sobre la API de seguridad.
- **Capítulo 7** incluye información sobre la API de administración.
- **Capítulo 8** incluye información sobre las consideraciones de rendimiento.
- **Capítulo 9** incluye información sobre la resolución de problemas.
- **Capítulo 10** incluye el glosario del producto.

Cómo obtener actualizaciones de esta publicación

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

Envío de comentarios

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

Capítulo 1. Cómo empezar con WebSphere eXtreme Scale

Tras instalar WebSphere eXtreme Scale en un entorno autónomo, utilice los pasos siguientes como una simple presentación de sus funciones como en una cuadrícula de datos de memoria.

La instalación autónoma de WebSphere eXtreme Scale incluye un ejemplo que puede utilizar para verificar la instalación y para ver cómo se puede utilizar un cliente y una cuadrícula sencilla de eXtreme Scale. El ejemplo de iniciación está en el directorio `installRoot/ObjectGrid/gettingstarted`.

El ejemplo de iniciación proporciona una rápida introducción a las características y al funcionamiento básico de eXtreme Scale. El ejemplo está formado por scripts de shell y de procesos por lotes diseñados para iniciar una cuadrícula sencilla con muy poca necesidad de personalización. Además, un programa cliente, incluido el origen, se proporciona para ejecutar las funciones CRUD (crear, leer, actualizar y suprimir) sencillas en esta cuadrícula básica.

Los scripts y sus funciones

Este ejemplo proporciona los cuatro scripts siguientes:

Los otros scripts llaman al script `env.sh|bat` para establecer las variables de entorno necesarias. Normalmente, no necesita cambiar este script.

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

`runcat.sh|bat` inicia el proceso del servicio de catálogo eXtreme Scale en el sistema local.

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

El script `runcontainer.sh|bat` inicia un proceso de servidor de contenedor. Puede ejecutar este script varias veces con nombres de servidor exclusivos especificados para iniciar cualquier número de contenedores. Estas instancias pueden trabajar juntas para alojar información con particiones y redundante de la cuadrícula.

- `UNIX Linux ./runcontainer.sh nombre_servidor_exclusivo`
- `Windows runcontainer.bat nombre_servidor_exclusivo`

El script `runclient.sh|bat` ejecute el cliente de CRUD sencillo e inicia la operación determinada.

- `UNIX Linux ./runclient.sh mandato valor1 valor2`
- `Windows runclient.sh mandato valor1 valor2`

Para *mandato*, utilice una de las siguientes opciones:

- Especifique como `i` para insertar el *valor2* en la cuadrícula con la clave *valor1*
- Especifique como `u` para actualizar el objeto con clave de *valor1* a *valor2*
- Especifique como `d` para suprimir el objeto con clave por *valor1*

- Especifique como `g` para recuperar y visualizar el objeto con clave por *valor1*

Nota: El archivo `installRoot/ObjectGrid/gettingstarted/src/Client.java` es el programa cliente que demuestra cómo conectarse a un servidor de catálogo, obtener una instancia de `ObjectGrid` y utiliza la API `ObjectMap`.

Pasos básicos

Utilice los siguientes pasos para iniciar la primera cuadrícula y ejecutar un cliente para interactuar con la cuadrícula.

1. Abra una ventana de sesión de terminal o de línea de mandatos.
2. Utilice el siguiente mandato para ir hasta el directorio `gettingstarted`:

```
cd installRoot/ObjectGrid/gettingstarted
```

Sustituya `installRoot` por la vía de acceso del directorio raíz de instalación de eXtreme Scale o la vía de acceso del archivo raíz de la versión de prueba de eXtreme Scale extraída `installRoot`.

3. Ejecute el siguiente script para iniciar un proceso de servicio de catálogo en el sistema principal local:

- `UNIX` `Linux` `./runcat.sh`

- `Windows` `runcat.bat`

El proceso de servicio de catálogo se ejecuta en la ventana actual de terminal.

4. Abra otra ventana de sesión de terminal o de línea de mandatos, y ejecute el siguiente mandato para iniciar una instancia de servidor de contenedor:

- `UNIX` `Linux` `./runcontainer.sh server0`

- `Windows` `runcontainer.bat server0`

El servidor de contenedor se ejecuta en la ventana actual del terminal. Puede repetir el paso 5 y 6 si desea iniciar más instancias de servidor de contenedor para soportar la réplica.

5. Abra otra ventana de sesión de terminal o de línea de mandatos para ejecutar los mandatos de cliente.

- Añada datos a la cuadrícula:

- `UNIX` `Linux` `./runclient.sh i key1 helloWorld`

- `Windows` `runclient.bat i key1 helloWorld`

- Busque y visualice el valor:

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

- `Windows` `runclient.bat g key1`

- Actualice el valor:

- `UNIX` `Linux` `./runclient.sh u key1 goodbyeWorld`

- `Windows` `runclient.bat u key1 goodbyeWorld`

- Suprima el valor:

- `UNIX` `Linux` `./runclient.sh d key1`

- `Windows` `runclient.bat d key1`

6. Utilice `<ctrl+c>` para detener el proceso del servicio de catálogo y los servidores de contenedor en las ventanas respectivas.

Definición de un ObjectGrid

El ejemplo utiliza los archivos `objectgrid.xml` y `deployment.xml` que están en el directorio `installRoot/ObjectGrid/gettingstarted/xml` para iniciar un servidor de contenedor. El archivo `objectgrid.xml` es el archivo XML de descriptor de ObjectGrid y el archivo `deployment.xml` es el archivo XML de descriptor de política de despliegue de ObjectGrid. Ambos archivos definen de forma conjunta una topología de ObjectGrid distribuida.

Archivo XML de descriptor ObjectGrid

Se utiliza un archivo XML de descriptor de ObjectGrid para definir la estructura del ObjectGrid que es utilizado por la aplicación. Incluye una lista de configuraciones de BackingMap. Estas BackingMaps son el almacenamiento real de datos para los datos almacenados en memoria caché. El ejemplo siguiente es un archivo `objectgrid.xml` de ejemplo. Las primeras líneas del archivo incluyen la cabecera necesaria para cada archivo XML de ObjectGrid. Este archivo de ejemplo define el ObjectGrid Grid con las BackingMaps Map1 y Map2.

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

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

</objectGridConfig>
```

Archivo XML de descriptor de la política de despliegue

Se pasa un archivo XML de descriptor de la política de despliegue a un servidor de contenedor ObjectGrid durante el arranque. Se debe utilizar una política de despliegue con un archivo XML de ObjectGrid y debe ser compatible con el XML de ObjectGrid que se utiliza con la misma. Para cada elemento `objectgridDeployment` de la política de despliegue, debe tener un elemento ObjectGrid correspondiente en el XML de ObjectGrid. Los elementos `backingMap` que están definidos dentro del elemento `objectgridDeployment` deben ser coherentes con las `backingMaps` que se encuentran en el XML de ObjectGrid. Debe hacerse referencia a cada `backingMap` dentro de únicamente un `mapSet`.

El archivo XML de descriptor de política de despliegue intenta emparejarse con el XML correspondiente de ObjectGrid, el archivo `objectgrid.xml`. En el siguiente ejemplo, las primeras líneas del archivo `deployment.xml` incluyen la cabecera necesaria para cada archivo XML de política de despliegue. El archivo define el elemento `objectgridDeployment` para el ObjectGrid Grid que está definido en el archivo `objectgrid.xml`. Ambas BackingMaps, Map1 y Map2, que están definidas dentro del ObjectGrid Grid se incluyen en el `mapSet` `mapSet` que tiene los atributos `numberOfPartitions`, `minSyncReplicas` y `maxSyncReplicas` configurados.

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

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

```

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

```

El atributo `numberOfPartitions` del elemento `mapSet` especifica el número de particiones para el `mapSet`. Es un atributo opcional y el valor predeterminado es 1. El número debe ser apropiado para la capacidad anticipada de la cuadrícula.

El atributo `minSyncReplicas` de `mapSet` especifica el número mínimo de réplicas síncronas para cada partición del `mapSet`. Se trata de un atributo opcional y el valor predeterminado es 0. El primario y la réplica no se colocan hasta que el dominio pueda soportar el número mínimo de réplicas síncronas. Para dar soporte al valor `minSyncReplicas`, es necesario un contenedor más que el valor de `minSyncReplicas`. Si el número de réplicas síncronas cae por debajo del valor de `minSyncReplicas`, ya no se permiten transacciones de grabación para esa partición.

El atributo `maxSyncReplicas` de `mapSet` especifica el número máximo de réplicas síncronas para cada partición del `mapSet`. Se trata de un atributo opcional y el valor predeterminado es 0. No se coloca ninguna otra réplica síncrona para una partición después de que un dominio alcance este número de réplicas síncronas para dicha partición específica. La adición de contenedores que puedan dar soporte a este `ObjectGrid` puede comportar un aumento en el número de réplicas síncronas si todavía no se ha alcanzado el valor de `maxSyncReplicas`. El ejemplo de valor `maxSyncReplicas` establecido en 1 significa que el dominio colocará, como mínimo, una réplica síncrona. Si inicia más de una instancia de servidor de contenedor, sólo habrá una réplica síncrona colocada en una de las instancias del servidor de contenedor.

Utilización de ObjectGrid

El archivo `Client.java` en el directorio `installRoot/ObjectGrid/gettingstarted/src/` es el programa cliente que demuestra cómo conectarse al servidor de catálogo, obtener la instancia de `ObjectGrid` y utilizar la API `ObjectMap`.

Desde la perspectiva de una aplicación cliente, el uso de `WebSphere eXtreme Scale` se puede dividir en los pasos siguientes.

1. Conexión al servicio de catálogo obteniendo una instancia de `ClientClusterContext`.
 2. Obtención de una instancia `ObjectGrid` cliente.
 3. Obtención de una instancia `Session`.
 4. Obtención de una instancia `ObjectMap`.
 5. Uso de los métodos `ObjectMap`.
1. **Conexión al servicio de catálogo obteniendo una instancia de `ClientClusterContext`**

Para conectarse al servidor de catálogo, utilice el método `connect` de la API `ObjectGridManager`. El método `connect` utilizado sólo requiere el punto final de servidor de catálogo en el formato de `nombrehost:puerto`, como, por ejemplo, `localhost:2809`. Si la conexión con el servidor de catálogo se establece correctamente, el método `connect` devuelve una instancia de `ClientClusterContext`. La instancia de `ClientClusterContext` es necesaria para

obtener el ObjectGrid de la API ObjectGridManager. El siguiente fragmento de código demuestra cómo conectarse a un servidor de catálogo y obtener una instancia de ClientClusterContext.

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

2. Obtención de una instancia de ObjectGrid

Para obtener una instancia de ObjectGrid, utilice el método getObjectGrid de la API ObjectGridManager. El método getObjectGrid requiere tanto la instancia de ClientClusterContext, como el nombre de la instancia de ObjectGrid. La instancia de ClientClusterContext se obtiene durante la conexión al servidor de catálogo. El nombre del ObjectGrid es Grid que se especifica en el archivo objectgrid.xml. En el siguiente fragmento de código se muestra cómo obtener el ObjectGrid llamando al método getObjectGrid de la API ObjectGridManager.

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

3. Obtención de una instancia de Session

Puede obtener una Session desde la instancia de ObjectGrid obtenida. Es necesaria una instancia de Session para obtener la instancia de ObjectMap, y realizar la demarcación de la transacción. En el siguiente fragmento de código se muestra cómo obtener una instancia de Session llamando al método getSession de la API ObjectGrid.

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

4. Obtención de una instancia de ObjectMap

Después de obtener una Session, podrá obtener una instancia de ObjectMap desde una instancia de Session llamando al método getMap de la API Session. Debe pasar el nombre de la correlación como parámetro al método getMap para poder obtener la instancia de ObjectMap. En el siguiente fragmento de código se muestra cómo obtener ObjectMap llamando al método getMap de la API Session.

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

5. Uso de los métodos ObjectMap

Después de obtener ObjectMap, puede utilizar la API ObjectMap. Recuerde que la interfaz ObjectMap es una correlación transaccional y requiere la demarcación de transacción utilizando los métodos begin y commit de la API Session. Si no hay ninguna demarcación de transacción explícita en la aplicación, las operaciones de ObjectMap se ejecutan con transacciones de confirmación automática.

En el siguiente fragmento de código se muestra cómo utilizar la API ObjectMap con la transacción de confirmación automática.

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

En el siguiente fragmento de código se muestra cómo utilizar la API ObjectMap con la demarcación de transacción explícita.

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

Información adicional

Este ejemplo demuestra cómo iniciar el servidor de catálogo y el servidor de contenedor y cómo utilizar la API ObjectMap en un entorno autónomo. También puede utilizar la API EntityManager.

En un entorno WebSphere Application Server con WebSphere eXtreme Scale instalado o habilitado, el escenario más común es una topología conectada a red. En una topología conectada a red, el servidor de catálogo se encuentran en el proceso del gestor de despliegue de WebSphere Application Server y cada instancia

de WebSphere Application Server aloja un servidor eXtreme Scale automáticamente. Las aplicaciones Java™ Platform, Enterprise Edition sólo deben incluir el archivo XML de descriptor de ObjectGrid y, también, el archivo XML de descriptor de la política de despliegue de ObjectGrid en el directorio META-INF de cualquier módulo y el ObjectGrid pasa a estar disponible de forma automática. Una aplicación se puede conectar a un servidor de catálogo disponible de forma local y obtener una instancia de ObjectGrid para utilizar.

Capítulo 2. Preparación para desarrollar aplicaciones

Configure el entorno de desarrollo y obtenga información sobre dónde puede encontrar los detalles sobre las interfaces de programación disponibles.

Interfaces de programación de WebSphere eXtreme Scale

WebSphere eXtreme Scale proporciona varias características a las que se accede a través de programa utilizando el lenguaje de programación Java a través de las interfaces de programación de aplicaciones (API) y las interfaces de programación del sistema.

API de WebSphere eXtreme Scale

Cuando se utilizan las API de eXtreme Scale, debe distinguirse entre operaciones transaccionales y no transaccionales. Una operación transaccional es una operación que se realiza dentro de una transacción. Las API ObjectMap, EntityManager, Query y DataGrid son API transaccionales que están contenidas dentro del objeto Session que es un contenedor transaccional. Las operaciones transaccionales no tienen nada que ver con una transacción, como por ejemplo las operaciones de configuración.

Las API ObjectGrid, BackingMap y plug-in son no transaccionales. Las API ObjectGrid, BackingMap y otras API de configuración se clasifican como API central de ObjectGrid. Los plug-ins son para personalizar la memoria caché para conseguir las funciones que desea y se categorizan como la API de programación del sistema. Un plug-in en eXtreme Scale es un componente que proporciona un determinado tipo de función a los componentes de eXtreme Scale que se pueden conectar que incluyen ObjectGrid y BackingMap. Una característica representa una función o característica específica de un componente de eXtreme Scale, que incluye ObjectGrid, Session, BackingMap, ObjectMap, etc. Normalmente, las características se pueden configurar con las API de configuración. Los plug-ins pueden estar incorporados, pero en algunas situaciones es posible que tenga que desarrollar sus propios plug-ins.

Normalmente, puede configurar ObjectGrid y BackingMap para cumplir los requisitos de la aplicación. Si la aplicación tiene unos requisitos especiales, considere el uso de plug-ins especializados. WebSphere eXtreme Scale podría tener los plug-ins incorporados que cumplen los requisitos. Por ejemplo, si necesita un modelo de réplica de igual a igual entre dos instancias de ObjectGrid locales y dos cuadrículas de eXtreme Scale distribuidas, está disponible el JMSObjectGridEventListener incorporado. Si ninguno de los plug-ins incorporados puede solucionar sus problemas empresariales, consulte la API de programación del sistema para conseguir sus propios plug-ins.

ObjectMap es una API sencilla basada en correlaciones. Si los objetos almacenados en memoria caché son sencillos y no tienen ninguna relación, la API ObjectMap es ideal para la aplicación. Si hubiera relaciones de objeto, utilice la API EntityManager, que soporta las relaciones como gráficos.

Query es un mecanismo muy sólido para encontrar datos en ObjectGrid. Tanto Session como EntityManager ofrecen la prestación tradicional de consulta.

La API de DataGrid es una potente prestación informática en un entorno distribuido de eXtreme Scale que implica muchas máquinas, réplicas y particiones. Las aplicaciones pueden ejecutar la lógica empresarial en paralelo en todos los nodos del entorno distribuido de eXtreme Scale. La aplicación puede obtener la API DataGrid a través de la API ObjectMap.

7.0.0.0 FIX 2+ El servicio de datos REST de WebSphere eXtreme Scale es un servicio HTTP Java compatible con Microsoft® WCF Data Services (anteriormente ADO.NET Data Services) e implementa el Protocolo de datos abierto (OData). El servicio de datos REST permite a cualquier cliente HTTP acceder a una cuadrícula de eXtreme Scale. Es compatible con el soporte de WCF Data Services que se proporciona con Microsoft .NET Framework 3.5 SP1. Se pueden desarrollar aplicaciones RESTful con las útiles herramientas proporcionadas por Microsoft Visual Studio 2008 SP1. Para obtener más información, consulte la guía del usuario del servicio de datos REST de eXtreme Scale.

Consideraciones del cargador de clases y la classpath

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

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

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

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

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

Configuración de un entorno de desarrollo

Configure un entorno de desarrollo integrado basado en Eclipse para crear y ejecutar una aplicación Java SE con eXtreme Scale.

Procedimiento

- Configure Eclipse para crear y ejecutar una aplicación Java SE con eXtreme Scale.
 1. Defina una biblioteca de usuario para permitir que la aplicación haga referencia a las interfaces de programación de aplicaciones de eXtreme Scale.
 - a. En el entorno Eclipse o IBM® Rational Application Developer, pulse **Ventana > Preferencias**.
 - b. Expanda la ramificación **Java > Vía de acceso de compilación** y seleccione **Bibliotecas de usuario**. Pulse **Nueva**.
 - c. Seleccione la biblioteca de usuario de eXtreme Scale. Pulse **Añadir JAR**.

- 1) Examine y seleccione los archivos `objectgrid.jar` y `cglib.jar` del directorio raíz_wxs/lib. Pulse **Aceptar**.
- 2) Para incluir el Javadoc de la API ObjectGrid, seleccione la ubicación del Javadoc para el archivo `objectgrid.jar` que ha añadido en el paso anterior. Pulse **Editar**. En el recuadro de vía de acceso de ubicación del Javadoc, especifique la dirección web siguiente:

`http://www.ibm.com/developerworks/wikis/extremescale/docs/api/`

- d. Pulse **Aceptar** para aplicar los valores y cerrar la ventana Preferencias.

Ahora las bibliotecas de eXtreme Scale estarán en la vía de acceso de compilación del proyecto.

2. Añada la biblioteca de usuario a su proyecto Java.
 - a. En el explorador de paquetes, pulse el botón derecho del ratón en el proyecto y seleccione **Propiedades**.
 - b. Seleccione el separador **Bibliotecas**.
 - c. Pulse **Añadir biblioteca**.
 - d. Seleccione **Biblioteca de usuario**. Pulse **Siguiente**.
 - e. Seleccione la biblioteca de usuario eXtreme Scale que ha configurado anteriormente.
 - f. Pulse **Aceptar** para aplicar los cambios y cerrar la ventana Propiedades.
- Ejecute una aplicación Java SE con eXtreme Scale con Eclipse. Cree una configuración de ejecución para ejecutar su aplicación.
 1. Configure Eclipse para crear y ejecutar una aplicación Java SE con eXtreme Scale. En el menú **Ejecutar** seleccione **Ejecutar configuraciones**.
 2. Pulse el botón derecho del ratón en la categoría Aplicación Java y seleccione **Nueva**.
 3. Seleccione la nueva configuración de ejecución, denominada *Nueva_configuración*.
 4. Configure el perfil.
 - **Proyecto** (en la página tabulada principal): *su_nombre_proyecto*
 - **Clase principal** (en la página tabulada principal): *su_clase_principal*
 - **Argumentos de VM** (en la página tabulada de argumentos):
-Djava.endorsed.dirs=raíz_wxs/lib/endorsed

Suelen ocurrir problemas con los **Argumentos VM** porque la vía de acceso de `java.endorsed.dirs` debe ser absoluta sin variables ni atajos.

Otros problemas comunes de configuración están relacionados con el intermediario de solicitud de objetos (ORB). Podría aparecer el error siguiente. Consulte Configuración de un intermediario de solicitud de objetos personalizado para obtener más información:

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

Si no tiene los archivos `objectGrid.xml` o `deployment.xml` accesibles para la aplicación, podría aparecer el error siguiente:

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

5. Pulse **Aplicar** y cierre la ventana o pulse **Ejecutar**.

Ejecución de una aplicación de servidor o cliente WebSphere eXtreme Scale con Apache Tomcat en Rational Application Developer

Tanto si tiene una aplicación de servidor o cliente, utilice los mismos pasos básicos para ejecutar la aplicación en Apache Tomcat de Rational Application Developer. Para una aplicación cliente, desea configurar y ejecutar una aplicación web para utilizar un cliente WebSphere eXtreme Scale en Rational Application Developer. Siga estas instrucciones para crear un proyecto web para ejecutar un servicio de catálogo o contenedor de WebSphere eXtreme Scale. Para una aplicación de servidor, desea habilitar una aplicación Java EE en la interfaz de Rational Application Developer con una instalación autónoma de WebSphere eXtreme Scale. Siga estas instrucciones para configurar un proyecto de aplicación Java EE para utilizar la biblioteca del cliente WebSphere eXtreme Scale.

Antes de empezar

Instale el producto WebSphere eXtreme Scale de prueba o completo.

- Instale la versión autónoma del producto WebSphere eXtreme Scale Versión 7.1.
- Descargue y extraiga la versión de prueba de WebSphere eXtreme Scale.
- Instale Apache Tomcat 6.0 o posterior.
- Instale Rational Application Developer y cree una aplicación web Java EE.

Procedimiento

1. Añada la biblioteca de tiempo de ejecución de WebSphere eXtreme Scale a su vía de acceso de compilación de Java EE.

Aplicación cliente En este caso, desea configurar y ejecutar una aplicación web para utilizar un cliente WebSphere eXtreme Scale en Rational Application Developer.

- a. **Ventana** → **Preferencias** → **Java** → **Vía de acceso de de compilación** → **Bibliotecas de usuario**. Pulse **Nueva**.
- b. Introduzca un **Nombre de biblioteca de usuario** de `eXtremeScaleClient` y pulse **Aceptar**.
- c. Pulse **Añadir Jars...** y navegue hasta el archivo `wxs_home/lib/ogclient.jar` y selecciónelo. Pulse **Abrir**.
- d. Opcional: (Opcional) Para añadir un Javadoc, seleccione su ubicación y pulse **Editar...** En la vía de acceso de ubicación del Javadoc, puede introducir el URL de la documentación de la API o puede descargar la documentación de la API.
 - Para utilizar la documentación de la API en línea, introduzca `http://www.ibm.com/developerworks/wikis/extremescale/docs/api/` en la vía de acceso de ubicación del Javadoc.
 - Para descargar la documentación de la API, vaya a la página de descargas WebSphere eXtreme Scale API documentation. En la vía de acceso de ubicación del Javadoc, introduzca su ubicación de descarga local.
- e. Pulse **Aceptar**.
- f. Pulse **Aceptar** para cerrar el diálogo Bibliotecas de usuario.
- g. Pulse **Proyecto** → **Propiedades**.
- h. Pulse en **Vía de acceso de compilación**.

- i. Pulse **Añadir biblioteca**.
 - j. Seleccione **Biblioteca de usuario**. Pulse **Siguiente**.
 - k. Seleccione la biblioteca **eXtremeScaleClient** y pulse **Finalizar**.
 - l. Pulse **Aceptar** para cerrar el diálogo **Propiedades del proyecto**.
- Aplicación de servidor En este caso, desea configurar y ejecutar una aplicación web para ejecutar un servidor WebSphere eXtreme Scale incorporado en Rational Application Developer.
- a. Pulse **Ventana** → **Preferencias** → **Java** → **Vía de acceso de de compilación** → **Bibliotecas de usuario**. Pulse **Nueva**.
 - b. Introduzca un **Nombre de biblioteca de usuario** de eXtremeScale y pulse **Aceptar**.
 - c. Pulse **Añadir Jars...** y navegue hasta `wxs_home/lib/objectgrid.jar` y selecciónelo. Pulse **Abrir**.
 - d. (Opcional) Para añadir un Javadoc, seleccione la ubicación del Javadoc y pulse **Editar...** En la vía de acceso de ubicación del Javadoc, introduzca `http://www.ibm.com/developerworks/wikis/extremescale/docs/api/`.
 - e. Pulse **Aceptar**.
 - f. Pulse **Aceptar** para cerrar el diálogo Bibliotecas de usuario.
 - g. Pulse **Proyecto** → **Propiedades**.
 - h. Pulse en **Vía de acceso de compilación**.
 - i. Pulse **Añadir biblioteca**.
 - j. Seleccione **Biblioteca de usuario**. Pulse **Siguiente**.
 - k. Seleccione la biblioteca **eXtremeScaleClient** y pulse **Finalizar**.
 - l. Pulse **Aceptar** para cerrar el diálogo **Propiedades del proyecto**.
2. Defina Tomcat Server para el proyecto.
 - a. Asegúrese de que se encuentra en la perspectiva J2EE y pulse en el separador **Servidores** en el panel inferior. Puede pulsar también **Ventana** → **Mostrar vista** → **Servidores**.
 - b. Pulse el botón derecho del ratón en el panel Servidores y seleccione **Nuevo** → **Servidor**.
 - c. Seleccione **Apache, Tomcat v6.0 Server**. Pulse **Siguiente**.
 - d. Pulse **Examinar..**, navegue hasta `tomcat_root` y selecciónelo. Pulse **Aceptar**.
 - e. Pulse **Siguiente**.
 - f. Seleccione la aplicación Java EE en el panel Disponible de la izquierda y pulse **Añadir** > para moverla al panel Configurado de la derecha en el servidor, seguidamente pulse **Finalizar**.
 3. Resuelva los errores restantes del proyecto. Todos los errores del proyecto deben ir en el panel Problemas. Si no es así, realice estos pasos.
 - a. Pulse **Proyecto** → **Limpiar** → *nombre_proyecto*. Pulse **Aceptar**. Compile el proyecto.
 - b. Pulse el botón derecho del ratón en el proyecto Java EE deseado y seleccione **Vía de acceso de compilación** → **Configurar vía de acceso de compilación**.
 - c. Pulse el separador **Bibliotecas**. Asegúrese de que la vía de acceso está configurada correctamente:
 - **Para las aplicaciones cliente:** asegúrese de que Apache Tomcat, eXtremeScaleClient y Java 1.5 JRE se encuentran en la vía de acceso.
 - **Para las aplicaciones de servidor:** asegúrese de que Apache Tomcat, eXtremeScale y Java 1.5 JRE se encuentran en la vía de acceso.

4. Cree una configuración de ejecución para ejecutar su aplicación.
 - a. En el menú **Ejecutar** seleccione **Ejecutar configuraciones**.
 - b. Pulse el botón derecho del ratón en la categoría Aplicación Java y seleccione **Nueva**.
 - c. Seleccione la nueva configuración de ejecución, denominada *Nueva_configuración*.
 - d. Configure el perfil.
 - **Proyecto** (en la página tabulada principal): *su_nombre_proyecto*
 - **Clase principal** (en la página tabulada principal): *su_clase_principal*
 - **Argumentos de VM** (en la página tabulada de argumentos):
-Djava.endorsed.dirs=raíz_wxs/lib/endorsed

Suelen ocurrir problemas con los **Argumentos VM** porque la vía de acceso de `java.endorsed.dirs` debe ser absoluta sin variables ni atajos.

Otros problemas comunes de configuración están relacionados con el intermediario de solicitud de objetos (ORB). Podría aparecer el error siguiente. Consulte Configuración de un intermediario de solicitud de objetos personalizado para obtener más información:

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

Si no tiene los archivos `objectGrid.xml` o `deployment.xml` accesibles para la aplicación, podría aparecer el error siguiente:

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

5. Pulse **Aplicar** y cierre la ventana o pulse **Ejecutar**.

Qué hacer a continuación

Si ha configurado y ejecutado una aplicación web para utilizar un cliente WebSphere eXtreme Scale en Rational Application Developer, ahora desarrolle un servlet que utiliza las API de WebSphere eXtreme Scale para almacenar y recuperar los datos de una cuadrícula WebSphere eXtreme Scale remota.

Si ha habilitado una aplicación Java EE en la interfaz de Rational Application Developer con una instalación autónoma de WebSphere eXtreme Scale, ahora desarrolle un servlet que utiliza las API de sistema de WebSphere eXtreme Scale para iniciar y detener los servicios de catálogo.

Ejecución de una aplicación de servidor o cliente integrada con WebSphere Application Server en Rational Application Developer

Configure y ejecute una aplicación Java EE con un cliente o servidor WebSphere eXtreme Scale con el módulo ejecutable de WebSphere Application Server incorporado en Rational Application Developer. Si va a configurar un servidor, si inicia WebSphere Application Server se inicia automáticamente WebSphere eXtreme Scale.

Antes de empezar

Los pasos siguientes son para WebSphere Application Server Versión 7.0 con Rational Application Developer Versión 7.5. Los pasos siguientes podrían variar si utiliza versiones distintas de estos productos.

Instalar Rational Application Developer con ampliaciones del entorno de prueba de WebSphere Application Server.

Instalar el cliente o el servidor WebSphere eXtreme Scale en el entorno de prueba de WebSphere Application Server, Versión 7.0 en el directorio *inicio_rad*\runtimes\base_v7.

Procedimiento

1. Defina el servidor eXtreme Scale integrado con WebSphere Application Server para su proyecto.
 - a. En la perspectiva J2EE, pulse **Ventana > Mostrar vista > Servidores**.
 - b. Pulse el botón derecho del ratón en el panel **Servidores**. Seleccione **Nuevo > Servidor**.
 - c. Seleccione **IBM WebSphere Application Server v7.0**. Pulse **Siguiente**.
 - d. Seleccione un perfil que desea utilizar. El valor predeterminado es `was70profile1`.
 - e. Introduzca el nombre de servidor. El valor predeterminado es `server1`.
 - f. Pulse **Siguiente**.
 - g. Seleccione la aplicación Java EE en el panel **Disponible**. Pulse **Añadir >** para moverla al panel **Configurado** en el servidor. Pulse **Finalizar**.
2. Para ejecutar la aplicación Java EE, inicie el servidor de aplicaciones. Pulse el botón derecho del ratón en **WebSphere Application Server v7.0** y seleccione **Iniciar**.

Capítulo 3. Acceso a los datos con aplicaciones cliente

Después de configurar el entorno de desarrollo, puede comenzar a desarrollar aplicaciones que crean, acceden y gestionan los datos de la cuadrícula de datos.

Acerca de esta tarea

Desde la perspectiva de una aplicación cliente, el uso de WebSphere eXtreme Scale lleva consigo los pasos siguientes:

- Conexión al servicio de catálogo obteniendo una instancia de `ClientClusterContext`.
- Obtención de una instancia `ObjectGrid` cliente.
- Obtención de una instancia `Session`.
- Obtención de una instancia `ObjectMap`.
- Uso de los métodos `ObjectMap`.

Interfaz `ObjectGrid`

Los siguientes métodos le permiten interactuar con una instancia de `ObjectGrid`.

Crear e inicializar

Consulte el tema sobre la interfaz `ObjectGridManager` para obtener información sobre los pasos necesarios para crear una instancia `ObjectGrid`. Existen dos métodos distintos para crear una instancia `ObjectGrid`: mediante programa o mediante configuración de los archivos XML. Consulte la documentación de la API si desea más información.

Get o set y métodos de fábrica

Antes de inicializar la instancia `ObjectGrid`, debe llamarse a los métodos `set`. Si llama a un método `set` después de que se llame al método `initialize`, se genera una excepción `java.lang.IllegalStateException`. Cada uno de los métodos `getSession` de la interfaz `ObjectGrid` llaman implícitamente al método `initialize`. Por lo tanto, debe llamar a los métodos `set` antes de llamar a los métodos `getSession`. La única excepción a esta regla se produce al establecer, añadir y eliminar los objetos `ObjectGridEventListener`. Estos objetos pueden procesarse después de terminado el proceso de inicialización.

La interfaz `ObjectGrid` contiene los siguientes métodos principales.

Tabla 1. Interfaz `ObjectGrid`. Métodos principales de `ObjectGrid`.

Método	Descripción
<code>BackingMap defineMap(String name);</code>	<code>defineMap</code> : es un método de fábrica para definir un <code>BackingMap</code> de nombre exclusivo. Para obtener más información sobre las correlaciones de respaldo, consulte la interfaz <code>BackingMap</code> .
<code>BackingMap getMap(String name);</code>	<code>getMap</code> : devuelve un objeto <code>BackingMap</code> previamente definido al llamar a <code>defineMap</code> . Al usar este método, puede configurar <code>BackingMap</code> , si no se ha configurado ya a través de la configuración del XML.
<code>BackingMap createMap(String name);</code>	<code>createMap</code> : crea un objeto <code>BackingMap</code> , pero no lo almacena en caché para que lo use este <code>ObjectGrid</code> . Utilice este método con el método <code>setMaps(List)</code> de la interfaz <code>ObjectGrid</code> , que almacena en la memoria caché <code>BackingMaps</code> para ser utilizados con este <code>ObjectGrid</code> . Utilice estos métodos al configurar un objeto <code>ObjectGrid</code> con la infraestructura Spring.

Tabla 1. Interfaz ObjectGrid (continuación). Métodos principales de ObjectGrid.

Método	Descripción
void setMaps(List mapList);	setMaps: borra los objetos BackingMap que se hayan definido previamente en este ObjectGrid y los sustituye por la lista de BackingMaps proporcionados.
public Session getSession() throws ObjectGridException, TransactionCallbackException;	getSession: devuelve un objeto Session, que proporciona funcionalidad de inicio, confirmación y retrotracción para una unidad de trabajo. Para obtener más información sobre los objetos Session, consulte la interfaz Session.
Session getSession(CredentialGenerator cg);	getSession(CredentialGenerator cg): obtiene una sesión con un objeto CredentialGenerator. Sólo puede llamar a este método un cliente ObjectGrid en un entorno cliente/servidor.
Session getSession(Subject subject);	getSession(Subject subject): permite el uso de un objeto Subject específico en lugar del configurado en ObjectGrid para obtener un objeto Session.
void initialize() throws ObjectGridException;	initialize: se inicializa ObjectGrid y está disponible para su uso general. Este método se llama de forma implícita cuando se llama al método getSession, si ObjectGrid no está en una fase inicializada.
void destroy();	destroy: la infraestructura se desensambla y no se puede utilizar después de llamar a este método.
void setTxTimeout(int timeout);	setTxTimeout: utilice este método para establecer la cantidad de tiempo, en segundos, que tardará en completarse una transacción iniciada por una sesión que creó esta instancia ObjectGrid. Si una transacción no se completa en el tiempo especificado, el objeto Session que inició la transacción se marca como "tiempo de espera excedido". Marcar un objeto Session como con el tiempo de espera excedido provoca que el siguiente método ObjectMap invocado por el Session de tiempo de espera genere una excepción. El objeto se marca sólo como retrotracción, lo cual provoca que la transacción se retrotraiga aunque la aplicación llame al método commit en lugar de al método rollback después de que la excepción TransactionTimeoutException haya sido captada por la aplicación. Un valor de tiempo de espera de 0 indica que a la transacción se le permite un tiempo ilimitado para completarse. Si se utiliza el valor de 0, no se producirá tiempo de espera excedido por parte de la transacción. Si no se llama a este método, cualquier objeto Session devuelto por el método getSession de esta interfaz tiene un valor predeterminado de tiempo de espera de la transacción establecido en 0. Una aplicación puede alterar temporalmente el valor de tiempo de espera de la transacción por objeto Session mediante el uso del método setTransactionTimeout de la interfaz com.ibm.websphere.objectgrid.Session. También puede configurar el tiempo de espera de transacción con el archivo objectGrid.xml en el caso distribuido.
int getTxTimeout();	getTxTimeout: devuelve el valor del tiempo de espera de la transacción en segundos. Este método devuelve el mismo valor pasado como parámetro de tiempo de espera en el método setTxTimeout. Si no se llama al método setTxTimeout, el método devuelve 0 para indicar que a la transacción se le permite un tiempo ilimitado para completarse.
public int getObjectGridType();	Devuelve el tipo de ObjectGrid. El valor devuelto equivale a una de las constantes declaradas en este interfaz: LOCAL, SERVER o CLIENT.
public void registerEntities(Class[] entities);	Registra una o más entidades basadas en los metadatos de la clase. El registro de entidades debe realizarse antes de la inicialización de ObjectGrid para enlazar un objeto Entity con BackingMap y cualquier índice definido. Este método puede llamarse varias veces.
public void registerEntities(URL entityXML);	Registra una o más entidades de un archivo XML de entidad. El registro de entidades debe realizarse antes de la inicialización de ObjectGrid para enlazar un objeto Entity con BackingMap y cualquier índice definido. Este método puede llamarse varias veces.
void setQueryConfig(QueryConfig queryConfig);	Establece el objeto QueryConfig para este ObjectGrid. Un objeto QueryConfig proporciona configuraciones de consulta para ejecutar consultas de objetos en las correlaciones de este ObjectGrid.
//Palabras clave	
void associateKeyword(Serializable parent, Serializable child);	En desuso: utilice la función de índice o consultas para obtener objetos con atributos específicos. El método associateKeyword proporciona un mecanismo de invalidación flexible basado en palabras clave. Este método enlaza las dos palabras clave en una relación direccional. Si el padre se invalida, el hijo también. La invalidación del hijo no afecta al padre.
//Seguridad	
void setSecurityEnabled();	setSecurityEnabled: habilita la seguridad. La seguridad está inhabilitada de manera predeterminada.

Tabla 1. Interfaz ObjectGrid (continuación). Métodos principales de ObjectGrid.

Método	Descripción
void setPermissionCheckPeriod(long period);	setPermissionCheckPeriod: este método acepta un solo parámetro que indica la frecuencia con la que debe comprobarse el permiso utilizado para permitir el acceso de un cliente. Si el parámetro es 0, todos los métodos solicitan el mecanismo de autorización, ya sea autorización JAAS o autorización personalizada, para comprobar si el sujeto actual tiene permiso. Puede que esta estrategia provoque problemas de rendimiento en función de la implementación de la autorización. No obstante, este tipo de autorización está disponible si se prefiere. De forma alternativa, si el parámetro es menor que 0, indica el número de milisegundos para almacenar en memoria caché un conjunto de permisos antes de volver al mecanismo de autorización para que los actualice. Con este parámetro mejora el rendimiento, pero si se modifican los permisos del programa de fondo durante este tiempo, ObjectGrid puede permitir o prohibir el acceso aunque se haya modificado el proveedor de seguridad de programa de fondo.
void setAuthorizationMechanism(int authMechanism);	setAuthorizationMechanism: establece el mecanismo de autorización. El valor predeterminado es SecurityConstants.JAAS_AUTHORIZATION.
setMapAuthorization(MapAuthorization ma);	En desuso: utilice el método setObjectGridAuthorization (ObjectGridAuthorization) en lugar de conectar las autorizaciones personalizadas. setMapAuthorization: establece el plug-in MapAuthorization para esta instancia ObjectGrid. Este plug-in se puede utilizar para autorizar los accesos de ObjectMap o JavaMap a los principales contenidos en el objeto Subject. Una implementación típica de este plug-in es recuperar los principales del objeto Subject y después comprobar si se han concedido los permisos especificados a los principales.
setSubjectSource(SubjectSource ss);	setSubjectSource: establece el plug-in SubjectSource. Este plug-in puede utilizarse para obtener un objeto Subject que represente el cliente ObjectGrid. Este objeto Subject se utiliza para la autorización de ObjectGrid. El tiempo de ejecución de ObjectGrid llama al método SubjectSource.getSubject cuando se utiliza el método ObjectGrid.getSession para obtener una sesión y se ha habilitado la seguridad. Este plug-in resulta útil para un cliente ya autenticado: puede recuperar un objeto Subject autenticado y pasar después a la instancia ObjectGrid. No se necesita otra autenticación.
setSubjectValidation(SubjectValidation sv);	setSubjectValidation: establece el plug-in SubjectValidation para esta instancia ObjectGrid. Este plug-in puede utilizarse para validar que un sujeto javax.security.auth.Subject pasado a ObjectGrid es un sujeto válido que no se ha manipulado de forma indebida. Una implementación de este plug-in necesita el soporte del creador del objeto Subject porque sólo el creador sabe si el objeto Subject ha sido manipulado indebidamente. Puede darse el caso, no obstante, de que el creador no sepa si el objeto Subject se ha manipulado de forma indebida. De ser así, este plug-in no debería utilizarse.
void setObjectGridAuthorization(ObjectGrid Authorization ogAuthorization);	Establece ObjectGridAuthorization para esta instancia ObjectGrid. Si se pasan valores nulos a este método, se elimina un objeto ObjectGridAuthorization ya establecido de una invocación anterior de este método, e indica que este <code><code>ObjectGrid</code></code> no está asociado a ningún objeto ObjectGridAuthorization. Este método sólo debe utilizarse cuando se ha habilitado la seguridad ObjectGrid. Si la seguridad ObjectGrid está inhabilitada, no se utilizará el objeto ObjectGridAuthorization proporcionado. Puede utilizarse un plug-in ObjectGridAuthorization para autorizar el acceso a ObjectGrid y las correlaciones. Desde XD 6.1, setMapAuthorization está en desuso y se recomienda el uso de setObjectGridAuthorization. No obstante, si se utilizan los plug-in MapAuthorization y ObjectGridAuthorization, ObjectGrid utilizará el plug-in MapAuthorization proporcionado para autorizar los accesos de las correlaciones, aunque esté en desuso.

Interfaz ObjectGrid: plug-ins

La interfaz ObjectGrid tiene varios puntos de plug-in opcionales para interacciones más ampliables.

```
void addEventListener(ObjectGridEventListener cb);
void setEventListeners(List cbList);
void removeEventListener(ObjectGridEventListener cb);
void setTransactionCallback(TransactionCallback callback);
int reserveSlot(String);
// Plug-ins relacionados con la seguridad
void setSubjectValidation(SubjectValidation subjectValidation);
void setSubjectSource(SubjectSource source);
void setMapAuthorization(MapAuthorization mapAuthorization);
```

- ObjectGridEventListener: se utiliza una interfaz ObjectGridEventListener para recibir notificaciones cuando se producen sucesos significativos en ObjectGrid.

Estos sucesos pueden ser inicialización de ObjectGrid, inicio de una transacción, fin de una transacción y destrucción de ObjectGrid. Para escuchar estos sucesos, cree una clase que implemente la interfaz ObjectGridEventListener y la añada a ObjectGrid. Estas escuchas se asocian con cada objeto Session. Consulte el tema que trata sobre escuchas y la interfaz Session para obtener más información.

- TransactionCallback: una interfaz de escucha TransactionCallback permite que se envíen sucesos transaccionales como señales de inicio, confirmación y retrotracción a esta interfaz. Por norma, se suele utilizar una interfaz de escucha TransactionCallback con un cargador. Para obtener más información, consulte el tema que trata sobre el plug-in TransactionCallback y los cargadores. Estos sucesos pueden utilizarse para coordinar transacciones con un recurso externo o dentro de varios cargadores.
- reserveSlot: permite que los plug-ins de este ObjectGrid reserven ranuras para su uso en instancias de objetos que tienen ranuras como TxID.
- SubjectValidation: si se habilita la seguridad, este plug-in puede utilizarse para validar una clase javax.security.auth.Subject que se pasa a ObjectGrid.
- MapAuthorization: si se habilita la seguridad, este plug-in puede utilizarse para autorizar los accesos de ObjectMap a los principales que se representan en el objeto Subject.
- SubjectSource: si se habilita la seguridad, este plug-in puede utilizarse para obtener un objeto Subject que represente el cliente ObjectGrid. A continuación, este objeto Subject se utiliza para la autorización de ObjectGrid.

Si desea más información sobre los plug-ins, consulte la introducción de los plug-ins en *Guía de programación*.

Interfaz BackingMap

Cada instancia de ObjectGrid contiene una colección de objetos BackingMap. Utilice el método defineMap o el método createMap de la interfaz ObjectGrid para nombrar y añadir cada BackingMap a una instancia de ObjectGrid. Estos métodos devuelven una instancia BackingMap que se utiliza después para definir el comportamiento de una correlación individual. Un objeto BackingMap puede considerarse como una memoria caché en memoria de datos confirmados para una correlación individual.

Interfaz Session

La interfaz Session se utiliza para iniciar una transacción y obtener el objeto ObjectMap o JavaMap necesario para realizar la interacción transaccional entre una aplicación y un objeto BackingMap. No obstante, los cambios de la transacción no se aplican al objeto BackingMap hasta que se confirme la transacción. Un objeto BackingMap puede considerarse como una memoria caché en memoria de datos confirmados para una correlación individual. Si desea más información, consulte la información sobre cómo utilizar Sessions para acceder a los datos de *Guía de programación*.

La interfaz BackingMap proporciona métodos para establecer los atributos BackingMap. Algunos de estos métodos permiten la ampliación de un objeto BackingMap a través de diversos plug-ins de diseño personalizado. Consulte la lista siguiente de los métodos set para definir atributos y proporcionar soporte de plug-ins de diseño personalizado:

```
// Para establecer atributos de
BackingMap.
public void setReadOnly(boolean readOnlyEnabled);
```

```

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

// Para establecer un plug-in personalizado y opcional que proporciona la
// aplicación.
public abstract void setObjectTransformer(ObjectTransformer t);
public abstract void setOptimisticCallback(OptimisticCallback checker);
public abstract void setLoader(Loader loader);
public abstract void setPreloadMode(boolean async);
public abstract void setEvictor(Evictor e);
public void setMapEventListeners( List /*MapEventListener*/ eventListenerList );
public void addMapEventListener(MapEventListener eventListener );
public void removeMapEventListener(MapEventListener eventListener );
public void addMapIndexPlugin(MapIndexPlugin index);
public void setMapIndexPlugins(List /*MapIndexPlugin */ indexList );
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb);
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback cb);
public void removeDynamicIndex(String name);

```

Para cada uno de los métodos set listados, existe un método get correspondiente.

Atributos de BackingMap

Cada BackingMap tiene los siguientes atributos que pueden establecerse para modificar o controlar el comportamiento del objeto BackingMap:

- **ReadOnly:** este atributo indica si la correlación es una correlación de sólo lectura o una correlación de lectura y grabación. Si este atributo no se establece, la correlación toma el valor predeterminado de lectura y grabación. Cuando un objeto BackingMap se establece para sólo lectura, ObjectGrid optimiza el rendimiento para sólo lectura siempre que sea posible.
- **NullValuesSupported:** este atributo indica si se puede asignar un valor nulo a la correlación. Si no se establece este atributo, la correlación no admite valores nulos. Si la correlación admite valores nulos, una operación get que devuelve un valor nulo significa que o bien el valor es nulo o bien la correlación no contiene la clave especificada por la operación get.
- **LockStrategy:** este atributo determina si este objeto BackingMap utiliza un gestor de bloqueos. Si se utiliza, el atributo LockStrategy se usa para indicar si se emplea un procedimiento de bloqueo optimista o pesimista para bloquear las entradas de correlación. Si no se establece este atributo, se utiliza la estrategia de bloqueo optimista. Consulte el tema sobre bloqueos para obtener más información sobre las estrategias de bloqueo permitidas.
- **CopyMode:** este atributo determina si BackingMap realiza una copia de un objeto de valor cuando se lee un valor de la correlación o éste se coloca en el objeto BackingMap durante el ciclo de confirmación de una transacción. Se admiten diversas modalidades de copia que permiten a la aplicación conseguir el equilibrio entre rendimiento e integridad de datos. Si no se establece el atributo, se utiliza la modalidad de copia COPY_ON_READ_AND_COMMIT. Esta modalidad de copia no tiene el mejor rendimiento, pero sí la mejor protección frente a problemas de integridad de los datos. Si BackingMap está asociado a una entidad de API EntityManager, el valor de CopyMode no tiene

ningún efecto, a menos que el valor se establezca en `COPY_TO_BYTES`. Si se define cualquier otro `CopyMode`, siempre estará establecido en `NO_COPY`. Para alterar temporalmente el valor de `CopyMode` para una correlación de entidad, utilice el método `ObjectMap.setCopyMode`. Si desea más información sobre las modalidades de copia, consulte la información sobre los procedimientos recomendados del método en *Guía de programación*.

- `CopyKey`: este atributo determina si `BackingMap` realiza una copia de un objeto de clave cuando se crea por primera vez una entrada en la correlación. De manera predeterminada, no se realiza ninguna copia de objetos de clave porque las claves suelen ser objetos que no se pueden cambiar.
- `NumberOfBuckets`: este atributo indica el número de grupos hash que `BackingMap` utiliza. La implementación de `BackingMap` utiliza una correlación hash para su implementación. Si existen muchas entradas en el objeto `BackingMap`, la existencia de más grupos mejora el rendimiento. El número de claves que tienen el mismo grupo disminuye a medida que crece el número de grupos. Un número mayor de grupos también implica mayor simultaneidad. Este atributo es útil para el ajuste del rendimiento. Se utiliza un valor predeterminado no definido si la aplicación no establece el atributo `NumberOfBuckets`.
- `NumberOfLockBuckets`: este atributo indica el número de grupos de bloqueo que utiliza el gestor de bloqueos para este objeto `BackingMap`. Cuando `LockStrategy` se establece en `OPTIMISTIC` o `PESSIMISTIC`, se crea un gestor de bloqueos para `BackingMap`. El gestor de bloqueos utiliza una correlación hash para realizar un seguimiento de las entradas bloqueadas por una o más transacciones. Si existen muchas entradas en la correlación hash, un número mayor de grupos de bloqueo mejora el rendimiento porque el número de claves que tienen el mismo grupo disminuye a medida que crece el número de grupos. Un número mayor de grupos de bloqueo también implica mayor simultaneidad. Si el atributo `LockStrategy` se establece en `NONE`, este objeto `BackingMap` no utiliza ningún gestor de bloqueos. En este caso, establecer `NumberOfLockBuckets` no tiene ningún efecto. Si no se establece este atributo, se utiliza un valor no definido.
- `LockTimeout`: este atributo se utiliza cuando `BackingMap` usa un gestor de bloqueos. `BackingMap` utiliza un gestor de bloqueos cuando el atributo `LockStrategy` está establecido en `OPTIMISTIC` o `PESSIMISTIC`. El valor del atributo se especifica en segundos y determina cuánto tiempo espera el gestor de bloqueos a que se otorgue un bloqueo. Si este atributo no está definido, se utilizan 15 segundos como el valor de `LockTimeout`. Consulte el tema que trata sobre el bloqueo pesimista para obtener más información sobre las excepciones de tiempo de espera de bloqueo que pueden producirse.
- `TtlEvictorType`: cada `BackingMap` tiene su propio desalojador de tiempo de vida incorporado que utiliza un algoritmo basado en la hora para determinar las entradas de la correlación que debe desalojar. De manera predeterminada, el desalojador de tiempo de vida incorporado no está activo. Para activarlo, llame al método `setTtlEvictorType` con uno de los estos valores: `CREATION_TIME`, `LAST_ACCESS_TIME`, `LAST_UPDATE_TIME` o `NONE`. Un valor `CREATION_TIME` indica que el desalojador añade el atributo `TimeToLive` a la hora en que se creó la entrada de correlación en `BackingMap` para determinar cuándo debe el desalojador desalojar la entrada de correlación del objeto `BackingMap`. Un valor `LAST_ACCESS_TIME` (o `LAST_UPDATE_TIME`) indica que el desalojador añade el atributo `TimeToLive` a la hora en la que una transacción, que ejecuta la aplicación, accedió (o accedió y actualizó) por última vez a la entrada de correlación para determinar cuándo debe el desalojador desalojar la entrada de correlación. La entrada de correlación sólo se desaloja si ninguna transacción accede a una entrada de correlación durante un período de

tiempo especificado por el atributo `TimeToLive`. Un valor `NONE` indica que el desalojador debe permanecer inactivo y no desalojar nunca ninguna de las entradas de correlación. Si no se establece este atributo, `NONE` se utiliza como el valor predeterminado y el desalojador de tiempo de vida no está activo. Consulte el tema sobre los desalojadores para obtener más información sobre el desalojador de tiempo de vida incorporado.

- **TimeToLive:** este atributo se utiliza para especificar el número de segundos que necesita el desalojador de tiempo de vida incorporado para añadir al tiempo de creación o de acceso por última vez de cada entrada, como se ha descrito en el atributo `TtlEvictorType`. Si no se establece este atributo, se utiliza el valor especial de cero para indicar que el tiempo de vida es infinito. Si este atributo se establece en infinito, el desalojador nunca desaloja las entradas de correlación.

El ejemplo siguiente muestra cómo definir `someMap` `BackingMap` en la instancia `someGrid` `ObjectGrid` y establecer varios atributos de `BackingMap` mediante los métodos `set` de la interfaz `BackingMap`:

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

...

ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("someGrid");
BackingMap bm = objectGrid.getMap("someMap");
bm.setReadOnly( true ); // alterar temporalmente el valor predeterminado de
                        // lectura/grabación
bm.setNullValuesSupported(false); // alterar temporalmente el valor
                        // predeterminado de permitir valores nulos
bm.setLockStrategy( LockStrategy.PESSIMISTIC ); // alterar temporalmente
                        // el valor predeterminado de OPTIMISTIC
bm.setLockTimeout( 60 ); // alterar temporalmente el valor predeterminado
                        // de 15 segundos.
bm.setNumberOfBuckets(251); // alterar temporalmente el valor predeterminado
                        // (números primos funcionan mejor)
bm.setNumberOfLockBuckets(251); // alterar temporalmente el valor
                        // predeterminado (números primos funcionan mejor)
```

Plug-ins de BackingMap

La interfaz `BackingMap` tiene diversos plug-ins opcionales que amplían las interacciones con `ObjectGrid`:

- **Plug-in ObjectTransformer:** para algunas operaciones de correlación, `BackingMap` podría necesitar serializar, deserializar o copiar una clave o valor de una entrada en el objeto `BackingMap`. `BackingMap` puede realizar estas acciones; para ello, proporcione una implementación predeterminada de la interfaz `ObjectTransformer`. Una aplicación puede mejorar el rendimiento si proporciona un plug-in `ObjectTransformer` de diseño personalizado que `BackingMap` utilice para serializar, deserializar o copiar una clave o valor de una entrada en `BackingMap`. Consulte la información sobre el plug-in `ObjectTransformer` en *Guía de programación* si desea más información.
- **Plug-in Evictor:** el desalojador de tiempo de vida incorporado utiliza un algoritmo basado en la hora para decidir cuándo se debe desalojar una entrada en `BackingMap`. Algunas aplicaciones podrían necesitar un algoritmo diferente para decidir cuándo debe desalojarse una entrada en `BackingMap`. El plug-in `Evictor` pone a disposición de `BackingMap` un desalojador de diseño personalizado. El plug-in `Evictor` se ofrece además del desalojador de tiempo de vida incorporado. No lo reemplaza. `ObjectGrid` proporciona un plug-in `Evictor`

personalizado que implementa los algoritmos conocidos de "menos utilizado recientemente" o "utilizado con menor frecuencia". Las aplicaciones pueden conectar uno de los plug-ins Evictor suministrados o pueden proporcionar su propio plug-in Evictor. Consulte la información sobre el desalojo en *Guía de programación*.

- **Plug-in MapEventListener:** es posible que una aplicación desee conocer los sucesos de BackingMap como, por ejemplo, un desalojo de una entrada de correlación o una precarga de una terminación de BackingMap. BackingMap llama a los métodos del plug-in MapEventListener para informar de sucesos de BackingMap a una aplicación. Una aplicación puede recibir una notificación de varios sucesos de BackingMap mediante el método setMapEventListener para proporcionar uno o más plug-ins MapEventListener de diseño personalizado a BackingMap. La aplicación puede modificar los objetos MapEventListener listados utilizando el método addMapEventListener o el método removeMapEventListener. Consulte la información sobre el plug-in MapEventListener en *Guía de programación* si desea más información.
- **Plug-in Loader:** BackingMap es una memoria caché en memoria de una correlación. Un plug-in Loader es una opción que utiliza BackingMap para mover los datos entre la memoria y un almacén persistente. Por ejemplo, se puede utilizar un cargador JDBC (Java database connectivity) para insertar y extraer datos de BackingMap y una o más tablas relacionales de una base de datos relacional. Una base de datos relacional no necesita utilizarse como almacén persistente de un objeto BackingMap. El cargador también se puede utilizar para mover datos entre BackingMap y un archivo, entre BackingMap y una correlación Hibernate, entre BackingMap y un bean de entidad de Java 2 Platform, Enterprise Edition (JEE), entre BackingMap y otro servidor de aplicaciones, etc. La aplicación debe proporcionar un plug-in Loader de diseño personalizado para mover datos entre BackingMap y el almacén persistente de cada tecnología que se use. Si no se proporciona un cargador, BackingMap se convierte en una sencilla memoria caché en memoria. Consulte la información sobre cómo utilizar un cargador en *Guía de programación* si desea más información.
- **Plug-in OptimisticCallback:** cuando el atributo LockStrategy para BackingMap está establecido en OPTIMISTIC, BackingMap o un plug-in Loader debe realizar operaciones de comparación para los valores de la correlación. BackingMap y Loader utilizan el plug-in OptimisticCallback para realizar las operaciones de comparación de versiones optimistas. Consulte la información sobre el plug-in OptimisticCallback en *Guía de programación* si desea más información.
- **Plug-in MapIndexPlugin:** un plug-in MapIndexPlugin, o un Index, es una opción utilizada por BackingMap para crear un índice que se basa en el atributo especificado del objeto almacenado. El índice permite a la aplicación buscar objetos por un valor específico o un intervalo de valores. Existen dos tipos de índice: estático y dinámico. Consulte el tema que trata sobre los índices para obtener más información.

Si desea más información en relación a los plug-ins, consulte la introducción a los plug-ins en *Guía de programación*.

Conexión a un ObjectGrid distribuido

Puede conectarse a un ObjectGrid distribuido con un punto final de conexión para el servicio de catálogo. Debe tener el nombre de host y el puerto de punto final del servidor de catálogo al que desea conectarse.

Para poder conectarse a una cuadrícula distribuida, debe haber configurado el entorno del lado del servidor con un servicio de catálogo y servidores de contenedor.

El método `getObjectGrid(ClientClusterContext ccc, String objectGridName)` se conecta al servicio de catálogo especificado y devuelve una instancia de `ObjectGrid` cliente correspondiente a una instancia `ObjectGrid` del lado del servidor.

El siguiente fragmento de código es un ejemplo de cómo conectarse a una cuadrícula distribuida.

```
// Crear una instancia ObjectGridManager.

ObjectGridManager ogm = ObjectGridManagerFactory.getObjectGridManager();

// Obtener un ClientClusterContext al conectar un
// ObjectGrid distribuido basado en servidor. Debe proporcionar
// un punto final de conexión para el servidor de catálogo con el formato
// de nombrehost:puertopuntofinal. El nombrehost es la máquina donde
// reside el servidor de catálogo y el puertopuntofinal es el puerto
// de escucha del servidor de catálogo, cuyo valor predeterminado es 2809.

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

// Obtener un ObjectGrid distribuido utilizando ObjectGridManager y
// proporcionando el ClientClusterContext.

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

Interacción con un ObjectGrid utilizando ObjectGridManager

La clase `ObjectGridManagerFactory` y la interfaz `ObjectGridManager` proporcionan un mecanismo para crear, acceder y almacenar en memoria caché instancias `ObjectGrid`. La clase `ObjectGridManagerFactory` es una clase ayudante estática que sirve para acceder a la interfaz `ObjectGridManager`, un singleton. La interfaz `ObjectGridManager` contiene varios métodos de simplificación para crear instancias de un objeto `ObjectGrid`. Además esta interfaz facilita la creación de las instancias `ObjectGrid` y su almacenamiento en memoria caché. Varios usuarios pueden acceder a estas instancias.

Modelo de programación

Antes de utilizar las funciones de eXtreme Scale como una cuadrícula de datos en memoria, debe crear e interactuar con instancias de `ObjectGrid` con métodos como, por ejemplo, los siguientes.

- Métodos `createObjectGrid`
- Métodos `getObjectGrid`
- Métodos `removeObjectGrid`
- Control del ciclo de vida de un objeto `ObjectGrid`

Métodos `createObjectGrid`

En este tema se describen los siete métodos `createObjectGrid` de la interfaz `ObjectGridManager`. Cada uno de estos métodos crea una instancia local de un `ObjectGrid`.

Instancia local en memoria

El siguiente fragmento de código ilustra cómo obtener y configurar una instancia local de ObjectGrid con eXtreme Scale.

```
// Obtener una referencia ObjectGrid local
// puede crear un ObjectGrid nuevo o obtener uno configurado
// definido en el archivo ObjectGrid.xml
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid ivObjectGrid =
objectGridManager.createObjectGrid("objectgridName");

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

// Definir un objeto BackingMap
// si BackingMap está configurado en el archivo ObjectGrid.xml,
// sólo deberá obtenerlo.
BackingMap ivBackingMap = ivObjectGrid.defineMap("myMap");

// Añadir un Loader a BackingMap
Loader ivLoader = new HeapCacheLoader();
ivBackingMap.setLoader(ivLoader);

// inicializar ObjectGrid
ivObjectGrid.initialize();

// Obtener una sesión que debe utilizar la hebra actual.
// La sesión no puede compartirse entre diversas hebras.
Session ivSession = ivObjectGrid.getSession();

// Obtener ObjectMap de ObjectGrid Session
ObjectMap objectMap = ivSession.getMap("myMap");
```

Configuración compartida predeterminada

El código siguiente es un caso sencillo de creación de un ObjectGrid que se compartirá entre numerosos usuarios.

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

El fragmento de código Java anterior crea y almacena en la memoria caché el Employees ObjectGrid. Employees ObjectGrid se inicializa con la configuración predeterminada y está listo para usar. El segundo parámetro del método createObjectGrid está establecido en true, que indica a ObjectGridManager que almacene en memoria caché la instancia ObjectGrid que crea. Si este parámetro estuviera establecido en false, la instancia no se almacenaría en memoria caché. Cada instancia ObjectGrid tiene un nombre, y la instancia puede compartirse entre diversos clientes o usuarios basándose en ese nombre.

Si la instancia ObjectGrid se utiliza en compartimiento de igual a igual, el almacenamiento en memoria caché debe establecerse en true. Si desea más

información sobre el compartimiento de igual a igual, consulte el tema Distribución de cambios entre máquinas virtuales Java de igual.

Configuración XML

WebSphere eXtreme Scale es altamente configurable. El ejemplo anterior demuestra cómo crear un objeto ObjectGrid sencillo, sin ninguna configuración. Este ejemplo muestra cómo crear una instancia ObjectGrid previamente configurada basada en un archivo de configuración XML. Puede configurar una instancia ObjectGrid mediante programa o mediante un archivo de configuración XML. También puede configurar ObjectGrid mediante una combinación de estos dos procedimientos. La interfaz ObjectGridManager permite la creación de una instancia ObjectGrid basada en la configuración XML. La interfaz ObjectGridManager tiene varios métodos que toman una dirección URL como argumento. Cada archivo XML que se pasa a ObjectGridManager debe estar validado en el esquema. La validación de XML puede inhabilitarse sólo cuando el archivo se ha validado previamente y no se han realizado cambios en el archivo desde su última validación. Si se inhabilita la validación, se evita una pequeña sobrecarga, pero podría utilizarse un archivo XML no válido. IBM Java Developer Kit (JDK) 1.4.2 tiene soporte para la validación de XML. Si se utiliza un JDK que no ofrezca este soporte, podría necesitarse Apache Xerces para validar el XML.

El siguiente fragmento de código Java demuestra cómo pasar un archivo de configuración de XML para crear un ObjectGrid.

```
import java.net.MalformedURLException;
import java.net.URL;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
boolean validateXML = true; // activar validación de XML
boolean cacheInstance = true; // Almacenar en memoria caché la instancia
String objectGridName="Employees"; // Nombre de ObjectGrid URL
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
    ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees =
    oGridManager.createObjectGrid(objectGridName, allObjectGrids,
        bvalidateXML, cacheInstance);
```

El archivo XML puede contener información de configuración para varios objetos ObjectGrid. El fragmento de código anterior devuelve específicamente ObjectGrid Employees, y presupone que la configuración de Employees se ha definido en el archivo. Para conocer la sintaxis XML, consulte la configuración de ObjectGrid. Existen siete métodos createObjectGrid, que se documentan en el bloque de código siguiente.

```
/**
 * Un método de fábrica sencillo para devolver una instancia de un
 * ObjectGrid. Se asigna un nombre exclusivo.
 * La instancia ObjectGrid no se almacena en memoria caché.
 * Los usuarios pueden usar {@link ObjectGrid#setName(String)} para cambiar el
 * nombre de ObjectGrid.
 *
 * @devuelve ObjectGrid Instancia ObjectGrid con un nombre exclusivo asignado
 * @emite ObjectGridException si encuentra un error durante la
 * creación de ObjectGrid
 */
public ObjectGrid createObjectGrid() throws ObjectGridException;

/**
```

```

* Un método de fábrica sencillo para devolver una instancia ObjectGrid con el
* nombre especificado. Las instancias ObjectGrid se pueden almacenar en memoria caché.
* Si un ObjectGrid con este nombre ya se ha almacenado en memoria caché, se producirá
* una excepción ObjectGridException.
*
* @establecer parámetro objectGridName, nombre del ObjectGrid que se va a crear.
* @establecer parámetro cacheInstance en true, si la instancia ObjectGrid debe
* almacenarse en memoria caché
* @devolver una instancia ObjectGrid
* @este nombre ya se ha almacenado en memoria caché o
* se ha producido un error durante la creación de ObjectGrid.
*/
public ObjectGrid createObjectGrid(String objectGridName, boolean cacheInstance)
    throws ObjectGridException;

/**
* Crear una instancia ObjectGrid con el nombre de ObjectGrid especificado. La
* instancia ObjectGrid creada se almacenará en memoria caché.
* @establecer parámetro objectGridName, nombre de la instancia ObjectGrid
* que se va a crear.
* @devolver una instancia ObjectGrid
* @emite ObjectGridException si un ObjectGrid con ese nombre ya se
* ha almacenado en memoria caché, o si ha encontrado un error durante la
* creación de ObjectGrid
*/
public ObjectGrid createObjectGrid(String objectGridName)
    throws ObjectGridException;

/**
* Crear una instancia ObjectGrid basada en el nombre ObjectGrid especificado y el
* archivo XML. La instancia ObjectGrid definida en el archivo XML con el nombre de
* ObjectGrid especificado se creará y se devolverá. Si dicho ObjectGrid
* no se encuentra en el archivo XML, se emitirá una excepción.
*
* Esta instancia ObjectGrid se puede almacenar en memoria caché.
*
* Si la dirección URL es nula, se pasará por alto. En este caso, este método
* se comporta de la misma manera que {@link #createObjectGrid(String, boolean)}.
*
* @establecer parámetro objectGridName, nombre de la instancia ObjectGrid
* que se va a devolver. No debe ser nulo.
* @establecer parámetro xmlFile, una dirección URL para un archivo XML de
* formato correcto basado en el esquema ObjectGrid.
* @establecer parámetro enableXmlValidation, si es true, el XML se valida
* @establecer parámetro cacheInstance, valor booleano que indica si las
* instancias ObjectGrid
* definidas en el XML se almacenarán o no en memoria caché. Si es true
* (verdadero), la instancia almacenará en memoria caché.
*
* @emite ObjectGridException si un ObjectGrid con el mismo nombre
* se ha almacenado en memoria caché previamente, no se puede encontrar ningún
* nombre de ObjectGrid en el archivo XML, ni ningún otro error durante la
* creación de ObjectGrid.
* @devolver una instancia ObjectGrid
* @ver ObjectGrid
*/
public ObjectGrid createObjectGrid(String objectGridName, final URL xmlFile,
final boolean enableXmlValidation, boolean cacheInstance)
    throws ObjectGridException;

/**
* Procesar un archivo XML y crear una lista de objetos ObjectGrid basados
* en el archivo.
* Estas instancias ObjectGrid pueden almacenarse en memoria caché.
* Se emitirá una excepción ObjectGridException al intentar almacenar en
* memoria caché un ObjectGrid recién creado
* que tenga el mismo nombre que un ObjectGrid que ya se haya almacenado en

```

```

* memoria caché.
*
* @establecer parámetro xmlFile, archivo que define un ObjectGrid o varios
* ObjectGrids
* @establecer parámetro enableXmlValidation, si el valor es true se validará
* el archivo XML en el esquema
* @establecer parámetro cacheInstances, si se establece en true, se
* almacenarán en memoria caché todas las instancias ObjectGrid creadas y
* basadas en el archivo
* @devolver una instancia ObjectGrid
* @emite ObjectGridException si se intenta crear y almacenar en memoria caché
* un ObjectGrid con el mismo nombre que
* un ObjectGrid que ya se haya almacenado en memoria caché, o si se produce
* cualquier otro error durante la
* creación de ObjectGrid
*/
public List createObjectGrids(final URL xmlFile, final boolean enableXmlValidation,
boolean cacheInstances) throws ObjectGridException;

/** Crear todos los ObjectGrid que se encuentran en el archivo XML. El archivo
* XML se validará en el esquema. Cada instancia ObjectGrid que se crea se
* almacenará en memoria caché. Se emitirá una excepción ObjectGridException al
* intentar almacenar en memoria caché un ObjectGrid recién creado que tenga el
* mismo nombre que un ObjectGrid que ya se haya almacenado en memoria caché.
* @establecer parámetro xmlFile, archivo XML que se va a procesar. Se crearán
* ObjectGrids basados en el contenido del archivo.
* @devolver una lista de instancias ObjectGrid que se han creado.
* @emite ObjectGridException si un ObjectGrid, con el mismo nombre que
* los encontrados en el XML, ya se ha almacenado en memoria caché, o si se
* produce otro error durante la creación de ObjectGrid.
*/
public List createObjectGrids(final URL xmlFile) throws ObjectGridException;

/**
* Procesar el archivo XML y crear una única instancia ObjectGrid con
* el nombre de ObjectGrid especificado sólo si se encuentra un ObjectGrid con
* ese nombre en el archivo. Si no se ha definido ningún ObjectGrid con ese
* nombre en el archivo XML, se producirá una excepción ObjectGridException.
* La instancia ObjectGrid creada se almacenará en memoria caché.
* @establecer parámetro objectGridName, nombre del ObjectGrid que se va a
* crear. Este ObjectGrid debe definirse en el archivo XML.
* @establecer parámetro xmlFile, archivo XML que se va a procesar
* @devuelve un ObjectGrid recién creado
* @emite ObjectGridException si un ObjectGrid con el mismo nombre se ha
* almacenado en memoria caché previamente, no se puede encontrar ningún nombre
* de ObjectGrid en el archivo XML, ni ningún otro error durante la creación de
* ObjectGrid.
*/
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
throws ObjectGridException;

```

El cliente se cuelga durante una llamada al método getObjectGrid

Podría parecer que un cliente se cuelga al llamar al método getObjectGrid en ObjectGridManager o que emite una excepción: com.ibm.websphere.projector.MetadataException. El repositorio EntityMetadata no está disponible y se ha alcanzado el umbral del tiempo de espera. La razón es que el cliente está esperando a que los metadatos de entidad del servidor ObjectGrid pasen a estar disponibles. Este error puede producirse cuando se ha iniciado un contenedor, pero no se ha alcanzado el número inicial de contenedores o un número mínimo de réplicas síncronas. Examine la política de despliegue de ObjectGrid y compruebe que el número de contenedores activos es mayor o igual

que los atributos numInitialContainers y minSyncReplicas del archivo de descriptor de política de despliegue.

Métodos de getObjectGrid

Utilice los métodos ObjectGridManager.getObjectGrid para recuperar las instancias almacenadas en memoria caché.

Recuperación de una instancia almacenada en memoria caché

Puesto que la interfaz ObjectGridManager almacenó en memoria caché la instancia Employees ObjectGrid, otro usuario puede acceder a ella mediante el siguiente fragmento de código:

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

A continuación, aparecen los dos métodos getObjectGrid que devuelven instancias de ObjectGrid almacenadas en memoria caché:

- **Recuperación de todas las instancias almacenadas en memoria caché**

Para obtener todas las instancias de ObjectGrid que se han almacenado en la memoria caché previamente, utilice el método getObjectGrids, que devuelve una lista de cada instancia. Si no existen instancias almacenadas en memoria caché, el método devolverá null.

- **Recuperación de una instancia almacenada en memoria caché por nombre**

Para obtener una única instancia almacenada en memoria caché de un ObjectGrid, utilice getObjectGrid(String objectGridName), pasando el nombre de la instancia almacenada en memoria caché en el método. El método devuelve la instancia de ObjectGrid con el nombre especificado, o bien el valor null, si no hay ninguna instancia de ObjectGrid con dicho nombre.

Nota: También puede utilizar el método getObjectGrid para conectarse a una cuadrícula distribuida. Si desea más información, consulte “Conexión a un ObjectGrid distribuido” en la página 22.

Métodos removeObjectGrid

Puede utilizar dos métodos removeObjectGrid distintos para eliminar las instancias de ObjectGrid de la memoria caché.

Eliminar una instancia de ObjectGrid

Para eliminar de la memoria caché instancias de ObjectGrid, utilice uno de los métodos removeObjectGrid. ObjectGridManager no mantiene una referencia de las instancias que se eliminan. Existen dos métodos de eliminación. Un método toma un parámetro booleano. Si el parámetro booleano está establecido en true, se llama al método destroy en el ObjectGrid. La llamada al método destroy en el ObjectGrid lo concluye y libera cualquier recurso que utilice el ObjectGrid. A continuación, aparece una descripción de cómo utilizar los dos métodos removeObjectGrid:

```
/**
 * Eliminar un ObjectGrid de la memoria caché de las instancias de ObjectGrid
 *
 * @param objectGridName el nombre de la instancia de ObjectGrid a eliminar
 * de la memoria caché
 *
 * @throws ObjectGridException si un ObjectGrid con objectGridName
 * no se ha encontrado en la memoria caché
 */
public void removeObjectGrid(String objectGridName) throws ObjectGridException;
```

```

/**
 * Eliminar un ObjectGrid de la memoria caché de las instancias de ObjectGrid y
 * destruir sus recursos asociados
 *
 * @param objectGridName el nombre de la instancia de ObjectGrid a eliminar
 * de la memoria caché
 *
 * @param destroy destruir la instancia de objectgrid y sus recursos
 * asociados
 *
 * @throws ObjectGridException si un ObjectGrid con objectGridName
 * no se ha encontrado en la memoria caché
 */
public void removeObjectGrid(String objectGridName, boolean destroy)
    throws ObjectGridException;

```

Control del ciclo de vida de un ObjectGrid

Puede utilizar la interfaz `ObjectGridManager` para controlar el ciclo de vida de una instancia de `ObjectGrid` utilizando un bean de arranque o un servlet.

Gestión del ciclo de vida con un bean de arranque

Se utiliza un bean de arranque para controlar el ciclo de vida de una instancia de `ObjectGrid`. Un bean de arranque se carga cuando se inicia una aplicación. Con un bean de arranque, el código puede ejecutarse cuando una aplicación se inicia o se detiene del modo previsto. Para crear un bean de arranque, utilice la interfaz `com.ibm.websphere.startupservice.AppStartupHome` inicial y utilice la interfaz `com.ibm.websphere.startupservice.AppStartup` remota. Implemente los métodos `start` y `stop` en el bean. El método `start` se invoca cuando la aplicación se inicia. El método `stop` se invoca cuando la aplicación concluye. El método `start` se utiliza para crear instancias de `ObjectGrid`. El método `stop` se utiliza para eliminar las instancias de `ObjectGrid`. A continuación aparece un fragmento de código que demuestra esta gestión del ciclo de vida de `ObjectGrid` en un bean de arranque:

```

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

    /* Los métodos de la interfaz SessionBean se han
     * omitido en este ejemplo por razones de brevedad*/

    public boolean start(){
        // Inicio del bean de arranque
        // Se llama a este método cuando se inicia la aplicación
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // crear 2 ObjectGrids y colocar en memoria caché estas instancias
            ObjectGrid bookstoreGrid =
            objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
            ObjectGrid videostoreGrid =
            objectGridManager.createObjectGrid("videostore", true);
            // dentro de la JVM,
            // estas ObjectGrids pueden recuperarse ahora del
            //ObjectGridManager mediante el método getObjectGrid(String)
        } catch (ObjectGridException e) {
            e.printStackTrace();
            return false;
        }

        return true;
    }

    public void stop(){

```

```

// Detención del bean de arranque
// Se llama a este método cuando se detiene la aplicación
try {
    // eliminar los ObjectGrids almacenadas en memoria caché y destruirlos
    objectGridManager.removeObjectGrid("bookstore", true);
    objectGridManager.removeObjectGrid("videostore", true);
} catch (ObjectGridException e) {
    e.printStackTrace();
}
}
}

```

Después de que se llame al método `start`, se recuperan las instancias de `ObjectGrid` recién creadas de la interfaz `ObjectGridManager`. Por ejemplo, si se incluye un servlet en la aplicación, el servlet accede a eXtreme Scale utilizando el siguiente fragmento de código:

```

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

```

Gestión de un ciclo de vida con un servlet

Para gestionar el ciclo de vida de un `ObjectGrid` en un servlet, puede utilizar el método `init` para crear una instancia de `ObjectGrid` y el método `destroy` para eliminar la instancia de `ObjectGrid`. Si la instancia de `ObjectGrid` se almacena en memoria caché, se recupera y manipula en el código del servlet. El código de ejemplo que demuestra la creación, manipulación y destrucción de `ObjectGrid` dentro de un servlet es el siguiente:

```

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

    public MyObjectGridServlet() {
        super();
    }

    public void init(ServletConfig arg0) throws ServletException {
        super.init();
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // crear y almacenar en memoria caché un ObjectGrid llamado bookstore
            ObjectGrid bookstoreGrid =
                objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }

    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
        Session session = bookstoreGrid.getSession();
        ObjectMap bookMap = session.getMap("book");
        // realizar operaciones en el ObjectGrid almacenado en memoria caché
        // ...
    }

    public void destroy() {
        super.destroy();
        try {
            // eliminar y destruir el ObjectGrid bookstore almacenado en memoria caché
            objectGridManager.removeObjectGrid("bookstore", true);
        } catch (ObjectGridException e) {

```

```

        e.printStackTrace();
    }
}

```

Acceso al fragmento de ObjectGrid

WebSphere eXtreme Scale consigue altas velocidades de proceso trasladando la lógica a donde están los datos y devolviendo sólo los resultados al cliente.

La lógica de la aplicación en una Máquina virtual Java (JVM) de cliente necesita extraer datos de la JVM del servidor que mantiene los datos y hacerlos retroceder cuando se confirma la transacción. Este proceso disminuye la velocidad en la que se procesan los datos. Si la lógica de la aplicación estaba en la misma JVM que el fragmento que contiene los datos, el coste de ordenación y latencia de red se elimina y puede proporcionar un aumento significativo de rendimiento.

Referencia local a datos del fragmento

Las API de ObjectGrid proporcionan una Session para el método del lado del servidor. Esta sesión es una referencia directa a los datos correspondientes a ese fragmento. No hay ninguna lógica de direccionamiento en esta vía de acceso. La lógica de aplicación puede utilizarse directamente con los datos para ese fragmento. La sesión no puede utilizarse para acceder a los datos de otra partición porque no existe ninguna lógica de redireccionamiento.

Un plug-in del cargador proporciona una forma de recibir un suceso cuando un fragmento se convierte en una partición primaria. Una aplicación puede implementar un cargador e implementar la interfaz ReplicaPreloadController. El método de estado de precarga de comprobación sólo se invoca cuando un fragmento pasa a ser un primario. La sesión proporcionada para ese método es una referencia local para los datos de fragmentos. Este enfoque normalmente se utiliza si un primario de partición debe iniciar algunas hebras o suscribirse a un tejido de mensajes para el tráfico relacionado con las particiones. Puede iniciar una hebra que esté a la escucha de mensajes en una correlación local utilizando la API de getNextKey.

Optimización de cliente-servidor de ubicación compartida

Si una aplicación utiliza las API de cliente para acceder a una partición que tiene que colocarse con el JVM que contiene el cliente, se evita la red, pero se sigue produciendo alguna ordenación debido a los problemas actuales de implementación. Si se utiliza una cuadrícula, no tiene ningún impacto en el rendimiento de la aplicación por que el número de llamadas (N-1)/N direcciona a una JVM distinta. Si siempre es necesario el acceso local con un fragmento, utilice las API del cargador o de ObjectGrid para invocar esa lógica.

Acceso a los datos en WebSphere eXtreme Scale

Una vez que una aplicación tenga una referencia a una instancia de ObjectGrid o a una conexión de cliente con una cuadrícula remota, se podrá acceder a datos de la configuración de WebSphere eXtreme Scale e interactuar con ellos. Con la API ObjectGridManager, utilice uno de los métodos createObjectGrid para crear una instancia local, o el método getObjectGrid para una instancia cliente con una cuadrícula distribuida.

Una hebra en una aplicación necesita su propia sesión (Session). Cuando una aplicación desea utilizar el ObjectGrid en una hebra, debe llamar sólo a uno de los métodos getSession para obtener una hebra. Esta operación no es costosa, en la mayoría de los casos, no es necesario agrupar estas operaciones. Si la aplicación utiliza una infraestructura de inyección de dependencia como, por ejemplo Spring, puede inyectar una Session en un bean de aplicación, cuando sea necesario.

Después de obtener una Sesión, la aplicación puede acceder a los datos almacenados en correlaciones en el ObjectGrid. Si el ObjectGrid utiliza entidades, puede utilizar la API EntityManager, que puede obtener con el método Session.getEntityManager. Puesto que es cercano a las especificaciones Java, la interfaz EntityManager es más sencilla que la API basada en correlación. Sin embargo, la API EntityManager conlleva una sobrecarga de rendimiento porque rastrea los cambios en los objetos. La API basada en correlación se obtiene a través del uso del método Session.getMap.

WebSphere eXtreme Scale utiliza transacciones. Cuando una aplicación interactúa con un elemento Session, debe ser en el contexto de una transacción. Una transacción se inicia y confirma o se retrotrae utilizando los métodos Session.begin, Session.commit y Session.rollback en el objeto Session. Las aplicaciones también pueden funcionar en modalidad de confirmación automática, según la cual el elemento Session se inicia automáticamente y confirma una transacción siempre que la aplicación interactúa con correlaciones. Sin embargo, la modalidad de confirmación automática es más lenta.

La lógica del uso de transacciones

Las transacciones pueden parecer lentas, pero eXtreme Scale utiliza transacciones por tres motivos:

1. Para permitir la retrotracción de cambios si se produce una excepción o si la lógica empresarial necesita deshacer cambios de estado.
2. Para mantener bloqueos en datos y liberar bloqueos dentro del ciclo de vida de una transacción, lo que permite que se realicen automáticamente un conjunto de cambios, es decir, o todos los cambios o ningún cambio.
3. Para producir una unidad atómica de réplica.

WebSphere eXtreme Scale permite a una Session personalizar el volumen de transacción que realmente necesita. Una aplicación puede desactivar el soporte de retrotracción y el bloqueo, pero ello conlleva un coste para la aplicación. La aplicación deberá manejar la falta de estas características.

Por ejemplo, una aplicación puede desactivar el bloqueo mediante el establecimiento del valor NONE en la estrategia de bloqueo de BackingMap. Esta estrategia es rápida, pero las transacciones simultáneas ahora pueden modificar los mismos datos sin protección entre ellas. La aplicación es responsable de la coherencia de los datos y el bloqueo cuando se utiliza NONE.

Una aplicación también puede cambiar la forma en que se copian los objetos cuando la transacción accede a éstos. La aplicación puede especificar cómo se copian los objetos con el método ObjectMap.setCopyMode. Con este método, puede desactivar CopyMode. Normalmente, la modalidad CopyMode desconectada se utiliza para las transacciones de sólo lectura, si se pueden devolver distintos valores para el mismo objeto dentro de una transacción. Se pueden devolver distintos valores para el mismo objeto dentro de una transacción.

Por ejemplo, si la transacción ha llamado al método `ObjectMap.get` para el objeto en T1, obtuvo el valor en ese momento puntual. Si vuelve a llamar al método `get` dentro de dicha transacción en otro momento posterior T2, otra hebra podría haber cambiado el valor. Puesto que el valor ha sido modificado por otras hebra, la aplicación ve un valor distinto. Si la aplicación modifica un objeto recuperado utilizando un valor `NONE` de `CopyMode`, está cambiando la copia confirmada de dicho objeto directamente. Retrotraer la transacción no tiene sentido en esta modalidad. Modifica la única copia de `ObjectGrid`. Aunque utilizar `NONE CopyMode` es rápido, debe ser consciente de sus consecuencias. Una aplicación que utiliza `NONE CopyMode` nunca debe retrotraer la transacción. Si la aplicación retrotrae la transacción, los índices no se actualizan con los cambios y los cambios no se duplican, si la réplica está activa. Los valores predeterminados son fáciles de utilizar y menos propensos a errores. Si inicia el rendimiento en favor de unos datos menos fiables, la aplicación necesita saber qué está haciendo para evitar problemas no deseados.

PRECAUCIÓN:

Extreme las precauciones cuando modifique el bloqueo o los valores `CopyMode`. Si cambia los valores, se producirá un comportamiento de la aplicación impredecible.

Interactuar con datos almacenados

Después de que se haya obtenido una sesión, puede utilizar el siguiente fragmento de código para utilizar la API de correlación para insertar datos.

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

El mismo ejemplo que utiliza la API `EntityManager` es el siguiente. Este código de ejemplo da por supuesto que el objeto `Person` está correlacionado con una entidad.

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

El patrón se ha diseñado para obtener referencias a `ObjectMaps` para las correlaciones con las que trabajará la hebra, iniciar una transacción, trabajar con los datos y, después, confirmar la transacción.

La interfaz `ObjectMap` tiene las típicas operaciones de correlación como, por ejemplo, `put`, `get` y `remove`. Sin embargo, utilice los nombres de operación más específicos como: `get`, `getForUpdate`, `insert`, `update` y `remove`. Estos nombres de método expresan la intención de forma más precisa que las API de correlación tradicionales.

También puede utilizar el soporte de `indexación`, que es flexible.

A continuación, aparece un ejemplo para actualizar un objeto:

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

Normalmente, la aplicación utiliza el método `getForUpdate` en lugar de un sencillo método `get` para bloquear el registro. El método `update` debe llamarse para proporcionar el valor actualizado a la correlación. Si no se llama este método, la correlación no se modificará. A continuación, aparece el mismo fragmento utilizando la API `EntityManager`:

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

La API `EntityManager` API es más sencilla que el enfoque de correlación. En este caso, `eXtreme Scale` encuentra la entidad y devuelve un objeto gestionado a la aplicación. La aplicación modifica el objeto y confirma la transacción, y `eXtreme Scale` rastrea los cambios en los objetos gestionados de forma automática durante la confirmación y realiza las actualizaciones necesarias.

Transacciones y particiones

Las transacciones de `WebSphere eXtreme Scale` sólo pueden actualizar una única partición. Las transacciones de un cliente pueden leer varias particiones, pero sólo pueden actualizar una partición. Si una aplicación intenta actualizar dos particiones, la transacción falla y se retrotrae. Una transacción que utiliza un `ObjectGrid` incorporado (lógica de cuadrícula) no tiene capacidad de direccionamiento y sólo puede ver los datos de la partición local. Esta lógica empresarial siempre puede obtener una segunda sesión que sea una verdadera sesión de cliente para acceder a otras particiones. Sin embargo, esta transacción debería ser una transacción independiente.

Consultas y particiones

Si una transacción ya ha buscado una entidad, la transacción se asocia a la partición de dicha entidad. Cualquier consulta que se ejecute en una transacción asociada a una entidad se direcciona a la partición asociada.

Si una consulta se ejecuta en una transacción antes de que se asocie a una partición, debe establecer el ID de partición para utilizar para la consulta. El ID de partición es un valor entero. La consulta se direcciona entonces a dicha partición.

Las consultas sólo buscan dentro de una única partición. Sin embargo, puede utilizar las API `DataGrid` para ejecutar la misma consulta en paralelo en todas las particiones o un subconjunto de particiones. Utilice las API `DataGrid` para encontrar una entrada que pudiera estar en una partición.

7.0.0.0 FIX 2+ El servicio de datos REST permite a cualquier cliente HTTP acceder a una cuadrícula de `WebSphere eXtreme Scale` y es compatible con `WCF Data Services` en `Microsoft .NET Framework 3.5 SP1`. Para obtener más información, consulte la guía del usuario del servicio de datos REST de `eXtreme Scale`.

Procedimientos recomendados de CopyMode

WebSphere eXtreme Scale realiza una copia del valor basado en los seis valores de CopyMode disponibles. Determine el valor que funcione mejor para sus necesidades de despliegue.

Puede utilizar el método `setCopyMode(CopyMode, valueInterfaceClass)` de la API `BackingMap` para establecer la modalidad de copia en uno de los siguientes campos estáticos finales que se definen en la clase `com.ibm.websphere.objectgrid.CopyMode`.

Cuando una aplicación utiliza la interfaz `ObjectMap` para obtener una referencia a una entrada de correlación, utilice dicha referencia sólo dentro de la transacción de WebSphere eXtreme Scale que obtuvo la referencia. El uso de la referencia en una transacción diferente puede conducir a errores. Por ejemplo, si utiliza la estrategia de bloqueo pesimista para `BackingMap`, una llamada de método `get` o `getForUpdate` adquiere un bloqueo S (compartido) o U (actualización), en función de la transacción. El método `get` devuelve la referencia al valor y el bloqueo que se obtiene se libera cuando se completa la transacción. La transacción debe llamar al método `get` o `getForUpdate` para bloquear la entrada de la correlación en una transacción diferente. Cada transacción debe obtener su propia referencia al valor llamando al método `get` o `getForUpdate`, en lugar de reutilizar la misma referencia de valor en varias transacciones.

CopyMode para correlaciones de entidad

Si se utiliza una correlación asociada con una entidad de API `EntityManager`, la correlación siempre devuelve los objetos tuple de entidad directamente sin realizar una copia, a menos que utilice la modalidad de copia `COPY_TO_BYTES`. Es importante que `CopyMode` se actualice o que se copie el objeto `Tuple` correctamente al realizar los cambios.

COPY_ON_READ_AND_COMMIT

La modalidad `COPY_ON_READ_AND_COMMIT` es la modalidad predeterminada. El argumento `valueInterfaceClass` se pasa por alto cuando se utiliza esta modalidad. Esta modalidad garantiza que una aplicación no contenga una referencia al objeto de valor que esté en la correlación `BackingMap`. En su lugar, la aplicación siempre trabaja con una copia del valor que esté en la correlación `BackingMap`. La modalidad `COPY_ON_READ_AND_COMMIT` garantiza que la aplicación nunca pueda dañar accidentalmente los datos almacenados en memoria caché en `BackingMap`. Cuando una transacción de la aplicación llama a un método `ObjectMap.get` de una clave determinada, y es el primer acceso de la entrada `ObjectMap` de esa clave, se devuelve una copia del valor. Cuando se confirma la transacción, los cambios confirmados por la aplicación se copian en `BackingMap` para garantizar que la aplicación no tenga una referencia al valor confirmado en `BackingMap`.

COPY_ON_READ

La modalidad `COPY_ON_READ` mejora el rendimiento en comparación con la modalidad `COPY_ON_READ_AND_COMMIT` al eliminar la copia que se produce cuando se confirma una transacción. El argumento `valueInterfaceClass` se pasa por alto cuando se utiliza esta modalidad. Para conservar la integridad de los datos de `BackingMap`, la aplicación garantiza que todas las referencias que tiene de una entrada se destruyan una vez confirmada la transacción. Con esta modalidad, el

método `ObjectMap.get` devuelve una copia del valor en lugar de una referencia al valor para garantizar que los cambios realizados por la aplicación en el valor no afecten al valor de `BackingMap` hasta que se confirme la transacción. No obstante, cuando se confirma, no se realiza una copia de los cambios. En su lugar, se almacena en `BackingMap` la referencia a la copia devuelta por el método `ObjectMap.get`. La aplicación destruye todas las referencias de la entrada de correlación una vez que se confirma la transacción. Si la aplicación no las destruye, la aplicación podría dañar los datos almacenados en memoria caché de `BackingMap`. Si una aplicación que utiliza esta modalidad experimenta problemas, cambie a la modalidad `COPY_ON_READ_AND_COMMIT` para ver si se sigue produciendo el problema. Si desaparece, significa que la aplicación no está destruyendo todas las referencias después de la confirmación de la transacción.

COPY_ON_WRITE

La modalidad `COPY_ON_WRITE` mejora el rendimiento en comparación con la modalidad `COPY_ON_READ_AND_COMMIT` al eliminar la copia que se produce cuando una transacción llama por primera vez al método `ObjectMap.get` para una clave determinada. El método `ObjectMap.get` devuelve un proxy al valor en lugar de una referencia directa al objeto de valor. El proxy garantiza que no se realice una copia del valor a no ser que la aplicación llame a un método `set` en la interfaz de valor especificada en el argumento `valueInterfaceClass`. El proxy proporciona una copia en la implementación de grabación. Cuando se confirma una transacción, `BackingMap` examina el proxy para determinar si se realizó una copia como resultado de haber llamado a un método `set`. Si se realizó una copia, la referencia a dicha copia se almacena en `BackingMap`. La ventaja de utilizar esta modalidad es que un valor nunca se copia en una operación de lectura o de confirmación si la transacción no ha llamado a un método `set` para cambiar el valor.

Las modalidades `COPY_ON_READ_AND_COMMIT` y `COPY_ON_READ` realizan una copia profunda cuando un valor se recupera de `ObjectMap`. Si una aplicación sólo actualiza algunos de los valores recuperados en una transacción, esta modalidad no es la ideal. La modalidad `COPY_ON_WRITE` admite este comportamiento de una manera eficaz, pero requiere que la aplicación utilice un patrón sencillo. Los objetos de valor son obligatorios para admitir una interfaz. La aplicación debe utilizar los métodos de esta interfaz cuando interactúe con el valor de una sesión de eXtreme Scale. Si éste fuera el caso, eXtreme Scale crea proxies para los valores devueltos a la aplicación. El proxy tiene una referencia a un valor real. Si la aplicación sólo realiza operaciones de lectura, éstas siempre se ejecutan contra la copia real. Si la aplicación modifica un atributo en el objeto, el proxy realiza una copia del objeto real y después realiza la modificación en la copia. A continuación, el proxy utilice la copia a partir de ese punto. El uso de la copia permite que no se realice la operación de copia para los objetos que sólo lee la aplicación. Todas las operaciones de modificación deben empezar con el prefijo `set`. Normalmente, los Enterprise JavaBeans™ se codifican para utilizar este estilo de denominación de método para los métodos que modifican los atributos de objetos. Debe seguirse este convenio. Los objetos que se modifican se copian en el momento en que los modifica la aplicación. Este escenario de lectura y escritura es el escenario más eficaz soportado por eXtreme Scale. Para configurar una correlación de modo que utilice la modalidad `COPY_ON_WRITE`, observe el ejemplo siguiente. En este ejemplo, la aplicación almacena objetos `Person` que utilizan el nombre en la correlación. El objeto `person` se representa en el siguiente fragmento de código.

```

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

```

La aplicación utiliza la interfaz IPerson sólo cuando interactúa con valores recuperados de ObjectMap. Modifique el objeto para utilizar una interfaz como en el ejemplo siguiente:

```

interface IPerson
{
    void setName(String n);
    String getName();
    void setAge(int a);
    int getAge();
}
// Modificar Person para implementar la interfaz IPerson
class Person implements IPerson {
    ...
}

```

La aplicación necesita configurar BackingMap para que utilice la modalidad COPY_ON_WRITE, como en este ejemplo:

```

ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// usar COPY_ON_WRITE para esta correlación con
// IPerson como valueProxyInfo Class
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// La aplicación debe utilizar el siguiente
// patrón al usar la correlación PERSON.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// la aplicación difunde el valor devuelto a IPerson y no Person
IPerson p = (IPerson)person.get("Billy");
p.setAge(p.getAge()+1);
...
// hacer Person nuevo y añadirlo a la correlación
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// el fragmento de código siguiente NO FUNCIONARÁ. Devolverá ClassCastException
sess.begin();
// el error ha sido utilizar Person en lugar de
// IPerson
Person a = (Person)person.get("Bobby");
sess.commit();

```

La primera sección muestra la aplicación que recupera un valor de nombre Billy en la correlación. La aplicación difunde el valor devuelto al objeto IPerson, no al objeto Person porque el proxy que se devuelve implementa dos interfaces:

- La interfaz especificada en la llamada al método `BackingMap.setCopyMode`.
- La interfaz `com.ibm.websphere.objectgrid.ValueProxyInfo`.

Puede difundir el proxy para dos tipos. La última parte del fragmento de código anterior muestra lo que no se permite en la modalidad `COPY_ON_WRITE`. La aplicación recupera el registro Bobby e intenta difundir el registro a un objeto Person. Esta acción produce una excepción de difusión de clase porque el proxy devuelto no es un objeto Person. El proxy devuelto implementa el objeto IPerson y ValueProxyInfo.

Interfaz ValueProxyInfo y soporte de actualización parcial: esta interfaz permite a una aplicación recuperar el valor confirmado de sólo lectura al que hace referencia el proxy o el conjunto de atributos modificado durante esta transacción.

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

El método `ibmGetRealValue` devuelve una copia de sólo lectura del objeto. La aplicación no debe modificar este valor. El método `ibmGetDirtyAttributes` devuelve una lista de series que representa los atributos modificados por la aplicación durante esta transacción. El principal caso de uso de `ibmGetDirtyAttributes` está en un cargador basado en CMP o JDBC (Java database connectivity). Sólo deben actualizarse los atributos de la lista, ya sea en la sentencia SQL o en el objeto correlacionado con la tabla; se obtiene así un SQL, generado por el cargador, más eficaz. Cuando se confirma una transacción "copy on write" y se conecta un cargador, éste puede difundir los valores de los objetos modificados a la interfaz ValueProxyInfo para obtener esta información.

Manejo del método `equals` al utilizar `COPY_ON_WRITE` o servidores proxy: por ejemplo, el código siguiente construye un objeto Person y lo inserta en un ObjectMap. A continuación, recupera el mismo objeto mediante el método ObjectMap.get. El valor se difunde a la interfaz. Si el valor se difunde a la interfaz Person, se produce una excepción ClassCastException porque el valor devuelto es un proxy que implementa la interfaz IPerson y no es un objeto Person. La comprobación de igualdad falla al utilizar la operación `==` porque no son el mismo objeto.

```
session.begin();
// objeto Person nuevo
Person p = new Person(...);
personMap.insert(p.getName, p);
// recuperarlo de nuevo, recordar usar la interfaz para la difusión
IPerson p2 = personMap.get(p.getName());
if(p2 == p) {
    // son iguales
} else {
    // no son iguales
}
```

Otra consideración a tener en cuenta es cuando debe alterarse temporalmente el método `equals`. Como se ilustra en el fragmento de código siguiente, el método `equals` debe verificar que el argumento es un objeto que implementa la interfaz IPerson y difunde el argumento para ser IPerson. Como el argumento puede ser

un proxy que implementa la interfaz IPerson, debe usar los métodos getAge y getName al comparar la igualdad de las variables de instancia.

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

Requisitos de configuración de ObjectQuery y HashIndex: cuando se utiliza COPY_ON_WRITE con el plug-in ObjectQuery o HashIndex, es importante configurar el esquema ObjectQuery y un plug-in HashIndex para acceder a los objetos a través de métodos de propiedades, que es el valor predeterminado. Si se ha configurado para usar el acceso de campos, el motor de consultas y el índice intentarán acceder a los campos en los objetos proxy, que siempre devolverán null o 0 ya que la instancia del objeto será un proxy.

NO_COPY

La modalidad NO_COPY permite que una aplicación nunca modifique un objeto de valor que se obtiene mediante el método ObjectMap.get a cambio de mejoras en el rendimiento. El argumento valueInterfaceClass se pasa por alto cuando se utiliza esta modalidad. Si se utiliza esta modalidad, no se produce nunca una copia del valor. Si la aplicación modifica los valores, los datos de BackingMap se dañarán. La modalidad NO_COPY es útil especialmente en el caso de correlaciones de sólo lectura en las que la aplicación nunca modifica los datos. Si la aplicación utiliza esta modalidad y experimenta problemas, cambie a la modalidad COPY_ON_READ_AND_COMMIT para ver si se sigue produciendo el problema. Si desaparece, significa que la aplicación está modificando el valor devuelto por el método ObjectMap.get, durante la transacción o una vez confirmada ésta. Todas las correlaciones asociadas a las entidades de la API EntityManager utilizan automáticamente esta modalidad, independientemente de lo que se ha especificado en la configuración de eXtreme Scale.

Todas las correlaciones asociadas a las entidades de la API EntityManager utilizan automáticamente esta modalidad, independientemente de lo que se ha especificado en la configuración de eXtreme Scale.

COPY_TO_BYTES

Puede almacenar los objetos en un formato serializado, en lugar del formato POJO. Mediante el uso del valor COPY_TO_BYTES, puede reducir la huella de la memoria que puede consumir un gran gráfico de objetos. Consulte “Correlaciones de matriz de bytes” en la página 40 si desea información adicional.

Uso incorrecto de CopyMode

Los errores se producen cuando la aplicación intenta mejorar el rendimiento al usar las modalidades de copia COPY_ON_READ, COPY_ON_WRITE o NO_COPY, como se ha descrito anteriormente. Los errores intermitentes no se producen al cambiar la modalidad de copia a la modalidad COPY_ON_READ_AND_COMMIT.

Problema

Los datos de la correlación ObjectGrid pueden resultar dañados como resultado de la violación por parte de la aplicación del contrato de programación de la modalidad de copia que se está utilizando. El daño en los datos puede ocasionar errores imprevisibles de forma intermitente o errores que se manifiestan de forma inexplicable o inesperada.

Solución

La aplicación debe cumplir el contrato de programación que se aplica para la modalidad de copia que se vaya a utilizar. Para las modalidades de copia COPY_ON_READ y COPY_ON_WRITE, la aplicación utiliza una referencia a un objeto de valor fuera del ámbito de la transacción del que se obtuvo la referencia del valor. Para utilizar estas modalidades, la aplicación debe eliminar la referencia al objeto de valor una vez completada la transacción, y obtener una nueva referencia al objeto de valor en cada transacción que acceda al objeto de valor. Para la modalidad de copia NO_COPY, la aplicación nunca debe modificar el objeto de valor. En este caso, escriba la aplicación de modo que no cambie el objeto de valor, o establezca la aplicación para utilizar otra modalidad de copia.

Correlaciones de matriz de bytes

Puede almacenar los pares de clave-valor en sus correlaciones en una matriz de bytes, en lugar del formato POJO, que reduce la huella de la memoria que puede consumir un gráfico grande de objetos.

Ventajas

La cantidad de memoria que se consume aumenta con el número de objetos de una gráfico de objetos. Reduciendo un gráfico complicado de objetos a una matriz de bytes, sólo se conserva un objeto en el almacenamiento dinámico, en lugar de varios objetos. Con esta reducción del número de objetos en el almacenamiento dinámico, el tiempo de ejecución Java tiene menos objetos para buscar durante la recogida de basura.

El mecanismo de copia predeterminado utilizado por WebSphere eXtreme Scale es la serialización, que es un mecanismo caro. Por ejemplo, si utiliza la modalidad de copia predeterminada de COPY_ON_READ_AND_COMMIT, se realiza una copia tanto en el momento de leer, como en el de obtener. En lugar de realizar una copia durante la lectura, con las matrices de bytes, el valor se infla a partir de los bytes y, en lugar de realizar una copia durante la confirmación, el valor se serializa en bytes. El uso de matrices de bytes genera una coherencia de datos equivalentes al valor predeterminado con una reducción de la memoria utilizada.

Si se utilizan las matrices de bytes, tenga en cuenta que tener un mecanismo de serialización optimizado es vital para ver una reducción en el consumo de memoria. Para obtener más información, consulte “Rendimiento de serialización” en la página 234.

Configuración de correlaciones de matrices de bytes

Puede habilitar las correlaciones de matrices de bytes con el archivo XML ObjectGrid modificando el atributo CopyMode utilizado por una correlación por el valor COPY_TO_BYTES, mostrado en el ejemplo siguiente:

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

Consulte el tema sobre el archivo XML de descriptor de ObjectGrid en la *Guía de administración* si desea más información.

Consideraciones

Debe considerar si va a utilizar o no las correlaciones de matrices de bytes en un escenario determinado. Aunque puede reducir el uso de la memoria, el uso del procesador puede aumentar cuando se utilizan las matrices de bytes.

La lista siguiente describe varios factores que se deben tener en cuenta antes de elegir utilizar la función de correlación de matrices de bytes.

Tipo de objeto

Comparativamente, la reducción de la memoria no es posible si se utilizan las correlaciones de matrices de bytes para algunos tipos de objeto. Por consecuencia, existen varios tipos de objeto para los que no deberá utilizar las correlaciones de matrices de bytes. Si utiliza algunos de los derivadores primitivos de Java como valores, o un POJO que no contiene referencias a ningún otro objeto (sólo almacenar campos primitivos), el número de objetos Java ya es tan bajo como sea posible, sólo hay uno. Puesto que la cantidad de memoria utilizada por el objeto ya se ha optimizado, no se recomienda el uso de una correlación de matrices de bytes para estos tipos de objetos. Las correlaciones de matrices de bytes son más idóneas para los tipos de objeto que contiene otros objetos o colecciones de objetos donde el número total de objetos POJO es mayor que uno.

Por ejemplo, si tiene un objeto Customer (Cliente) que tenía una dirección empresarial y una dirección personal, así como una colección de Orders (Pedidos), el número de objetos en el almacenamiento dinámico y el número de bytes utilizados por dichos objetos se pueden reducir mediante el uso de correlaciones de matrices de bytes.

Acceso local

Cuando se utilizan otras modalidades de copia, las aplicaciones se pueden optimizar, cuando las copias se realizan, si los objetos son Cloneable con el ObjectTransformer predeterminado o cuando se proporciona un ObjectTransformer personalizado con un método copyValue optimizado. En comparación con otras modalidades de copia, la copia en operaciones de lecturas, escrituras o confirmación tendrá un coste adicional al acceder a los objetos de forma local. Por ejemplo, si tiene una memoria caché cercana en una topología distribuida o al acceder directamente a una instancia de ObjectGrid de servidor o local, el tiempo de acceso y confirmación se aumentará si se utilizan las correlaciones de matrices de bytes debido al coste de serialización. Verá un coste similar en una topología distribuida, si utiliza los agentes de cuadrícula de datos o si accede al primario del servidor, al utilizar el plug-in ObjectGridEventGroup.ShardEvents.

Interacciones de plug-in

Con las correlaciones de matrices de bytes, los objetos no se inflan cuando se establece una comunicación entre un cliente y un servidor, a menos que el servidor necesite un formato POJO. Los plug-ins que interactúan con el valor de correlación experimentará una reducción en el rendimiento debido a la necesidad de inflar el valor.

Cualquier plug-in que utiliza `LogElement.getCacheEntry` o `LogElement.getCurrentValue` verá este coste adicional. Si desea obtener la clave, podrá utilizar `LogElement.getKey`, que impide la sobrecarga adicional asociada al método `LogElement.getCacheEntry().getKey`. En las siguientes secciones se tratan los plug-ins para clarificar el uso de las matrices de bytes.

Índices y consultas

Cuando los objetos se almacenan en el formato POJO, el coste de los índices y las consultas es mínimo, porque el objeto no necesita ser inflado. Cuando se utiliza una correlación de matrices de bytes tendrá el coste adicional de inflar el objeto. En general, si la aplicación utiliza índices o consultas, no se recomienda utilizar las correlaciones de matrices de bytes, a menos que sólo ejecute consultas sobre atributos de clave.

Bloqueo optimista

Si se utiliza la estrategia de bloqueo optimista, tendrá el coste adicional durante las actualizaciones y las operaciones invalidar. Esto procede de tener que inflar el valor en el servidor para obtener el valor de la versión para realizar una comprobación de colisión optimista. Si simplemente utiliza el bloqueo optimista para garantizar las operaciones de obtención de información y no necesita una comprobación de colisión optimista, puede utilizar `com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback` para inhabilitar la comprobación de versiones.

Cargador

Con un cargador, también tendrá el coste en el tiempo de ejecución de eXtreme Scale de inflar y serializar el valor si es utilizado por el cargador. Puede seguir utilizando las correlaciones de matrices de bytes con los cargadores, pero tenga en cuenta el coste de realizar cambios en el valor en dicho escenario. Por ejemplo, puede utilizar la característica de matriz de bytes en el contexto de una memoria caché que se lee con frecuencia. En este caso, la ventaja de tener menos objetos en el almacenamiento dinámico y utilizar menos memoria superará el coste generado del uso de las matrices de bytes en las operaciones insertar y actualizar.

ObjectGridEventListener

Cuando se utiliza el método `transactionEnd` en el plug-in `ObjectGridEventListener`, tendrá un coste adicional en el lado del servidor para las solicitudes remotas al acceder a una `CacheEntry` del `LogElement` o al valor actual. Si la implementación del método no accede a estos campos, no tendrá el coste adicional.

Utilización de sesiones para acceder a los datos de la cuadrícula

Las aplicaciones pueden empezar y terminar transacciones a través de la interfaz `Session`. La interfaz `Session` también proporciona acceso a las interfaces `ObjectMap` y `JavaMap` basadas en la aplicación.

Todas las instancias de `ObjectMap` o `JavaMap` están unidas a un objeto `Session` determinado. Cada hebra que desea acceder a un eXtreme Scale debe, en primer lugar, obtener una sesión del objeto `ObjectGrid`. Una instancia de `Session` no puede compartirse de modo concurrente entre las hebras. WebSphere eXtreme Scale no utiliza ningún almacenamiento local de hebras, pero las restricciones de plataforma podrían limitar la oportunidad de pasar una sesión de una hebra a otra.

Métodos

Los siguientes métodos están disponibles con la interfaz `Session`. Consulte la documentación de la API para obtener más información sobre los siguientes métodos:

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

    void begin() throws TransactionAlreadyActiveException, TransactionException;

    void beginNoWriteThrough() throws TransactionAlreadyActiveException, TransactionException;
    public void commit() throws NoActiveTransactionException, TransactionException;

    public void rollback() throws NoActiveTransactionException, TransactionException;

    public void flush() throws TransactionException;

    TxID getTxID() throws NoActiveTransactionException;

    boolean isWriteThroughEnabled();

    void setTransactionType(String tranType);

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

    ObjectGrid getObjectGrid();

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

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

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

Método `get`

Una aplicación obtiene una instancia de `Session` de un objeto `ObjectGrid` utilizando el método `ObjectGrid.getSession`. El siguiente ejemplo demuestra cómo utilizar una instancia de `Session`:

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

Después de obtener una `Session`, la hebra mantiene una referencia a la sesión para uso propio. La llamada al método `getSession` en varias ocasiones devuelve un nuevo objeto `Session` cada vez.

Transacciones y métodos `Session`

Una `Session` puede utilizarse para iniciar, confirmar o retrotraer transacciones. Las operaciones realizadas en `BackingMaps` utilizando `ObjectMaps` y `JavaMaps` se realizan con mayor eficacia dentro de una transacción de `Session`. Después de que se haya iniciado una transacción, cualquier cambio a una o más `BackingMaps` de ese ámbito de transacción se almacenan en una memoria caché de transacciones especial hasta que se confirme la transacción. Cuando se confirma una transacción, los cambios pendientes se aplican a las `BackingMaps` y los cargadores y se hacen visibles a los demás clientes `ObjectGrid`.

WebSphere eXtreme Scale también soporta la capacidad de confirmar automáticamente transacciones, también se conoce como confirmación automática. Si se realiza cualquier operación de `ObjectMap` fuera del contexto de una transacción activa, se inicia una transacción implícita antes de la operación y la transacción se confirma automáticamente antes de devolver el control a la aplicación.

```

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

```

Método Session.flush

El método `Session.flush` sólo tiene sentido cuando se asocia un cargador a una `BackingMap`. El método `flush` invoca el cargador con el conjunto de cambios actuales de la memoria caché de transacciones. El cargador aplica los cambios al programa de fondo. Estos cambios no se confirman cuando se invoca el desecho. Si una transacción de `Session` se confirma después de una invocación de desecho, sólo las actualizaciones que ocurran después de esa invocación se aplican al cargador. Si una transacción de `Session` se retrotrae después de una invocación de desecho, los cambios desechados se descartan junto con los demás cambios pendientes de la transacción. Utilice el método `Flush` con moderación ya que limita la posibilidad de las operaciones por lotes en el cargador. A continuación, aparece un ejemplo del uso del método `Session.flush`:

```

Session session = objectGrid.getSession();
session.begin();
// realizar algunos cambios
...
session.flush(); // pasar estos cambios al cargador, sin confirmarlos todavía
// realizar más cambios
...
session.commit();

```

Método NoWriteThrough

Algunas de las correlaciones de `eXtreme Scale` están respaldadas por un cargador, que proporciona un almacenamiento persistente para los datos de la correlación. A veces, es útil confirmar los datos sólo en la correlación de `eXtreme Scale` y no pasar los datos al cargador. La interfaz `Session` proporciona el método `beginNoWriteThrough` con este fin. El método `beginNoWriteThrough` inicia una transacción como el método `begin`. Con el método `beginNoWriteThrough`, cuando se confirma la transacción, los datos sólo se confirman en la correlación en memoria de `eXtreme Scale` y no se confirman en el almacenamiento persistente proporcionado por el cargador. Este método es muy útil al realizar la precarga de datos en la correlación.

Cuando se utiliza una instancia de `ObjectGrid` distribuida, el método `beginNoWriteThrough` es muy útil para realizar cambios sólo en la memoria caché cercana, sin modificar la memoria caché lejana en el servidor. Si se sabe que los datos están obsoletos en la memoria caché cercana, el uso del método `beginNoWriteThrough` permite invalidar entradas en la memoria caché cercana sin invalidarlas también en el servidor.

La interfaz `Session` también proporciona el método `isWriteThroughEnabled` para determinar qué tipo de transacción está activa actualmente.

```

Session session = objectGrid.getSession();
session.beginNoWriteThrough();
// realizar algunos cambios ...
session.commit(); // estos cambios no se pasarán al cargador

```

Obtención del método del objeto TxID

El objeto TxID es un objeto opaco que identifica la transacción activa. Utilice el objeto TxID para los siguientes objetivos:

- Para comparar cuando busque una determinada transacción.
- Para almacenar datos compartidos entre los objetos TransactionCallback y Loader.

Consulte el plug-in TransactionCallback y los cargadores para obtener información adicional sobre la característica de ranuras de los objetos.

Método de supervisión del rendimiento

Si utiliza eXtreme Scale dentro de WebSphere Application Server, podría ser necesario restablecer el tipo de transacción para la supervisión del rendimiento. Puede establecer el tipo de transacción con el método `setTransactionType`. Consulte Supervisión del rendimiento de ObjectGrid con PMI (Performance Monitoring Infrastructure) de WebSphere Application Server si desea más información sobre el método `setTransactionType`.

Proceso de un método LogSequence completo

WebSphere eXtreme Scale puede propagar conjuntos de cambios de correlaciones a los escuchas de ObjectGrid como formas para distribuir correlaciones de una Máquina virtual Java a otra. Para facilitar al receptor el proceso de las LogSequences recibidas, la interfaz Session proporciona el método `processLogSequence`. Este método examina todos los LogElements de la LogSequence y realiza la operación adecuada como, por ejemplo, `insert`, `update`, `invalidate`, etc., en la BackingMap identificada por el MapName de LogSequence. Debe estar disponible una Session de ObjectGrid antes de que se invoque el método `processLogSequence`. La aplicación también es responsable de emitir las llamadas de confirmación o retrotracción adecuadas para completar la Session. El proceso de confirmación automática no está disponible para esta invocación de método. El proceso normal del ObjectGridEventListener receptor de la JVM sería iniciar una Session mediante el método `beginNoWriteThrough`, que evita la propagación incesante de cambios, llamar seguidamente a este método `processLogSequence` y, finalmente, confirmar o retrotraer la transacción.

```
// Utilizar el objeto Session que se ha pasado durante
//ObjectGridEventListener.initialization...
session.beginNoWriteThrough();
// procesar la LogSequence recibida
try {
    session.processLogSequence(receivedLogSequence);
} catch (Exception e) {
    session.rollback(); throw e;
}
// confirmar los cambios
session.commit();
```

Método markRollbackOnly

Este método se utiliza para marcar la transacción actual como "sólo de retrotracción". Marcar una transacción como "sólo de retrotracción" garantiza la retrotracción de la transacción aunque la aplicación invoque el método `commit`. Este método lo utiliza normalmente ObjectGrid o la aplicación cuando sabe que se pueden dañar los datos si se permite confirmar la transacción. Una vez invocado el método, el objeto Throwable que se pasa a este método se encadena a la excepción `com.ibm.websphere.objectgrid.TransactionException` que produce el método `commit` si se invoca en una sesión que se ha marcado previamente como "sólo

retrotracción". Las siguientes llamadas a este método para una transacción marcada previamente como "sólo retrotracción" se ignora. Es decir, sólo se utiliza la primera llamada que pasa una referencia a Throwable no nula. Una vez finalizada la transacción marcada, se elimina la marca "sólo retrotracción" para que se pueda confirmar la siguiente transacción iniciada por la sesión.

Método `isMarkedRollbackOnly`

Devuelve si la sesión está marcada actualmente como "solo retrotracción". Este método devuelve boolean true si y sólo si se ha invocado previamente el método `markRollbackOnly` en esta sesión y la transacción iniciada por la sesión continúa activa.

Método `setTransactionTimeout`

Establezca el tiempo de espera de transacción para la siguiente transacción iniciada por esta sesión en un número específico de segundos. Este método no afecta al tiempo de espera de transacción de las transacciones iniciadas previamente por esta sesión. Sólo afecta a las transacciones iniciadas después de invocar este método. Si este método no se invoca nunca, se utiliza el valor de tiempo de espera que se ha pasado al método `setTxTimeout` del método `com.ibm.websphere.objectgrid.ObjectGrid`.

Método `getTransactionTimeout`

Este método devuelve el valor de tiempo de espera de la transacción en segundos. Este método devuelve el último valor que se ha pasado como valor de tiempo de espera al método `setTransactionTimeout`. Si el método `setTransactionTimeout` no se invoca nunca, se utiliza el valor de tiempo de espera que se ha pasado al método `setTxTimeout` del método `com.ibm.websphere.objectgrid.ObjectGrid`.

`transactionTimedOut`

Este método devuelve boolean true si la transacción actual iniciada por esta sesión ha excedido el tiempo de espera.

Método `isFlushing`

Este método devuelve un valor boolean true si y sólo si todos los cambios transaccionales se desechan en el plug-in Loader como resultado del método `flush` de la interfaz `Session` que se está invocando. Un plug-in Loader puede encontrar este método práctico si necesita saber por qué se ha invocado el método `batchUpdate`.

Método `isCommitting`

Este método devuelve boolean true si y sólo si todos los cambios de transacción se confirman como resultado del método `commit` de la interfaz `Session` que se está invocando. Este método es muy útil para los plug-ins del cargador cuando necesitan saber por qué se ha invocado el método `batchUpdate`.

Método `setRequestRetryTimeout`

Este método establece el valor de tiempo de espera de reintento de solicitud para la sesión en milisegundos. Si el cliente establece un tiempo de espera de reintento de solicitud, el valor de la sesión altera temporalmente el valor del cliente.

Método `getRequestRetryTimeout`

Este método obtiene el valor actual de tiempo de espera de reintento de solicitud en la sesión. Un valor de -1 indica que el tiempo de espera no se ha establecido. Un valor de 0 indica que está en la modalidad fail-fast. Un valor mayor que 0 indica el valor de tiempo de espera en milisegundos.

SessionHandle para el direccionamiento

Al utilizar una política de ubicación de particiones por contenedor, puede utilizar un `SessionHandle`. Una instancia de `SessionHandle` contiene información de partición para la sesión actual y se puede reutilizar para una nueva sesión.

Un `SessionHandle` incluye información para la partición a la que está vinculada la sesión actual. `SessionHandle` es extremadamente útil para la política de ubicación de particiones por contenedor y se puede serializar con la serialización Java estándar.

Si tiene una instancia de `SessionHandle`, puede aplicar dicho descriptor de contexto en una sesión con el método `setSessionHandle(SessionHandle de destino)`, que pasa el descriptor de contexto como el destino. Puede recuperar el `SessionHandle` con el método `Session.getSessionHandle`.

Puesto que sólo es aplicable en un escenario de colocación por contenedor, al obtener el `SessionHandle` lanza una `IllegalStateException` si una `ObjectGrid` determinada tiene varios conjuntos de correlaciones por contenedor o no tiene ninguno. Si no invoca el método `setSessionHandle` antes de llamar al método `getSessionHandle`, se seleccionará el `SessionHandle` basándose en la configuración de `ClientProperties`.

También puede utilizar la clase ayudante `SessionHandleTransformer` para convertir el descriptor de contexto en distintos formatos. Los métodos de esta clase pueden cambiar la representación de un descriptor de contexto de matriz de bytes a instancia, de serie a instancia y viceversa en ambos casos y, también, pueden escribir los contenidos del descriptor de contexto en la corriente de salida.

Si desea ver un ejemplo sobre cómo poder utilizar un `SessionHandle`, consulte el tema de direccionamiento preferido por zonas en la *Visión general del producto*.

Integración de SessionHandle

Un objeto `SessionHandle` incluye la información de partición de la sesión a la que está enlazado y facilita el direccionamiento de solicitudes. Los objetos `SessionHandle` sólo se aplican al escenario de ubicación de particiones por contenedor.

Objeto SessionHandle para el direccionamiento de solicitudes

Puede enlazar un objeto `SessionHandle` a una sesión de los modos siguientes:

Consejo: En cada una de las llamadas de método siguientes, cuando un objeto `SessionHandle` está enlazado a una sesión, el objeto `SessionHandle` se puede obtener del método `Session.getSessionHandle` que se utiliza con el método `Session.setSessionHandle`.

- Invocación del método `Session.getSessionHandle`: En el momento en el que se invoca este método, si no hay ningún `SessionHandle` enlazado a la sesión, se seleccionará un `SessionHandle` aleatoriamente y se enlazará a la sesión.

- Invocación de operaciones de creación, lectura, actualización y supresión (CRUD) transaccionales: En el momento en el que se invocan estos métodos, si no hay ningún SessionHandle enlazado a la sesión, se seleccionará un SessionHandle aleatoriamente y se enlazará a la sesión.
- Invocación del método ObjectMap.getNextKey: En el momento en el que se invoca este método, si no hay ningún SessionHandle enlazado a la sesión, la solicitud de operación se direccionará aleatoriamente a particiones individuales hasta que se obtenga una clave. Si se devuelve una clave de una partición, se enlazará a la sesión un SessionHandle correspondiente a dicha partición. Si no se encuentra ninguna clave, no se enlazará ningún SessionHandle a la sesión.
- Invocación del método QueryQueue.getNextEntity o QueryQueue.getNextEntities: En el momento en el que se invoca este método, si no hay ningún SessionHandle enlazado a la sesión, la solicitud de operación se direccionará aleatoriamente a particiones individuales hasta que se obtenga un objeto. Si se devuelve un objeto de una partición, se enlazará a la sesión un SessionHandle correspondiente a dicha partición. Si no se encuentra ningún objeto, no se enlazará ningún SessionHandle a la sesión.
- Definir un SessionHandle con el método Session.setSessionHandle(SessionHandle sh): Si se obtiene un SessionHandle del método Session.getSessionHandle, el SessionHandle se puede enlazar a una Sesión. La definición de un SessionHandle influye en el direccionamiento de solicitudes en el ámbito de la sesión a la cual está enlazado.

El método Session.getSessionHandle siempre devolverá un SessionHandle y enlazará automáticamente un SessionHandle sobre la sesión si no hay ningún SessionHandle enlazado a la sesión. Si sólo desea verificar si una sesión tiene un SessionHandle, llame al método Session.isSessionHandleSet. Si este método devuelve un valor de false, significa que actualmente no hay ningún SessionHandle enlazado a la sesión.

nuevo método añadido a la API Session

```
/**
 * Determina si hay un SessionHandle definido actualmente en esta sesión.
 *
 * @devolver true si hay un SessionHandle definido actualmente en esta sesión.
 *
 * @desde 7.1
 */
public boolean isSessionHandleSet();
```

Tipos de operación principales en el escenario de colocación por contenedor

A continuación se resume el comportamiento de direccionamiento de los tipos de operación principales en el escenario de ubicación de particiones por contenedor con respecto a los objetos SessionHandle.

- **Objeto Session con objeto SessionHandle enlazado**
 - Index - API MapIndex y MapRangeIndex: SessionHandle
 - Query y ObjectQuery: SessionHandle
 - Agent - API MapGridAgent y ReduceGridAgent: SessionHandle
 - ObjectMap.Clear: SessionHandle
 - ObjectMap.getNextKey: SessionHandle
 - QueryQueue.getNextEntity, QueryQueue.getNextEntities: SessionHandle
 - Operaciones CRUD transaccionales (API ObjectMap y API EntityManager): SessionHandle

- **Objeto Session sin objeto SessionHandle enlazado**
 - Index - API MapIndex y MapRangeIndex: Todas las particiones activas actuales
 - Query y ObjectQuery: Partición especificada mediante el método setPartition de Query y ObjectQuery
 - Agent - MapGridAgent y ReduceGridAgent
 - No soportados: Método ReduceGridAgent.reduce(Session s, ObjectMap map, Collection keys) y MapGridAgent.process(Session s, ObjectMap map, Object key).
 - Todas las particiones activas actuales: Método ReduceGridAgent.reduce(Session s, ObjectMap map) y MapGridAgent.processAllEntries(Session s, ObjectMap map).
 - ObjectMap.clear: Todas las particiones activas actuales.
 - ObjectMap.getNextKey: Enlaza un SessionHandle a la sesión si se devuelve una clave de una de las particiones seleccionadas aleatoriamente. Si no se devuelve ninguna clave, la sesión no se enlaza a un SessionHandle.
 - QueryQueue: Especifica una partición con el método QueryQueue.setPartition. Si no hay ninguna partición definida, el método seleccionará aleatoriamente una partición para devolverla. Si se devuelve un objeto, la sesión actual se enlaza con el SessionHandle enlazado a la partición que devuelve el objeto. Si no se devuelve ningún objeto, la sesión no se enlaza a un SessionHandle.
 - Operaciones CRUD transaccionales (API ObjectMap y API EntityManager): Seleccionar una partición aleatoriamente.

En la mayoría de los casos, se debe utilizar SessionHandle para controlar el direccionamiento a una determinada partición. Puede recuperar y almacenar en memoria caché el SessionHandle de la sesión que inserta datos. Después de almacenar en memoria caché el SessionHandle, puede ponerlo en otra sesión de modo que pueda direccionar solicitudes a la partición especificada por el SessionHandle almacenado en la memoria caché. Para realizar operaciones como, por ejemplo, ObjectMap.clear sin SessionHandle, puede definir temporalmente SessionHandle con un valor nulo llamando a Session.setSessionHandle(null). Sin un SessionHandle especificado, las operaciones se ejecutarán en todas las particiones activas actualmente.

- **Comportamiento de direccionamiento de QueryQueue**

En el escenario de ubicación de particiones por contenedor, se puede utilizar SessionHandle para controlar el direccionamiento de getNextEntity y los métodos getNextEntities de la API QueryQueue. Si la sesión está enlazada a un SessionHandle, las solicitudes se direccionan a la partición a la cual está enlazada el SessionHandle. Si la sesión no está enlazada a un SessionHandle, las solicitudes se direccionan a la partición definida con el método QueryQueue.setPartition si se ha definido una partición de este modo. Si la sesión no tiene un SessionHandle ni una partición enlazados, se devolverá una partición seleccionada aleatoriamente. Si o se encuentra tal partición, el proceso se detiene y no se enlaza ningún SessionHandle a la sesión actual.

El fragmento de código siguiente muestra cómo utilizar SessionHandle.

Ejemplo de programación del objeto SessionHandle

```
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();
// almacenar en memoria caché el SessionHandle que inserta datos
SessionHandle cachedSessionHandle = ogSession.getSessionHandle();

// verificar si SessionHandle está definido en la sesión
boolean isSessionHandleSet = ogSession.isSessionHandleSet();

// desenlazar temporalmente el SessionHandle de la sesión
if(isSessionHandleSet) {
    ogSession.setSessionHandle(null);
}

// si la sesión no tiene ningún SessionHandle enlazado, la operación
// de borrado se ejecutará en todas las particiones activas actualmente
// y, de este modo, eliminará todos los datos de la correlación en toda
// la cuadrícula
map.clear();

// después de realizar el borrado, restablecer el SessionHandle, si
// la sesión necesita utilizar el SessionHandle anterior.
// Opcionalmente, al llamar a getSessionHandle se puede obtener un
// SessionHandle nuevo
ogSession.setSessionHandle(cachedSessionHandle);

```

Consideraciones sobre el diseño de aplicaciones

En el escenario de estrategia de ubicación por contenedor, se debe utilizar SessionHandle para la mayor parte de las operaciones. SessionHandle controla el direccionamiento a las particiones. Para recuperar datos, el SessionHandle que enlaza a la sesión debería ser el mismo SessionHandle de cualquier transacción de inserción de datos.

Cuando desee realizar una operación sin un SessionHandle definido en la sesión, puede desenlazar un SessionHandle de una sesión realizando una llamada de método Session.setSessionHandle(null).

Cuando una sesión está enlazada a un SessionHandle, todas las solicitudes de operación se direccionarán a la partición especificada por el SessionHandle. Sin SessionHandle definido, las operaciones se direccionan a todas las particiones o a una partición seleccionada aleatoriamente.

Almacenamiento en memoria caché de objetos sin relaciones implicadas (API ObjectMap)

ObjectMaps son como correlaciones Java que permiten a los datos almacenarse como pares clave-valor. Los ObjectMap proporcionan un acercamiento sencillo e intuitivo para el almacenamiento de los datos de la aplicación. Un ObjectMap es ideal para almacenar en memoria caché los objetos que no tienen relaciones. Si hubiera relaciones de objeto, debería utilizar la API EntityManager.

Si desea más información sobre la API EntityManager, consulte “Almacenamiento en memoria caché de objetos y sus relaciones (API EntityManager)” en la página 61.

Las aplicaciones suelen obtener una referencia de WebSphere eXtreme Scale y después un objeto Session de la referencia de cada hebra. Las sesiones no pueden compartirse entre hebras. El método `getMap` de Session devuelve una referencia a un `ObjectMap` para su uso en esta hebra.

Introducción a ObjectMap

La interfaz `ObjectMap` se utiliza para la interacción transaccional entre aplicaciones y `BackingMaps`.

Finalidad

Una instancia de `ObjectMap` se obtiene de un objeto `Session` que se corresponde con la hebra actual. La interfaz `ObjectMap` es el vehículo principal que utilizan las aplicaciones para realizar cambios en las entradas de una `BackingMap`.

Obtener una instancia de ObjectMap

Una aplicación obtiene una instancia de `ObjectMap` de un objeto `Session` mediante el método `Session.getMap(String)`. El siguiente fragmento de código demuestra cómo se obtiene una instancia de `ObjectMap`:

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

Cada instancia de `ObjectMap` se corresponde con un determinado objeto `Session`. Al llamar varias veces al método `getMap` en un determinado objeto `Session` con el mismo nombre `BackingMap`, siempre se devuelve la misma instancia de `ObjectMap`.

Confirmar automáticamente transacciones

Las operaciones realizadas en `BackingMaps` que utilizan `ObjectMaps` y `JavaMaps` se realizan con mayor eficacia dentro de una transacción de `Session`. WebSphere eXtreme Scale proporciona el soporte de confirmación automática cuando se llama a los métodos de las interfaces `ObjectMap` y `JavaMap` fuera de una transacción de `Session`. Los métodos inician una transacción implícita, realizan la operación solicitada y confirman la transacción implícita.

Semántica de los métodos

A continuación se proporciona una explicación de la semántica en la que se basan todos los métodos de las interfaces `ObjectMap` y `JavaMap`. El método `setDefaultKeyword`, el método `invalidateUsingKeyword` y los métodos que tienen un argumento `Serializable` se tratan en el tema `Palabras clave`. El método `setTimeToLive` se describe en el tema `Desalojadores`. Consulte la documentación de la API para obtener más información sobre estos métodos.

Método `containsKey`

El método `containsKey` determina si una clave tiene un valor en `BackingMap` o `Loader`. Si una aplicación da soporte a valores nulos, este método puede utilizarse para determinar si una referencia nula devuelta por una operación `get` hace referencia a un valor nulo o indica que la `BackingMap` y el `Loader` no contienen la clave.

Método `flush`

La semántica del método `flush` es parecida al método `flush` en la interfaz

Session. La diferencia importante es que el desecho de Session se aplica a los cambios pendientes actuales de todas las correlaciones que se han modificado en la sesión actual. Con este método, sólo los cambios de esta instancia de ObjectMap se desechan en el Loader.

Método get

El método get capta la entrada de la instancia de BackingMap. Si la entrada no se encuentra en la instancia de BackingMap pero existe un Loader asociado a la instancia de BackingMap, la instancia de BackingMap intenta captar la entrada del Loader. El método getAll se proporciona para permitir el proceso de captación de lotes.

Método getForUpdate

El método getForUpdate es igual al método get, aunque si se utiliza el método getForUpdate se indica a la BackingMap y al Loader que la intención es actualizar la entrada. Un Loader puede utilizar esta sugerencia para emitir un consulta SELECT for UPDATE a un programa de fondo de base de datos. Si se define una LockingStrategy pesimista para la BackingMap, el gestor de bloqueos bloquea la entrada. El método getAllForUpdate se proporciona para permitir el proceso de captación de lotes.

Método insert

El método insert inserta una entrada en la BackingMap y el Loader. Si se utiliza este método se indica a la BackingMap y al Loader que desea insertar una entrada que no existía previamente. Al invocar este método en una entrada existente, se genera una excepción cuando se invoca el método o se confirma la transacción actual.

Método invalidate

La semántica del método invalidate depende del valor del parámetro **isGlobal** que se pase al método. El método invalidateAll se proporciona para permitir el proceso de anulación de lotes.

La anulación local se especifica cuando se pasa el valor false como parámetro **isGlobal** del método de anulación. La anulación local descarta todos los cambios realizados en la entrada en la memoria caché de transacción. Si la aplicación emite un método get, la entrada se capta del último valor confirmado en la BackingMap. Si no existe ninguna entrada en la BackingMap, la entrada se capta del último valor confirmado o desechado del Loader. Cuando se confirma una transacción, todas las entradas que se han marcado como anuladas localmente no tienen ningún impacto en la BackingMap. Todos los cambios que se hayan desechado al Loader siguen estando comprometidos incluso si se ha desechado la entrada.

La anulación global se especifica cuando se pasa true como parámetro **isGlobal** del método invalidate. La anulación global descarta cualquier cambio pendiente de la entrada de la memoria caché de transacciones y omite el valor de BackingMap en operaciones posteriores que se realicen en la entrada. Cuando se confirma una transacción, todas las entradas que se han marcado como anuladas globalmente se desalojan de la BackingMap. Considere el siguiente caso de uso de anulación como ejemplo: la BackingMap está respaldada por una base de datos que tiene una columna de incremento automático. Las columnas de incremento son útiles para asignar números exclusivos a los registros. La aplicación inserta una entrada. Después de la inserción, la aplicación necesita saber el número de secuencia de la fila insertada. Sabe que su copia del objeto es

antigua, así que utiliza la anulación global para obtener el valor del Loader. El siguiente código demuestra este caso de uso:

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

Este ejemplo de código añade una entrada para Billy. El atributo de versión de Person se establece mediante una columna de incremento automático de la base de datos. La aplicación realiza primero un mandato de inserción. Después emite un desecho, que hace que la inserción se envíe al Loader y a la base de datos. La base de datos establece la columna de versión en el siguiente número de la secuencia, lo que provoca que el objeto Person quede obsoleto. Para actualizar el objeto, la aplicación se anula globalmente. El siguiente método get que se emite obtiene la entrada del Loader, e ignora el valor de transacción. La entrada se capta de la base de datos con el valor de versión actualizado.

Método put

La semántica del método put depende de si se ha invocado previamente un método get en la transacción para la clave. Si la aplicación emite una operación get que devuelve una entrada existente de la BackingMap o el Loader, la invocación del método put se interpreta como una actualización y devuelve el valor anterior en la transacción. Si se ha ejecutado la invocación a un método put sin una invocación al método get anterior, o una invocación al método get anterior no ha encontrado una entrada, la operación se interpreta como una inserción. La semántica de los métodos insert y update se aplica cuando se confirma la operación put. El método putAll se proporciona para habilitar el proceso de actualización e inserción de lotes.

Método remove

El método remove elimina la entrada de BackingMap y el cargador, si hay un cargador conectado. Este método devuelve el valor del objeto que se eliminó. Si el objeto no existe, este método devuelve un valor nulo. El método removeAll se proporciona para habilitar el proceso de supresión de lotes sin los valores de retorno.

Método setCopyMode

El método setCopyMode especifica un valor CopyMode para esta ObjectMap. Con este método, una aplicación puede alterar temporalmente el valor CopyMode que se especifica en la BackingMap. El valor CopyMode especificado está en vigor hasta que se invoca el método clearCopyMode. Ambos métodos se invocan fuera de los límites transaccionales. Un valor CopyMode no puede cambiarse en la mitad de una transacción.

Método touch

El método touch actualiza la hora del último acceso para una entrada. Este método no recupera el valor de la BackingMap. Utilice este método en su propia transacción. Si la clave proporcionada no existe en la BackingMap debido a la anulación o eliminación, se produce una excepción durante el proceso de confirmación.

Método update

El método update actualiza de forma explícita una entrada en la BackingMap y el Loader. Si se utiliza este método se indica a la BackingMap y al Loader que desea actualizar una entrada existente. Si se invoca este método en una entrada que no existe cuando se invoca el método o durante el proceso de confirmación, se producirá una excepción.

Método getIndex

El método getIndex intenta obtener un índice con nombre que se basa en la BackingMap. El índice no se puede compartir entre hebras y se ocupa de las mismas reglas que una Session. El objeto de índice devuelto se debe convertir a la interfaz de índice de aplicación correcta como, por ejemplo, la interfaz MapIndex, la interfaz MapRangeIndex o una interfaz de índice personalizada.

Método clear

El método clear elimina todas las entradas de la memoria caché de una correlación desde todas las particiones. Esta operación es una función de confirmación automática, por ello no debe haber ninguna transacción activa cuando se invoca el método clear.

Nota: el método clear sólo borra la correlación en la que se llama, y las correlaciones de entidad relacionadas no se ven afectadas. Este método no invoca el plug-in Loader.

Correlaciones dinámicas

Con la función de correlaciones dinámicas, puede crear correlaciones después de que la cuadrícula ya se haya inicializado.

En versiones anteriores, para eXtreme Scale es necesario que defina correlaciones antes de inicializar ObjectGrid. Como resultado, ha tenido que crear todas las correlaciones que se utilizarán antes de ejecutar transacciones respecto a cualquier de las correlaciones.

Ventajas de las correlaciones dinámicas

La introducción de las correlaciones dinámicas reduce la restricción de tener que definir todas las correlaciones antes de la inicialización. A través del uso de las correlaciones de plantilla, ahora se pueden crear las correlaciones después de que se haya inicializado ObjectGrid.

Las correlaciones de plantilla se definen en el archivo XML ObjectGrid. Las comparaciones de plantilla se ejecutan cuando una Sesión solicita una correlación que no se ha definido previamente. Si el nuevo nombre de correlación coincide con la expresión regular de una correlación de plantilla, la correlación se crea dinámicamente y se le asigna el nombre de la correlación solicitada. Esta correlación creada recientemente hereda todos los valores de la correlación de plantilla tal como los ha definido el archivo XML de ObjectGrid.

Creación de correlaciones dinámicas

La creación de correlaciones dinámicas está vinculada al método Session.getMap(String). Las llamadas a este método devuelven un ObjectMap basado en la BackingMap que configuró el archivo XML de ObjectGrid.

Pasarlo en una serie que coincide con la expresión regular de una correlación de plantilla generará la creación de un ObjectMap y una BackingMap asociada.

Consulte la documentación de la API si desea más información sobre el método `Session.getMap(String cacheName)`.

Definir una correlación de plantilla en XML es tan sencillo como establecer un atributo booleano de plantilla en el elemento `backingMap`. Cuando la plantilla está establecida en `true`, el nombre de `backingMap` se interpreta como una expresión regular.

WebSphere eXtreme Scale utiliza la coincidencia de patrón de la expresión regular de Java. Si desea más información sobre el motor de expresiones regulares en Java, consulte la documentación de la API para ver el paquete y las clases `java.util.regex`.

A continuación, aparece un archivo XML de ObjectGrid de ejemplo con una correlación de plantilla definida.

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

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

El archivo XML anterior define una correlación de plantilla y una correlación sin plantilla. El nombre de la correlación de plantilla es una expresión regular: `templateMap.*`. Cuando se llama al método `Session.getMap(String)` con un nombre de correlación que coincide con esta expresión regular, la aplicación crea una nueva correlación.

Nota: Si ha definido más de una correlación de plantilla, asegúrese de que el nombre de ningún argumento del método `Session.getMap(String)` coincide con más de una correlación de plantilla.

Ejemplo

La configuración de una correlación de plantilla es necesaria para poder crear una correlación dinámica. Añada el booleano de la plantilla a una `backingMap` en el archivo XML de ObjectGrid.

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

El nombre de la correlación de plantilla se trata como una expresión regular.

Llamar al método `Session.getMap(String cacheName)` con un `cacheName` que es una coincidencia para la expresión regular genera la creación de la correlación dinámica. Se devuelve un objeto `ObjectMap` de la llamada de este método y se crea un objeto `BackingMap` asociado.

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

La correlación creada recientemente se configura con todos los atributos y plug-ins que se definieron en la definición de la correlación de plantilla. Vuelva a considerar el archivo XML de `ObjectGrid` anterior.

Una correlación dinámica creada basándose en la correlación de plantilla de este archivo XML tendría un desalojador configurado y su estrategia de bloqueo sería pesimista.

Nota: Una plantilla no es un elemento `BackingMap` real. Es decir, el `ObjectGrid` "que cuenta" no contiene ninguna correlación "templateMap.*" real. La plantilla sólo se utiliza como base para la creación de correlaciones dinámicas. No obstante, debe incluir la correlación dinámica en el elemento `mapRef` del archivo XML de política de despliegue que tiene exactamente el mismo nombre que el XML de `ObjectGrid`. De este modo se identifica en qué `mapSet` estarán las correlaciones dinámicas.

Considere el cambio de comportamiento del método `Session.getMap(String cacheName)` al utilizar las correlaciones de plantilla. Antes de `WebSphere eXtreme Scale` versión 7.0, todas las llamadas al método `Session.getMap(String cacheName)` generaron una excepción `UndefinedMapException`, si no existía la correlación solicitada. Con las correlaciones dinámicas, todos los nombres que coinciden con la expresión regular para una correlación de plantilla generan una creación de correlación. Asegúrese de anotar el número de correlaciones que crea la aplicación, sobre todo, si la expresión regular es genérica.

Además, `ObjectGridPermission.DYNAMIC_MAP` es necesario para la creación de correlaciones dinámicas cuando la seguridad de `eXtreme Scale` está habilitada. Este permiso se comprueba cuando se llama al método `Session.getMap(String)`. Si desea más información, consulte la información sobre la autorización del cliente de aplicaciones en la *Visión general del producto*.

Ejemplos adicionales

objectGrid.xml

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

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

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

```

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

</objectGrids>
</objectGridConfig>

```

objectGridDeployment.xml

```

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

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

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

</deploymentPolicy>

```

Limitaciones y consideraciones:

Limitaciones:

- No puede utilizar las correlaciones dinámicas con la consulta.
- QuerySchema no soporta la plantilla para mapName.
- No puede utilizar las entidades con las correlaciones dinámicas.
- Una entidad BackingMap se define de forma implícita, se correlaciona con la entidad a través del nombre de clase.

Consideraciones:

- Muchos plug-ins no tienen ningún modo para determinar la correlación con la que se asocia cada plug-in.
- Otros plug-ins se diferencian entre sí utilizando un nombre de correlación o un nombre de BackingMap como argumento.

ObjectMap y JavaMap

Una instancia de JavaMap se obtiene de un objeto ObjectMap. La interfaz JavaMap tiene las mismas firmas de método que ObjectMap, pero con un manejo de excepciones distinto. JavaMap amplía la interfaz java.util.Map, por lo que todas las excepciones son instancias de la clase java.lang.RuntimeException. Como JavaMap amplía la interfaz java.util.Map, es fácil utilizar rápidamente WebSphere eXtreme Scale con una aplicación existente que utiliza una interfaz java.util.Map para almacenar los objetos en la memoria caché.

Obtener una instancia de JavaMap

Una aplicación obtiene una instancia de JavaMap de un objeto ObjectMap utilizando el método ObjectMap.getJavaMap. El siguiente fragmento de código demuestra cómo obtener una instancia de JavaMap.

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

Una JavaMap está respaldada por la ObjectMap de la que se ha obtenido. Si llama al método getJavaMap varias veces utilizando una ObjectMap concreta siempre se devuelve la misma instancia de JavaMap.

Métodos

La interfaz de JavaMap sólo da soporte a un subconjunto de los métodos en la interfaz java.util.Map. La interfaz java.util.Map da soporte a los siguientes métodos:

Método containsKey(java.lang.Object)

Método get(java.lang.Object)

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

Método putAll(java.util.Map)

Método remove(java.lang.Object)

clear()

Los demás métodos heredados de la interfaz java.util.Map generan una excepción java.lang.UnsupportedOperationException.

Correlaciones como colas FIFO

Con WebSphere eXtreme Scale, puede proporcionar una prestación parecida a una cola FIFO (primero en entrar, primero en salir) para todas las correlaciones. WebSphere eXtreme Scale realiza un seguimiento del orden de inserción de todas las correlaciones. Un cliente puede solicitar una correlación para la siguiente entrada desbloqueada en una correlación en el orden de inserción y bloquear la entrada. Este proceso permite a varios clientes consumir entradas de la correlación de una forma eficaz.

Ejemplo FIFO

El siguiente fragmento de código muestra un cliente entrando un bucle para procesar entradas en la correlación hasta que la correlación se agota. El bucle inicia

una transacción y luego llama al método `ObjectMap.getNextKey(5000)`. Este método devuelve la clave de la siguiente entrada desbloqueada disponible y la bloquea. Si la transacción está bloqueada durante más de 5000 milisegundos, el método devuelve un valor nulo.

```
Session session = ...;
ObjectMap map = session.getMap("xxx");
// esto es necesario establecerlo en algún lugar para detener este bucle
boolean timeToStop = false;

while(!timeToStop)
{
    session.begin();
    Object msgKey = map.getNextKey(5000);
    if(msgKey == null)
    {
        // la partición actual se ha agotado, invóquela de nuevo en
        // una nueva transacción para pasar a la partición siguiente
        session.rollback();
        continue;
    }
    Message m = (Message)map.get(msgKey);
    // ahora consumir el mensaje
    ...
    // es necesario suprimirlo
    map.remove(msgKey);
    session.commit();
}
```

Modalidad local frente a modalidad de cliente

Si la aplicación utiliza un núcleo principal, es decir, no es un cliente, el mecanismo funciona tal como se describe anteriormente.

Para la modalidad de cliente, si la JVM (Java Virtual Machine) es un cliente, el cliente se conecta inicialmente a un primario de partición aleatoria. Si no hay trabajo en esa partición, el cliente pasa a la siguiente en busca de trabajo. El cliente encuentra una partición con entradas o realiza un bucle y vuelve a la partición aleatoria inicial. Si el cliente realiza un bucle y vuelve a la partición inicial, devolverá un valor nulo a la aplicación. Si el cliente encuentra una partición con una correlación que tenga entradas, consumirá entradas de la misma hasta que no haya entradas disponibles durante el periodo de tiempo de espera. Una vez que ha transcurrido el tiempo de espera, se devuelve el valor nulo. Esta acción significa que cuando se devuelve nulo y se utiliza una correlación particionada, se debe iniciar una nueva transacción y reanudar la escucha. El fragmento de ejemplo de código anterior tiene este comportamiento.

Ejemplo

Cuando ejecute como cliente y se devuelva una clave, esa transacción ahora está enlazada a la partición con la entrada para esa clave. Si no desea actualizar ninguna otra correlación durante la transacción, no existe ningún problema. Si no desea actualizar, sólo puede actualizar correlaciones de la misma partición que la correlación de la que ha obtenido la clave. La entrada devuelta del método `getNextKey` debe ofrecer a la aplicación un método para descubrir los datos relevantes de dicha partición. Como ejemplo, si tiene dos correlaciones, una para los sucesos y otra para los trabajos que se ven afectados por los sucesos. Defina las dos correlaciones con las siguientes entidades:

```
Job.java
package tutorial.fifo;
```

```

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

@Entity
public class Job
{
    @Id String jobId;

    int jobState;
}

JobEvent.java
package tutorial.fifo;

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

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

```

El trabajo tiene un ID y un estado, que es un entero. Suponga que desea incrementar el estado cuando llega un suceso. Los sucesos se almacenan en la correlación JobEvent. Cada entrada tiene una referencia al trabajo asociado con el suceso. El código para que el escucha haga esto se parece al siguiente ejemplo:

```

JobEventListener.java
package tutorial.fifo;

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

public class JobEventListener
{
    boolean stopListening;

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

    synchronized boolean isStopped()
    {
        return stopListening;
    }

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

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

```

```

    JobEvent event = (JobEvent)em.find(JobEvent.class, jobEventKey);
    // procesar el suceso, aquí sólo se incrementa el
    // estado de trabajo
    event.job.jobState++;
    em.getTransaction().commit();
  }
}

```

La aplicación inicia el escucha en una hebra. El escucha se ejecuta hasta que se llama al método `stopListening`. El método `processJobEvents` se ejecuta en una hebra hasta que se llama al método `stopListening`. El bucle bloquea la espera de `eventKey` de la correlación `JobEvent` y luego utiliza `EntityManager` para acceder al objeto de suceso, elimina la referencia al trabajo e incrementa el estado.

La API de `EntityManager` no tiene un método `getNextKey`, pero `ObjectMap` sí lo tiene. Por lo tanto, el código utiliza la `ObjectMap` para que `JobEvent` obtenga la clave. Si se utiliza una correlación con entidades, no almacenará más objetos. En su lugar, almacenará tuples; un objeto `Tuple` para la clave y un objeto `Tuple` para el valor. El método `EntityManager.find` acepta un `Tuple` para la clave.

El código para crear un suceso se parece al siguiente ejemplo:

```

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

```

Para buscar el trabajo para el suceso, construya un suceso, haga que apunte al trabajo, insértelo en la correlación `JobEvent` y confirme la transacción.

Cargadores y correlaciones FIFO

Si desea respaldar una correlación que se utiliza como cola FIFO con un cargador, puede que sea necesario realizar algún trabajo adicional. Si el orden de las entradas de la correlación no es importante, no tendrá trabajo adicional. Si el orden es importante, es necesario añadir un número de secuencia a todos los registros insertados cuando se persisten los registros en el programa de fondo. El mecanismo de precarga debe grabarse para insertar los registro durante el inicio utilizando este orden.

Almacenamiento en memoria caché de objetos y sus relaciones (API `EntityManager`)

La mayoría de los productos de memoria caché utilizan las API basadas en correlaciones para almacenar los datos como pares de clave-valor. La API `ObjectMap` y la memoria caché dinámica de `WebSphere Application Server`, entre otras, utilizan este acercamiento. No obstante, las API basadas en correlaciones tienen limitaciones. La API `EntityManager` simplifica la interacción con la memoria caché de `eXtreme Scale` proporcionando una forma fácil de declarar e interactuar con un gráfico complejo de objetos relacionados.

Limitaciones de las API basadas en correlaciones

Si utiliza una API basada en correlaciones, como la memoria caché dinámica de WebSphere Application Server o la API ObjectMap, deberá tener en cuenta las limitaciones siguientes:

- La memoria caché debe utilizar un reflejo para extraer los datos de los objetos de la memoria caché, lo cual tiene implicaciones en el rendimiento.
- Dos aplicaciones no pueden compartir una memoria caché si utilizan diferentes objetos para los mismos datos.
- No se puede utilizar la evolución de datos porque no se puede añadir fácilmente un atributo a un objeto Java almacenado en la memoria caché.
- Trabajar con gráficos de objetos es difícil. La aplicación debe almacenar referencias artificiales entre los objetos y unirlos de forma manual.

Uso de EntityManager

La API EntityManager utiliza la infraestructura basada en correlaciones, pero convierte los objetos de entidad en tuples y viceversa antes de almacenarlos y leerlos en la correlación. Un objeto de entidad se transforma en un tuple de clave y un tuple de valor, que después de almacenar como pares de clave-valor. Un tuple es una matriz de atributos primitivos.

Este conjunto de API facilita significativamente el uso de eXtreme Scale al seguir el estilo POJO (Plain Old Java Object) de programación que se adopta en la mayoría de las infraestructuras.

Definición de un esquema de entidad

Un ObjectGrid puede tener varios un número ilimitado de esquemas de entidades lógicas. Las entidades se definen utilizando las clases Java anotadas, XML o una combinación de XML y clases Java. Las entidades definidas se registran con un servidor eXtreme Scale y se enlazan a BackingMaps, índices y otros plug-ins.

Al diseñar un esquema de entidad, debe completar las siguientes tareas:

1. Definir las entidades y sus relaciones.
2. Configurar eXtreme Scale.
3. Registrar las entidades.
4. Crear aplicaciones basadas en entidades que interactúan con las API EntityManager de eXtreme Scale.

Configuración de esquema de entidad

Un esquema de entidad es un conjunto de entidades y las relaciones entre las entidades. En una aplicación de eXtreme Scale con varias particiones, se aplican las siguientes restricciones y opciones a los esquemas de entidades:

- Cada esquema de entidad debe tener definida una sola raíz. Esto se conoce como raíz de esquema.
- Todas las entidades para un esquema dado deben estar en el mismo conjunto de correlaciones, lo que significa que todas las entidades que se pueden alcanzar desde una raíz de esquema con relaciones de clave o no de clave deben definirse en el mismo conjunto de correlaciones como raíz del esquema.
- Cada entidad puede pertenecer sólo a un esquema de entidad.
- Cada aplicación de eXtreme Scale puede tener varios esquemas.

Las entidades se registran con una instancia de ObjectGrid antes de que se inicialice. Cada entidad definida debe tener un nombre exclusivo y se enlaza automáticamente a una BackingMap de ObjectGrid con el mismo nombre. El método de inicialización varía en función de la configuración que se utilice:

Configuración de eXtreme Scale local

Si utiliza un ObjectGrid local, puede configurar mediante programación la entidad de esquema. En esta modalidad, puede utilizar los métodos ObjectGrid.registerEntities para registrar clases de entidad anotadas o un archivo de descriptor de metadatos de entidad.

Configuración de eXtreme Scale distribuido

Si utiliza una configuración de eXtreme Scale distribuido, debe proporcionar un archivo de descriptor de metadatos de entidad con el esquema de entidad.

Para obtener más detalles, consulte "EntityManager en un entorno distribuido" en la página 73.

Requisitos de la entidad

Los metadatos de entidad se configuran utilizando archivos de clase Java, un archivo XML descriptor de entidad o ambos. Como mínimo, se requiere el XML el descriptor de entidad para identificar qué BackingMaps de eXtreme Scale se deben asociar con entidades. Los atributos persistentes de la entidad y sus relaciones con otras entidades se describen en una clase Java anotada (clase de metadatos de entidad) o en el archivo XML descriptor de entidad. La clase de metadatos de entidad, cuando se especifica, también es utilizada por la API EntityManager interactuar con los datos en la cuadrícula.

7.0.0.0 FIX 2+ Una cuadrícula de eXtreme Scale se puede definir sin proporcionar ninguna clase de entidad. Esto puede ser beneficioso cuando el servidor y el cliente interactúan directamente con los datos de tuple almacenados en las correlaciones subyacentes. Estas entidades se definen completamente en el archivo XML descriptor de entidad y se conocen como entidades sin clase.

Entidades sin clase

Las entidades sin clase son útiles cuando no es posible incluir clases de aplicación en el servidor o en la vía de acceso de clases del cliente. Estas entidades se definen en el archivo XML descriptor de metadatos de entidad, donde el nombre de clase de la entidad se especifica utilizando un identificador de entidad sin clase en el formato siguiente: <identificador de entidad>. El símbolo @ identifica la entidad como entidad sin clase y se utiliza para correlacionar asociaciones entre entidades. Consulte la figura "Metadatos de entidad sin clase" para ver un ejemplo de un archivo XML descriptor de metadatos de entidad con dos entidades sin clase definidas.

Si un servidor o cliente de eXtreme Scale no tienen acceso a las clases, cualquiera de los dos puede seguir utilizando la API EntityManager utilizando entidades sin clase. Entre los casos de uso común se encuentran los siguientes:

- El contenedor de eXtreme Scale se aloja en un servidor que no permite clases de aplicación en la vía de acceso de clases. En este caso, los clientes pueden seguir accediendo a la cuadrícula utilizando la API EntityManager desde un cliente donde las clases estén permitidas.
- El cliente de eXtreme Scale no necesita acceso a las clases de entidad porque el cliente utiliza o un cliente que no es Java, como el servicio de datos REST de eXtreme Scale, o el cliente accede a los datos de tuple de la cuadrícula utilizando la API ObjectMap.

Si los metadatos de entidad son compatibles entre el cliente y servidor, se pueden crear metadatos de entidad utilizando clases de metadatos de entidad, un archivo XML, o a ambos.

Por ejemplo, la "clase de entidad mediante programa" de la figura siguiente es compatible con el código de metadatos sin clase de la siguiente sección.

Clase de entidad mediante programa

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

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

Campos, claves y versiones sin clase

Tal como se ha mencionado anteriormente, las entidades sin clases se configuran completamente en el archivo descriptor XML de entidad. Las entidades basadas en clases definen sus atributos utilizando campos propiedades y anotaciones Java. Por lo tanto, las entidades sin clases necesitan definir la estructura de claves y atributos en el descriptor XML de entidad con las etiquetas <basic> y <id>.

Metadatos de entidad sin clase

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

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

<entity class-name="@Department" name="Department" >
    <attributes>
```

```

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

```

Observe que cada entidad anterior tiene un elemento <id>. Una entidad sin clase debe tener uno o varios elementos <id> definidos o una asociación de un solo valor que represente la clave para la entidad. Los campos de la entidad se representan mediante elementos <basic>. Los elementos <id>, <version> y <basic> requieren un nombre y un tipo en las entidades sin clase. Consulte la sección siguiente sobre los tipos de atributos soportados para obtener información sobre los tipos soportados.

Requisitos de clases de entidades

Las entidades basadas en clases se identifican mediante la asociación de distintos metadatos con una clase Java. Los metadatos pueden especificarse utilizando anotaciones de Java Platform, Standard Edition 5, un archivo de descriptor de metadatos de entidad o una combinación de anotaciones y el archivo de descriptor. Las clases de entidad deben satisfacer los siguientes criterios:

- La anotación @Entity se especifica en el archivo descriptor XML de entidad.
- La clase tiene un constructor sin argumentos público o protegido.
- Debe ser una clase de nivel superior. Las interfaces y tipos enumerados no son clases de entidad válidas.
- No se puede utilizar la palabra clave final.
- No se puede utilizar la herencia.
- Debe tener un tipo y nombre exclusivos para cada instancia de ObjectGrid.

Todas las entidades tienen un nombre y tipo exclusivos. El nombre, si utiliza anotaciones, es el nombre simple (corto) de la clase de forma predeterminada, pero puede alterarse temporalmente utilizando el atributo name de la anotación @Entity.

Atributos persistentes

Los clientes y el gestor de entidades accede al estado persistente de una entidad utilizando campos (variables de instancia) o descriptores de acceso de propiedad de estilo Enterprise JavaBeans. Cada entidad debe definir el acceso basado en el campo o en la propiedad. Las entidades anotadas son de acceso a campos si los campos de clases se anotan y son de acceso a propiedades si el método de obtención de la propiedad es anotado. No se permite una combinación de acceso basado en campos y en propiedades. Si el tipo no se puede determinar automáticamente, se puede utilizar el atributo **accessType** de la anotación @Entity o XML equivalente para identificar el tipo de acceso.

Campos persistentes

A las variables de instancias de entidades de acceso a campos se accede directamente desde el gestor de entidades y los clientes. Los campos que se marcan con el modificador transient o la anotación transient se ignoran. Los campos persistentes no deben tener modificadores final o static.

Propiedades persistentes

Las entidades de acceso a propiedades se deben adherir a los convenios de firma de JavaBeans para las propiedades de lectura y grabación. Se ignoran los métodos que no siguen los convenios de JavaBeans o que tienen la anotación `Transient` en el método `getter`. Para una propiedad de tipo `T`, debe haber un método de obtención `getProperty` que devuelva un valor de tipo `T` y un método de establecimiento vacío `setProperty(T)`. Para los tipos booleanos, el método de obtención puede expresarse como `isProperty` devolviendo `true` o `false`. Las propiedades persistentes no pueden tener el modificador `static`.

Tipos de atributos soportados

Se da soporte a los siguientes tipos de propiedades y campos persistentes:

- Los tipos primitivos Java que incluyen derivadores:
- `java.lang.String`
- `java.math.BigInteger`
- `java.math.BigDecimal`
- `java.util.Date`
- `java.util.Calendar`
- `java.sql.Date`
- `java.sql.Time`
- `java.sql.Timestamp`
- `byte[]`
- `java.lang.Byte[]`
- `char[]`
- `java.lang.Character[]`
- `enum`

Se da soporte a los tipos de atributos serializables de usuario pero tienen limitaciones de rendimiento, consulta y detección de cambios. Los datos persistentes que no pueden enviarse a través de proxy, como las matrices y objetos serializables de usuario, deben volver a asignarse a la entidad en caso de que se modifiquen.

Los atributos serializables se representan en el archivo XML descriptor de entidad utilizando el nombre de clase del objeto. Si el objeto es una matriz, el tipo de datos se representa utilizando el formato interno Java. Por ejemplo, si un tipo de datos de atributo es `java.lang.Byte[][]`, la representación de serie será `[[Ljava.lang.Byte;`

Los tipos serializables de usuario deben ceñirse a los procedimientos recomendados siguientes:

- Implementar métodos de serialización de alto rendimiento. Implementar la interfaz `java.lang.Cloneable` y el método `clone` público.
- Implementar la interfaz `java.io.Externalizable`.
- Implementar `equals` y `hashCode`

Asociaciones de entidad

Las asociaciones de entidades bidireccionales y unidireccionales, o las relaciones entre entidades se pueden definir como uno con uno, muchos con uno, uno con muchos y muchos con muchos. El gestor de entidades resuelve automáticamente las relaciones de entidades en las referencias de clave adecuadas al almacenar las entidades.

La cuadrícula de eXtreme Scale es la memoria caché de datos y no fuerza la integridad referencial como una base de datos. Aunque las relaciones permiten las operaciones de persistencia y eliminación en cascada para entidades hijas, no detecta ni impone enlaces rotos con los objetos. Cuando se elimina un objeto hijo, la consulta a ese objeto debe eliminarse del padre.

Si define una asociación bidireccional entre dos entidades, debe identificar el propietario de la relación. En una asociación a muchos, el lado de muchos de la relación siempre es el lado propietario. Si la propiedad no puede determinarse automáticamente, se debe especificar el atributo **mappedBy** de la anotación o el equivalente XML. El atributo **mappedBy** identifica el campo en la entidad de destino que es el propietario de la relación. Este atributo también ayuda a identificar los campos relacionados donde hay varios atributos del mismo tipo y cardinalidad.

Asociaciones con un solo valor

Las asociaciones de uno con uno o de muchos con uno se indican utilizando las anotaciones `@OneToOne` y `@ManyToOne` o los atributos XML equivalentes. El tipo de entidad de destino lo determina el tipo de atributo. El ejemplo siguiente define una asociación unidireccional entre `Person` y `Address`. La entidad `Customer` tiene una referencia a una entidad `Address`. En este caso, la asociación también podría ser muchos con uno dado que no hay ninguna relación inversa.

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

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

Para especificar una relación bidireccional entre las clases `Customer` y `Address`, añade una referencia a la clase `Customer` desde la clase `Address` y añade la anotación adecuada para tachar el lado inverso de la relación. Dado que esta asociación es uno con uno, debe especificar un propietario de la relación utilizando el atributo `mappedBy` en la anotación `@OneToOne`.

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

Asociaciones valoradas por colecciones

Las asociaciones uno con muchos y muchos con muchos se indican utilizando las anotaciones `@OneToMany` y `@ManyToMany` o atributos XML equivalentes. Todas las relaciones de muchos se representan utilizando los tipos: `java.util.Collection`, `java.util.List` o `java.util.Set`. El tipo de entidad de destino se determina por el tipo genérico de `Collection`, `List` o `Set`, o utilizando de forma explícita el atributo **targetEntity** en la anotación `@OneToMany` o `@ManyToMany` (o XML equivalente).

En el ejemplo anterior, no es práctico tener un objeto de dirección por cada cliente porque es posible que muchos clientes compartan una dirección o puedan tener varias direcciones. Esta situación se resuelve mejor utilizando una asociación de muchos:

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

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

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

En este ejemplo, existen dos relaciones distintas entre las mismas entidades: una relación de dirección particular y de trabajo. Se utiliza una colección no genérica para el atributo **workCustomers** para demostrar cómo utilizar el atributo **targetEntity** cuando no hay genéricos disponibles.

Asociaciones sin clase

Las asociaciones de entidad sin clase se definen en el archivo XML descriptor de metadatos de entidad de modo similar a como se definen las asociaciones basadas en clases. La única diferencia es que, en lugar de la entidad de destino que apunta a una clase real, apunta al identificador de entidad sin clase utilizado para el nombre de clase de la entidad.

A continuación se muestra un ejemplo:

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

Claves primarias

Todas las entidades deben tener una clave primaria, que puede ser una clave simple (un solo atributo) o compuesta (varios atributos). Los atributos de clave se indican utilizando la anotación ID o se definen en el archivo de descriptor XML de entidad. Los atributos de clave tienen los siguientes requisitos:

- El valor de una clave primaria no puede cambiar.
- Un atributo de clave primaria debe adoptar uno de los tipos siguientes: tipo primitivo Java y derivadores, `java.lang.String`, `java.util.Date` o `java.sql.Date`.
- Una clave primaria puede contener cualquier número de asociaciones de valor único. La entidad de destino de la asociación de clave primaria no debe tener una asociación inversa directa o indirectamente a la entidad de origen.

Las claves primarias compuestas pueden, de forma opcional, definir una clase de clave primaria. Una entidad se asocia a una clase de clave primaria utilizando la

anotación `@IdClass` o el archivo de descriptor XML de entidad. Una anotación `@IdClass` resulta útil conjuntamente con el método `EntityManager.find`.

Las clases de claves primarias tienen los siguientes requisitos:

- Deben ser públicas con un constructor sin argumentos.
- El tipo de acceso de la clase de clave primaria lo determina la entidad que declara la clase de clave primaria.
- Si es acceso a propiedades, las propiedades de la clave primaria deben ser públicas o protegidas.
- Las propiedades o campos de claves primarias deben coincidir con los nombres y tipos de los atributos de claves definidas en la entidad que hace la referencia.
- Las clases de claves primarias deben implementar los métodos `equals` y `hashCode`.

A continuación se muestra un ejemplo:

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

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

public class CustomerKey {
    Zone zone;
    int custId;

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

Claves primarias sin clase

Las entidades sin clase deben cualquiera tener al menos un elemento `<id>` o una asociación en el archivo XML con el atributo `id=true`. Un ejemplo de ambos casos sería el siguiente:

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

Recuerde:

La etiqueta XML `<id-class>` no se soporta para las entidades sin clase.

Proxies de entidad e interceptación de campos

Las clases de entidad y los tipos de atributos soportados mutables se amplían mediante clases de proxy para entidades de acceso a propiedades y se han mejorado mediante códigos de bytes para entidades de acceso a campos de Java

Development Kit (JDK) 5. Cualquier acceso a la entidad, incluso por los métodos de negocios internos y los métodos equals, deben utilizar los métodos de acceso a propiedades o campos adecuados.

Los interceptores de proxies y campos se utilizan para permitir al gestor de entidades realizar un seguimiento del estado de la entidad, determinar si la entidad ha cambiado y mejorar el rendimiento. Los interceptores de campos sólo están disponibles en plataformas de Java SE 5 cuando se configura el agente de instrumentación de entidad.

Atención: al utilizar entidades de acceso a propiedades, el método equals debe utilizar el operador instanceof para comparar la instancia actual con el objeto de entrada. Toda la introspección del objeto de destino debe ser a través de las propiedades del objeto y no de los propios campos, ya que la instancia de objeto será el proxy.

Archivo emd.xsd

Utilice la definición de esquema XML de metadatos de entidad para crear un archivo XML de descriptor y definir un esquema de entidad para WebSphere eXtreme Scale.

Consulte la información sobre el archivo descriptor de los metadatos de entidad en *Guía de administración* si desea ver la descripciones de cada elemento y atributo del archivo emd.xsd.

Archivo emd.xsd

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

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

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

  <!-- ***** -->
  <xsd:complexType name="attributes">
    <xsd:sequence>
      <xsd:choice>
```

```

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

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

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

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

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

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

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

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

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

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

<!-- ***** -->
<xsd:complexType name="many-to-many">

```

```

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

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

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

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

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

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

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

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

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

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

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

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

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

```

```

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

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

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

</xsd:schema>

```

EntityManager en un entorno distribuido

Puede utilizar EntityManager con un ObjectGrid local o en un entorno de eXtreme Scale distribuido. La diferencia principal es el modo de conectarse a este entorno remoto. Después de establecer una conexión, no hay ninguna diferencia entre el uso de un objeto Session o el uso de la API EntityManager.

Archivos de configuración obligatorios

Son necesarios los siguientes archivos de configuración XML:

- Archivo XML de descriptor ObjectGrid
- Archivo XML de descriptor de entidad
- Archivo XML de descriptor de despliegue o cuadrícula

Estos archivos especifican qué entidades y BackingMaps albergará un servidor.

El archivo de descriptor de metadatos de entidad contiene una descripción de las entidades que se utilizan. Como mínimo, debe especificar el nombre y la clase de entidad. Si se ejecuta en un entorno Java Platform, Standard Edition 5, eXtreme Scale lee automáticamente la clase de entidad y sus anotaciones. Puede definir atributos XML adicionales si la clase de entidad no tiene anotaciones o si es necesario que altere temporalmente los atributos de clase. Si registra las entidades sin clases, proporcione toda información de la entidad sólo en el archivo XML.

Puede utilizar el siguiente fragmento de configuración XML para definir una cuadrícula de datos con entidades. En este fragmento de código, el servidor crea un ObjectGrid con el nombre bookstore y una correlación de respaldo asociada con el nombre order. Tenga en cuenta que el fragmento de código del archivo objectgrid.xml hace referencia al archivo entity.xml. En este caso, el archivo entity.xml contiene una entidad, la entidad Order.

objectgrid.xml

```

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

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

</objectGridConfig>

```

Este archivo `objectgrid.xml` se refiere al archivo `entity.xml` con el atributo `entityMetadataXMLFile`. La ubicación de este archivo es relativa a la ubicación del archivo `objectgrid.xml`. A continuación se muestra un ejemplo del archivo `entity.xml`:

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

En este ejemplo se supone que la clase `Order` tendría los campos `orderNumber` y `desc` anotados de manera similar.

Un archivo `entity.xml` sin clase equivalente sería como el que se muestra a continuación

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

Si desea más información sobre cómo iniciar un servidor eXtreme Scale, consulte *Inicio de procesos de servidor WebSphere eXtreme Scale en Guía de administración*, que utiliza ambos archivos, `deployment.xml` y `objectgrid.xml`, para iniciar el servidor de catálogo.

Conexión con un servidor eXtreme Scale distribuido

El siguiente código habilita el mecanismo de conexión para un cliente y un servidor en el mismo sistema:

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

En el fragmento de código anterior, tenga en cuenta la referencia al servidor eXtreme Scale remoto. Tras establecer la conexión, puede invocar métodos de la API `EntityManager` como, por ejemplo, `persist`, `update`, `remove` y `find`.

Atención: Cuando utilice entidades, pase el archivo XML de descriptor de ObjectGrid de alteración temporal del cliente en el método `connect`. Si se pasa un valor nulo en la propiedad `clientOverrideURL` y el cliente tiene una estructura de directorios diferente a la del servidor, el cliente podría no encontrar los archivos XML de descriptor de entidad o de ObjectGrid. Como mínimo, los archivos XML de ObjectGrid y de entidad para el servidor se pueden copiar en el cliente.

Antes, el uso de entidades en un cliente ObjectGrid exigía que se pusiera el XML de ObjectGrid y el XML de entidad a disposición del cliente de uno de los dos modos siguientes:

1. Pasar un XML de ObjectGrid de sustitución al método `ObjectGridManager.connect(String catalogServerAddresses, ClientSecurityConfiguration securityProps, URL overRideObjectGridXml)`.

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

2. Pasar un valor nulo para el archivo de sustitución y asegurarse de que el XML de ObjectGrid y el XML de la entidad referenciada estén disponibles para el cliente en la misma vía de acceso que en el servidor.

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

7.0.0.0 FIX 2+ Los archivos XML eran necesarios independientemente de si se deseaba o no utilizar entidades de subconjunto subset en el lado del cliente. Ya no es necesario que estos archivos utilicen las entidades definidas por el servidor. En su lugar, pase un valor nulo como parámetro `overrideObjectGridXml` como en la opción 2 de la sección anterior. Si el archivo XML no se encuentra en la misma vía de acceso establecida en el servidor, el cliente utilizará la configuración de entidad del servidor.

No obstante, si utiliza entidades de subconjunto en el cliente, debe proporcionar un XML de ObjectGrid de sustitución como en la opción 1.

Esquemas de cliente y de servidor

El esquema del lado del servidor define el tipo de datos que se almacenan en las correlaciones en un servidor. El esquema de cliente es una correlación a los objetos de aplicación en el esquema del servidor. Por ejemplo, podría tener el siguiente esquema de servidor:

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

Un cliente podría tener un objeto anotado como en el siguiente ejemplo:

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

Este cliente toma una entidad del lado del servidor y proyecta el subconjunto de la entidad en el objeto del cliente. Esta proyección implica un ahorro de memoria y de ancho de banda en un cliente porque el cliente sólo tiene la información que necesita, en lugar de toda la información que está en la entidad del lado del servidor. Aplicaciones diferentes pueden usar sus propios objetos en lugar de forzar a todas las aplicaciones a compartir un conjunto de clases para el acceso a los datos.

El archivo XML descriptor de entidad del cliente se requiere en los casos siguientes: si el servidor se ejecuta con entidades basadas en clases mientras el cliente se ejecuta sin clases; o si el servidor es sin clases y el cliente utiliza entidades basadas en clases. Una modalidad de cliente sin clases permite que el cliente siga ejecutando consultas de entidad sin tener acceso a las clases físicas. Presuponiendo que el servidor ha registrado la entidad `ServerPerson` anterior, el cliente sustituiría la cuadrícula por un archivo `entity.xml` como el siguiente:

```

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

```

Este archivo consigue una entidad de subconjunto equivalente en el cliente, sin necesidad de que el cliente proporcione la clase anotada real. Si el servidor es sin clases y el cliente no, el cliente proporciona un archivo XML descriptor de entidad de sustitución. Este archivo XML descriptor de entidad contiene una sustitución de la referencia de archivo de clase.

Referencia al esquema

Si la aplicación se ejecuta en Java SE 5, la aplicación puede añadirse a los objetos utilizando anotaciones. EntityManager puede leer el esquema de las anotaciones en dichos objetos. La aplicación proporciona al tiempo de ejecución de eXtreme Scale referencias a estos objetos utilizando el archivo entity.xml, al que se hace referencia en el archivo objectgrid.xml. El archivo entity.xml lista todas las entidades, cada una de las cuales está asociada con una clase o un esquema. Si se especifica un nombre de clase apropiado, la aplicación intenta leer las anotaciones de Java SE 5 de esas clases para determinar el esquema. Si no se anota el archivo de clase o si se especifica un identificador sin clase, el esquema se toma del archivo XML. El archivo XML se utiliza para especificar todos los atributos, claves y relaciones para cada entidad.

Una cuadrícula local no necesita archivos XML. El programa puede obtener una referencia de ObjectGrid e invocar el método ObjectGrid.registerEntities para especificar una lista de clases anotadas de Java SE 5 o un archivo XML.

El tiempo de ejecución utiliza el archivo XML o una lista de clases anotadas para encontrar los nombres de entidad, nombres y tipos de atributo, campos clave y tipos y relaciones entre entidades. Si eXtreme Scale se ejecuta en un servidor o en la modalidad autónoma, se realiza automáticamente una correlación cuyo nombre será el de cada entidad. Estas correlaciones puede personalizarse más mediante el archivo objectgrid.xml o las API establecidas por la aplicación o infraestructuras de inyección como Spring.

Archivo de descriptor de metadatos de entidad

Consulte “Archivo emd.xsd” en la página 70 si desea más información sobre el archivo descriptor de metadatos.

Interacción con EntityManager

Normalmente, las aplicaciones en primer lugar obtienen una referencia de ObjectGrid y, a continuación, una Session de dicha referencia para cada hebra. Las sesiones no pueden compartirse entre hebras. Hay disponible un método adicional en el elemento Session, el método getEntityManager. Este método devuelve una referencia a un gestor de entidades para utilizar para esta hebra. La interfaz EntityManager puede sustituir las interfaces Session y ObjectMap para todas las aplicaciones. Puede utilizar estas API EntityManager si el cliente tiene acceso a las clases de entidad definidas.

Cómo obtener una instancia de EntityManager desde una sesión

El método `getEntityManager` está disponible en un objeto `Session`. El siguiente código de ejemplo ilustra cómo crear una instancia `ObjectGrid` local y acceder a `EntityManager`. Consulte la interfaz `EntityManager` en la documentación de la API para ver detalles sobre todos los métodos soportados.

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

Existe una relación de uno a uno entre el objeto `Session` y el objeto `EntityManager`. Puede utilizar el objeto `EntityManager` más de una vez.

Persistencia de una entidad

Persistir una entidad quiere decir guardar el estado de una entidad nueva en una memoria caché `ObjectGrid`. Después de llamar al método de persistencia, la entidad pasa a estado gestionado. Persistir es una operación transaccional, y la nueva entidad se almacena en una memoria caché `ObjectGrid` después de que se confirme la transacción.

Cada entidad tiene un elemento `BackingMap` correspondiente en el que se almacenan los tuples. `BackingMap` tiene el mismo nombre que la entidad, y se crea al registrarse la clase. El siguiente ejemplo de código demuestra cómo crear un objeto `Order` utilizando la operación `persist`.

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

El objeto `Order` se crea con la clave 123, y el objeto se pasa al método de persistencia. Puede seguir modificando el estado del objeto antes de confirmar la transacción.

Importante: El ejemplo anterior no incluye ningún límite transaccional necesario como, por ejemplo, `begin` y `commit`. Consulte la guía de aprendizaje del gestor de entidades en *Visión general del producto* si desea más información.

Búsqueda de una entidad

Puede localizar la entidad en la memoria caché de `ObjectGrid` con el método `find` proporcionando una clave después de que la entidad se almacene en la memoria caché. Este método no requiere ningún límite transaccional, que es útil para la semántica de sólo lectura. El siguiente ejemplo ilustra que sólo se necesita una línea de código para buscar la entidad.

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

Eliminación de una entidad

El método `remove`, igual que el método `persist`, es una operación transaccional. El ejemplo siguiente muestra el límite transaccional llamando a los métodos `begin` y `commit`.

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

La entidad debe, en primer lugar, ser gestionada antes de que se pueda eliminar, para ello llame al método `find` dentro del límite transaccional. Llame al método `remove` en la interfaz `EntityManager`.

Invalidación de una entidad

El método `invalidate` se comporta de forma parecida al método `remove`, pero no invoca a los plug-ins `Loader`. Utilice este método para eliminar las entidades del `ObjectGrid`, sino para conservarlas en el almacén de datos de programa de fondo.

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

La entidad debe, en primer lugar, ser gestionada antes de que se pueda invalidar, para ello llame al método `find` dentro del límite transaccional. Después de llamar al método `find`, puede llamar al método `invalidate` en la interfaz `EntityManager`.

Actualización de una entidad

El método `update` también es una operación transaccional. Para poder aplicar una actualización, primero se debe gestionar la entidad.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
foundOrder.date = new Date(); // actualiza la fecha del pedido
em.getTransaction().commit();
```

En el ejemplo anterior, no se llama al método `persist` después de que se actualice la entidad. La entidad se actualiza en la memoria caché `ObjectGrid` cuando la transacción se confirma.

Consultas y colas de consulta

Con el motor de consultas flexible, puede recuperar entidades mediante la API `EntityManager`. Cree consultas de tipo `SELECT` en una entidad o esquema basado en objetos mediante el lenguaje de consulta de `ObjectGrid`. La interfaz de consultas explica en detalle cómo ejecutar las consultas mediante la API `EntityManager`. Consulte el apartado sobre la API `Query` si desea información sobre cómo utilizar las consultas.

Una entidad `QueryQueue` es una estructura de datos en forma de cola asociada con una consulta de entidad. Selecciona todas las entidades que coinciden con la condición `WHERE` en el filtro de la consulta y coloca las entidades resultantes en una cola. Los clientes puede recuperar de manera iterativa las entidades de esta cola. Si desea más información, consulte “Colas de consulta de entidades” en la página 90.

Métodos de devolución de llamada y escuchas de entidad

Se puede notificar a las aplicaciones cuando el estado de una entidad pasa de un estado a otro. Existen dos mecanismos de devolución de llamada para los sucesos de cambio de estado: los métodos de devolución de llamada de ciclo de vida que están definidos en una clase de entidad y se invocan siempre que cambia el estado de la entidad, y los escuchas de entidad, que son más generales porque el escucha de entidad se puede registrar en varias entidades.

Ciclo de vida de una instancia de entidad

Una instancia de entidad tiene los siguientes estados:

- **New** (nuevo): instancia de entidad creada recientemente que no existe en la memoria caché de eXtreme Scale.
- **Managed** (gestionado): instancia de entidad que existe en la memoria caché de eXtreme Scale y que se recupera y persiste mediante el gestor de entidades. Para que una entidad esté en estado gestionado, ésta debe asociarse a una transacción activa.
- **Detached** (desconectado): la instancia de entidad existe en la memoria caché de eXtreme Scale, pero ya no está asociada a una transacción activa.
- **Removed** (eliminado): instancia de entidad que se elimina, o se programa para que se elimine, de la memoria caché de eXtreme Scale cuando la transacción se vacía o se confirma.
- **Invalidated** (invalidado): instancia de entidad que se invalida, o se programa para que se invalide, en la memoria caché de eXtreme Scale cuando la transacción se vacía o se confirma.

Cuando las entidades cambian de un estado a otro, puede invocar los métodos de devolución de llamada de ciclo de vida.

En los apartados siguientes se describe el significado de los estados New, Managed, Detached, Removed e Invalidated según se van aplicando los estados a una entidad.

Métodos de devolución de llamada del ciclo de vida de entidad

Los métodos de devolución de llamada del ciclo de vida de la entidad se pueden definir en la clase de entidad y se invocan cuando cambia el estado de la entidad. Estos métodos son útiles para validar campos de entidad y actualizar el estado temporal que normalmente no es persistente con la entidad. Los métodos de devolución de llamada del ciclo de vida de la entidad también se pueden definir en las clases que no utilizan entidades. Dichas clases son clases de escucha de entidad que pueden asociarse a varios tipos de entidad. Los métodos de devolución de llamada del ciclo de vida se pueden definir utilizando anotaciones de metadatos y, también, un archivo XML descriptor de metadatos de entidad.

- **Anotaciones:** los métodos de devolución de llamada de ciclo de vida se pueden indicar utilizando las anotaciones PrePersist, PostPersist, PreRemove, PostRemove, PreUpdate, PostUpdate y PostLoad en una clase de entidad.
- **Descriptor XML de entidad:** los métodos de devolución de llamada de ciclo de vida se pueden describir utilizando XML cuando las anotaciones no están disponibles.

Escuchas de entidad

Una clase de escucha de entidad es una clase que no utiliza entidades que define uno o más métodos de devolución de llamada de ciclo de vida de entidad. Las escuchas de entidad son útiles para las aplicaciones de registro o auditoría de uso general. Las escuchas de entidad pueden definirse utilizando anotaciones de metadatos y un archivo de descriptor XML de metadatos de entidad:

- **Anotación:** la anotación EntityListeners puede utilizarse para indicar una o más clases de escuchas de entidad en una clase de entidad. Si se definen varios escuchas de entidad, el orden en el que se invocan lo determina el orden en el que se especifican en la anotación EntityListeners.

- **Descriptor XML de entidad:** el descriptor XML puede utilizarse como alternativa para especificar el orden de invocación de los escuchas de entidad o para alterar temporalmente el orden que se especifica en las anotaciones de metadatos.

Requisitos del método de devolución de llamada

Cualquier subconjunto o combinación de anotaciones se puede especificar en una clase de entidad o una clase de escucha. Una sola clase no puede tener más de un método de devolución de llamada de ciclo de vida para el mismo suceso de ciclo de vida. Sin embargo, se puede utilizar el mismo método para varios sucesos de devolución de llamada. La clase de escucha de entidad debe tener un constructor sin argumentos público. Los escuchas de entidad no tienen estado. El ciclo de vida de un escucha de entidad no se especifica. eXtreme Scale no da soporte a la herencia de entidades, de modo que los métodos de devolución de llamada sólo se pueden definir en la clase de entidad, pero no en la superclase.

Firma de método de devolución de llamada

Los métodos de devolución de llamada de ciclo de vida de entidad se pueden definir en una clase de escucha de entidad, directamente en una clase de entidad, o en ambas. Los métodos de devolución de llamada del ciclo de vida de entidad se pueden definir utilizando las anotaciones de metadatos y, también, el descriptor XML de la entidad. Las anotaciones utilizadas para los métodos de devolución de llamada de la clase de entidad y la clase de escucha de entidad son las mismas. Las firmas de los métodos de devolución de llamada son distintos cuando se definen en una clase de entidad contra una clase de escucha de entidad. Los métodos de devolución de llamada definidos en una clase de entidad o superclase correlacionada tiene la siguiente firma:

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

Los métodos de devolución de llamada que se definen en una clase de escucha de entidad tienen la siguiente firma:

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

El argumento Object es la instancia de entidad para la que se invoca el método de devolución de llamada. El argumento Object puede declararse como objeto `java.lang.Object` o el tipo de entidad real.

Los métodos de devolución de llamada pueden tener el acceso de nivel público, privado, protegido o paquete, pero no debe ser estático o final.

Las siguientes anotaciones se definen para designar los métodos de devolución de llamada de suceso de ciclo de vida de los tipos correspondientes:

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

Consulte la documentación de la API para obtener información detallada. Cada anotación tiene un atributo XML equivalente definido en el archivo de descriptor XML de metadatos de entidad.

Semánticas del método de devolución de llamada de ciclo de vida

Cada uno de los métodos distintos de devolución de llamada de ciclo de vida tiene un objetivo distinto y se llama en distintas fases del ciclo de vida de entidad:

PrePersist

Se invoca para una entidad antes de que la entidad haya persistido en el almacén, lo que incluye las entidades que han persistido debido a una operación en cascada. Este método se invoca en la hebra de la operación `EntityManager.persist`.

PostPersist

Se invoca para una entidad después de que la entidad haya persistido en el almacén, lo que incluye las entidades que han persistido debido a una operación en cascada. Este método se invoca en la hebra de la operación `EntityManager.persist`. Se invoca después de llamar a `EntityManager.flush` o `EntityManager.commit`.

PreRemove

Se invoca para una entidad antes de que la entidad se haya eliminado, lo que incluye las entidades que se han eliminado debido a una operación en cascada. Este método se invoca en la hebra de la operación `EntityManager.remove`.

PostRemove

Se invoca para una entidad después de que la entidad se haya eliminado, lo que incluye las entidades que se han eliminado debido a una operación en cascada. Este método se invoca en la hebra de la operación `EntityManager.remove`. Se invoca después de llamar a `EntityManager.flush` o `EntityManager.commit`.

PreUpdate

Se invoca para una entidad antes de que la entidad se haya actualizado en el almacén. Este método se invoca en la hebra de la operación de desecho o confirmación.

PostUpdate

Se invoca para una entidad después de que la entidad se haya actualizado en el almacén. Este método se invoca en la hebra de la operación de desecho o confirmación.

PostLoad

Se invoca para una entidad después de que la entidad se haya cargado del almacén, lo que todas las entidades que se cargan a través de una asociación. Este método se invoca en la hebra de la operación de carga, como `EntityManager.find` o una consulta.

Métodos de devolución de llamada de ciclo de vida duplicados

Si se definen varios métodos de devolución de llamada para un suceso de ciclo de vida de entidad, el orden de la invocación de estos métodos es el siguiente:

1. **Métodos de devolución de llamada del ciclo de vida definidos en los escuchas de entidad:** los métodos de devolución de llamada de ciclo de vida que están definidos en las clases de escucha de entidad para una clase de

entidad se invocan en el mismo orden que la especificación de las clases de escucha de entidad en la anotación EntityListeners o el descriptor XML.

2. **Superclase de escucha:** los métodos de devolución de llamada definidos en la superclase del escucha de entidad se invocan antes que los hijos.
3. **Métodos de ciclo de vida de entidad:** WebSphere eXtreme Scale no soporta la herencia de entidad, así que los métodos de ciclo de vida de entidad sólo se pueden definir en la clase de entidad.

Excepciones

Los métodos de devolución de llamada del ciclo de vida podría generar excepciones de tiempo de ejecución. Si un método de devolución de llamada de ciclo de vida genera una excepción de tiempo de ejecución dentro de una transacción, la transacción se retrotrae. No se invoca ningún método de devolución de llamada de ciclo de vida adicional después de que se genere una excepción de tiempo de ejecución.

Ejemplos de escucha de entidad

Puede escribir EntityListeners según sus necesidades. A continuación se ofrecen varios scripts de ejemplo.

Ejemplo de EntityListeners utilizando anotaciones

El siguiente ejemplo muestra las invocaciones al método de devolución de llamada de ciclo de vida y el orden de las invocaciones. Suponga que existe una clase de entidad Employee y dos escuchas de entidad: EmployeeListener y EmployeeListener2.

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

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

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

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

Si se produce un suceso PrePersist en una instancia Employee, se invocan los siguientes métodos en orden:

1. Método onEmployeePrePersist

2. Método onPersonPrePersist
3. Método onEmployeePrePersist2
4. Método checkEmployeeID

Ejemplos de escuchas de entidad utilizando XML

En el siguiente ejemplo se muestra cómo establecer un escucha de entidad en una entidad utilizando el archivo XML de descriptor de entidad:

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

La entidad Employee se configura con una clase de escucha de entidad com.ibm.websphere.objectgrid.sample.EmployeeListener, que tiene definidos dos métodos de devolución de llamada de ciclo de vida. El método onListenerPrePersist es para el suceso PrePersist y el método onListenerPostPersist es para el suceso PostPersist. Además, el método checkEmployeeID en la clase Employee se configura para escuchar el suceso PrePersist.

Soporte de planes de captación de EntityManager

Un FetchPlan es la estrategia que el gestor de entidades utiliza para recuperar los objetos asociados si la aplicación tiene que acceder a las relaciones.

Ejemplo

Supongamos por ejemplo que la aplicación tiene dos entidades: Department (Departamento) y Employee (Empleado). La relación entre la entidad Department y la entidad Employee es una relación bidireccional de uno a muchos: Un departamento tiene muchos empleados y un empleado pertenece a un solo departamento. Ya que la mayor parte de las veces en que se capte la entidad Department es probable que se capturen sus empleados, el tipo de captación de esta relación de uno a muchos se define como EAGER.

A continuación se ofrece un fragmento de código de la clase Department.

```
@Entity
public class Department {

    @Id
    private String deptId;

    @Basic
    String deptName;

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

}
```

En un entorno distribuido cuando una aplicación llama a `em.find(Department.class, "dept1")` para buscar una entidad `Department` con la clave "dept1", esta operación de búsqueda obtendrá la entidad `Department` y todas sus relaciones captadas de tipo EAGER. En el caso del fragmento de código anterior, estos son todos los empleados del departamento "dept1".

Antes de WebSphere eXtreme Scale 6.1.0.5, la recuperación de una entidad `Department` y `N` entidades `Employee` incurría en `N+1` trayectos cliente-servidor porque el cliente recuperaba una sola entidad por trayecto cliente-servidor. Puede mejorar el rendimiento si recupera estas `N+1` entidades en un solo trayecto.

Plan de captación

Un plan de captación se puede utilizar para personalizar cómo captar relaciones EAGER personalizando la profundidad máxima de las relaciones. La profundidad de captación sustituye las relaciones EAGER con una profundidad superior a la especificada por relaciones LAZY. De forma predeterminada, la profundidad de captación es la profundidad de captación máxima. Esto significa que se recuperarán todas las relaciones EAGER de todos los niveles navegables mediante EAGER desde la entidad raíz. Una relación EAGER es navegable mediante EAGER desde una entidad raíz si y sólo si todas las relaciones que se inician desde la entidad raíz a ella están configuradas como captadas mediante EAGER.

En el ejemplo anterior, la entidad `Employee` es navegable mediante EAGER desde la entidad `Department` porque la relación `Department-Employee` está configurada como captada mediante EAGER.

Si la entidad `Employee` tiene otra relación EAGER con una entidad `Address`, por ejemplo, entonces la entidad `Address` también será navegable mediante EAGER desde la entidad `Department`. Sin embargo, si las relaciones `Department-Employee` se configuraron como de captación LAZY, entonces la entidad `ADDRESS` no es navegable mediante EAGER desde la entidad `Department` porque la relación `Department-Employee` rompe la cadena de captación EAGER.

Se puede recuperar un objeto `FetchPlan` desde la instancia de `EntityManager`. La aplicación puede utilizar el método `setMaxFetchDepth` para cambiar la profundidad de captación máxima.

Un plan de captación está asociado con una instancia de `EntityManager`. El plan de captación se aplica a cualquier operación de captación, más concretamente como se indica a continuación.

- Operaciones de `EntityManager` `find(Class class, Object key)` y `findForUpdate(Class class, Object key)`
- Operaciones `Query`
- Operaciones `QueryQueue`

El objeto `FetchPlan` es mutable. Una vez cambiado, el valor cambiado se aplicará a las operaciones de captación ejecutadas posteriormente.

Un plan de captación es importante para un despliegue distribuido porque decide si las entidades de relación captadas mediante EAGER se recuperan con la entidad raíz en un solo trayecto cliente-servidor o más de uno.

Continuando con el ejemplo anterior, considere ahora que el plan de captación tiene la profundidad máxima definida como infinita. En este caso, cuando una

aplicación llama a `em.find(Department.class, "dept1")` para encontrar una entidad `Department`, esta operación de búsqueda obtendrá una entidad `Department` y `N` entidades `Employee` en un solo trayecto cliente-servidor. Sin embargo, para un plan de captación con una profundidad de captación máximo definida como cero, sólo se recuperará del servidor el objeto `Department`, mientras que las entidades `Employee` se recuperan del servidor sólo cuando se accede a la colección de empleados del objeto `Department`.

Planes de captación diferentes

Dispone de varios planes de captación diferentes según sus necesidades, que se explican en las secciones siguientes.

Impacto sobre una cuadrícula distribuida

- *Plan de captación de profundidad infinita:* Un plan de captación de profundidad infinita tiene su profundidad de captación máxima definida como `FetchPlan.DEPTH_INFINITE`.

En un entorno de cliente-servidor, si se utiliza un plan de captación de profundidad infinita, entonces se recuperarán todas las relaciones que sean navegables mediante `EAGER` desde la entidad raíz en un solo trayecto cliente-servidor.

Ejemplo: Si la aplicación está interesada en todas las entidades `Address` de todos los empleados de un determinado departamento, utiliza el plan de captación de profundidad infinita para recuperar todas las entidades `Address` asociadas. El código siguiente sólo incurre en un trayecto cliente-servidor.

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

tran.begin();
Department dept = (Department) em.find(Department.class, "dept1");
// hacer algo con el objeto Address.
for (Employee e: dept.employees) {
    for (Address addr: e.addresses) {
        // hacer algo con las direcciones.
    }
}
tran.commit();
```

- *Plan de captación de profundidad cero:* Un plan de captación de profundidad cero tiene su profundidad de captación máxima definida como 0.

En un entorno de cliente-servidor, si se utiliza un plan de captación cero, entonces sólo se recuperará la entidad raíz en el primer trayecto cliente-servidor. Todas las relaciones `EAGER` se tratan como si fueran `LAZY`.

Ejemplo: En este ejemplo, la aplicación sólo está interesada en el atributo de la entidad `Department` (Departamento). No necesita acceder a sus empleados, de modo que la aplicación define 0 como profundidad del plan de captación.

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

tran.begin();
Department dept = (Department) em.find(Department.class, "dept1");
// hacer algo con el objeto dept.
tran.commit();
```

- *Plan de captación con profundidad k:*

Un plan de captación con profundidad k tiene su profundidad de captación máxima definida como k .

En un entorno cliente-servidor de eXtreme Scale, si se utiliza un plan de captación de profundidad k -, entonces todas las relaciones EAGER navegables de la entidad raíz dentro de k pasos se recuperarán en el primer trayecto cliente-servidor.

El plan de captación de profundidad infinita ($k = \infty$) y el plan de captación de profundidad cero ($k = 0$) son sólo dos ejemplos del plan de captación de profundidad k -.

Para continuar ampliando el ejemplo anterior, supongamos que hay otra relación EAGER de la entidad Employee (Empleado) a la Address (Dirección). Si el plan de captación tiene la profundidad de captación máxima definida como 1, entonces la operación `em.find(Department.class, "dept1")` recuperará la entidad Department y todas sus entidades Employee en un solo trayecto cliente-servidor. No obstante, las entidades Address no se recuperarán porque no son navegables mediante EAGER a la entidad Department con 1 solo paso, sino en 2 pasos.

Si utiliza un plan de captación con una profundidad definida como 2, entonces la operación `em.find(Department.class, "dept1")` recuperará la entidad Department y todas sus entidades Employee, y todas las entidades Address asociadas con las entidades Employee en un solo trayecto cliente-servidor.

Consejo: El plan de captación predeterminado tiene una profundidad de captación máxima definida como infinito, de modo que el comportamiento predeterminado de una operación de captación puede cambiar. Se recuperan todas las relaciones navegables mediante EAGER de la entidad raíz. En lugar de varios trayectos, ahora la operación de captación sólo incurre en un trayecto cliente-servidor con el plan de captación predeterminado. Para mantener los valores para el producto de la versión anterior, defina la profundidad de captación como 0.

- *Plan de captación utilizado en la consulta:*

Si ejecuta una consulta de entidad, también puede utilizar un plan de captación para personalizar la recuperación de relaciones.

Por ejemplo, el resultado de la consulta `SELECT d FROM Department d WHERE "d.deptName='Department'"` tiene una relación con la entidad Department. Observe que la profundidad del plan de captación empieza con la asociación del resultado de la consulta: En este caso, la entidad Department, y no el propio resultado de la consulta. Es decir, la entidad Department está en el nivel de profundidad de captación 0. Por lo tanto, un plan de captación con una profundidad de captación máxima de 1 recuperará la entidad Department y sus entidades Employee en un solo trayecto cliente-servidor.

Ejemplo: En este ejemplo, la profundidad del plan de captación se define como 1, de modo que la entidad Department y sus entidades Employee se recuperan en un trayecto cliente-servidor, pero las entidades Address no se recuperarán en el mismo trayecto.

Importante: Si se ordena una relación, utilizando la anotación o la configuración `OrderBy`, entonces se considera una relación EAGER aunque se haya configurado como una captación LAZY.

Consideraciones sobre el rendimiento en un entorno distribuido

De forma predeterminada, todas las relaciones navegables mediante EAGER desde la entidad raíz se recuperarán en un solo trayecto cliente-servidor. Esto puede mejorar el rendimiento si se van a utilizar todas las relaciones. Sin embargo, en ciertos escenarios de uso, no se utilizan todas las relaciones navegables mediante

EAGER desde la entidad raíz, de modo que incurren tanto en una sobrecarga de tiempo de ejecución como de ancho de banda al recuperar las entidades no utilizadas.

Para estos casos, la aplicación puede definir un número pequeño como profundidad de captación máxima para disminuir la profundidad de las entidades que se deben recuperar realizando todas las relaciones EAGER después de dicha profundidad LAZY. Este valor puede aumentar el rendimiento.

Siguiendo aún más con el ejemplo Department-Employee-Address, de forma predeterminada, se recuperarán todas las entidades Address asociadas con empleados del departamento "dept1" cuando se llame a `em.find(Department.class, "dept1")`. Si la aplicación no utiliza entidades Address, puede definir la profundidad de captación máxima como 1, de modo que las entidades Address no se recuperarán con la entidad Department.

Impacto en el rendimiento de la interfaz EntityManager

Un entorno que obligase a cada aplicación a utilizar los mismos objetos de acceso a datos para una determinada base de datos resultaría muy poco práctico. Por lo contrario, la interfaz EntityManager que se proporciona con WebSphere eXtreme Scale separa las aplicaciones del estado mantenido en su almacén de datos de cuadrícula de servidor.

El coste de utilizar la interfaz EntityManager no es alto y depende del tipo de trabajo que se esté realizando. Utilice siempre la interfaz EntityManager y optimice la lógica empresarial decisiva una vez que se ha completado la aplicación. Puede reprocesar cualquier código que utilice interfaces EntityManager para utilizar correlaciones y tuples. Normalmente, este reproceso del código puede ser necesario para un diez por ciento del código.

Si utiliza relaciones entre objetos, el impacto sobre el rendimiento es menor porque una aplicación que utilice correlaciones necesita gestionar estas relaciones de forma parecida a la interfaz EntityManager.

Las aplicaciones que utilizan la interfaz EntityManager no necesitan proporcionar un ObjectTransformer porque se optimiza automáticamente.

Reproceso del código de EntityManager para correlaciones

A continuación se muestra una entidad de ejemplo:

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

A continuación se muestra parte del código para buscar la entidad y actualizarla:

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

A continuación se muestra el mismo código utilizando correlaciones y tuples:

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

// La modalidad de copia siempre es NO_COPY para las correlaciones de entidad
// si no se utiliza COPY_TO_BYTES.
// O bien necesitamos copiar el tuple o bien podemos solicitar que ObjectGrid
// lo haga:
map.setCopyMode(CopyMode.COPY_ON_READ);
s.begin();
Tuple value = (Tuple)map.get(key);
value.setAttribute(1, String.valueOf(inner));
map.update(key, value);
value = null;
s.commit();
```

Los dos fragmentos de código tienen el mismo resultado y una aplicación puede utilizar cualquiera de los dos fragmentos de código o ambos a la vez.

El segundo fragmento de código muestra cómo utilizar correlaciones directamente y cómo trabajar con tuples (los pares de clave y valor). El tuple de valor tiene tres atributos: firstName, middlename y surname, indexados en 0, 1 y 2 respectivamente. El tuple de clave tiene un solo atributo, el número de ID se indexa a cero. Puede ver cómo se crean los tuples utilizando los métodos EntityMetadata#getKeyMetaData o EntityMetadata#getValueMetaData. Debe utilizar estos métodos para crear tuples para una entidad. No puede implementar la interfaz Tuple y pasar una instancia de la implementación de tuple.

Agente de instrumentación

Puede mejorar el rendimiento de las entidades de acceso a campos habilitando el agente de instrumentación de WebSphere eXtreme Scale cuando se utiliza Java Development Kit (JDK) versión 1.5 o posterior.

Habilitación del agente de eXtreme Scale en JDK Versión 1.5 o posterior

El agente ObjectGrid se puede habilitar con una opción de línea de mandatos Java con la siguiente sintaxis>

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

El valor de *jarpath* es la vía de acceso a un archivo JAR (Java Archive) del tiempo de ejecución de eXtreme Scale que contiene una clase de agente eXtreme Scale y clases de soporte como, por ejemplo, los archivos objectgrid.jar, wsubjectgrid.jar, ogclient.jar, wsogclient.jar y ogagent.jar. Normalmente, en un programa autónomo de Java o en un entorno de Java Platform, Enterprise Edition que no ejecuta WebSphere Application Server, utilice el archivo objectgrid.jar o ogclient.jar. En un entorno de WebSphere Application Server o de varios cargadores de clases, debe utilizar el archivo ogagent.jar en la opción del agente de la línea de mandatos Java. Proporcione el archivo ogagent.config en la classpath o utilice las opciones de agente para especificar información adicional.

Opciones del agente de eXtreme Scale

config

Altera temporalmente el nombre de archivo de configuración.

include

Especifica o altera temporalmente la definición de dominio de transformación que es la primera parte del archivo de configuración.

exclude

Especifica o altera temporalmente la definición @Exclude.

fieldAccessEntity

Especifica o altera temporalmente la definición @FieldAccessEntity.

trace Especifica un nivel de rastreo. Los niveles pueden ser ALL, CONFIG, FINE, FINER, FINEST, SEVERE, WARNING, INFO y OFF.

trace.file

Especifica la ubicación del archivo de rastreo.

El punto y coma (;) se utiliza como delimitador para separar cada opción. La coma (,) se utiliza como delimitador para separar cada elemento dentro de una opción. El siguiente ejemplo demuestra la opción del agente eXtreme Scale para un programa Java:

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

Archivo ogagent.config

El archivo ogagent.config es el nombre del archivo de configuración del agente eXtreme Scale designado. Si el nombre de archivo está en la classpath, el agente eXtreme Scale encuentra y analiza el archivo. Puede alterar temporalmente el nombre de archivo designado a través de la opción config del agente de eXtreme Scale. En el siguiente ejemplo se muestra cómo especificar el archivo de configuración:

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

Un archivo de agente de configuración de eXtreme Scale tiene las partes siguientes:

- **Dominio de transformación:** la parte del dominio de transformación es la primera del archivo de configuración. El dominio de transformación es una lista de paquetes y clases que se incluyen en el proceso de transformación de clase. Este dominio de transformación debe incluir todas las clases que son clases de entidad de acceso a campos y otras clases que hacen referencia a estas clases de entidad de acceso a campos. Las clases de entidad de acceso a campos y las clases que hacen referencia a estas clases de entidad de acceso a campos construyen el dominio de transformación. Si piensa especificar clases de entidad de acceso a campos en la parte @FieldAccessEntity, no es necesario que aquí incluya clases de entidad de acceso a campos. El dominio de transformación debe haberse completado. Si no, es posible que vea una excepción FieldAccessEntityNotInstrumentedException.
- **@Exclude:** la señal @Exclude indica que los paquetes y las clases que se listan después de esta señal se excluyen del dominio de transformación.
- **@FieldAccessEntity:** la señal @FieldAccessEntity indica que los paquetes y las clases que se listan después de esta señal son clases y paquetes de entidad de acceso a campos. Si no existe ninguna línea después de la señal @FieldAccessEntity, su equivalente es "Ninguna @FieldAccessEntity especificada". El agente de eXtreme Scale determina que no se ha definido ninguna clase y paquete de entidad de acceso a campos. Si hay líneas después de la señal @FieldAccessEntity, estas representan las clases y paquetes de la entidad de acceso a campos especificada por el usuario. Por ejemplo, "dominio

de entidad de acceso a campos". El dominio de entidad de acceso a campos es un subdominio del dominio de transformación. Los paquetes y clases que se listan en el dominio de entidad de acceso a campos forman parte del dominio de transformación, incluso cuando no se listan en el dominio de transformación. La señal @Exclude, que lista paquetes y clases que se excluyen de la transformación, no tiene ningún impacto en el dominio de entidad de acceso a campos. Cuando se especifica la señal @FieldAccessEntity, todas las entidades de acceso a campos deben estar en este dominio de entidad de acceso a campos. De lo contrario, puede producirse una excepción FieldAccessEntityNotInstrumentedException.

Archivo de configuración de agente de ejemplo (ogagent.config)

```
#####
# El símbolo # indica línea de comentario
#####
# Es un archivo de configuración de agente ObjectGrid (el nombre de archivo designado es ogagent.config) que puede encontrar y analizar el agente ObjectGrid
# si está en la vía de acceso de clases.
# Si el nombre de archivo es "ogagent.config" y está en la vía de acceso de clases, las ejecuciones del programa Java -javaagent:objectgridRoot/ogagent.jar
# tendrán habilitado el objeto ObjectGrid.
# Si el nombre de archivo no es "ogagent.config" pero está en la vía de acceso de clases, puede especificar el nombre de archivo en la opción config del
# agente ObjectGrid
# -javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
# Vea los comentarios a continuación para obtener más información sobre cómo alterar temporalmente el valor de instrumentación.

# La primera parte de la configuración es la lista de paquetes y clases que deben incluirse en el dominio de transformación.
# Las inclusiones (paquetes/clases, construyen el dominio de instrumentación) deben estar al principio del archivo.
com.testpackage
com.testClass

# Dominio de transformación: las líneas anteriores son paquetes/clases que construyen el dominio de transformación.
# El sistema procesará clases con el nombre que empiece con los paquetes/clases anteriores para la transformación.
#
# Señal @Exclude: excluir del dominio de transformación.
# La señal @Exclude indica que los paquetes/clases que aparecen después de esa línea deben excluirse del dominio de transformación.
# Se utiliza cuando el usuario desea excluir algunos paquetes/clases de los paquetes incluidos especificados anteriormente
#
# Señal @FieldAccessEntity: dominio de entidad de acceso a campos.
# La señal @FieldAccessEntity indica que los paquetes/clases que aparecen después de esa línea son paquetes/clases de entidad de acceso
# a campos.
# Si no hay ninguna línea después de la señal @FieldAccessEntity, equivale a "Ninguna @FieldAccessEntity especificada".
# El tiempo de ejecución considerará que el usuario no especifica paquetes/clases de entidad de acceso a campos.
# El "dominio de entidad de acceso a campos" es un subdominio del dominio de transformación.
#
# Los paquetes/clases que se listan en el "dominio de entidad de acceso a campos" siempre formará parte del dominio de transformación,
# incluso cuando no se listen en el dominio de transformación.
# La señal @Exclude, que lista paquetes/clases que se han excluido de la transformación, no tiene ningún impacto en el "dominio de entidad de acceso
# a campos".
# Nota: cuando se especifica @FieldAccessEntity, todas las entidades de acceso a campos deben estar en este dominio de entidad de acceso a campos,
# de lo contrario, puede producirse una FieldAccessEntityNotInstrumentedException.
#
# El archivo de configuración del agente ObjectGrid es ogagent.config
# El tiempo de ejecución buscará este archivo como recurso en la vía de acceso de clases y lo procesará.
# Los usuarios pueden alterar temporalmente el nombre del archivo de configuración del agente ObjectGrid a través de la opción config del agente.
#
# por ejemplo,
# javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
#
# La definición de instrumentación, incluido el dominio de instrumentación, @Exclude y @FieldAccessEntity se pueden alterar individualmente
# mediante las correspondientes opciones de agente.
# Las opciones de agente designadas incluyen:
# include      -> se utiliza para alterar temporalmente la definición del dominio de instrumentación que es la primera parte del
#               archivo de configuración
# exclude     -> se utiliza para alterar temporalmente la definición @Exclude
# fieldAccessEntity -> se utiliza para alterar temporalmente la definición @FieldAccessEntity
#
# Cada opción de agente debe separarse mediante ","
# Dentro de la opción de agente, el paquete o la clase deben separarse mediante "."
#
# A continuación se muestra un ejemplo que no altera temporalmente el nombre del archivo de configuración:
# -javaagent:objectgridRoot/lib/objectgrid.jar=include=includedPackage;exclude=excludedPackage;fieldAccessEntity=package1,package2
#####

@Exclude
com.excludedPackage
com.excludedClass

@FieldAccessEntity
```

Consideración sobre el rendimiento

Para obtener un mejor rendimiento, especifique el dominio de transformación y el dominio de entidad de acceso a campos.

Colas de consulta de entidades

Las colas de consulta permiten a las aplicaciones crear una cola calificada por una consulta en el servidor o en eXtreme Scale local para una entidad. Las entidades del resultado de la consulta se almacenan en esta cola. Actualmente, la cola de consulta sólo se admite en una correlación que utilice la estrategia de bloqueo pesimista.

Varios clientes y transacciones comparten una cola de consulta. Una vez que la cola de consulta se queda vacía, la consulta de entidad asociada con esta cola se vuelve a ejecutar y los nuevos resultados se añaden a la cola. Una cola de consulta se identifica de forma exclusiva mediante la serie de consulta de entidad y los parámetros. Sólo hay una instancia para cada cola de consulta exclusiva en una instancia de ObjectGrid. Consulte la documentación de la API EntityManager para obtener más información.

Ejemplo de cola de consulta

El ejemplo siguiente muestra cómo se puede utilizar la cola de consulta.

```
/**
 * Obtener una tarea de tipo pregunta sin asignar
 */
private void getUnassignedQuestionTask() throws Exception {
    EntityManager em = og.getSession().getEntityManager();
    EntityTransaction tran = em.getTransaction();

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

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

El ejemplo anterior crea una cola de consulta QueryQueue con una serie de consulta de entidad, "SELECT t FROM Task t WHERE t.type=?1 AND t.status=?2". A continuación, establece los parámetros del objeto QueryQueue. Esta cola de consulta representa todas las tareas no asignadas del tipo "question" (pregunta). El objeto QueryQueue es muy parecido al objeto Query de entidad.

Una vez creado QueryQueue, se inicia una transacción de entidad y se invoca el método getNextEntity, que recupera la siguiente entidad disponible con un valor de tiempo de espera establecido en 10 segundos. Una vez recuperada la entidad, se procesa en el método assignTask. El método assignTask modifica la instancia de la entidad Task y cambia el estado a "assigned" (asignado), lo cual la elimina eficazmente de la cola, puesto que ya no coincide con el filtro de QueryQueue. Una vez asignada, la transacción se confirma.

Con este ejemplo, puede ver que una cola de consulta es similar a una consulta de entidad. Se diferencia, no obstante, en lo siguiente:

1. Las entidades de la cola de consulta pueden recuperarse de forma iterativa. La aplicación de usuario decide el número de entidades que se va a recuperar. Por ejemplo, si se utiliza QueryQueue.getNextEntity(timeout), sólo se recupera una entidad, y si se utiliza QueryQueue.getNextEntities(5, timeout), se recuperan 5 entidades. En un entorno distribuido, el número de entidades decide directamente el número de bytes que se transferirán del servidor al cliente.
2. Cuando se recupera una entidad de la cola de consulta, se coloca un bloqueo U en la entidad de modo que ninguna otra transacción pueda acceder a ella.

Recuperación de entidades en un bucle

Puede recuperar entidades en un bucle. A continuación se muestra un ejemplo que ilustra cómo obtener todas las tareas de tipo pregunta sin asignar.

```
/**
 * Obtener todas las tareas de tipo pregunta sin asignar
 */
private void getAllUnassignedQuestionTask() throws Exception {
    EntityManager em = og.getSession().getEntityManager();
    EntityTransaction tran = em.getTransaction();

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

    Task nextTask = null;

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

Si hay 10 tareas de tipo pregunta sin asignar en la correlación de entidad, esperaría tener 10 entidades impresas en la consola. No obstante, si ejecuta este ejemplo, observará que el programa nunca sale, que es lo contrario de lo que esperaba.

Cuando se crea una cola de consulta y se llama a `getNextEntity`, la consulta de entidad asociada con la cola se ejecuta y en la cola se muestran 10 resultados. Al llamar a `getNextEntity`, una entidad se extrae de la cola. Después de ejecutar 10 llamadas a `getNextEntity`, la cola se queda vacía. La cola de la entidad se volverá a ejecutar automáticamente. Puesto que estas 10 entidades siguen existiendo y coinciden con el criterio del filtro de la cola de consulta, se vuelven a colocar en la cola.

Si se añade la línea siguiente después de la sentencia `println()`, sólo verá impresas 10 entidades.

```
em.remove(nextTask);
```

Para obtener información sobre cómo utilizar `SessionHandle` con `QueryQueue` en un despliegue de colocación por contenedor, lea la información sobre Integración de `SessionHandle`.

Colas de consulta desplegadas en todas las particiones

En un entorno distribuido de eXtreme Scale, una cola de consulta puede crearse para una partición o para todas las particiones. Si se crea una cola de consulta para todas las particiones, habrá una instancia de cola de consulta en cada partición.

Cuando un cliente intenta obtener la siguiente entidad mediante el método `QueryQueue.getNextEntity` o `QueryQueue.getNextEntities`, el cliente envía una solicitud a una de las particiones. Un cliente envía solicitudes PEEK y PIN al servidor.

- Con una solicitud PEEK, el cliente envía una solicitud a una partición y el servidor responde inmediatamente. Si hay una entidad en la cola, el servidor envía una respuesta con la entidad; si no hay ninguna entidad, el servidor envía una respuesta sin ninguna entidad. En cualquier caso, el servidor responde inmediatamente.
- Con una solicitud PIN, el cliente envía una solicitud a una partición y el servidor espera hasta que haya una entidad disponible. Si hay una entidad en la cola, el servidor envía una respuesta con la entidad inmediatamente; si no hay ninguna entidad, el servidor espera en la cola hasta que haya una entidad disponible o hasta que la solicitud exceda el tiempo de espera.

El ejemplo siguiente muestra cómo se recupera una entidad de una cola de consulta que se despliega en todas las particiones (n):

1. Cuando se llama un método `QueryQueue.getNextEntity` o `QueryQueue.getNextEntities`, el cliente elige un número de partición aleatorio de 0 a n-1.
2. El cliente envía una solicitud PEEK a la partición aleatoria.
 - Si hay una entidad disponible, el método `QueryQueue.getNextEntity` o `QueryQueue.getNextEntities` sale después de devolver la entidad.
 - Si no hay ninguna entidad disponible y no es la última partición sin visitar, el cliente envía una solicitud PEEK a la siguiente partición.
 - Si no hay ninguna entidad disponible y es la última partición sin visitar, el cliente envía una solicitud PIN.
 - Si la solicitud PIN a la última partición excede el tiempo de espera y sigue sin haber ningún dato disponible, el cliente enviará una solicitud PEEK a todas las particiones en serie una vez más. Por lo tanto, si hubiera una entidad disponible en las particiones anteriores, el cliente podría obtenerla.

Entidad de subconjunto y soporte de no entidad

El método para crear un objeto `QueryQueue` en el gestor de entidades es el siguiente:

```
public QueryQueue createQueryQueue(String qlString, Class entityClass);
```

El resultado de la cola de la consulta se debe proyectar en el objeto definido por el segundo parámetro en el método, Clase `entityClass`.

Si se especifica este parámetro, la clase debe tener el mismo nombre de entidad que el especificado en la serie de consulta. Esto resulta útil si desea proyectar una entidad en una entidad de subconjunto. Si se utiliza un valor nulo como clase de entidad, el resultado no se proyectará. El valor almacenado en la correlación tendrá un formato de tuple de entidad.

Colisión de claves de cliente

En un entorno distribuido de eXtreme Scale, la cola de consulta sólo se admite en correlaciones de eXtreme Scale con modalidad de bloqueo pesimista. Por lo tanto, no hay memoria caché cercana en el cliente. No obstante, un cliente podría tener datos (clave y valor) en la correlación transaccional. Esto podría desembocar potencialmente en una colisión de claves cuando una entidad recuperada del servidor comparte la misma clave que una entrada de la correlación transaccional.

Cuando se produce una colisión de claves, el tiempo de ejecución del cliente de eXtreme Scale utiliza la siguiente norma para lanzar una excepción o alterar temporalmente los datos de forma silenciosa.

1. Si la clave de colisión es la clave de la entidad especificada en la consulta de entidad asociada con la cola de consulta, se emitirá una excepción. En este caso, la transacción se retrotrae, y el bloqueo U de esta clave de entidad se liberará en el servidor.
2. Por el contrario, si la clave de colisión es la clave de la asociación de la entidad, los datos de la correlación transaccional se alterarán temporalmente sin aviso.

La colisión de claves sólo sucede cuando hay datos en la correlación transaccional. Es decir, sólo tiene lugar cuando se llama a `getNextEntity` o `getNextEntities` en una transacción que ya estaba sucia (se han insertado datos nuevos o se han actualizado datos). Si una aplicación prefiere que no se produzcan colisiones de claves, debe llamar siempre a `getNextEntity` o `getNextEntities` en una transacción que no se haya ensuciado.

Anomalías de cliente

Una vez que un cliente envía una solicitud `getNextEntity` o `getNextEntities` al servidor, se puede producir una anomalía en el cliente de la siguiente manera:

1. El cliente envía una solicitud al servidor y concluye.
2. El cliente obtiene una o más entidades del servidor y después concluye.

En el primer caso, el servidor descubre que el cliente va a concluir cuando intenta responder al cliente. En el segundo caso, cuando el cliente obtiene una o más entidades del servidor, se coloca un bloqueo X en estas entidades. Si el cliente concluye, la transacción excederá el tiempo de espera y se liberará el bloqueo X.

Consulta con la cláusula `ORDER BY`

Por norma, las colas de consulta no reconocen la cláusula `ORDER BY`. Si llama a `getNextEntity` o `getNextEntities` en la cola de consulta, no se garantiza que las entidades se devuelvan en función del orden. La razón es que las entidades no se pueden ordenar en las particiones. En el caso de que la cola de consulta se despliegue en todas las particiones, cuando se ejecuta una llamada `getNextEntity` o `getNextEntities`, se elige una partición aleatoria para procesar la solicitud. Por lo tanto, no se garantiza el orden.

`ORDER BY` se reconoce si se despliega una cola de consulta en una sola partición.

Si desea más información consulte “API EntityManager Query” en la página 106.

Una llamada por transacción

Cada llamada a `QueryQueue.getNextEntity` o `QueryQueue.getNextEntities` recupera las entidades coincidentes de una partición aleatoria. Las aplicaciones deben llamar exactamente a una `QueryQueue.getNextEntity` o `QueryQueue.getNextEntities` en una transacción. De lo contrario, eXtreme Scale podría finalizar afectando a entidades de varias particiones, que provoca que se genere una excepción durante la confirmación.

Interfaz EntityTransaction

Puede utilizar la interfaz `EntityTransaction` para delimitar transacciones.

Finalidad

Para delimitar una transacción, puede utilizar la interfaz `EntityTransaction`, que está asociada a una instancia de gestor de entidad. Utilice el método `EntityManager.getTransaction` para recuperar la instancia `EntityTransaction` para el gestor de entidad. Cada instancia `EntityManager` y `EntityTransaction` está asociada a la `Session`. Puede delimitar transacciones con `EntityTransaction` o `Session`. Los métodos de la interfaz `EntityTransaction` no tienen ninguna excepción seleccionada. Sólo resultarán las excepciones de tiempo de ejecución de tipo `PersistenceException` o sus subclases.

Si desea más información sobre la interfaz `EntityTransaction`, consulte la documentación de la API la interfaz `EntityTransaction` en la documentación de la API.

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.

Recuperación de entidades y objetos (API de consulta)

`WebSphere eXtreme Scale` proporciona un motor de consultas flexible para recuperar entidades utilizando la API `EntityManager` y los objetos `Java` mediante la API `ObjectQuery`.

Funciones de consulta de `WebSphere eXtreme Scale`

Con el motor de consultas de `eXtreme Scale`, puede realizar las consultas del tipo `SELECT` en una entidad o esquema basado en objeto mediante el lenguaje de consulta de `eXtreme Scale`.

Este lenguaje de consulta ofrece las funciones siguientes:

- Resultados de valor único o de varios valores
- Funciones de agregación
- Ordenación y agrupación
- Uniones
- Expresiones condicionales con subconsultas
- Parámetros con nombre y posicionales
- Uso de índices de `eXtreme Scale`
- Sintaxis de expresiones `path` para la navegación de objetos
- Paginación

Interfaz de consultas

Utilice la interfaz de consultas para controlar la ejecución de consultas de entidad.

Utilice el método `EntityManager.createQuery(String)` para crear una consulta. Puede utilizar cada una de las instancias de consulta varias veces con la instancia de `EntityManager` en la que se recuperó.

Cada resultado de consulta genera una entidad, donde la clave de la entidad es el ID de la fila (de tipo `long`) y el valor de la entidad contiene los resultados del campo de la cláusula `SELECT`. Puede utilizar cada resultado de consulta en posteriores consultas.

Los métodos siguientes están disponibles en la interfaz `com.ibm.websphere.objectgrid.em.Query`.

public ObjectMap getResultMap()

El método `getResultMap` ejecuta una consulta `SELECT` y devuelve los resultados en un objeto `ObjectMap` con los resultados en el orden especificado por la consulta. El objeto `ObjectMap` resultante es sólo válido para la transacción actual.

La clave de la correlación es el número de resultado, de tipo `long`, que empieza por 1. El valor de la correlación es de tipo `com.ibm.websphere.projector.Tuple` donde cada atributo y asociación se denomina en función de su posición ordinal dentro de la cláusula `SELECT` de la consulta. Utilice este método para recuperar el objeto `EntityMetadata` para el objeto `Tuple` almacenado dentro de la correlación.

El método `getResultMap` es el método más rápido para recuperar los datos del resultado de la consulta donde pueden existir varios resultados. Puede recuperar el nombre de la entidad resultante mediante los métodos `ObjectMap.getEntityMetadata()` y `EntityMetadata.getName()`.

Ejemplo: la consulta siguiente devuelve dos filas.

```
String q1 = SELECT e.name, e.id, d from Employee e join e.dept d WHERE d.number=5
Query q = em.createQuery(q1);
ObjectMap resultMap = q.getResultMap();
long rowID = 1; // empieza con el índice 1
Tuple tResult = (Tuple) resultMap.get(new Long(rowID));
while(tResult != null) {
    // El primer atributo es name y tiene un nombre de atributo de 1
    // Pero tiene una posición ordinal de 0.
    String name = (String)tResult.getAttribute(0);
    Integer id = (String)tResult.getAttribute(1);

    // Dept es una asociación con un nombre 3, pero
    // una posición ordinal de 0 ya que es la primera asociación.
    // La asociación es siempre una relación de uno a uno,
    // por lo que sólo hay una clave.
    Tuple deptKey = tResult.getAssociation(0,0);
    ...
    ++rowID;
    tResult = (Tuple) resultMap.get(new Long(rowID));
}
}
```

public Iterator getResultIterator

El método `getResultIterator` ejecuta una consulta `SELECT` y devuelve los resultados de la consulta utilizando un `Iterator` donde cada resultado es un `Object` para una consulta de valor único, o una matriz de `Object` para una consulta de varios valores. Los valores del resultado `Object[]` se almacenan en el orden de la consulta. El objeto `Iterator` resultante es sólo válido para la transacción actual.

Este método es preferido para recuperar los resultados de la consulta dentro del contexto EntityManager. Puede utilizar de forma opcional el método setResultEntityName(String) para nombrar la entidad resultante, de modo que pueda utilizarse en otras consultas.

Ejemplo: la consulta siguiente devuelve dos filas.

```
String q1 = SELECT e.name, e.id, e.dept from Employee e WHERE e.dept.number=5
Query q = em.createQuery(q1);
Iterator results = q.getResultIterator();
while(results.hasNext()) {
    Object[] curEmp = (Object[]) results.next();
    String name = (String) curEmp[0];
    Integer id = (Integer) curEmp[1];
    Dept d = (Dept) curEmp[2];
    ...
}
```

public Iterator getResultIterator(Class resultType)

El método getResultIterator(Class resultType) ejecuta una consulta SELECT y devuelve los resultados de la consulta utilizando una instancia de Iterator. El tipo de entidad está determinado por el parámetro resultType. El objeto Iterator resultante es sólo válido para la transacción actual.

Utilice este método cuando desee utilizar las API EntityManager para acceder a las entidades resultantes.

Ejemplo: la consulta siguiente devuelve todos los empleados de una división y el departamento al que pertenecen, ordenados por sueldo. Para imprimir los cinco empleados con el sueldo más alto y seleccionar trabajar con empleados de sólo un departamento en el mismo conjunto de trabajo, utilice este código:

```
String string_q1 = "SELECT e.name, e.id, e.dept from Employee e
WHERE e.dept.division='Manufacturing' ORDER BY e.salary DESC";
Query query1 = em.createQuery(string_q1);
query1.setResultEntityName("AllEmployees");
Iterator results1 = query1.getResultIterator(EmployeeResult.class);
int curEmployee = 0;
System.out.println("Highest paid employees");
while (results1.hasNext() && curEmployee++ < 5) {
    EmployeeResult curEmp = (EmployeeResult) results1.next();
    System.out.println(curEmp);
    // Eliminar empleado del conjunto de resultados.
    em.remove(curEmp);
}

// Vaciar los cambios en la correlación de resultados.
em.flush();

// Ejecutar una consulta en el conjunto de trabajo local sin los empleados
// eliminados
String string_q2 = "SELECT e.name, e.id, e.dept from AllEmployees
e WHERE e.dept.name='Hardware'";
Query query2 = em.createQuery(string_q2);
Iterator results2 = query2.getResultIterator(EmployeeResult.class);
System.out.println("Subset list of Employees");
while (results2.hasNext()) {
    EmployeeResult curEmp = (EmployeeResult) results2.next();
    System.out.println(curEmp);
}
```

public Object getSingleResult

El método getSingleResult ejecuta una consulta SELECT que devuelve un único resultado.

Si la cláusula SELECT tiene más de un campo definido, el resultado es una matriz de objetos, donde cada elemento de la matriz se basa en su posición ordinal dentro de la cláusula SELECT de la consulta.

```
String q1 = "SELECT e from Employee e WHERE e.id=100"
Employee e = em.createQuery(q1).getSingleResult();

String q1 = "SELECT e.name, e.dept from Employee e WHERE e.id=100"
Object[] empData = em.createQuery(q1).getSingleResult();
String empName= (String) empData[0];
Department empDept = (Department) empData[1];
```

public Query setResultEntityName(String entityName)

El método setResultEntityName(String entityName) especifica el nombre de la entidad del resultado de la consulta.

Cada vez que se invocan los métodos getResultIterator o getResultMap, se crea dinámicamente una entidad con ObjectMap para que contenga los resultados de la consulta. Si la entidad no se especifica, o es nula, el nombre de ObjectMap y la entidad se generan automáticamente.

Como todos los resultados de la consulta están disponibles durante una transacción, no puede volver a usarse un nombre de consulta en una única transacción.

public Query setPartition(int partitionId)

Establezca la partición a la que se direcciona la consulta.

Este método es necesario si las correlaciones de la consulta se particionan y si el gestor de entidades no tiene afinidad con una partición única de entidad raíz de esquema.

Utilice PartitionManager Interface para determinar el número de particiones para la correlación de respaldo de una entidad dada.

La siguiente tabla proporciona descripciones de otros métodos que están disponibles a través de la interfaz de la consulta.

Tabla 2. Otros métodos

Método	Resultado
public Query setMaxResults(int maxResult)	Establece el número máximo de resultados que se va a recuperar.
public Query setFirstResult(int startPosition)	Establece la posición del primer resultado que se va a recuperar.
public Query setParameter(String name, Object value)	Enlaza un argumento con un parámetro con nombre.
public Query setParameter(int position, Object value)	Enlaza un argumento con un parámetro posicional.
public Query setFlushMode(FlushModeType flushMode)	Establece el tipo de modalidad de vaciado que se va a utilizar cuando se ejecuta la consulta, que alterará temporalmente el tipo de modalidad de vaciado establecido en EntityManager.

Elementos de las consultas de eXtreme Scale

Con el motor de consultas de eXtreme Scale, puede utilizar un lenguaje de consultas para realizar búsquedas en la memoria caché de eXtreme Scale. Este lenguaje de consulta puede consultar los objetos Java que están almacenados en los objetos ObjectMap y los objetos Entity. Use la sintaxis siguiente para crear una serie de consulta.

Una consulta de eXtreme Scale es una serie que contiene los elementos siguientes:

- Una cláusula SELECT que especifica los objetos y valores que se van a devolver.
- Una cláusula FROM que nombra las colecciones de objetos.
- Una cláusula WHERE opcional que contiene predicados de búsqueda en las colecciones.
- Una cláusula GROUP BY y HAVING opcional (consulte las funciones de agregación de consultas de eXtreme Scale).
- Una cláusula ORDER BY opcional que especifica el orden de la colección de resultados.

Las colecciones de objetos Java se identifican en las consultas a través del uso del nombre en la cláusula FROM de la consulta.

Los elementos del lenguaje de consultas se describen con más detalle en los siguientes temas relacionados:

- “BNF (Backus-Naur Form) de consulta de ObjectGrid” en la página 119
- “Referencia para consultas de eXtreme Scale” en la página 110

Los temas siguientes describen los métodos para utilizar la API Query:

- “API EntityManager Query” en la página 106
- “Utilización de la API ObjectQuery” en la página 101

Consulta de datos en varios husos horarios

En un escenario distribuido, las consultas se ejecutan en los servidores. Al consultar datos con predicados de tipo calendar, java.util.Date y timestamp, el valor de fecha y hora especificado en una consulta se basa en el huso horario local del servidor.

En un sistema de un solo huso horario donde todos los clientes y servidores se ejecutan en un mismo huso horario, no es necesario tener en cuenta cuestiones relacionadas con los tipos de predicado con calendar, java.util.Date y timestamp. No obstante, cuando los clientes y los servidores están en husos horarios diferentes, el valor de fecha y hora especificado en las consultas se basa en el huso horario del servidor y puede devolver datos no deseados al cliente. Si se desconoce el huso horario del servidor, el valor de fecha y hora especificado no tiene sentido. Por lo tanto, el valor de fecha y hora especificado debe tener en cuenta la diferencia de desplazamiento de huso horario entre el huso horario de destino y el huso horario del servidor.

Desplazamiento de huso horario

Por ejemplo, supongamos que un cliente está en el huso horario [GMT-0] y el servidor está en el huso horario [GMT-6]. El huso horario del servidor está 6 horas por detrás del cliente. El cliente querría ejecutar la consulta siguiente:

```
SELECT e FROM Employee e WHERE e.birthDate='1999-12-31 06:00:00'
```

Suponiendo que la entidad Employee (empleado) tiene un atributo birthDate (fecha de nacimiento) que es del tipo java.util.Date, el cliente está en el huso horario [GMT-0] y desea recuperar empleados con un valor de birthDate como, por ejemplo '1999-12-31 06:00:00 [GMT-0]' de acuerdo con su huso horario.

La consulta se ejecutará en el servidor y el valor de birthDate utilizado por el motor de consulta será '1999-12-31 06:00:00 [GMT-6]', que equivale a '1999-12-31 12:00:00 [GMT-0]'. Los empleados con un valor de birthDate igual a '1999-12-31 12:00:00 [la GMT-0]' se devolverán al cliente. De este modo, el cliente dejará de obtener empleados deseados con un valor de birthDate de '1999-12-31 06:00:00 [GMT-0]'.

El problema descrito ocurre debido a la diferencia de huso horario entre cliente y servidor. Para solucionar este problema, un método consiste en calcular el desplazamiento de huso horario entre el cliente y el servidor y aplicar el desplazamiento de huso horario al valor de fecha y hora de destino en la consulta. En el ejemplo de consulta anterior, el desplazamiento de huso horario es de -6 horas y el predicado birthDate ajustado debe ser "birthDate='1999-12-31 00:00:00'" si el cliente tiene la intención de recuperar a empleados con el valor de birthDate '12-31 06:00:00 [GMT-0]'. Con el valor de birthDate ajustado, el servidor utilizará '1999-12-31 00:00:00 [GMT-6]', que equivale al valor de destino '12-31 06:00:00 [GMT-0]', y se devolverán al cliente los empleados necesarios.

Despliegue distribuido en varios husos horarios

Si la cuadrícula de eXtreme Scale distribuida se despliega en varios servidores ObjectGrid en varios husos horarios, el método de de ajuste de desplazamiento del huso horario no funcionará porque el cliente no sabrá qué servidor ejecutará la consulta y, por lo tanto, no podrá determinar el desplazamiento de huso horario que debe utilizar. La única solución consiste en utilizar el sufijo 'Z' (no sensible a mayúsculas y minúsculas) en la fecha JDBC y el formato de escape de hora para indicar que se utiliza el valor de fecha y hora basado en el huso horario GMT. El sufijo 'Z' (no sensible a mayúsculas y minúsculas) indica que se debe utilizar el valor de fecha y hora basado en el huso horario GMT. Sin el sufijo 'Z', se utilizará el valor de fecha y hora basado en el huso horario local en el proceso que ejecuta la consulta.

La consulta siguiente equivale al ejemplo anterior, pero en su lugar utiliza el sufijo 'Z':

```
SELECT e FROM Employee e WHERE e.birthDate='1999-12-31 06:00:00Z'
```

La consulta debe encontrar empleados con un valor de birthDate '1999-12-31 06:00:00'. El sufijo 'Z' indica que el valor de birthDate especificado está basado en el huso horario GMT, de modo que el motor de consulta utilizará el valor de birthDate '1999-12-31 06:00:00 GMT-0]' para encontrar los valores de criterio. Los empleados con un valor de atributo birthDate igual a este valor de birthDate '1999-12-31 06:00:00 [GMT-0]' basado en GMT se incluirán en el resultado de la consulta. Utilizar el sufijo 'Z' en el formato de escape de fecha y hora JDBC en cualquier consulta resulta crucial para conseguir que el huso horario de las aplicaciones resulte seguro. Sin este método, el valor de fecha y hora se basa en el huso horario del servidor y no tiene sentido desde la perspectiva del cliente cuando los clientes y los servidores están en husos horarios diferentes.

Si desea más información, consulte el tema sobre la inserción de datos para husos horarios distintos en la *Visión general del producto*

Inserción de datos para husos horarios diferentes

Al insertar datos con los atributos `calendar`, `java.util.Date` y `timestamp` en un `ObjectGrid`, debe asegurarse de que estos atributos de fecha y hora se creen basándose en el mismo huso horario, sobre todo cuando se realiza el despliegue en diversos servidores en varios husos horarios. El uso de los mismos objetos de fecha y hora basados en el mismo huso horario puede garantizar que la aplicación sea segura por lo que respecta al huso horario y los datos se puedan consultar mediante los predicados `calendar`, `java.util.Date` y `timestamp`.

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

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

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

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

Utilización de la API ObjectQuery

La API `ObjectQuery` proporciona métodos para consultar datos en el `ObjectGrid` que se almacenan utilizando la API `ObjectMap`. Cuando se define un esquema en la instancia de `ObjectGrid`, la API `ObjectQuery` se puede utilizar para crear y ejecutar consultas sobre los objetos heterogéneos almacenados en las correlaciones de objeto.

Consultas y correlaciones de objeto

Puede utilizar una capacidad de consulta ampliada para los objetos que se han almacenado utilizando la API `ObjectMap`. Mediante estas consultas, puede recuperar objetos mediante atributos que no son de clave y realizar agregaciones simples, como `sum`, `avg`, `min` y `max` en todos los datos que coinciden con una consulta. Las aplicaciones pueden construir una consulta utilizando el método `Session.createObjectQuery`. Este método devuelve un objeto `ObjectQuery` que se puede interrogar para obtener los resultados de la consulta. Con el objeto `Query` también puede personalizar la consulta antes de ejecutarla. La consulta se ejecuta automáticamente cuando se llama a cualquier método que devuelva el resultado.

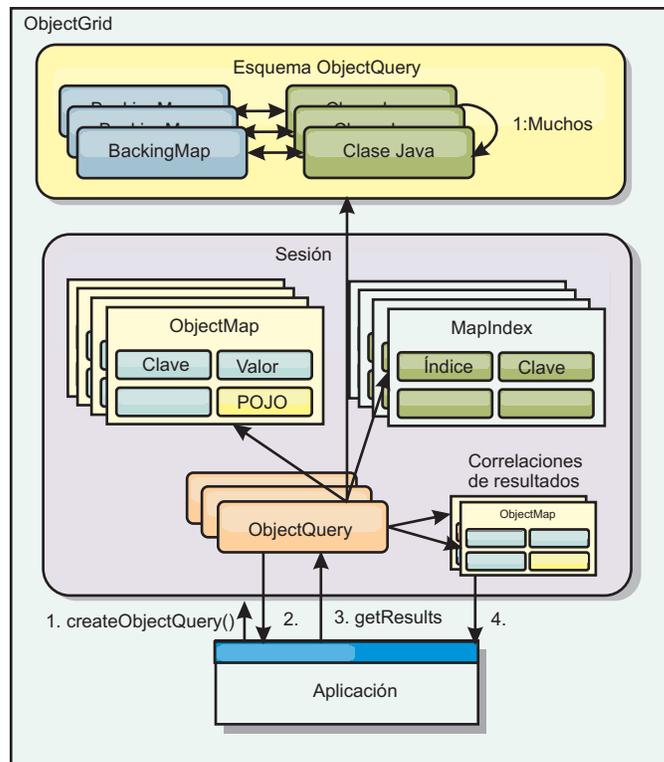


Figura 1. La interacción de la consulta con las correlaciones de objeto ObjectGrid y cómo se define un esquema para las clases y se asocia a una correlación de ObjectGrid

Definición de un esquema ObjectMap

Las correlaciones de objeto se utilizan para almacenar objetos en distintos formatos, de los que no son conscientes. Un esquema debe definirse en el objeto ObjectGrid que define el formato de los datos. Un esquema está formado por las siguientes partes:

- El tipo de objeto almacenado en ObjectMap.
- Las relaciones entre ObjectMaps.
- El método con el que cada consulta accederá a los atributos de los datos de los objetos (métodos de campos o propiedades).
- El nombre del atributo de la clave primaria del objeto.

Consulte el apartado Configuración de un esquema ObjectQuery para obtener más información.

Si desea un ejemplo sobre cómo crear un esquema a través de programa o utilizando el archivo XML descriptor de ObjectGrid, consulte la guía de aprendizaje sobre el ObjectQuery en *Visión general del producto*.

Consulta de objetos con la API ObjectQuery

La interfaz ObjectQuery permite consultar objetos que no son de entidad, que son objetos heterogéneos almacenados directamente en las ObjectMaps de ObjectGrid. La API ObjectQuery proporciona una forma fácil de encontrar objetos ObjectMap sin utilizar directamente los mecanismos de palabra clave e índice.

Existen dos métodos de recuperar resultados de un objeto `ObjectQuery`: `getResultIterator` y `getResultMap`.

Recuperación de resultados de la consulta mediante `getResultIterator`

Los resultados de la consulta son básicamente una lista de atributos. Imagine que la consulta era seleccionar a,b,c de X donde y=z. Esta consulta devuelve una lista de filas que contiene a, b y c. Esta lista se almacena en una correlación con ámbito de transacciones, que significa que debe asociar una clave artificial con cada fila y utilizar un entero que aumente con cada fila. Esta correlación se obtiene mediante el método `ObjectQuery.getResultMap()`. Puede acceder a los elementos de cada fila con un código similar al siguiente:

```
ObjectQuery q = session.createQuery(
    "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");

q.setParameter(1, "Claus");

Iterator iter = q.getResultIterator();
while(iter.hasNext())
{
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id "
        + row[objectgrid: 0 ] + ", firstName: "
        + row[objectgrid: 1 ] + ", surname: "
        + row[objectgrid: 2 ]);
}
```

Recuperación de resultados de la consulta mediante `getResultMap`

Los resultados de la consulta también se pueden recuperar mediante la correlación de resultados directamente. En el ejemplo siguiente se muestra una consulta que recupera partes específicas de clientes (Customers) coincidentes y muestra cómo acceder a las filas de resultados. Si utiliza el objeto `ObjectQuery` para acceder a los datos, el identificador de fila `long` generado no se muestra. La fila de tipo `long` sólo se muestra al utilizar `ObjectMap` para acceder al resultado.

Cuando finaliza la transacción, esta correlación desaparece. La correlación sólo está visible para la sesión utilizada, es decir, normalmente sólo para la hebra que la ha creado. La correlación utiliza una clave de tipo `Long` que representa el ID de la fila. Los valores almacenados en la correlación son de tipo `Object` u `Object[]`, donde cada elemento coincide con el tipo de elemento de la cláusula `select` de la consulta.

```
ObjectQuery q = em.createQuery(
    "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
q.setParameter(1, "Claus");
ObjectMap qmap = q.getResultMap();
for(long rowId = 0; true; ++rowId)
{
    Object[] row = (Object[]) qmap.get(new Long(rowId));
    if(row == null) break;
    System.out.println(" I Found a Claus with id " + row[0]
        + ", firstName: " + row[1]
        + ", surname: " + row[2]);
}
```

Si desea ejemplos sobre cómo utilizar `ObjectQuery`, consulte la la guía de aprendizaje de la API `ObjectQuery` en *Visión general del producto*.

Configuración de un esquema ObjectQuery

ObjectQuery se basa en información de esquema o de forma para realizar la comprobación semántica y evaluar expresiones path. En este apartado se describe cómo definir el esquema en el archivo XML o mediante programa.

Definición del esquema

El esquema ObjectMap se define en el archivo XML de descriptor de despliegue ObjectGrid o mediante programa con las técnicas normales de configuración de eXtreme Scale. Para obtener un ejemplo de cómo crear un esquema, consulte el apartado “Configuración de un esquema ObjectQuery”

La información del esquema describe los objetos POJO (plain old Java object): los atributos de los que se compone y los tipos de atributos, si los atributos son campos de clave primaria, relaciones de un valor o de varios valores o relaciones bidireccionales. La información de esquema indica a ObjectQuery que use el acceso de campos o el acceso de propiedades.

Atributos consultables

Cuando se define un esquema en ObjectGrid, se realiza una introspección en los objetos del esquema mediante el uso de un reflejo para determinar qué atributos están disponibles para realizar la consulta. Puede consultar los siguientes tipos de atributo:

- Los tipos primitivos Java que incluyen derivadores:
- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.util.Calendar
- byte[]
- java.lang.Byte[]
- char[]
- java.lang.Character[]
- J2SE enum

Los tipos serializables incorporados distintos de los mencionados anteriormente también pueden incluirse en un resultado de la consulta, pero no pueden incluirse en la cláusula WHERE o FROM de la consulta. Los atributos serializables no son navegables.

Los tipos de atributo pueden excluirse del esquema si el tipo no es serializable, el campo o propiedad es estático o el campo es transitorio. Puesto que todos los objetos de correlación se deben serializar, el ObjectGrid sólo incluye atributos que se pueden persistir en el objeto. Los otros objetos se pasan por alto.

Atributos de campos

Cuando el esquema se configura para acceder al objeto mediante campos, todos los campos serializables, no transitorios se incorporan automáticamente al esquema. Para seleccionar un atributo de campo en una consulta, utilice el nombre del identificador de campo tal y como existe en la definición de clase.

Todos los campos protegidos, protegidos por paquetes, públicos y privados se incluyen en el esquema.

Atributos de propiedades

Cuando el esquema se configura para acceder al objeto mediante propiedades, todos los métodos serializables que siguen los convenios de denominación de la propiedad JavaBeans se incorporarán automáticamente en el esquema. Para seleccionar un atributo de propiedad para la consulta, utilice los convenios de denominación de propiedad del estilo JavaBeans.

Todas las propiedades protegidas, protegidas por paquetes, públicas y privadas se incluyen en el esquema.

En la clase siguiente, se han añadido al esquema estos atributos: name, birthday, valid.

```
public class Person {
    public String getName(){}
    private java.util.Date getBirthday(){}
    boolean isValid(){}
    public NonSerializableObject getData(){}
}
```

Si se utiliza CopyMode de COPY_ON_WRITE, el esquema de la consulta siempre debe utilizar el acceso basado en la propiedad. COPY_ON_WRITE crea objetos proxy siempre que los objetos se recuperen de la correlación y sólo puede acceder a dichos objetos mediante los métodos de propiedad. Si no se hace de esa manera, cada resultado de la consulta se establecerá en el valor nulo.

Relaciones

Cada relación se debe definir explícitamente en la configuración del esquema. El tipo de atributo determina automáticamente la cardinalidad de la relación. Si el atributo implementa la interfaz java.util.Collection, la relación es una relación de uno a muchos o de muchos a muchos.

A diferencia de las consultas de entidad, los atributos que se refieren a otros objetos almacenados en memoria caché no deben almacenar referencias directas al objeto. Las referencias a otros objetos se serializan como parte de los datos del objeto que contienen. Almacene la clave para el objeto relacionado.

Por ejemplo, si hay una relación de muchos a uno entre Customer y Order:

Incorrecto.

Almacenar una referencia de objeto.

```
public class Customer {
    String customerId;
    Collection<Order> orders;
}
```

```
public class Order {
    String orderId;
    Customer customer;
}
```

Correcto. Clave para el objeto relacionado.

```
public class Customer {
    String customerId;
    Collection<String> orders;
}
```

```
public class Order {
    String orderId;
    String customer;
}
```

Cuando se ejecuta una consulta que une dos objetos de correlación, la clave se infla automáticamente. Por ejemplo, la consulta siguiente devuelve objetos Customer:

```
SELECT c FROM Order o JOIN Customer c WHERE orderId=5
```

Uso de índices

ObjectGrid utiliza plug-ins de índice para añadir índices a correlaciones. El motor de consultas incorpora automáticamente los índices definidos en un elemento de correlación de esquemas del tipo:

`com.ibm.websphere.objectgrid.plugins.index.HashIndex` y la propiedad `rangeIndex` se establece en `true`. Si el tipo de índice no es `HashIndex` y la propiedad `rangeIndex` no se establece en `true`, la consulta pasa por alto el índice. Consulte *Guía de aprendizaje de consulta de objeto* la guía de aprendizaje de `ObjectQuery` en *Visión general del producto* si desea ver un ejemplo sobre cómo añadir un índice al esquema.

API EntityManager Query

La API `EntityManager` proporciona métodos para consultar datos en `ObjectGrid` almacenados mediante la API `EntityManager`. La API `EntityManager Query` se utiliza para crear y ejecutar consultas sobre una o más entidades definidas en `eXtreme Scale`.

Consulta y ObjectMaps de entidades

`WebSphere Extended Deployment v6.1` ha presentado y ampliado la capacidad de consulta para las entidades almacenadas en `eXtreme Scale`. Estas consultas permiten recuperar objetos utilizando atributos no de clave y realizar agregaciones sencillas como, por ejemplo, `sum`, `average`, `minimum` y `maximum` en todos los datos que coinciden con una consulta. Las aplicaciones construyen una consulta mediante la API `EntityManager.createQuery`. Ésta devuelve un objeto `Query`, que puede después interrogarse para obtener los resultados de la consulta. Con el objeto `Query` también puede personalizar la consulta antes de ejecutarla. La consulta se ejecuta automáticamente cuando se llama a cualquier método que devuelva el resultado.

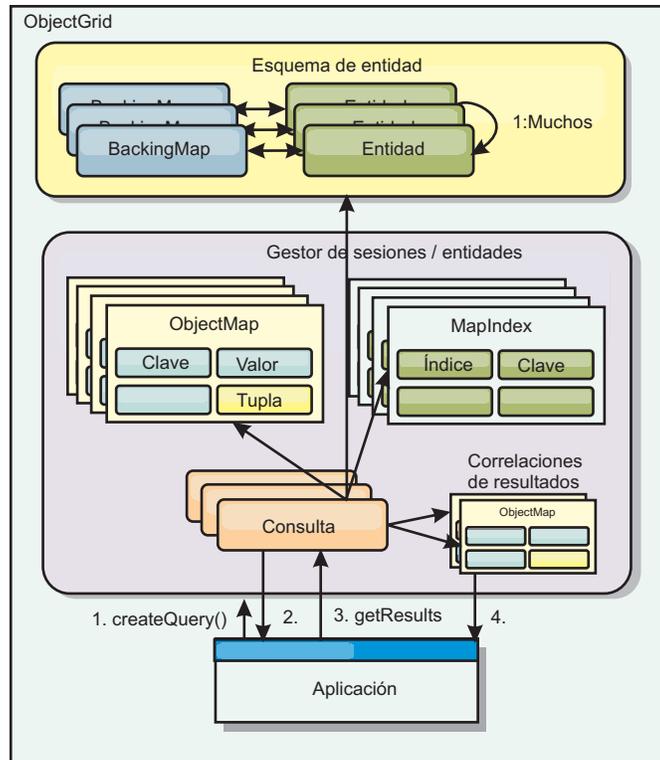


Figura 2. La interacción de la consulta con las correlaciones de objeto ObjectGrid y cómo se define y asocia el esquema de entidad con una correlación de ObjectGrid.

Recuperaciones de resultados de consulta utilizando el método getResultIterator

Los resultados de la consulta son una lista de atributos. Si la consulta era seleccionar a,b,c de X donde y=z, se devolverá una lista de filas que contengan a, b y c. Esta lista se almacena en una correlación con ámbito de transacciones, que significa que debe asociar una clave artificial con cada fila y utilizar un entero que aumente con cada fila. Esta correlación se obtiene a través del método Query.getResultMap. La correlación tiene EntityMetaData, que describe cada fila de la correlación asociada con ella. Puede acceder a los elementos de cada fila con un código similar al siguiente:

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
q.setParameter(1, "Claus");

Iterator iter = q.getResultIterator();
while(iter.hasNext())
{
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id " + row[objectgrid: 0 ]
        + ", firstName: " + row[objectgrid: 1 ]
        + ", surname: " + row[objectgrid: 2 ]);
}
```

Recuperación de resultados de la consulta mediante getResultMap

El siguiente código muestra la recuperación de partes específicas de clientes (Customers) coincidentes y muestra cómo acceder a las filas de resultados. Si utiliza el objeto Query para acceder a los datos, el identificador de fila long generado no se muestra. El tipo long sólo se muestra al utilizar ObjectMap para acceder al resultado. Cuando la transacción se completa, esta Map desaparece. La

correlación sólo está visible para el objeto Session utilizado, es decir, normalmente sólo para la hebra que la ha creado. La correlación utiliza el tuple para la clave con un atributo único, un tipo long con el ID de fila. El valor es otro tuple con un atributo para cada columna del conjunto de resultados.

Observe el siguiente código de ejemplo:

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from
Customer c where c.surname=?1");
q.setParameter(1, "Claus");
ObjectMap qmap = q.getResultMap();
Tuple keyTuple = qmap.getEntityMetadata().getKeyMetadata().createTuple();
for(long i = 0; true; ++i)
{
    keyTuple.setAttribute(0, new Long(i));
    Tuple row = (Tuple)qmap.get(keyTuple);
    if(row == null) break;
    System.out.println(" I Found a Claus with id " + row.getAttribute(0)
        + ", firstName: " + row.getAttribute(1)
        + ", surname: " + row.getAttribute(2));
}
```

Recuperación de resultados de la consulta mediante un iterador de resultados de entidad

El código siguiente muestra la consulta y el bucle que recupera cada fila de resultado mediante el uso de las API de correlación normales. La clave de la correlación es un tuple. Por lo tanto, si se construye uno de los tipos correctos mediante el método createTuple, resulta en keyTuple. Intente recuperar todas las filas con los ID de fila de 0 en adelante. Cuando get devuelva null (que indica que no se ha encontrado la clave), el bucle termina. Establezca el primer atributo de keyTuple para que tenga la longitud que desea encontrar. El valor devuelto por get también es un tuple con un atributo para cada columna en el resultado de la consulta. Después, extraiga cada atributo del valor Tuple mediante getAttribute.

A continuación se muestra un fragmento de código que ilustra lo descrito:

```
Query q2 = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
q2.setResultEntityName("CustomerQueryResult");
q2.setParameter(1, "Claus");

Iterator iter2 = q2.getResultIterator(CustomerQueryResult.class);
while(iter2.hasNext())
{
    CustomerQueryResult row = (CustomerQueryResult)iter2.next();
    // firstName es el ID no el valor de firstName.
    System.out.println("Found a Claus with id " + row.id
        + ", firstName: " + row.firstName
        + ", surname: " + row.surname);
}

em.getTransaction().commit();
```

Se especifica un valor de ResultEntityName en la consulta. Este valor indica al motor de consultas que desea proyectar cada fila en un objeto específico, CustomerQueryResult en este caso. La clase es la siguiente:

```
@Entity
public class CustomerQueryResult {
    @Id long rowId;
    String id;
    String firstName;
    String surname;
};
```

En el primer fragmento de código, observe que cada fila de la consulta se devuelve como objeto CustomerQueryResult en lugar de como Object[]. Las columnas de

resultado de la consulta se proyectan al objeto CustomerQueryResult. Proyectar el resultado es ligeramente más lento en el tiempo de ejecución, pero más legible. Las entidades del resultado de la consulta no se deben registrar con eXtreme Scale en el arranque. Si las entidades se registran, se crea una correlación global con el mismo nombre y la consulta falla con un error que indica un nombre de correlación duplicado.

Consultas sencillas con EntityManager

WebSphere eXtreme Scale incluye la API de consulta EntityManager.

La API de consulta EntityManager es muy similar a otros motores de consulta SQL que realizan consultas en objetos. Se define una consulta, y el resultado se recupera de la consulta mediante diversos métodos getResult.

Los siguientes ejemplos hacen referencia a las entidades utilizadas en la guía de aprendizaje EntityManager en la visión general del producto.

Ejecución de una consulta sencilla

En este ejemplo, se realiza una consulta en los clientes con el apellido Claus:

```
em.getTransaction().begin();

    Query q = em.createQuery("select c from Customer c where c.surname='Claus'");

    Iterator iter = q.getResultIterator();
    while(iter.hasNext())
    {
        Customer c = (Customer)iter.next();
        System.out.println("Found a claus with id " + c.id);
    }

    em.getTransaction().commit();
```

Uso de parámetros

Puesto que desea buscar todos los clientes que se apelliden Claus, se utiliza un parámetro para especificar el apellido ya que puede que deba realizar esta consulta más de una vez.

Ejemplo de parámetro posicional

```
Query q = em.createQuery("select c from Customer c where c.surname=?1");
    q.setParameter(1, "Claus");
```

El uso de parámetros es muy importante si la consulta se va a utilizar más de una vez. EntityManager debe analizar la serie de consulta y crear un plan para la consulta, lo cual resulta costoso. Mediante el uso de un parámetro, EntityManager almacena en memoria caché el plan de la consulta, por lo que se reduce el tiempo que se tarda en ejecutar una consulta.

Se utilizan los parámetros posicionales y los parámetros con nombre:

Ejemplo de parámetro con nombre

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
    q.setParameter("name", "Claus");
```

Uso de un índice para mejorar el rendimiento

Si hubiera millones de clientes, la consulta anterior tendría que explorar todas las filas de la correlación de clientes. Esto no es muy eficiente. Pero eXtreme Scale proporciona un mecanismo para definir índices en atributos individuales en una entidad. La consulta utiliza de forma automática este índice cuando corresponda, lo cual acelera las consultas significativamente.

Puede especificar qué atributos indizar; el procedimiento es sencillo, basta con utilizar la anotación `@Index` en el atributo de la entidad:

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

EntityManager crea un índice ObjectGrid apropiado para el atributo de apellido en la entidad Customer, que el motor de consultas utiliza automáticamente. De esta manera, se reduce drásticamente el tiempo de la consulta.

Uso de paginación para mejorar el rendimiento

Si hubiera un millón de clientes con el apellido Claus, no sería útil mostrar una página con el millón de clientes. Lo más conveniente sería mostrar 10 o 25 clientes cada vez.

Con los métodos Query `setFirstResult` y `setMaxResults` se puede especificar que sólo se devuelva un subconjunto de los resultados.

Ejemplo de paginación

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
q.setParameter("name", "Claus");
// Mostrar la primera página
q.setFirstResult=1;
q.setMaxResults=25;
displayPage(q.getResultIterator());

// Mostrar la segunda página
q.setFirstResult=26;
displayPage(q.getResultIterator());
```

Referencia para consultas de eXtreme Scale

WebSphere eXtreme Scale tiene su propio lenguaje mediante el cual el usuario puede consultar datos.

Cláusula FROM de consulta de ObjectGrid

La cláusula FROM especifica las colecciones de objetos a los que se aplica la consulta. Cada colección se identifica por un nombre de esquema abstracto y una variable de identificación, llamada variable de rango, o por una declaración de miembro de colección que identifica una relación de un solo valor o de varios valores y una variable de identificación.

Conceptualmente, la semántica de la consulta es primero formar una colección temporal de tuples, denominados R. Los tuples están compuestos de elemento de las colecciones que se identifican en la cláusula FROM. Cada tuple contiene un elemento de cada una de las colecciones en la cláusula FROM. Se forman todas las combinaciones posibles sujetas a las limitaciones que se imponen por las declaraciones de miembros de colección. Si algún nombre de esquema identifica una colección para la que no hay ningún registro en el almacén persistente, la colección temporal R está vacía.

Ejemplos del uso de FROM

El objeto DeptBean contiene los registros 10, 20 y 30. El objeto EmpBean contiene los registros 1, 2 y 3 que están relacionados con el departamento 10 y los registros 4 y 5 que están relacionados con el departamento 20. El departamento 30 no tiene empleados asociados.

```
FROM DeptBean d, EmpBean e
```

Esta cláusula forma una colección temporal R que contiene 15 tuples.

```
FROM DeptBean d, DeptBean d1
```

Esta cláusula forma una colección temporal R que contiene 9 tuples.

```
FROM DeptBean d, IN (d.emps) AS e
```

Esta cláusula forma una colección temporal R que contiene 5 tuples. El departamento 30 no está en la colección temporal R porque no contiene ningún empleado. El departamento 10 está contenido tres veces en la colección temporal R y el departamento está contenido dos veces en R.

En lugar de usar IN(d.emps) as e, puede usar un predicado JOIN:

```
FROM DeptBean d JOIN d.emps as e
```

Después de formar la colección temporal, las condiciones de búsqueda de cláusula WHERE se aplican a la colección temporal R, lo que da una nueva colección temporal R1. Las cláusulas ORDER BY y SELECT se aplican a R1 para producir el conjunto de resultados final.

Una variable de identificación es una variable que se declara en la cláusula FROM utilizando el operador IN o el operador AS opcional.

```
FROM DeptBean AS d, IN (d.emps) AS e
```

es equivalente a:

```
FROM DeptBean d, IN (d.emps) e
```

Una variable de identificación que se declara de modo que sea un nombre de esquema abstracto se llama variable de rango. En la consulta anterior, "d" es una variable de rango. Una variable de identificación que se declara de modo que sea una expresión path de varios valores se llama declaración de miembro de colección. Los valores "d" y "e" en el ejemplo anterior son declaraciones de miembro de colección.

A continuación se muestra un ejemplo del uso de una expresión path de un solo valor en la cláusula FROM:

```
FROM EmpBean e, IN(e.dept.mgr) as m
```

Cláusula SELECT de consulta de ObjectGrid

La sintaxis de la cláusula SELECT se ilustra en el siguiente ejemplo:

```
SELECT { ALL | DISTINCT } [ selection , ]* selection
selection ::= {single_valued_path_expression |
               identification_variable |
               OBJECT ( identification_variable) |
               aggregate_functions } [[ AS ] id ]
```

La cláusula SELECT consta de uno o más de los siguientes elementos: una sola variable de identificación que se define en la cláusula FROM, una expresión path de un solo valor que se evalúa en valores o referencias de objetos, y una función agregada. Puede usar la palabra clave DISTINCT para eliminar las referencias duplicadas.

Una subselección de escala es una subselección que devuelve un valor individual:

Ejemplos del uso de SELECT

Buscar todos los empleados que ganan más que el empleado John:

```
SELECT OBJECT(e) FROM EmpBean ej, EmpBean e WHERE ej.name = 'John' and
e.salary > ej.salary
```

Buscar todos los departamentos que tienen uno o más empleados que ganan menos que 20000:

```
SELECT DISTINCT e.dept FROM EmpBean e where e.salary < 20000
```

Una consulta puede tener una expresión path que se evalúa en un valor arbitrario:

```
SELECT e.dept.name FROM EmpBean e where e.salary < 20000
```

La consulta anterior devuelve una colección de valores de nombre para los departamentos que tienen empleados que ganan menos de 20000.

Una consulta puede devolver un valor agregado:

```
SELECT avg(e.salary) FROM EmpBean e
```

A continuación se muestra una consulta que recupera los nombres y referencias de objetos para los empleados con sueldo bajo:

```
SELECT e.name as name , object(e) as emp from EmpBean e where e.salary <
50000
```

Cláusula WHERE de consulta de ObjectGrid

La cláusula WHERE contiene condiciones de búsqueda que están compuestas de los elementos indicados a continuación. Cuando una condición de búsqueda se evalúa en TRUE, el tuple se añade al conjunto de resultados.

Literales de consulta de ObjectGrid

Un literal de serie se especifica en comillas simples. Una comilla simple que se encuentra dentro de un literal de serie se representa mediante dos comillas simples, por ejemplo: "Tom"'s'.

Un literal numérico puede ser cualquiera de los siguientes valores:

- Un valor exacto, como 57, -957 o +66
- Cualquier valor soportado por el tipo long de Java
- Un literal decimal como 57,5 o -47,02
- Un valor numérico aproximado como 7E3 o -57,4E-2
- Los tipos Float deben incluir el cualificador "F"; por ejemplo 1.0F
- Los tipos Long deben incluir el cualificador "L"; por ejemplo 123L

Los literales booleanos son TRUE y FALSE.

Los literales temporales siguen la sintaxis de escape JDBC en base al tipo de atributo:

- java.util.Date: aaaa-mm-ss
- java.sql.Date: aaaa-mm-ss
- java.sql.Time: hh-mm-ss
- java.sql.Timestamp: aaaa-mm-dd hh:mm:ss.f...
- java.util.Calendar: aaaa-mm-dd hh:mm:ss.f...

Los literales enum se expresan utilizando la sintaxis de literales enum de Java utilizando el nombre de clase enum plenamente calificado.

Parámetros de entrada de consulta de ObjectGrid

Puede especificar parámetros de entrada utilizando una posición ordinal o utilizando un nombre de variable. Se recomienda grabar consultas que utilicen parámetros de entrada, porque si se usan parámetros de entrada se aumentará el rendimiento permitiendo a ObjectGrid captar el plan de consulta entre acciones en ejecución.

Un parámetro de entrada puede adoptar cualquiera de los tipos siguientes: Byte, Short, Integer, Long, Float, Double, BigDecimal, BigInteger, String, Boolean, Char, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar, un Java SE 5 enum, una objeto Entity o POJO, o una serie de datos binarios con el formato de Java byte[].

Un parámetro de entrada no debe tener un valor nulo. Para buscar la aparición de un valor nulo (NULL), utilice el predicado NULL.

Parámetros posicionales

Los parámetros de entrada posicionales se definen utilizando el signo de interrogación seguido de un número positivo:

?[entero positivo].

Los parámetros de entrada posicionales se enumeran empezando por 1 y corresponden a los argumentos de la consulta; por lo tanto, una consulta no puede contener un parámetro de entrada que supera el número de argumentos de entrada.

Ejemplo: `SELECT e FROM Employee e WHERE e.city = ?1 and e.salary >= ?2`

Parámetros con nombre

Los parámetros de entrada con nombre se definen utilizando un nombre de variable en el formato: `:[nombre de parámetro]`.

Ejemplo: `SELECT e FROM Employee e WHERE e.city = :city and e.salary >= :salary`

Predicado BETWEEN de consulta de ObjectGrid

El predicado BETWEEN determina si un valor dado está comprendido entre dos otros valores dados.

`expression [NOT] BETWEEN expression-2 AND expression-3`

Ejemplo 1

`e.salary BETWEEN 50000 AND 60000`

es equivalente a:

`e.salary >= 50000 AND e.salary <= 60000`

Ejemplo 2

`e.name NOT BETWEEN 'A' AND 'B'`

es equivalente a:

`e.name < 'A' OR e.name > 'B'`

Predicado IN de consulta de ObjectGrid

El predicado IN compara un valor con un conjunto de valores. Puede utilizar el predicado IN de dos formas distintas:

`expression [NOT] IN (subselect) expression [NOT] IN (value1, value2,)`

El valor ValueN puede ser un valor literal o un parámetro de entrada. La expresión no se puede evaluar en un tipo de referencia.

Ejemplo 1

e.salary IN (10000, 15000)

es equivalente a

(e.salary = 10000 OR e.salary = 15000)

Ejemplo 2

e.salary IN (select e1.salary from EmpBean e1 where e1.dept.deptno = 10)

es equivalente a

e.salary = ANY (select e1.salary from EmpBean e1 where e1.dept.deptno = 10)

Ejemplo 3

e.salary NOT IN (select e1.salary from EmpBean e1 where e1.dept.deptno = 10)

es equivalente a

e.salary <> ALL (select e1.salary from EmpBean e1 where e1.dept.deptno = 10)

Predicado LIKE de consulta de ObjectGrid

El predicado LIKE busca un valor de serie para un patrón determinado.

string-expression [NOT] LIKE pattern [ESCAPE escape-character]

El valor del patrón es un literal de serie o un marcador de parámetro de tipo serie en el que el subrayado (_) representa un carácter individual y un signo de porcentaje (%) representa cualquier secuencia de caracteres, incluida una secuencia vacía. Cualquier otro carácter se representa a sí mismo. El carácter de escape se puede utilizar para buscar el carácter _ y %. El carácter de escape se puede especificar como literal de serie o como parámetro de entrada.

Si la expresión de serie es nula, entonces el resultado se desconoce.

Si ambas expresiones de serie y de patrón están vacías, entonces el resultado es true.

Ejemplo

' ' LIKE ' ' is true

' ' LIKE '%' is true

e.name LIKE '12%3' is true for '123' '12993' and false for '1234'

e.name LIKE 's_me' is true for 'some' and 'same', false for 'soome'

e.name LIKE '/_foo' escape '/' is true for ' _foo', false for 'afoo'

e.name LIKE '/_foo' escape '/' is true for ' /afoo' and for ' /bfoo'

e.name LIKE '///_foo' escape '/' is true for ' /_foo' but false for ' /afoo'

Predicado NULL de consulta de ObjectGrid

El predicado NULL comprueba los valores nulos.

{single-valued-path-expression | input_parameter} IS [NOT] NULL

Ejemplo

```
e.name IS NULL  
e.dept.name IS NOT NULL  
e.dept IS NOT NULL
```

Predicado de colección EMPTY de consulta de ObjectGrid

Utilice el predicado de colección EMPTY para comprobar una colección vacía.

Para comprobar si una relación de varios valores está vacía, utilice la siguiente sintaxis:

```
collection-valued-path-expression IS [NOT] EMPTY
```

Ejemplo

Predicado de colección vacía. Para buscar todos los departamentos que no tienen empleados:

```
SELECT OBJECT(d) FROM DeptBean d WHERE d.emps IS EMPTY
```

Predicado MEMBER OF de consulta de ObjectGrid

La siguiente expresión comprueba si la consulta de objeto especificada por el parámetro de entrada o la expresión path de un solo valor es miembro de la colección indicada. Si la expresión path con valor de colección designa una colección vacía, el valor de la expresión MEMBER OF es FALSE.

```
{ single-valued-path-expression | input_parameter } [ NOT ] MEMBER [ OF ]  
collection-valued-path-expression
```

Ejemplo

Buscar empleados que no sean miembros de un número de departamento dado:

```
SELECT OBJECT(e) FROM EmpBean e , DeptBean d  
WHERE e NOT MEMBER OF d.emps AND d.deptno = ?1
```

Buscar empleados cuyo gestor es un miembro de un número de departamento dado:

```
SELECT OBJECT(e) FROM EmpBean e, DeptBean d  
WHERE e.dept.mgr MEMBER OF d.emps and d.deptno=?1
```

Predicado EXISTS de consulta de ObjectGrid

El predicado EXISTS comprueba si existe o no una condición especificada por una subselección.

```
EXISTS ( subselect )
```

El resultado de EXISTS es true si la subselección devuelve como mínimo un valor, de lo contrario el resultado es false.

Para negar un predicado EXISTS, debe precederlo con el operador lógico NOT.

Ejemplo

Devuelve los departamentos que tienen como mínimo un empleado que gana más de 1000000:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE EXISTS ( SELECT e FROM IN (d.emps) e WHERE e.salary > 1000000 )
```

Devuelve los departamentos que no tienen empleados:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE NOT EXISTS ( SELECT e FROM IN (d.emps) e)
```

También puede volver a escribir la consulta anterior como en el siguiente ejemplo:

```
SELECT OBJECT(d) FROM DeptBean d WHERE SIZE(d.emps)=0
```

Cláusula ORDER BY de consulta de ObjectGrid

La cláusula ORDER BY especifica una ordenación de los objetos en la colección de resultados. A continuación se muestra un ejemplo:

```
ORDER BY [ order_element ,]* order_element order_element ::= { path-expression } [
ASC | DESC ]
```

La expresión path debe especificar un campo de valor individual que sea de un tipo primitivo de byte, short, int, long, float, double, char, o de un tipo de derivador de Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp y java.util.Calendar. El elemento de orden ASC especifica que los resultados se visualizan en orden ascendente, que es el valor predeterminado. Un elemento de orden DESC especifica que los resultados se visualicen en orden descendente.

Ejemplo

Devuelve objetos de departamento. Muestra los números de departamento en orden descendente:

```
SELECT OBJECT(d) FROM DeptBean d ORDER BY d.deptno DESC
```

Devuelve los objetos de empleado, ordenados por nombre y número de departamento:

```
SELECT OBJECT(e) FROM EmpBean e ORDER BY e.dept.deptno ASC, e.name DESC
```

Funciones de agregación de consulta de ObjectGrid

Las funciones de agregación operan en un conjunto de valores para devolver un solo valor escalar. Puede utilizar estas funciones en los métodos select y subselect. En el siguiente ejemplo se muestra una agregación:

```
SELECT SUM (e.salary) FROM EmpBean e WHERE e.dept.deptno =20
```

Esta agregación calcula el sueldo total del departamento 20.

Las funciones de agregación son: AVG, COUNT, MAX, MIN y SUM. La sintaxis de una función de agregación se muestra en el siguiente ejemplo:

```
aggregation-function ( [ ALL | DISTINCT ] expression )
```

o:

```
COUNT( [ ALL | DISTINCT ] identification-variable )
```

La opción DISTINCT elimina valores duplicados antes de aplicar la función. La opción ALL es la opción predeterminada, y no elimina valores duplicados. Los valores nulos se ignoran durante el cálculo de la función de agregación excepto cuando se utiliza la función COUNT(identification-variable), que devuelven un recuento de todos los elementos del conjunto.

Definición del tipo de retorno

Las funciones MAX y MIN pueden aplicarse a cualquier tipo de datos numérico, serie o fecha-hora y devolver el correspondiente tipo de datos. Las funciones SUM y AVG aceptan un tipo numérico como entrada. La función AVG devuelve un tipo double. La función SUM devuelve un tipo long si el tipo de entrada es un tipo de entero, excepto si la entrada es un tipo BigInteger de Java, en ese caso la función devuelve un tipo BigInteger de Java. La función SUM devuelve un tipo double, si el tipo de entrada no es un tipo de entero, excepto si la entrada de un tipo BigDecimal de Java, en ese caso, la función devuelve un tipo BigDecimal de Java. La función COUNT puede aceptar cualquier tipo de datos excepto colecciones, y devuelve un tipo long.

Cuando se aplican a un conjunto vacío, las funciones SUM, AVG, MAX y MIN pueden devolver un valor nulo. La función COUNT devuelve cero (0) cuando se aplica a un conjunto vacío.

Utilización de cláusulas GROUP BY y HAVING

El conjunto de valores que se utiliza para la función agregada lo determina la colección que resulta de la cláusula FROM y WHERE de la consulta. Puede dividir el conjunto en grupos y aplicar la función de agregación a cada grupo. Para realizar esta acción, utilice una cláusula GROUP BY en la consulta. La cláusula GROUP BY define la agrupación de miembros, que incluyen una lista de las expresiones path. Cada expresión de vía de acceso especifica un campo que es un tipo primitivo de byte, short, int, long, float, double, boolean, char, o un tipo de derivador de Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar o un Java SE 5 enum.

El siguiente ejemplo muestra el uso de la cláusula GROUP BY en una consulta que calcula el sueldo promedio de cada departamento:

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e GROUP BY e.dept.deptno
```

Al dividir un conjunto en grupos, se considera un valor NULL igual a otro valor NULL.

Los grupos se pueden filtrar utilizando una cláusula HAVING que comprueba las propiedades del grupo antes de requerir funciones de agregación o agrupación de miembros. Este filtro es parecido a cómo la cláusula WHERE filtra tuples (es decir, los registros de los valores de colección devueltos) de la cláusula FROM. A continuación se muestra un ejemplo de la cláusula HAVING:

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e
GROUP BY e.dept.deptno
HAVING COUNT(e) > 3 AND e.dept.deptno > 5
```

Esta consulta devuelve el sueldo promedio de los departamentos que tiene más de tres empleados y el número de departamento es mayor que cinco.

Puede utilizar una cláusula HAVING sin una cláusula GROUP BY. En este caso, todo el conjunto se considera un grupo individual, al que se aplica la cláusula HAVING.

BNF (Backus-Naur Form) de consulta de ObjectGrid

A continuación se muestra un resumen de la notación BNF (Backus-Naur Form) de consulta de ObjectGrid.

Tabla 3. Clave para el resumen de BNF

Representación	Descripción
{...}	Agrupación
[...]	Construcciones opcionales
negrita	Palabras clave
*	Cero o más
	Alternativos

```
ObjectGrid QL ::=select_clause from_clause [where_clause]
[group_by_clause] [having_clause] [order_by_clause]
from_clause
::=FROM identification_variable_declaration [,identification_variable_declaration]*
identification_variable_declaration
::=collection_member_declaration | range_variable_declaration
collection_member_declaration
::=IN ( collection_valued_path_expression | single_valued_navigation)
[AS] identifier | [LEFT [OUTER] | INNER] JOIN collection_valued_path_expression
| single_valued_navigation [AS] identifier
range_variable_declaration
::=abstract_schema_name [AS] identifier
single_valued_path_expression
::={single_valued_navigation | identification_variable}. { state_field
| state_field.value_object_attribute } | single_valued_navigation
single_valued_navigation
::=identification_variable.[ single_valued_association_field. ]*
single_valued_association_field
collection_valued_path_expression
::=identification_variable.[ single_valued_association_field. ]* collection_valued_association_field
select_clause
::= SELECT [DISTINCT] [ selection , ]* selection
selection
::= {single_valued_path_expression |identification_variable | OBJECT (
identification_variable) |aggregate_functions } [[AS] id
]
order_by_clause ::= ORDER BY [ {identification_variable.[
single_valued_association_field. ]*state_field} [ASC|DESC],]*
{identification_variable.[ single_valued_association_field. ]*state_field}[ASC|DESC]
where_clause
::= WHERE conditional_expression
conditional_expression
::= conditional_term | conditional_expression OR conditional_term
conditional_term
::= conditional_factor | conditional_term AND conditional_factor
conditional_factor
::= [NOT] conditional_primary
conditional_primary ::=
simple_cond_expression | (conditional_expression)
```

```

simple_cond_expression
 ::= comparison_expression | between_expression | like_expression |
in_expression | null_comparison_expression | empty_collection_comparison_expression
 | exists_expression | collection_member_expression

between_expression
 ::= numeric_expression [NOT] BETWEEN numeric_expression AND numeric_expression
 | string_expression [NOT] BETWEEN string_expression AND string_expression
 | datetime_expression [NOT] BETWEEN datetime_expression AND datetime_expression

in_expression
 ::= identification_variable.[ single_valued_association_field. ]state_field
 [*NOT] IN { (subselect) | ( atom ,)* atom }

atom
 ::= { string_literal | numeric_literal | input_parameter }

like_expression
 ::=string_expression [NOT] LIKE {string_literal | input_parameter}
 [ESCAPE {string_literal | input_parameter}]

null_comparison_expression
 ::= {single_valued_path_expression | input_parameter} IS [ NOT ] NULL

empty_collection_comparison_expression
 ::= collection_valued_path_expression IS [NOT] EMPTY

collection_member_expression
 ::= { single_valued_path_expression | input_parameter } [ NOT ] MEMBER [
OF ]collection_valued_path_expression

exists_expression ::= EXISTS {(subselect)}

subselect
 ::= SELECT [{ ALL | DISTINCT }] subselection
 from_clause [where_clause] [group_by_clause] [having_clause]

subselection
 ::= {single_valued_path_expression |identification_variable | aggregate_functions
}

group_by_clause ::= GROUP BY[single_valued_path_expression,]*
single_valued_path_expression

having_clause ::= HAVING conditional_expression

comparison_expression
 ::= numeric_expression comparison_operator { numeric_expression |
{SOME | ANY | ALL}(subselect) } | string_expression
comparison_operator {
string_expression | {SOME | ANY | ALL}(subselect)
} |
datetime_expression comparison_operator {
datetime_expression
{SOME | ANY | ALL}(subselect) } |
boolean_expression
{=|<>} {
boolean_expression {SOME | ANY | ALL}(subselect)
} |
entity_expression {=|<>} {
entity_expression {SOME| ANY | ALL}(subselect)
}
comparison_operator ::= = | > | >= | < | <= | <>

string_expression
 ::= string_primary | (subselect)

string_primary ::=state_field_path_expression
|string_literal | input_parameter | functions_returning_strings

datetime_expression
 ::= datetime_primary |(subselect)

datetime_primary ::=state_field_path_expression
| string_literal | long_literal | input_parameter | functions_returning_datetime

boolean_expression
 ::= boolean_primary |(subselect)

boolean_primary ::=state_field_path_expression
| boolean_literal | input_parameter

entity_expression ::=single_valued_association_path_expression |
identification_variable | input_parameter

```

```

numeric_expression
 ::= simple_numeric_expression |(subselect)
simple_numeric_expression
 ::= numeric_term | numeric_expression {+|-} numeric_term
numeric_term
 ::= numeric_factor | numeric_term {*/} numeric_factor
numeric_factor
 ::= {+|-} numeric_primary
numeric_primary ::= single_valued_path_expression
 | numeric_literal | ( numeric_expression ) | input_parameter | functions
aggregate_functions
 ::=
AVG([ALL|DISTINCT] identification_variable.[
single_valued_association_field. ]*state_field) |
COUNT([ALL|DISTINCT]
{single_valued_path_expression | identification_variable}) |
MAX([ALL|DISTINCT]
identification_variable.[ single_valued_association_field. ]*state_field) |
MIN([ALL|DISTINCT] identification_variable.[
single_valued_association_field. ]*state_field) |
SUM([ALL|DISTINCT]
identification_variable.[ single_valued_association_field. ]*state_field)
functions
 ::=
ABS (simple_numeric_expression) |
CONCAT (string_primary
, string_primary) |
LOWER (string_primary) |
LENGTH(string_primary)
|
LOCATE(string_primary, string_primary [, simple_numeric_expression])
|
MOD (simple_numeric_expression, simple_numeric_expression)
|
SIZE (collection_valued_path_expression) |
SQRT (simple_numeric_expression)
|
SUBSTRING (string_primary, simple_numeric_expression[,
simple_numeric_expression]) |
UPPER (string_primary)
|
TRIM ([[LEADING | TRAILING | BOTH]
[trim_character] FROM] string_primary)

```

Ajuste del rendimiento de consultas

Para ajustar el rendimiento de las consultas, utilice estas técnicas y sugerencias.

Uso de parámetros

Cuando se ejecuta una consulta, la serie de consultas se debe analizar y debe desarrollarse un plan para ejecutar la consulta, ambas operaciones pueden resultar costosas. WebSphere eXtreme Scale almacena en la memoria caché los planes de consulta por la serie de consulta. Puesto que la memoria caché tiene un tamaño finito, es importante reutilizar las series de consulta siempre que sea posible. El uso de parámetros posicionales o con nombre favorece el rendimiento al fomentar la reutilización de los planes de consulta.

Ejemplo de parámetro posicional | Query q = em.createQuery("select c from Customer c where c.surname=?1"); q.setParameter(1, "Claus");

Uso de índices

El uso de índices adecuados en una correlación puede tener un impacto significativo en el rendimiento de las consultas, a pesar de que los índices puedan implicar una sobrecarga en el rendimiento global de la correlación. Si no se dispone de índices en los atributos de objeto de las consultas, el motor de consultas realiza una exploración de la tabla de cada atributo. La exploración de tabla es la operación más cara durante la ejecución de una consulta. El uso de índices en atributos de objetos de las consultas permite que el motor de consultas no tenga que realizar una exploración innecesaria de la tabla, lo cual mejora el rendimiento global de la consulta. Si se ha diseñado la aplicación para usar las consultas de forma intensiva en una correlación sobre todo de lectura, configure los índices para atributos de objeto involucrados en la consulta. Si la correlación se actualiza continuamente, debe sopesar entre mejorar el rendimiento de la consulta y la sobrecarga de índices en la correlación. Si desea más información, consulte Indexación.

Cuando se almacenan objetos POJO (plain old Java Object) en una correlación, una indexación correcta puede evitar un reflejo Java. En la consulta del ejemplo siguiente, la cláusula WHERE se sustituye por la búsqueda de índices de intervalo, si el campo de presupuesto tiene un índice. De lo contrario, la consulta explora toda la correlación y evalúa la cláusula WHERE de la siguiente manera: primero obtiene el presupuesto mediante un reflejo de Java y después lo compara con el valor 50000:

```
SELECT d FROM DeptBean d WHERE d.budget=50000
```

Consulte el apartado “Plan de consulta” en la página 123 para obtener información sobre cómo ajustar consultas individuales y cómo pueden afectar sintaxis diferentes, modelos de objetos e índices al rendimiento de la consulta.

Uso de la paginación

En entornos cliente/servidor, el motor de consultas transporta toda la correlación de resultados al cliente. Los datos devueltos deben dividirse en partes razonables. Las interfaces EntityManager Query y ObjectMap ObjectQuery admiten los métodos setFirstResult y setMaxResults que permiten que la consulta devuelva un subconjunto de resultados.

Devolución de valores primitivos en lugar de entidades

Con la API de consulta EntityManager, las entidades se devuelven como parámetros de consulta. El motor de consultas devuelve actualmente las claves de estas entidades al cliente. Cuando el cliente itera en estas entidades mediante el uso de Iterator del método getResultIterator, cada entidad se infla automáticamente y se gestiona como si se creara con el método find de la interfaz EntityManager. Todo el gráfico de la entidad se crea a partir del objeto ObjectMap de entidad en el cliente. Los atributos del valor de entidad y las entidades relacionadas se resuelven rápidamente.

Para no tener que crear el tan costoso gráfico, modifique la consulta de modo que devuelva los atributos individuales con navegación de vías de acceso.

Por ejemplo:

```
//
Devuelve una entidad
SELECT p FROM Person p
// Devuelve atributos SELECT p.name, p.address.street, p.address.city, p.gender
FROM Person p
```

Plan de consulta

Todas las consultas de eXtreme Scale tienen un plan de consulta. El plan describe cómo interactúa el motor de consultas con ObjectMaps y con los índices. Consulte el plan de consulta para determinar si los índices o serie de consulta se están utilizando correctamente. El plan de consulta también puede utilizarse para examinar las diferencias que los cambios sutiles en una serie de consulta suponen en la forma en que eXtreme Scale ejecuta una consulta.

El plan de consulta puede verse de dos modos:

- Métodos de la API EntityManager Query o ObjectQuery getPlan
- Rastreo de diagnóstico de ObjectGrid

Método getPlan

El método getPlan de las interfaces ObjectQuery y Query devuelve una serie que describe el plan de consulta. Esta serie puede verse en una salida estándar o en un archivo de anotaciones cronológicas. Nota: en un entorno distribuido, el método getPlan no se ejecuta contra el servidor y no reflejará ningún índice definido. Para ver el plan, utilice un agente para visualizar el plan en el servidor.

Rastreo del plan de consulta

El plan de consulta puede verse mediante el rastreo de ObjectGrid. Para habilitar el rastreo del plan de consulta, utilice la especificación de rastreo siguiente:

```
QueryEnginePlan=debug=enabled
```

Consulte el apartado “Registros y rastreo” en la página 389 para obtener más información sobre cómo habilitar el rastreo y buscar los archivos de anotaciones cronológicas de rastreo.

Ejemplos de plan de consulta

El plan de consulta utiliza la palabra for para indicar que la consulta itera a través de una colección ObjectMap o a través de una colección derivada, como por ejemplo: q2.getEmps(), q2.dept o una colección temporal devuelta por un bucle interno. Si la colección es de ObjectMap, el plan de consulta muestra si se utiliza una exploración secuencial (que se indica mediante INDEX SCAN), un índice exclusivo o un índice no exclusivo. El plan de consulta utiliza una serie de filtro para listar las expresiones de condición que se aplican a una colección.

En una consulta de objeto no se suele utilizar un producto cartesiano. La consulta siguiente explora toda la correlación EmpBean en el bucle externo y explora toda la correlación DeptBean en el bucle interno:

```
SELECT e, d FROM EmpBean e, DeptBean d
```

Rastreo de plan:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  for q3 in DeptBean ObjectMap using INDEX SCAN
    returning new Tuple( q2, q3 )
```

La consulta siguiente recupera todos los nombres de los empleados de un departamento determinado; para ello, explora secuencialmente la correlación EmpBean para obtener un objeto employee. En el objeto employee, la consulta navega a su objeto department y aplica el filtro d.no=1. En este ejemplo, cada empleado tiene una referencia de objeto de departamento, de modo que el bucle interno se ejecuta sólo una vez:

```
SELECT e.name FROM EmpBean e JOIN e.dept d WHERE d.no=1
```

Rastreo de plan:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  for q3 in q2.dept
    filter ( q3.getNo() = 1 )
  returning new Tuple( q2.name )
```

La consulta siguiente equivale a la anterior. No obstante, esta consulta tiene un mejor rendimiento porque primero limita el resultado a un objeto department mediante el uso de un índice exclusivo definido en el número de campo de clave primaria DeptBean. A partir del objeto department, la consulta navega hasta los objetos employee para obtener sus nombres:

```
SELECT e.name FROM DeptBean d JOIN d.emps e WHERE d.no=1
```

Rastreo de plan:

```
for q2 in DeptBean ObjectMap using UNIQUE INDEX key=(1)
  for q3 in q2.getEmps()
  returning new Tuple( q3.name )
```

La consulta siguiente busca todos los empleados que trabajan en desarrollo o ventas. La consulta explora toda la correlación EmpBean y realiza un filtrado adicional al evaluar las expresiones: d.name = 'Sales' o d.name='Dev'

```
SELECT e FROM EmpBean e, in (e.dept) d WHERE d.name = 'Sales' or d.name='Dev'
```

Rastreo de plan:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  for q3 in q2.dept
    filter (( q3.getName() = Sales ) OR ( q3.getName() = Dev ) )
  returning new Tuple( q2 )
```

La consulta siguiente equivale a la anterior, pero esta consulta ejecuta un plan de consulta diferente y utiliza un índice de intervalo en el nombre de campo. En general, esta consulta tiene un mejor rendimiento porque el índice del campo de nombre se utiliza para limitar los objetos department, que se ejecuta rápidamente si sólo unos pocos departamentos son de desarrollo o ventas.

```
SELECT e FROM DeptBean d, in(d.emps) e WHERE d.name='Dev' or d.name='Sales'
```

Rastreo de plan:

IteratorUnionIndex of

```
for q2 in DeptBean ObjectMap using INDEX on name = (Dev)
  for q3 in q2.getEmps()
```

```
for q2 in DeptBean ObjectMap using INDEX on name = (Sales)
  for q3 in q2.getEmps()
```

La consulta siguiente busca departamentos que no tienen ningún empleado:

```
SELECT d FROM DeptBean d WHERE NOT EXISTS(select e from d.emps e)
```

Rastreo de plan:

```
for q2 in DeptBean ObjectMap using INDEX SCAN
  filter ( NOT EXISTS (   correlated collection defined as
      for q3 in q2.getEmps()
      returning new Tuple( q3
    )
  )
  returning new Tuple( q2 )
```

La consulta siguiente equivale a la anterior, pero utiliza la función escalar SIZE. Esta consulta tiene un rendimiento similar pero es más fácil de escribir.

```
SELECT d FROM DeptBean d WHERE SIZE(d.emps)=0
for q2 in DeptBean ObjectMap using INDEX SCAN
  filter (SIZE( q2.getEmps()) = 0 )
  returning new Tuple( q2 )
```

El ejemplo siguiente es otra manera de escribir la misma consulta que la anterior con un rendimiento similar, pero esta consulta es más fácil de escribir también:

```
SELECT d FROM DeptBean d WHERE d.emps is EMPTY
```

Rastreo de plan:

```
for q2 in DeptBean ObjectMap using INDEX SCAN
  filter ( q2.getEmps() IS EMPTY )
  returning new Tuple( q2 )
```

La consulta siguiente busca empleados con un domicilio que coincida al menos con una de las direcciones del empleado cuyo nombre sea igual al valor del parámetro. El bucle interno no tiene dependencia del bucle externo. La consulta ejecuta una vez el bucle interno.

```
SELECT e FROM EmpBean e WHERE e.home = any (SELECT e1.home FROM EmpBean e1
WHERE e1.name=?1)
for q2 in EmpBean ObjectMap using INDEX SCAN
  filter ( q2.home =ANY   temp collection defined as
      for q3 in EmpBean ObjectMap using INDEX on name = ( ?1)
      returning new Tuple( q3.home
    )
  )
  returning new Tuple( q2 )
```

La consulta siguiente es igual a la anterior, pero tiene una subconsulta correlacionada; además, el bucle interno se ejecuta repetidamente.

```
SELECT e FROM EmpBean e WHERE EXISTS(SELECT e1 FROM EmpBean e1 WHERE
e.home=e1.home and e1.name=?1)
```

Rastreo de plan:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
  filter ( EXISTS (   correlated collection defined as
      for q3 in EmpBean ObjectMap using INDEX on name = (?1)
      filter ( q2.home = q3.home )
      returning new Tuple( q3
    )
  )
  returning new Tuple( q2 )
```

Optimización de consultas mediante el uso de índices

La definición y el uso correctos de índices puede mejorar significativamente el rendimiento de las consultas.

Las consultas de WebSphere eXtreme Scale pueden utilizar los plug-ins HashIndex incorporados para mejorar el rendimiento de las consultas. Los índices se pueden definir en atributos de entidad o de objeto. El motor de consultas utilizará automáticamente los índices definidos si su cláusula WHERE utiliza una de las series siguientes:

- Una expresión de comparación con estos operadores: =, <, >, <= o >= (cualquier expresión de comparación excepto el símbolo distinto <>).
- Una expresión con BETWEEN.
- Los operandos de las expresiones son constantes o términos simples.

Requisitos

Los índices tienen los siguientes requisitos cuando son utilizados por Query:

- Todos los índices deben utilizar el plug-in HashIndex incorporado.
- Todos los índices deben estar definidos estáticamente. Los índices dinámicos no están soportados.
- La anotación @Index se puede utilizar para crear automáticamente plug-ins HashIndex estáticos.
- Todos los índices de atributo único deben tener la propiedad RangeIndex establecida en true.
- Todos los índices compuestos deben tener la propiedad RangeIndex establecida en false.
- Todos los índices de asociación (relación) deben tener la propiedad RangeIndex establecida en false.

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

Para obtener información sobre la creación de índices, consulte el apartado Índices.

Para obtener información sobre un modo eficaz de buscar objetos almacenados en memoria caché, consulte el apartado “Índice compuesto HashIndex” en la página 248.

Uso de sugerencias para elegir un índice

Se puede seleccionar manualmente un índice utilizando el método setHint en las interfaces Query y ObjectQuery con la constante HINT_USEINDEX. Esto puede ser útil al optimizar una consulta para utilizar el índice con mejor rendimiento.

Ejemplos de consulta que utilizan los índices de atributo

Los ejemplos siguientes utilizan términos simples: e.empid, e.name, e.salary, d.name, d.budget y e.isManager. En el ejemplo se da por supuesto que los índices se han definido en los campos de nombre, salario y presupuesto de un objeto de entidad o valor. El campo empid es una clave primaria e isManager no tiene ningún índice definido.

La consulta siguiente utiliza los índices en los campos de nombre y salario. Devuelve todos los empleados con nombres que son iguales al valor del primer parámetro o un salario que es igual al valor del segundo parámetro:

```
SELECT e FROM EmpBean e where e.name=?1 or e.salary=?2
```

La siguiente consulta utiliza ambos índices en los campos de nombre y presupuesto. La consulta devuelve todos los departamentos con nombre 'DEV' que tienen un presupuesto que es mayor que 2000.

```
SELECT d FROM DeptBean d where d.name='DEV' and d.budget>2000
```

La consulta siguiente devuelve todos los empleados con un salario mayor que 3000 y con un valor de distintivo isManager que es igual al valor del parámetro. La consulta utiliza el índice que se ha definido en el campo de salario y realiza un filtrado adicional al evaluar la expresión de comparación: e.isManager=?1.

```
SELECT e FROM EmpBean e where e.salary>3000 and e.isManager=?1
```

La consulta siguiente busca todos los empleados que ganan más que el primer parámetro o los empleados que son jefes. Aunque el campo de salario tiene un índice definido, la consulta explora el índice incorporado que se ha creado en las claves primarias del campo EmpBean y evalúa la expresión: e.salary>?1 o e.isManager=TRUE.

```
SELECT e FROM EmpBean e WHERE e.salary>?1 or e.isManager=TRUE
```

La consulta siguiente devuelve los empleados con un nombre que contiene la letra a. Aunque el campo de nombre tiene un índice definido, la consulta no utiliza el índice porque el campo de nombre se utiliza en la expresión LIKE.

```
SELECT e FROM EmpBean e WHERE e.name LIKE '%a%'
```

La consulta siguiente busca todos los empleados con un nombre que no sea "Smith". Aunque el campo de nombre tiene un índice definido, la consulta no utiliza el índice porque la consulta utiliza el operador de comparación distinto (<>).

```
SELECT e FROM EmpBean e where e.name<>'Smith'
```

La consulta siguiente busca todos los departamentos con un presupuesto inferior al valor del parámetro, y con un salario de empleados superior a 3000. La consulta utiliza un índice para el salario, pero no utiliza un índice para el presupuesto porque dept.budget no es un término simple. Los objetos dept se derivan de la colección e. No es necesario utilizar el índice de presupuesto para buscar los objetos dept.

```
SELECT dept from EmpBean e, in (e.dept) dept where e.salary>3000 and dept.budget<?
```

La consulta siguiente busca todos los empleados con un salario superior al salario de los empleados que tienen empid de 1, 2 y 3. No se utiliza el salario de índice porque la comparación tiene una subconsulta. empid es una clave primaria, y se utiliza para una búsqueda de índice único porque todas las claves primarias tienen un índice incorporado definido.

```
SELECT e FROM EmpBean e WHERE e.salary > ALL (SELECT e1.salary FROM EmpBean e1 WHERE e1.empid=1 or e1.empid =2 or e1.empid=99)
```

Para comprobar si la consulta utiliza el índice, puede consultar el apartado "Plan de consulta" en la página 123. A continuación se muestra un plan de consulta de ejemplo de la consulta anterior:

```

for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( q2.salary >ALL temp collection defined as
            IteratorUnionIndex of

                for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(1)
                )

                for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(2)
                )

                for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(99)
                )
    returning new Tuple( q3.salary )
returning new Tuple( q2 )

for q2 in EmpBean ObjectMap using RANGE INDEX on salary with range(3000,)
    for q3 in q2.dept
        filter ( q3.budget < ?1 )
    returning new Tuple( q3 )

```

Atributos de indexación

Los índices se pueden definir con un único tipo de atributo con las restricciones definidas previamente.

Definición de índices de entidad utilizando @Index

Para definir un índice en una entidad, simplemente defina una anotación:

Entidades que utilizan anotaciones

```

@Entity
public class Employee {
    @Id int empid;
    @Index String name
    @Index double salary
    @ManyToOne Department dept;
}
@Entity
public class Department {
    @Id int deptid;
    @Index String name;
    @Index double budget;
    boolean isManager;
    @OneToMany Collection<Employee> employees;
}

```

Con XML

Los índices también se pueden definir utilizando XML:

Entidades sin anotaciones

```

public class Employee {
    int empid;
    String name
    double salary
    Department dept;
}

public class Department {
    int deptid;
    String name;
}

```

```

double budget;
boolean isManager;
Collection employees;
}

```

XML de ObjectGrid con índices de atributo

```

<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="DepartmentGrid" entityMetadataXMLFile="entity.xml">
<backingMap name="Employee" pluginCollectionRef="Emp"/>
<backingMap name="Department" pluginCollectionRef="Dept"/>
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="Emp">
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Employee.name"/>
<property name="AttributeName" type="java.lang.String" value="name"/>
<property name="RangeIndex" type="boolean" value="true"
description="Se deben establecer los rangos en true para los atributos." />
</bean>
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Employee.salary"/>
<property name="AttributeName" type="java.lang.String" value="salary"/>
<property name="RangeIndex" type="boolean" value="true"
description="Se deben establecer los rangos en true para los atributos." />
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="Dept">
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Department.name"/>
<property name="AttributeName" type="java.lang.String" value="name"/>
<property name="RangeIndex" type="boolean" value="true"
description="Se deben establecer los rangos en true para los atributos." />
</bean>
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Department.budget"/>
<property name="AttributeName" type="java.lang.String" value="budget"/>
<property name="RangeIndex" type="boolean" value="true"
description="Se deben establecer los rangos en true para los atributos." />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

XML de entidad

```

<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ../emd.xsd">
<description>Department entities</description>
<entity class-name="acme.Employee" name="Employee" access="FIELD">
<attributes>
<id name="empid" />
<basic name="name" />
<basic name="salary" />
<many-to-one name="department"
target-entity="acme.Department"
fetch="EAGER">
<cascade><cascade-persist/></cascade>
</many-to-one>
</attributes>
</entity>
<entity class-name="acme.Department" name="Department" access="FIELD">
<attributes>
<id name="deptid" />
<basic name="name" />
<basic name="budget" />
<basic name="isManager" />
<one-to-many name="employees"
target-entity="acme.Employee"
fetch="LAZY" mapped-by="parentNode">
<cascade><cascade-persist/></cascade>
</one-to-many>
</attributes>
</entity>
</entity-mappings>

```

Definición de índices para no entidades utilizando XML

Los índices para tipos que no son entidad están definidos en XML. No hay ninguna diferencia al crear MapIndexPlugin para correlaciones con entidad y correlaciones sin entidad.

Bean de Java

```
public class Employee {
    int empid;
    String name;
    double salary;
    Department dept;

    public class Department {
        int deptid;
        String name;
        double budget;
        boolean isManager;
        Collection employees;
    }
}
```

XML de ObjectGrid con índices de atributo

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="DepartmentGrid">
      <backingMap name="Employee" pluginCollectionRef="Emp"/>
      <backingMap name="Department" pluginCollectionRef="Dept"/>
      <querySchema>
        <mapSchemas>
          <mapSchema mapName="Employee" valueClass="acme.Employee"
            primaryKeyField="empid" />
          <mapSchema mapName="Department" valueClass="acme.Department"
            primaryKeyField="deptid" />
        </mapSchemas>
        <relationships>
          <relationship source="acme.Employee"
            target="acme.Department"
            relationField="dept" invRelationField="employees" />
        </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="Emp">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Employee.name"/>
        <property name="AttributeName" type="java.lang.String" value="name"/>
        <property name="RangeIndex" type="boolean" value="true"
          description="Se deben establecer los rangos en true para los atributos." />
      </bean>
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Employee.salary"/>
        <property name="AttributeName" type="java.lang.String" value="salary"/>
        <property name="RangeIndex" type="boolean" value="true"
          description="Se deben establecer los rangos en true para los atributos." />
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="Dept">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Department.name"/>
        <property name="AttributeName" type="java.lang.String" value="name"/>
        <property name="RangeIndex" type="boolean" value="true"
          description="Se deben establecer los rangos en true para los atributos." />
      </bean>
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="Department.budget"/>
        <property name="AttributeName" type="java.lang.String" value="budget"/>
        <property name="RangeIndex" type="boolean" value="true"
          description="Se deben establecer los rangos en true para los atributos." />
      </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

Relaciones de índices

WebSphere eXtreme Scale almacena las claves foráneas para las entidades relacionadas dentro del objeto padre. En el caso de entidades, las claves se

almacenan en el tuple subyacente. En el caso de objetos que no son de entidad, las claves se almacenan explícitamente en el objeto padre.

Si se añade un índice a un atributo de relación, se pueden acelerar las consultas que utilizan referencias cíclicas o los filtros de consulta IS NULL, IS EMPTY, SIZE y MEMBER OF. Las asociaciones de un valor o de varios valores pueden tener la anotación @Index o una configuración de plug-in HashIndex en un archivo XML de descriptor ObjectGrid.

Definición de los índices de relación de entidad utilizando @Index

El ejemplo siguiente define las entidades con las anotaciones @Index:

Entidad con anotación

```
@Entity
public class Node {
    @ManyToOne @Index
    Node parentNode;

    @OneToMany @Index
    List<Node> childrenNodes = new ArrayList();

    @OneToMany @Index
    List<BusinessUnitType> businessUnitTypes = new ArrayList();
}
```

Definición de índices de relación utilizando XML

El ejemplo siguiente define las mismas entidades e índices utilizando XML con los plug-ins HashIndex:

Entidad sin anotaciones

```
public class Node {
    int nodeId;
    Node parentNode;
    List<Node> childrenNodes = new ArrayList();
    List<BusinessUnitType> businessUnitTypes = new ArrayList();
}
```

XML de ObjectGrid

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
  <objectGrid name="ObjectGrid_Entity" entityMetadataXMLFile="entity.xml">
  <backingMap name="Node" pluginCollectionRef="Node"/>
  <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
  </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
  <backingMapPluginCollection id="Node">
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="parentNode"/>
  <property name="AttributeName" type="java.lang.String" value="parentNode"/>
  <property name="RangeIndex" type="boolean" value="false"
  description="Los rangos no están soportados para los índices de asociación." /> </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="businessUnitType"/>
  <property name="AttributeName" type="java.lang.String" value="businessUnitTypes"/>
  <property name="RangeIndex" type="boolean" value="false"
  description="Los rangos no están soportados para los índices de asociación." />
  </bean>
  </backingMapPluginCollection>
```

```

</backingMapPluginCollections>
</objectGridConfig>

```

XML de entidad

```

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

<description>My entities</description>
<entity class-name="acme.Node" name="Account" access="FIELD">
<attributes>
<id name="nodeId" />
<one-to-many name="childrenNodes"
target-entity="acme.Node"
fetch="EAGER" mapped-by="parentNode">
<cascade><cascade-all/></cascade>
</one-to-many>
<many-to-one name="parentNodes"
target-entity="acme.Node"
fetch="LAZY" mapped-by="childrenNodes">
<cascade><cascade-none/></cascade>
</one-to-many>
<many-to-one name="businessUnitTypes"
target-entity="acme.BusinessUnitType"
fetch="EAGER">
<cascade><cascade-persist/></cascade>
</many-to-one>
</attributes>
</entity>
<entity class-name="acme.BusinessUnitType" name="BusinessUnitType" access="FIELD">
<attributes>
<id name="buId" />
<basic name="TypeDescription" />
</attributes>
</entity>
</entity-mappings>

```

A través de los índices definidos previamente, se optimizan los siguientes ejemplos de consulta de entidad:

```

SELECT n FROM Node n WHERE n.parentNode is null
SELECT n FROM Node n WHERE n.businessUnitTypes is EMPTY
SELECT n FROM Node n WHERE size(n.businessUnitTypes)>=10
SELECT n FROM BusinessUnitType b, Node n WHERE b member of n.businessUnitTypes
and b.name='TELECOM'

```

Definición de índices de relación sin entidad

En el ejemplo siguiente se define un plug-in HashIndex para correlaciones que no son de entidad en un archivo XML de descriptor ObjectGrid:

```

<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="ObjectGrid_POJO">
<backingMap name="Node" pluginCollectionRef="Node"/>
<backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
<querySchema>
<mapSchemas>
<mapSchema mapName="Node" valueClass="com.ibm.websphere.objectgrid.samples.entity.Node"
primaryKeyField="id" />
<mapSchema mapName="BusinessUnitType"
valueClass="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
primaryKeyField="id" />
</mapSchemas>
<relationships>
<relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
target="com.ibm.websphere.objectgrid.samples.entity.Node"
relationField="parentId" invRelationField="childrenNodeIds" />
<relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
target="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
relationField="businessUnitTypeKeys" invRelationField="" />
</relationships>
</querySchema>
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="Node">
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="parentNode"/>
<property name="Name" type="java.lang.String" value="parentId"/>
<property name="AttributeName" type="java.lang.String" value="parentId"/>
<property name="RangeIndex" type="boolean" value="false"

```

```

description="Los rangos no están soportados para los índices de asociación." />
</bean>
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="businessUnitType"/>
  <property name="AttributeName" type="java.lang.String" value="businessUnitTypeKeys"/>
</bean>
<property name="RangeIndex" type="boolean" value="false"
description="Los rangos no están soportados para los índices de asociación." />
</bean>
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="childrenNodeIds"/>
<property name="AttributeName" type="java.lang.String" value="childrenNodeIds"/>
<property name="RangeIndex" type="boolean" value="false"
description="Los rangos no están soportados para los índices de asociación." />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

Dadas las configuraciones de índice anteriores, se optimizan los ejemplos de consulta de objetos siguientes:

```

SELECT n FROM Node n WHERE n.parentNodeId is null
SELECT n FROM Node n WHERE n.businessUnitTypeKeys is EMPTY
SELECT n FROM Node n WHERE size(n.businessUnitTypeKeys)>=10
SELECT n FROM BusinessUnitType b, Node n WHERE
  b member of n.businessUnitTypeKeys and b.name='TELECOM'

```

Utilización de objetos que no sean claves para encontrar las particiones (interfaz PartitionableKey)

Cuando una configuración de eXtreme Scale utiliza la estrategia de ubicación 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.

Programación de transacciones

Aplicaciones que requieren que las transacciones introduzcan tales consideraciones como gestión de bloqueos, gestión de colisiones y aislamiento de transacciones.

Visión general del proceso de transacciones

Sesiones y proceso de transacciones

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

Para interactuar con los datos, la hebra de la aplicación necesita su propio objeto Session. Si la aplicación desea utilizar el ObjectGrid en una hebra, llame a uno de los métodos ObjectGrid.getSession para obtener una hebra. Con la sesión, la aplicación puede trabajar con los datos almacenados en las correlaciones de ObjectGrid.

Cuando una aplicación utiliza un objeto Session, la sesión debe estar en el contexto de una transacción. Una transacción empieza o se confirma y retrotrae mediante los métodos begin, commit y rollback en el objeto Session. Las aplicaciones también pueden funcionar en la modalidad de confirmación automática, en la que Session empieza automáticamente y confirma una transacción, siempre que se realiza una operación en la correlación. Una modalidad de confirmación automática no puede agrupar varias operaciones en una única transacción, de forma que es la opción más lenta si crea un proceso por lotes de varias operaciones en una única transacción. Sin embargo, para las transacciones que sólo contienen una operación, la confirmación automática es la opción más rápida.

Transacciones

Las transacciones tienen muchas ventajas para el almacenamiento de datos y la manipulación. Puede utilizar las transacciones para proteger la cuadrícula de los cambios simultáneos, para aplicar varios cambios como una unidad simultánea, para replicar datos y para implementar un ciclo de vida para los bloqueos en los cambios.

Cuando se inicia una transacción, WebSphere eXtreme Scale asigna una correlación de diferencias especial para mantener los cambios o copias actuales de pares de clave y valor que la transacción utiliza. Normalmente, cuando se accede a un par de clave y valor, el valor se copia antes de que la aplicación reciba el valor. La correlación de diferencias rastrea todos los cambios para las operaciones como, por ejemplo, insert, update, get, remove, etc. Las claves no se copian porque se da por supuesto que son inmutables. Si se especifica un objeto ObjectTransformer, este objeto se utiliza para copiar el valor. Si la transacción utiliza el bloqueo optimista, también se realiza un seguimiento de las imágenes anteriores de los valores para su comparación cuando se confirma la transacción.

Si se retrotrae una transacción, se descarta la información de correlación de diferencias y se liberan los bloqueos de las entradas. Cuando se confirma una transacción, los cambios se aplican a las correlaciones y se liberan los bloqueos. Si se utiliza el bloqueo optimista, eXtreme Scale compara las versiones de imágenes anteriores de los valores con los valores incluidos en la correlación. Estos valores deben coincidir para que la transacción se confirme. Esta comparación permite un esquema de bloqueo de varias versiones, pero a costa de que se realicen dos copias cuando la transacción accede a la entrada. Se vuelven a copiar todos los valores y

se almacena la nueva copia en la correlación. WebSphere eXtreme Scale realiza esta copia para evitar que la aplicación cambie la referencia de la aplicación por el valor después de una confirmación.

Puede evitar utilizar varias copias de la información. La aplicación puede guardar una copia utilizando el bloqueo pesimista en lugar del bloqueo optimista como coste de limitar la concurrencia. También se puede evitar la copia del valor durante la confirmación si la aplicación acepta no cambiar un valor después de la confirmación.

Ventajas de las transacciones

Utilice transacciones por las siguientes razones:

Mediante el uso de transacciones, puede:

- Retrotraer cambios si se produce una excepción o si la lógica empresarial necesita deshacer los cambios de estado.
- Para aplicar varios cambios como una unidad atómica durante la confirmación.
- Mantener y liberar bloqueos en los datos para aplicar varios cambios como una unidad atómica durante la confirmación.
- Proteger una hebra de los cambios simultáneos.
- Implementar un ciclo de vida para los bloqueos en cambios.
- Producir una unidad atómica de duplicación.

Tamaño de transacción

Las transacciones de mayor tamaño son más eficaces, especialmente para la réplica. Sin embargo, las transacciones de mayor tamaño pueden afectar de forma adversa a la concurrencia porque se mantienen durante más tiempo los bloqueos sobre entradas. Si utiliza transacciones de mayor tamaño, puede aumentar el rendimiento de la réplica. El aumento de este rendimiento es importante cuando se precarga una correlación. Pruebe con distintos tamaños de lotes para determinar lo que funciona mejor en cada caso.

Las transacciones de mayor tamaño también son útiles con los cargadores. Si se está utilizando un cargador que puede realizar el proceso por lotes de SQL, son posibles aumentos significativos de rendimiento en función de la transacción y las reducciones significativas de la carga en el lado de la base de datos. Esta ganancia en el rendimiento dependerá de la implementación del cargador.

Modalidad de confirmación automática

Si no se ha iniciado de forma activa ninguna transacción, cuando una aplicación interactúa con un objeto ObjectMap, empieza una operación automática de inicio y confirmación en nombre de la aplicación. Esta operación automática de inicio y confirmación funciona, pero impide que la retrotracción y el bloqueo funcionen de forma eficaz. La velocidad de réplica síncrona se ve afectado debido al tamaño de transacción muy pequeño. Si utiliza una aplicación de gestor de entidades, no utilice la modalidad de confirmación automática porque los objetos que busca el método EntityManager.find se convierten inmediatamente en no gestionados en la devolución del método y dejan de poderse utilizar.

Coordinadores de transacciones externos

Normalmente, las transacciones se inician con el método `session.begin` y finalizan con el método `session.commit`. Sin embargo, cuando se incorpora eXtreme Scale, las transacciones podrían iniciarse y terminarse a través de un coordinador de transacciones externo. Si utiliza un coordinador de transacciones externas, no tendrá que llamar al método `session.begin` y finalizar el método `session.commit`. Consulte *Guía de programación* si desea más información sobre eXtreme Scale y la interacción de transacciones externas. Si utiliza WebSphere Application Server, puede utilizar el plug-in `WebSphereTransactionCallback`. Consulte *Guía de programación* si desea más información sobre los plug-ins que están disponibles con WebSphere eXtreme Scale.

Atributo `CopyMode`

Puede ajustar el número de copias definiendo el atributo `CopyMode` de los objetos `BackingMap` u `ObjectMap`.

Puede ajustar el número de copias definiendo el atributo `CopyMode` de los objetos `BackingMap` u `ObjectMap`. La modalidad de copia tiene los siguientes valores:

- `COPY_ON_READ_AND_COMMIT`
- `COPY_ON_READ`
- `NO_COPY`
- `COPY_ON_WRITE`
- `COPY_TO_BYTES`

El valor `COPY_ON_READ_AND_COMMIT` es el valor predeterminado. El valor `COPY_ON_READ` copia los datos iniciales recuperados, pero no copia durante la confirmación. Esta modalidad es segura si la aplicación no modifica un valor después de confirmar una transacción. El valor `NO_COPY` no copia datos, que sólo es seguro para los datos de sólo lectura. Si los datos nunca cambian, no tendrá que copiarlos por razones de aislamiento.

Tenga cuidado cuando utilice el valor del atributo `NO_COPY` con las correlaciones que se pueden actualizar. WebSphere eXtreme Scale utiliza la copia en el primer toque para permitir la retrotracción de la transacción. La aplicación sólo ha cambiado la copia y, como resultado, eXtreme Scale descarta la copia. Si se utiliza el valor de atributo `NO_COPY`, y la aplicación modifica el valor confirmado, no es posible completar una retrotracción. Si se modifica el valor confirmado comportará problemas con índices, réplica, etc, porque los índices y las réplicas se actualizan cuando se confirma la transacción. Si modifica los datos confirmados y, a continuación, retrotrae la transacción, que en realidad no se retrotrae, los índices no se actualizan y la réplica no tiene lugar. Otras hebras pueden ver los cambios no confirmados inmediatamente, incluso si tienen bloqueos. Utilice el valor de atributo `NO_COPY` para las correlaciones de sólo lectura o para aplicaciones que completan la copia apropiada antes de modificar el valor. Si utiliza el valor de atributo `NO_COPY` y llama al soporte de IBM con un problema de integridad de datos, se le solicitará que reproduzca el problema con la modalidad de copia establecida en `COPY_ON_READ_AND_COMMIT`.

El valor `COPY_TO_BYTES` almacena valores en la correlación de un formato serializado. En el momento de lectura, eXtreme Scale infla el valor a partir de un formato serializado y en el momento de confirmación almacena el valor en un formato serializado. Con este método, se produce una copia durante la lectura y la confirmación.

La modalidad de copia predeterminada para una correlación se puede configurar en el objeto `BackingMap`. También puede cambiar la modalidad de copia en las correlaciones antes de iniciar una transacción mediante el uso del método `ObjectMap.setCopyMode`.

A continuación, aparece un ejemplo de un fragmento de código de la correlación de respaldo de un archivo `objectgrid.xml` que muestra cómo establecer la modalidad de copia para una correlación de respaldo dada. Este ejemplo da por supuesto que utiliza `cc` como espacio de nombres de `objectgrid/config`.

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

Consulte la información sobre los procedimientos recomendados de `copyMode` en *Guía de programación* si desea más información.

Bloqueo de entrada de correlación

Una `BackingMap` de `ObjectGrid` admite diversas estrategias de bloqueo para que las correlaciones mantengan la coherencia de las entradas en memoria caché.

Cada `BackingMap` puede configurarse de modo que utilice una de las estrategias de bloqueo siguientes:

1. Modalidad de bloqueo optimista
2. Modalidad de bloqueo pesimista
3. Ninguno

La estrategia de bloqueo predeterminada es `OPTIMISTIC`. Utilice el bloqueo optimista cuando los datos no se modifican frecuentemente. Los bloqueos sólo se mantienen durante un tiempo breve mientras los datos se leen de la memoria caché y se copian en la transacción. Cuando la memoria caché de la transacción se sincroniza con la memoria caché principal, los objetos de la memoria caché actualizados se comprueban contra la versión original. Si la comprobación falla, la transacción se retrotrae y se produce la excepción `OptimisticCollisionException`.

La estrategia de bloqueo `PESSIMISTIC` adquiere bloqueos para las entradas de memoria caché y debe utilizarse cuando los datos se cambian con frecuencia. Cada vez que se lee una entrada de la memoria caché, se adquiere un bloqueo, que puede mantenerse condicionalmente hasta que se complete la transacción. La duración de algunos de los bloqueos pueden ajustarse mediante el uso de niveles de aislamiento para la sesión.

Si el bloqueo no es necesario porque los datos nunca se actualizan o sólo se actualizan durante períodos tranquilos, puede inhabilitar el bloqueo mediante el uso de la estrategia de bloqueo `NONE`. Esta estrategia es muy rápida porque no se necesita ningún gestor de bloqueos. La estrategia de bloqueo `NONE` es ideal en tablas de búsqueda o en correlaciones de sólo lectura.

Si desea más información sobre las estrategias de bloqueo, consulte la información sobre las estrategias de bloqueo en *Visión general del producto*.

Especificación de una estrategia de bloqueo

El siguiente ejemplo demuestra cómo se puede establecer la estrategia de bloqueo en las correlaciones `BackingMaps` `map1`, `map2` y `map3`, donde cada correlación utiliza una estrategia de bloqueo diferente. El primer fragmento de código muestra cómo utilizar el XML para la configuración de la estrategia de bloqueo y el segundo fragmento de código muestra un enfoque mediante programa.

Enfoque de XML

Configuración de BackingMap - XML de ejemplo

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

  <objectGrids>
    <objectGrid name="test">
      <backingMap name="map1"
        lockStrategy="PESSIMISTIC" numberOfLockBuckets="31"/>
      <backingMap name="map2"
        lockStrategy="OPTIMISTIC" numberOfLockBuckets="409"/>
      <backingMap name="map3"
        lockStrategy="NONE"/>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Enfoque mediante programa

Configuración de BackingMap - ejemplo mediante programa

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
  ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setNumberOfLockBuckets(31);
bm = og.defineMap("map2");
bm.setNumberOfLockBuckets(409);
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

Para evitar una excepción de `java.lang.IllegalStateException`, se debe llamar al método `setLockStrategy` antes de utilizar los métodos `initialize` o `getSession` en una instancia local de `ObjectGrid`.

Para obtener más información, consulte el tema sobre las estrategias de bloqueo en la *Visión general del producto*.

Configuración del gestor de bloqueos

Cuando se utiliza una estrategia de bloqueo `PESSIMISTIC` u `OPTIMISTIC`, se crea un gestor de bloqueos para `BackingMap`. El gestor de bloqueos utiliza una correlación hash para realizar un seguimiento de las entradas bloqueadas por una o más transacciones. Cuantas más entradas de correlación existan en la correlación hash, mayor será el grupo de bloqueos con un buen rendimiento. El riesgo de las colisiones de sincronización de Java es menor a medida que crece el número de grupos. Un número mayor de grupos también implica mayor simultaneidad. Los ejemplos anteriores muestran cómo una aplicación puede establecer el número de grupos de bloqueos que se deben utilizar en una instancia determinada de `BackingMap`.

Para evitar una excepción `java.lang.IllegalStateException`, debe llamarse al método `setNumberOfLockBuckets` antes que a los métodos `initialize` o `getSession` en la instancia `ObjectGrid`. El parámetro del método `setNumberOfLockBuckets` es un entero primitivo de Java que especifica el número de grupos de bloqueo para utilizar. El uso de un número primo puede permitir una distribución uniforme de

entradas de correlación en los grupos de bloqueos. Un buen punto de partida para obtener un mejor rendimiento es establecer el número de grupos de bloqueos en un 10 por ciento del número esperado de entradas de BackingMap.

LockDeadlockException

A continuación se ofrece un ejemplo de código que muestra la obtención de la excepción y luego se muestra el mensaje resultante.

```
try {  
    ...  
} catch (ObjectGridException oe) {  
    System.out.println(oe);  
}
```

El resultado es el siguiente:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: _Message
```

Este mensaje representa la serie que se pasa como parámetro cuando se crea y se emite la excepción.

Causa de la excepción

El tipo más común de excepción de punto muerto se produce cuando se utiliza la estrategia de bloqueo pesimista y dos clientes distintos tienen un bloqueo compartido sobre un determinado objeto. Entonces, ambos clientes intentan ascender a un bloqueo exclusivo sobre dicho objeto. El diagrama siguiente ilustra esta situación, incluidos los bloques de transacción que causan la emisión de la excepción.

El siguiente fragmento de código Java demuestra cómo pasar un archivo de configuración de XML para crear un ObjectGrid.

Se trata de una visión abstracta de lo que sucede en su programa cuando se produce la excepción. En una aplicación con muchas hebras que actualizan la misma ObjectMap, es posible que se produzca esta situación. A continuación se ofrece un ejemplo de dos clientes que ejecutan los bloques de código de transacción, tal como se ilustra en la figura anterior.

Soluciones posibles

Podría encontrarse con la situación ilustrada en la Figura 1 cuando haya numerosas hebras que inicien transacciones en una determinada correlación. En este caso, la excepción se emite para impedir que su programa se cuelgue. Puede enviarse una notificación y añadir código al bloque catch para obtener más detalles sobre la causa. Puesto que sólo verá esta excepción en una estrategia de bloqueo pesimista, una solución sencilla consiste simplemente en utilizar una estrategia de bloqueo optimista. Si, no obstante, requiere una estrategia de bloqueo pesimista, puede utilizar el método `getForUpdate` en lugar del método `get`. De este modo se elimina la recepción de las excepciones para la situación descrita anteriormente.

Estrategias de bloqueo

Las estrategias de bloqueo pueden ser de tipo pesimista, optimista o ninguno. Para elegir la estrategia de bloqueo, debe tener en cuenta cuestiones como el porcentaje de cada tipo de operaciones que realizará, si utilizará un cargador o no, etc.

Los bloqueos son enlazados por transacciones. Puede especificar los siguientes valores de bloqueo:

- **Sin bloqueo:** la ejecución sin el valor de bloqueo es la más rápida. Si utiliza datos de sólo lectura, es posible que no necesite el bloqueo.
- **Bloqueo pesimista:** adquiere bloqueos sobre entradas y luego mantiene los bloqueos hasta que se realiza la confirmación. Esta estrategia de bloqueo proporciona una mayor coherencia a costa del rendimiento.
- **Bloqueo optimista:** toma una imagen anterior de cada registro que toca la transacción y compara la imagen con los valores de entrada actuales cuando se confirma la transacción. Si los valores de entrada cambian, la transacción se retrotrae. No se mantiene ningún bloqueo hasta el momento de la confirmación. Esta estrategia de bloqueo proporciona una mejor concurrencia que la estrategia pesimista, con el riesgo de que la transacción se retrotraiga y el coste de memoria de realizar una copia adicional de la entrada.

Establezca la estrategia de bloqueo en la BackingMap. No puede cambiar la estrategia de bloqueo para cada transacción. A continuación, aparece un fragmento de código XML de ejemplo que muestra cómo establecer la modalidad de bloqueo en una correlación utilizando el archivo XML, que da por supuesto que cc es el espacio de nombres para el espacio de nombres de objectgrid/config:

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Bloqueo pesimista

Utilice la estrategia de bloqueo pesimista en operaciones de correlación de lectura y grabación cuando no es posible utilizar otra estrategia de bloqueo. Cuando se configura una correlación ObjectGrid para utilizar la estrategia de bloqueo pesimista, se obtiene un bloqueo de transacción pesimista para una entrada de correlación cuando una transacción obtiene por primera vez la entrada de BackingMap. El bloqueo pesimista se mantiene hasta que la aplicación completa la transacción. Por lo general, la estrategia de bloqueo pesimista se utiliza en las situaciones siguientes:

- Cuando BackingMap se configura con o sin un cargador y la información de creación de versiones no está disponible.
- Cuando BackingMap se utiliza directamente en una aplicación que necesita ayuda de eXtreme Scale para el control de simultaneidad.
- Cuando la información de creación de versiones está disponible, pero las transacciones de actualización colisionan con frecuencia en las entradas de respaldo, lo cual produce anomalías optimistas de actualización.

Como la estrategia de bloqueo pesimista tiene el mayor impacto sobre el rendimiento y la escalabilidad, esta estrategia sólo debe utilizarse para correlaciones de lectura y grabación, cuando no es viable ninguna otra estrategia de bloqueo. Por ejemplo, estas situaciones podrían incluir cuando se producen con frecuencia anomalías optimistas de actualización, o cuando es difícil para una aplicación gestionar la recuperación de una anomalía optimista.

Bloqueo optimista

En la estrategia de bloqueo optimista, se presupone que dos transacciones no pueden intentar actualizar la misma entrada de correlación mientras se ejecutan simultáneamente. Por este motivo, la modalidad de bloqueo no necesita mantenerse para el ciclo de vida de la transacción, porque ya que es improbable

que más de una transacción actualice la entrada de correlación simultáneamente. Por lo general, la estrategia de bloqueo optimista se utiliza en las situaciones siguientes:

- Cuando BackingMap se configura con o sin un cargador y la información de creación de versiones está disponible.
- Cuando BackingMap tiene mayoritariamente transacciones que realizan operaciones de lectura. En BackingMap, las operaciones insert, update o remove no se producen con frecuencia en las entradas de correlaciones.
- Cuando una correlación BackingMap se inserta, actualiza o elimina con más frecuencia de lo que se lee, pero las transacciones rara vez colisionan en la misma entrada de correlación.

Al igual que en la estrategia de bloqueo pesimista, los métodos de la interfaz ObjectMap determinan cómo eXtreme Scale intenta automáticamente adquirir una modalidad de bloqueo para una entrada de correlación a la que se accede. No obstante, las diferencias entre las estrategias optimistas y pesimistas son:

- Al igual que la estrategia de bloqueo pesimista, los métodos get y getAll adquieren una modalidad de bloqueo S cuando se invoca el método. Sin embargo, con el bloqueo optimista, la modalidad de bloqueo S no se mantiene hasta que finaliza la transacción, sino que se libera antes de que el método vuelva a la aplicación. El propósito de adquirir la modalidad de bloqueo es que eXtreme Scale pueda garantizar que sólo los datos confirmados de otras transacciones sean visibles a la transacción actual. Después de que eXtreme Scale haya comprobado que los datos se han confirmado, la modalidad de bloqueo S se libera. Durante el ciclo de confirmación, se realiza una comprobación de la creación de versiones optimista para garantizar que ninguna otra transacción haya modificado la entrada de correlación después de que la transacción actual haya liberado su modalidad de bloqueo S. Si no se capta una entrada de la correlación antes de que se actualice, invalide o suprima, el tiempo de ejecución de eXtreme Scale capta de forma implícita la entrada de la correlación. Esta operación get implícita se realiza para obtener el valor actual en el momento en que la entrada se solicitó para modificarse.
- A diferencia de la estrategia de bloqueo pesimista, los métodos getForUpdate y getAllForUpdate se manejan exactamente igual que los métodos get y getAll cuando se utiliza la estrategia de bloqueo optimista. Es decir, se adquiere una modalidad de bloqueo S al inicio del método y se libera la modalidad de bloqueo S antes de que se devuelva a la aplicación.

Todos los otros métodos ObjectMap se manejan exactamente igual que si se manejaran para la estrategia de bloqueo pesimista. Es decir, cuando se invoca el método commit, se obtiene una modalidad de bloqueo X para cualquier entrada de correlación que se inserte, actualice, elimine, manipule o invalide, y la modalidad de bloqueo X se mantiene hasta que la transacción complete el proceso de confirmación.

En la estrategia de bloqueo optimista, se presupone que ninguna transacción que se ejecute simultáneamente con otra intentará actualizar la misma entrada de correlación. Por este motivo, la modalidad de bloqueo no necesita mantenerse durante toda la vida de la transacción ya que es improbable que más de una transacción actualice la entrada de correlación simultáneamente. Sin embargo, como no se ha mantenido la modalidad de bloqueo, otra transacción simultánea podría actualizar de forma potencial la entrada de la correlación, después de que la transacción actual haya liberado su modalidad de bloqueo S.

Para manejar esta posibilidad, eXtreme Scale obtiene un bloqueo X durante el ciclo de confirmación y realiza una comprobación de la creación de versiones optimista para verificar que ninguna otra transacción haya modificado la entrada de correlación después de que la transacción actual haya leído la entrada de correlación en BackingMap. Si otra transacción ha modificado la entrada de correlación, se produce una anomalía en la comprobación de versiones y se muestra una excepción `OptimisticCollisionException`. Esta excepción obliga a la transacción actual retrotraerse y la aplicación debe volver a intentar toda la transacción. La estrategia de bloqueo optimista es muy útil cuando se producen sobre todo lecturas de una correlación y rara vez se producen actualizaciones de la misma entrada de correlación.

Sin bloqueo

Cuando un objeto `BackingMap` se configura para que no use ninguna estrategia de bloqueo, no se obtiene ningún bloqueo de transacción para una entrada de correlación.

No usar ninguna estrategia de bloqueo es útil cuando una aplicación es un gestor de persistencia como, por ejemplo, un contenedor EJB (Enterprise JavaBeans) o cuando una aplicación utiliza Hibernate para obtener los datos persistentes. En este escenario, `BackingMap` se configura sin cargador y el gestor de persistencia utiliza `BackingMap` como memoria caché de datos. En este escenario, el gestor de persistencia proporciona control de simultaneidad entre las transacciones que están accediendo a las mismas entradas de correlación.

WebSphere eXtreme Scale no necesita obtener ningún bloqueo de transacción para el control de simultaneidad. Esta situación presupone que el gestor de persistencia no libera sus bloqueos de transacción antes de actualizar la correlación de `ObjectGrid` con los cambios confirmados. Si el gestor de persistencia libera sus bloqueos, debe utilizarse una estrategia de bloqueo optimista o pesimista. Por ejemplo, suponga que el gestor de persistencia de un contenedor EJB está actualizando una correlación de `ObjectGrid` con los datos que se confirmaron en la transacción gestionada por el contenedor EJB. Si la actualización de la correlación de `ObjectGrid` se produce antes de que se liberen los bloqueos de transacción del gestor de persistencia, podrá utilizar la estrategia sin bloqueos. Si la actualización de la correlación de `ObjectGrid` se produce después de que se liberen los bloqueos de transacción del gestor de persistencia, debe utilizar la estrategia de bloqueo optimista o pesimista.

Otro escenario en el que se puede utilizar una estrategia sin bloqueos es cuando la aplicación utiliza `BackingMap` directamente y se ha configurado un cargador para la correlación. En este escenario, el cargador utiliza el soporte de control de simultaneidad proporcionado por un sistema de gestión de bases de datos relacionales (RDBMS) mediante el uso de Java Database Connectivity (JDBC) o Hibernate para acceder a los datos de la base de datos relacional. La implementación de cargador puede utilizar un acercamiento optimista o pesimista. Un cargador que utiliza un bloqueo optimista o un procedimiento de creación de versiones favorece un alto nivel de simultaneidad y rendimiento. Si desea más información sobre cómo implementar un enfoque de bloqueo optimista, consulte la sección `OptimisticCallback` en la información sobre las consideraciones de cargador en *Guía de administración*. Si utiliza un cargador que utiliza el soporte de bloqueo pesimista de un 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 140.

JMS para distribuir los cambios de transacciones

Utilice JMS (Java Message Service) para los cambios de transacciones distribuidas entre las distintas capas o en entornos en plataformas combinadas.

JMS es un protocolo ideal para distribuir cambios entre distintos niveles o en entornos con diferentes plataformas. Por ejemplo, algunas aplicaciones que utilizan eXtreme Scale se podrían desplegar en IBM WebSphere Application Server Community Edition, Apache Geronimo o Apache Tomcat, mientras que otras aplicaciones se podrían ejecutar en WebSphere Application Server versión 6.x. JMS es ideal para los cambios distribuidos entre los iguales de eXtreme Scale en estos distintos entornos. El transporte de mensajes de High Availability Manager es muy rápido, pero sólo puede distribuir cambios a Máquinas virtuales Java que estén en un único grupo principal. JMS es más lento, pero permite que conjuntos más grandes y más diversos de clientes de aplicación puedan compartir un ObjectGrid. JMS es ideal si se comparten datos en un ObjectGrid entre un cliente grueso de Swing y una aplicación desplegada en WebSphere Extended Deployment.

El mecanismo de invalidación de clientes incorporado y la réplica de igual a igual son ejemplos de distribución de cambios transaccionales basados en JMS. Consulte la información sobre cómo configurar la réplica de igual a igual con JMS en *Guía de administración* si desea más información.

Implementación de JMS

JMS se implementa para distribuir los cambios de transacciones utilizando un objeto Java que se comporta como un ObjectGridEventListener. Este objeto puede propagar el estado de las cuatro formas siguientes:

1. Invalidación: las entradas desalojadas, actualizadas o suprimidas se eliminan de todas las 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

El mismo plug-in también inicia una hebra que forma un bucle y recibe todos los mensajes publicados para un tema conocido. Cuando llega un mensaje, se pasa el contenido del mensaje a la clase LogSequenceTransformer, donde se convierte a un conjunto de objetos LogSequence. A continuación, se inicia una transacción de no escritura a través. Cada objeto LogSequence se proporciona al método Session.processLogSequence, que actualiza las correlaciones locales con los cambios. El método processLogSequence entiende la modalidad de distribución. La transacción se confirma y la memoria caché local refleja los cambios. Si desea más información sobre cómo utilizar JMS para distribuir cambios de transacción, consulte la información sobre cómo distribuir los cambios entre máquinas virtuales Java iguales en *Guía de administración*.

Transacciones de partición única y de partición entre cuadrícula

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

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

Transacciones de partición única

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

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

devuelva a la cuadrícula, la memoria caché vuelve al 100 por ciento de capacidad de rendimiento.

Transacciones entre cuadrícula

En términos de rendimiento, disponibilidad y escalabilidad, las transacciones entre cuadrícula son el opuesto de las transacciones de partición única. Las transacciones entre cuadrícula acceden a cada partición y, por lo tanto, a todos los sistemas de la configuración. Se solicita a cada sistema de la cuadrícula que busque algunos datos y, a continuación, devuelva el resultado. La transacción no se puede completar hasta que hayan contestado todos los sistemas y, por lo tanto, el rendimiento de toda la cuadrícula está limitado al sistema 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 aumenta 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 todas 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, añade sistemas y añade otra cuadrícula a las solicitudes de servicio. Si el proceso de datos se debe escalar de forma vertical, añade más sistemas y mantenga constante el número de cuadrículas.

Manejo de bloqueos

Los bloqueos tienen ciclos de vida y tipos de bloqueos diferentes son compatibles con otros de distintas formas. Los bloqueos deben manejarse en el orden correcto para evitar escenarios de punto muerto.

Tiempos de espera de bloqueo

Cada BackingMap tiene un valor de tiempo de espera de bloqueo predeterminado. El valor de tiempo de espera se utiliza para garantizar que una aplicación no espere de forma ilimitada a que se otorgue una modalidad de bloqueo porque se

producirá una condición de punto muerto debido a un error de la aplicación. La aplicación puede utilizar la interfaz BackingMap para alterar temporalmente el valor predeterminado de tiempo de espera de bloqueo. El ejemplo siguiente ilustra cómo establecer el valor de tiempo de espera de bloqueo para la correlación de respaldo map1 en 60 segundos:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
    ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setLockTimeout( 60 );
```

Para evitar una excepción `java.lang.IllegalStateException`, llame al método `setLockStrategy` y, también, al método `setLockTimeout`, antes de llamar a los métodos `initialize` o `getSession` en la instancia `ObjectGrid`. El parámetro del método `setLockTimeout` es un entero primitivo Java que especifica el número de segundos que eXtreme Scale espera a que se otorgue una modalidad de bloqueo. Si una transacción espera más tiempo que el especificado en el valor de tiempo de espera de bloqueo configurado para `BackingMap`, se produce la excepción `com.ibm.websphere.objectgrid.LockTimeoutException`.

Cuando se produce una excepción `LockTimeoutException`, la aplicación debe determinar si el tiempo de espera excedido se produce porque la aplicación se ejecuta más despacio de lo normal, o si el tiempo de espera excedido se produce debido a una condición de punto muerto. Si se ha producido una condición de punto muerto, aumentar el valor de tiempo de espera de bloqueo no elimina la excepción. Si se incrementa el valor de tiempo de espera, la excepción tarda más en producirse. No obstante, si al aumentar el valor de tiempo de espera de bloqueo se elimina la excepción, la fuente del problema era que la aplicación se estaba ejecutando más despacio de lo esperado. La aplicación en este caso debe determinar por qué el rendimiento es lento.

Tiempo de espera de bloqueo para ObjectMaps

El valor de tiempo de espera de bloqueo puede alterarse temporalmente en una única instancia `ObjectMap` mediante el uso del método `ObjectMap.setLockTimeout`. El valor de tiempo de espera de bloqueo afecta a todas las transacciones que se inician después de haber establecido el nuevo valor de tiempo de espera. Este método puede ser útil si es posible que se produzcan o se esperen colisiones de bloqueos en transacciones de tipo `select`.

Bloqueos compartidos, actualizables y exclusivos

Cuando una aplicación llama a cualquier método de la interfaz `ObjectMap`, utiliza los métodos de búsqueda en un índice, o realiza una consulta, eXtreme Scale intenta automáticamente adquirir un bloqueo para la entrada de correlación a la que se está accediendo. WebSphere eXtreme Scale utiliza las siguientes modalidades de bloqueo basadas en el método al que llama la aplicación en la interfaz `ObjectMap`.

- Los métodos `get` y `getAll` en la interfaz `ObjectMap`, los métodos de índice y las consultas adquieren un *bloqueo S*, o modalidad de bloqueo compartido para la clave de una entrada de correlación. La duración que mantiene el bloqueo *S* depende del nivel de aislamiento de la transacción utilizado. Una modalidad de

bloqueo S permite la simultaneidad de las transacciones que intentan adquirir una modalidad de bloqueo S o de bloqueo actualizable (bloqueo U) para la misma clave, pero bloquea otras transacciones que intenten obtener una modalidad de bloqueo exclusivo (bloqueo X) para la misma clave.

- Los métodos `getForUpdate` y `getAllForUpdate` adquieren un *bloqueo U*, o modalidad de bloqueo actualizable para la clave de una entrada de correlación. El bloqueo U se mantiene hasta que se completa la transacción. Una modalidad de bloqueo U permite la simultaneidad de las transacciones que adquieren una modalidad de bloqueo S para la misma clave, pero bloquea otras transacciones que intenten obtener una modalidad de bloqueo U o X para la misma clave.
- Los métodos `put`, `putAll`, `remove`, `removeAll`, `insert`, `update` y `touch` adquieren un *bloqueo X*, o modalidad de bloqueo exclusivo para la clave de una entrada de correlación. El bloqueo X se mantiene hasta que se completa la transacción. Una modalidad de bloqueo X garantiza que sólo una transacción inserte, actualice o elimine una entrada de correlación de un valor de clave dado. Un bloqueo X bloquea todas las otras transacciones que intenten adquirir una modalidad de bloqueo S, U o X para la misma clave.
- Los métodos globales `invalidate` e `invalidateAll` adquieren un bloqueo X para cada entrada de correlación que se invalida. El bloqueo X se mantiene hasta que se completa la transacción. No se adquiere ningún bloqueo para los métodos locales `invalidate` e `invalidateAll` porque ninguna de las entradas de `BackingMap` se invalida mediante llamadas a métodos `invalidate` locales.

A partir de las definiciones anteriores, es obvio que una modalidad de bloqueo S es más débil que una modalidad de bloqueo U ya que permite que se ejecuten simultáneamente más transacciones al acceder a la misma entrada de correlación. La modalidad de bloqueo U es ligeramente más restrictiva que la modalidad de bloqueo S ya que bloquea las otras transacciones que soliciten una modalidad de bloqueo U o X. La modalidad de bloqueo S sólo bloquea a las otras transacciones que soliciten una modalidad de bloqueo X. Esta pequeña diferencia es importante para evitar situaciones de punto muerto. La modalidad de bloqueo X es la más fuerte porque bloquea todas las otras transacciones que intenten obtener una modalidad de bloqueo S, U o X para la misma entrada de correlación. La modalidad de bloqueo X garantiza que sólo una transacción pueda insertar, actualizar o eliminar una entrada de correlación. De este modo, se evita que se pierdan actualizaciones cuando más de una transacción intenta actualizar la misma entrada de correlación.

En la tabla siguiente se ofrece una matriz de compatibilidad de modalidades de bloqueo que resume las modalidades de bloqueo descritas, que puede utilizar para determinar las modalidades de bloqueo que son compatibles entre sí. La fila de la matriz indica una modalidad de bloqueo que ya se ha otorgado. La columna indica la modalidad de bloqueo que solicita otra transacción. Si en la columna aparece Sí, se otorga la modalidad de bloqueo solicitada por otra transacción porque es compatible con la modalidad de bloqueo que ya se ha otorgado. Si aparece No, indica que la modalidad de bloqueo no es compatible y, por tanto, la otra transacción debe esperar a que la primera transacción libere el bloqueo.

Tabla 4. Matriz de compatibilidad de modalidad de bloqueo

Bloqueo	Tipo de bloqueo S (compartido)	Tipo de bloqueo U (actualizable)	Tipo de bloqueo X (exclusivo)	Fuerza
S (compartido)	Sí	Sí	No	más débil
U (actualizable)	Sí	No	No	normal
X (exclusivo)	No	No	No	más fuerte

Puntos muertos de bloqueo

Considere la siguiente secuencia de peticiones de modalidad de bloqueo:

1. Se otorga el bloqueo X a la transacción 1 para key1.
2. Se otorga el bloqueo X a la transacción 2 para key2.
3. La transacción 1 solicita el bloqueo X para key2. (La transacción 1 se bloquea a la espera del bloqueo en propiedad de la transacción 2).
4. La transacción 2 solicita el bloqueo X para key1. (La transacción 2 se bloquea a la espera del bloqueo en propiedad de la transacción 1).

La secuencia anterior es el ejemplo clásico de punto muerto en el que dos transacciones intentan adquirir más de un solo bloqueo, y cada una de ellas adquiere los bloqueos en un orden diferente. Para evitar esta situación de punto muerto, cada transacción debe obtener los diversos bloqueos en el mismo orden. Si se utiliza la estrategia de bloqueo OPTIMISTIC y la aplicación nunca utiliza el método flush de la interfaz ObjectMap, la transacción sólo solicita las modalidades de bloqueo durante el ciclo de confirmación. Durante este ciclo, eXtreme Scale determina las claves de las entradas de correlación que deben bloquearse y solicita las modalidades de bloqueo en la secuencia de claves (comportamiento determinista). Con este método, eXtreme Scale evita la gran mayoría de los puntos muertos clásicos. No obstante, eXtreme Scale no puede evitar todos los escenarios posibles de punto muerto. Existen unos pocos escenarios que la aplicación debe tener en cuenta. A continuación se muestran algunos de éstos para que la aplicación pueda tomar acciones preventivas.

Se da un escenario en el que eXtreme Scale puede detectar un punto muerto sin tener que esperar a que se produzca un tiempo de espera de bloqueo. Si se da este escenario, se produce una excepción `com.ibm.websphere.objectgrid.LockDeadlockException`. Observe el fragmento de código siguiente:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Lynn");
// Ha sido el cumpleaños de Lynn, por lo que es 1 año mayor.
p.setAge( p.getAge() + 1 );
person.put( "Lynn", p );
sess.commit();
```

En esta situación, el novio de Lynn quiere que sea mayor de lo que lo es ahora y tanto Lynn, como su novio ejecutan esta transacción simultáneamente. En esta situación, ambas transacciones poseen una modalidad de bloqueo S en la entrada de Lynn de la correlación PERSON como resultado de la invocación del método `person.get("Lynn")`. Como resultado de la llamada del método `person.put("Lynn", p)`, ambas transacciones intentan actualizar la modalidad de bloqueo S a una modalidad de bloqueo X. Las dos transacciones se bloquean a la espera de que la otra transacción libere la modalidad de bloqueo S. Por lo tanto, se produce un punto muerto al darse una condición de espera circular entre las dos transacciones. Esta condición de espera circular se produce cuando más de una transacción intenta promover un bloqueo de una modalidad más débil a una más fuerte para la misma entrada de correlación. En este escenario, se produce una excepción `LockDeadlockException` en lugar de una excepción `LockTimeoutException`.

En el ejemplo anterior, la aplicación puede evitar la excepción `LockDeadlockException` si utiliza una estrategia de bloqueo optimista en lugar de la estrategia de bloqueo pesimista. El uso de una estrategia de bloqueo optimista

es la solución preferida cuando básicamente se realizan lecturas en la correlación, y las actualizaciones no son frecuentes. Si debe utilizarse la estrategia de bloqueo pesimista, utilice el método `getForUpdate` en lugar del método `get` del ejemplo anterior o un nivel de aislamiento de transacción de `TRANSACTION_READ_COMMITTED`.

Si desea más información, consulte el tema sobre las estrategias de bloqueo en *Visión general del producto*.

El uso del nivel de aislamiento de la transacción `TRANSACTION_READ_COMMITTED` impide que se obtenga el bloqueo S adquirido por el método `get` hasta que se complete la transacción. Si nunca se invalida la clave en la memoria caché transaccional, se siguen garantizando las lecturas repetibles.

Consulte el tema sobre el bloqueo de la entrada de correlación en la *Guía de administración* si desea más información.

Un procedimiento alternativo para cambiar el nivel de aislamiento de la transacción es utilizar el método `getForUpdate`. La primera transacción que llama al método `getForUpdate` adquiere una modalidad de bloqueo U en lugar de un bloqueo S. Esta modalidad de bloqueo hace que se bloquee la segunda transacción al llamar al método `getForUpdate` porque sólo se otorga una modalidad de bloqueo U a una transacción. Puesto que la segunda transacción está bloqueada, no posee ninguna modalidad de bloqueo en la entrada de la correlación de Lynn. La primera transacción no se bloquea cuando intenta actualizar la modalidad de bloqueo U a una modalidad de bloqueo X como resultado de la llamada de método `put` de la primera transacción. Esta característica demuestra por qué la modalidad de bloqueo U se llama *actualizable*. Cuando se completa la primera transacción, la segunda transacción se desbloquea y se le otorga la modalidad de bloqueo U. Cuando se utiliza una estrategia de bloqueo pesimista, una aplicación puede evitar que se produzca un escenario de punto muerto de promoción de bloqueo si utiliza el método `getForUpdate` en lugar del método `get`.

Importante: esta solución no impide que las transacciones de sólo lectura puedan leer una entrada de correlación. Las transacciones de sólo lectura llaman al método `get`, pero nunca llaman a los métodos `put`, `insert`, `update` ni `remove`. La simultaneidad es tan alta como cuando el utiliza el método `get`. La única reducción en la simultaneidad se produce cuando más de una transacción llama al método `getForUpdate` para la misma entrada de correlación.

Debe saber cuándo una transacción llama al método `getForUpdate` para más de una entrada de correlación y así garantizar que cada transacción adquiera los bloqueos U en el mismo orden. Por ejemplo, suponga que la primera transacción llama al método `getForUpdate` para la clave 1 y al método `getForUpdate` para la clave 2. Otra transacción simultánea llama al método `getForUpdate` para las mismas claves, pero en orden inverso. Esta secuencia dará lugar a una situación clásica de punto muerto ya que varias transacciones obtienen bloqueos en distinto orden. La aplicación debe asegurarse de que cada transacción accede a varias entradas de correlación en la secuencia de claves para garantizar que no se produzca un punto muerto. Como se obtiene el bloqueo U en el momento en el que se llama al método `getForUpdate` en lugar de en el ciclo de confirmación, eXtreme Scale no puede ordenar las solicitudes de bloqueo como lo hace durante el ciclo de confirmación. La aplicación debe controlar el orden de los bloqueos en este caso.

El uso del método flush de la interfaz ObjectMap antes de una confirmación presenta consideraciones de orden de bloqueos adicionales. El método flush suele utilizarse para forzar cambios realizados en la correlación para el programa de fondo a través del plug-in Loader. En esta situación, el programa de fondo utiliza su propio gestor de bloqueos para controlar la simultaneidad, de modo que la condición de espera de bloqueos y el punto muerto pueden producirse en el programa de fondo en lugar de en el gestor de bloqueos de eXtreme Scale. Observe la transacción siguiente:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
    sess.begin();
    activeTran = true;
    Person p = (IPerson)person.get("Lynn");
    p.setAge( p.getAge() + 1 );
    person.put( "Lynn", p );
    person.flush();
    ...
    p = (IPerson)person.get("Tom");
    p.setAge( p.getAge() + 1 );
    sess.commit();
    activeTran = false;
}
finally
{
    if ( activeTran ) sess.rollback();
}
```

Suponga que otra transacción también ha actualizado la persona Tom, llamó al método flush y, a continuación, actualizó la persona Lynn. Si se produjera esta situación, el intercalado de las dos transacciones provocaría una condición de punto muerto de la base de datos:

Se otorga el bloqueo X a la transacción 1 para "Lynn" cuando se ejecuta el método flush. Se otorga el bloqueo X a la transacción 2 para "Tom" cuando se ejecuta el método flush.

La transacción 1 solicita el bloqueo X para "Tom" durante el proceso de confirmación. (La transacción 1 se bloquea a la espera del bloqueo en propiedad de la transacción 2). El bloqueo X solicitado por la transacción 2 para "Lynn" durante el proceso de confirmación.

(La transacción 2 se bloquea a la espera del bloqueo en propiedad de la transacción 1).

Este ejemplo demuestra que el uso del método flush puede causar un punto muerto en la base de datos en lugar de en eXtreme Scale. Este ejemplo de punto muerto puede ocurrir independientemente del tipo de estrategia de bloqueo utilizado. La aplicación debe evitar que se produzca este punto muerto al utilizar el método flush y cuando un objeto Loader se conecta a BackingMap. El ejemplo anterior también ilustra otra razón por la que eXtreme Scale tiene un mecanismo de tiempo de espera de bloqueo. Una transacción que espera un bloqueo de la base de datos podría estar esperando mientras posee un bloqueo de entrada de correlación de eXtreme Scale. En consecuencia, los problemas a nivel de base de datos pueden ocasionar tiempos de espera excesivos para una modalidad de bloqueo de eXtreme Scale y terminar en una excepción LockTimeoutException.

Escenarios comunes de puntos muertos

Las siguientes secciones describen algunos de los escenarios más comunes de punto muerto y algunas sugerencias para evitarlos.

Escenario: puntos muertos de una sola clave

Los siguientes escenarios describen cómo se pueden producir puntos muertos cuando se accede a una única clave mediante un bloqueo S y que se actualiza más adelante. Cuando esto se produce desde dos transacciones que se ejecutan simultáneamente, se produce un punto muerto.

Tabla 5. Escenario de puntos muertos de llave única

	Hebra 1	Hebra 2	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.get(key1)	map.get(key1)	Se otorga el bloqueo S a las dos transacciones para key1.
3	map.update(Key1,v)		No se produce un bloqueo U. La actualización se realiza en la memoria caché transaccional.
4		map.update(key1,v)	No se produce un bloqueo U. La actualización se realiza en la memoria caché transaccional.
5	session.commit()		Bloqueado: el bloqueo S de key1 no se puede actualizar a un bloqueo X porque la hebra 2 tiene un bloqueo S.
6		session.commit()	Punto muerto: el bloqueo S de key1 no se puede actualizar a un bloqueo X porque la hebra 1 tiene un bloqueo S.

Tabla 6. Puntos muertos de llave única, continuación

	Hebra 1	Hebra 2	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.get(key1)		Se otorga el bloqueo S para key1.
3	map.getForUpdate(key1,v)		El bloqueo S se actualiza a un bloqueo U para key1.
4		map.get(key1)	Se otorga el bloqueo S para key1.
5		map.getForUpdate(key1,v)	Bloqueado: la hebra 1 ya tiene el bloqueo U.
6	session.commit()		Punto muerto: el bloqueo U de key1 no se puede actualizar.
7		session.commit()	Punto muerto: no se puede actualizar el bloqueo S para la clave key1.

Tabla 7. Puntos muertos de llave única, continuación

	Hebra 1	Hebra 2	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.get(key1)		Se otorga el bloqueo S para key1.
3	map.getForUpdate(key1,v)		El bloqueo S se actualiza a un bloqueo U para key1.

Tabla 7. Puntos muertos de llave única, continuación (continuación)

	Hebra 1	Hebra 2	
4		map.get(key1)	Se otorga el bloqueo S para key1.
5		map.getForUpdate(key1,v)	Bloqueado: la hebra 1 ya tiene el bloqueo U.
6	session.commit()		Punto muerto: el bloqueo U de key1 no se puede actualizar a un bloqueo X porque la hebra 2 tiene un bloqueo S.

Si se utiliza ObjectMap.getForUpdate para evitar el bloqueo S, no tendrá lugar el punto muerto:

Tabla 8. Puntos muertos de llave única, continuación

	Hebra 1	Hebra 2	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.getForUpdate(key1)		Se otorga el bloqueo U a la hebra 1 para key1.
3		map.getForUpdate(key1)	Se bloquea la solicitud de bloqueo U.
4	map.update(key1,v)	<bloqueado>	
5	session.commit()	<bloqueado>	El bloqueo U de key1 puede actualizarse correctamente a un bloqueo X.
6		<liberado>	El bloqueo U se otorga finalmente a key1 para la hebra 2.
7		map.update(key2,v)	Se otorga el bloqueo U a la hebra 2 para key2.
8		session.commit()	El bloqueo U de key1 puede actualizarse correctamente a un bloqueo X.

Soluciones

1. Use el método getForUpdate en lugar del método get para adquirir un bloqueo U en lugar de un bloqueo S.
2. Use un nivel de aislamiento de transacción de lectura confirmada para evitar mantener bloqueos S. Al reducir el nivel de aislamiento de la transacción, aumenta la posibilidad de lecturas no repetibles. No obstante, las lecturas no repetibles sólo son posibles si la memoria caché de la transacción se invalida explícitamente.
3. Use la estrategia de bloqueo optimista. El uso de la estrategia de bloqueo optimista requiere el manejo de excepciones de colisión optimista.

Escenario: punto muerto de varias claves ordenadas

En este escenario se describe qué sucede si dos transacciones intentan actualizar la misma entrada directamente y mantienen bloqueos S para otras entradas.

Tabla 9. Escenario de punto muerto de varias llaves ordenadas

	Hebra 1	Hebra 2	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.get(key1)	map.get(key1)	Se otorga el bloqueo S a las dos transacciones para key1.
3	map.get(key2)	map.get(key2)	Se otorga el bloqueo S a las dos transacciones para key2.
4	map.update(key1,v)		No se produce un bloqueo U. La actualización se realiza en la memoria caché transaccional.
5		map.update(key2,v)	No se produce un bloqueo U. La actualización se realiza en la memoria caché transaccional.
6.	session.commit()		Bloqueado: el bloqueo S de key1 no se puede actualizar a un bloqueo X porque la hebra 2 tiene un bloqueo S.
7		session.commit()	Punto muerto: el bloqueo S de key2 no se puede actualizar porque la hebra 1 tiene un bloqueo S.

Puede utilizar el método `ObjectMap.getForUpdate` para evitar el bloqueo S, después puede evitar el punto muerto:

Tabla 10. Escenario de punto muerto de varias llaves ordenadas, continuación

	Hebra 1	Hebra 2	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.getForUpdate(key1)		Se otorga el bloqueo U a la transacción de hebra 1 para key1.
3		map.getForUpdate(key1)	Se bloquea la solicitud de bloqueo U.
4	map.get(key2)	<bloqueado>	Se otorga el bloqueo S a la hebra 1 para key2.
5	map.update(key1,v)	<bloqueado>	
6	session.commit()	<bloqueado>	El bloqueo U de key1 puede actualizarse correctamente a un bloqueo X.
7		<liberado>	El bloqueo U se otorga finalmente a key1 para la hebra 2.
8		map.get(key2)	Se otorga el bloqueo S a la hebra 2 para key2.
9		map.update(key2,v)	Se otorga el bloqueo U a la hebra 2 para key2.
10		session.commit()	El bloqueo U de key1 puede actualizarse correctamente a un bloqueo X.

Soluciones

1. Use `getForUpdate` en lugar del método `get` para adquirir un bloqueo U directamente para la primera clave. Esta estrategia sólo funciona si el orden de los métodos es determinista.
2. Use un nivel de aislamiento de transacción de lectura confirmada para evitar mantener bloqueos S. Esta solución es la más fácil de implementar si el orden de los métodos no es determinista. Al reducir el nivel de aislamiento de la transacción, aumenta la posibilidad de lecturas no repetibles. No obstante, las lecturas no repetibles sólo son posibles si la memoria caché de la transacción se invalida explícitamente.
3. Use la estrategia de bloqueo optimista. El uso de la estrategia de bloqueo optimista requiere el manejo de excepciones de colisión optimista.

Escenario: sin orden con bloqueo U

Si no se puede garantizar el orden en que se solicitan las claves, podría darse una situación de punto muerto:

Tabla 11. Escenario de fuera de servicio con bloqueo U

	Hebra 1	Hebra 2	
1	<code>session.begin()</code>	<code>session.begin()</code>	Cada hebra establece una transacción independiente.
2	<code>map.getforUpdate(key1)</code>	<code>map.getForUpdate(key2)</code>	Se otorgan correctamente bloqueos U para <code>key1</code> y <code>key2</code> .
3	<code>map.get(key2)</code>	<code>map.get(key1)</code>	Se otorga el bloqueos S para <code>key1</code> y <code>key2</code> .
4	<code>map.update(key1,v)</code>	<code>map.update(key2,v)</code>	
5	<code>session.commit()</code>		El bloqueo U no se puede actualizar a un bloqueo X porque la hebra 2 tiene un bloqueo S.
6		<code>session.commit()</code>	El bloqueo U no se puede actualizar a un bloqueo X porque la hebra 1 tiene un bloqueo S.

Soluciones

1. Envuelva todo el trabajo con un único bloqueo U global (mútex). Este método reduce la simultaneidad, pero maneja todos los escenarios cuando el acceso y el orden no son deterministas.
2. Use un nivel de aislamiento de transacción de lectura confirmada para evitar mantener bloqueos S. Esta solución es la más fácil de implementar si el orden de los métodos no es determinista y proporciona un alto nivel de simultaneidad. Al reducir el nivel de aislamiento de la transacción, aumenta la posibilidad de lecturas no repetibles. No obstante, las lecturas no repetibles sólo son posibles si la memoria caché de la transacción se invalida explícitamente.
3. Use la estrategia de bloqueo optimista. El uso de la estrategia de bloqueo optimista requiere el manejo de excepciones de colisión optimista.

Manejo de excepciones en escenarios de bloqueo

Los ejemplos siguientes no tienen ningún manejo de excepciones. Para evitar que los bloqueos se mantengan durante un tiempo excesivo cuando se produzcan las excepciones `LockTimeoutException` o `LockDeadlockException`, una aplicación debe captar las excepciones no esperadas y llamar al método de retrotracción cuando sucede alguna acción inesperada. Cambie el fragmento de código anterior como se muestra en el ejemplo siguiente:

```

Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
    sess.begin();
    activeTran = true;
    Person p = (IPerson)person.get("Lynn");
    // Ha sido el cumpleaños de Lynn, por lo que es 1 año mayor.
    p.setAge( p.getAge() + 1 );
    person.put( "Lynn", p );
    sess.commit();
    activeTran = false;
}
finally
{
    if ( activeTran ) sess.rollback();
}

```

El bloque finally del fragmento de código garantiza que una transacción se retrotraiga cuando se produzca una excepción inesperada. No sólo maneja la excepción LockDeadlockException, sino cualquier otra excepción inesperada que pueda producirse. El bloque finally maneja el caso en el que se produce una excepción durante la invocación del método commit. Este ejemplo no es el único modo de tratar las excepciones inesperadas, y podrían darse casos en los que una aplicación desea captar algunas de las excepciones inesperadas que puedan producirse y mostrar una de las excepciones de la aplicación. Puede añadir bloques catch como desee, pero la aplicación debe garantizar que el fragmento de código no salga sin completar la transacción.

Estrategias de bloqueo

Las estrategias de bloqueo pueden ser de tipo pesimista, optimista o ninguno. Para elegir la estrategia de bloqueo, debe tener en cuenta cuestiones como el porcentaje de cada tipo de operaciones que realizará, si utilizará un cargador o no, etc.

Los bloqueos son enlazados por transacciones. Puede especificar los siguientes valores de bloqueo:

- **Sin bloqueo:** la ejecución sin el valor de bloqueo es la más rápida. Si utiliza datos de sólo lectura, es posible que no necesite el bloqueo.
- **Bloqueo pesimista:** adquiere bloqueos sobre entradas y luego mantiene los bloqueos hasta que se realiza la confirmación. Esta estrategia de bloqueo proporciona una mayor coherencia a costa del rendimiento.
- **Bloqueo optimista:** toma una imagen anterior de cada registro que toca la transacción y compara la imagen con los valores de entrada actuales cuando se confirma la transacción. Si los valores de entrada cambian, la transacción se retrotrae. No se mantiene ningún bloqueo hasta el momento de la confirmación. Esta estrategia de bloqueo proporciona una mejor concurrencia que la estrategia pesimista, con el riesgo de que la transacción se retrotraiga y el coste de memoria de realizar una copia adicional de la entrada.

Establezca la estrategia de bloqueo en la BackingMap. No puede cambiar la estrategia de bloqueo para cada transacción. A continuación, aparece un fragmento de código XML de ejemplo que muestra cómo establecer la modalidad de bloqueo en una correlación utilizando el archivo XML, que da por supuesto que cc es el espacio de nombres para el espacio de nombres de objectgrid/config:

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Bloqueo pesimista

Utilice la estrategia de bloqueo pesimista en operaciones de correlación de lectura y grabación cuando no es posible utilizar otra estrategia de bloqueo. Cuando se configura una correlación `ObjectGrid` para utilizar la estrategia de bloqueo pesimista, se obtiene un bloqueo de transacción pesimista para una entrada de correlación cuando una transacción obtiene por primera vez la entrada de `BackingMap`. El bloqueo pesimista se mantiene hasta que la aplicación completa la transacción. Por lo general, la estrategia de bloqueo pesimista se utiliza en las situaciones siguientes:

- Cuando `BackingMap` se configura con o sin un cargador y la información de creación de versiones no está disponible.
- Cuando `BackingMap` se utiliza directamente en una aplicación que necesita ayuda de `eXtreme Scale` para el control de simultaneidad.
- Cuando la información de creación de versiones está disponible, pero las transacciones de actualización colisionan con frecuencia en las entradas de respaldo, lo cual produce anomalías optimistas de actualización.

Como la estrategia de bloqueo pesimista tiene el mayor impacto sobre el rendimiento y la escalabilidad, esta estrategia sólo debe utilizarse para correlaciones de lectura y grabación, cuando no es viable ninguna otra estrategia de bloqueo. Por ejemplo, estas situaciones podrían incluir cuando se producen con frecuencia anomalías optimistas de actualización, o cuando es difícil para una aplicación gestionar la recuperación de una anomalía optimista.

Bloqueo optimista

En la estrategia de bloqueo optimista, se presupone que dos transacciones no pueden intentar actualizar la misma entrada de correlación mientras se ejecutan simultáneamente. Por este motivo, la modalidad de bloqueo no necesita mantenerse para el ciclo de vida de la transacción, porque ya que es improbable que más de una transacción actualice la entrada de correlación simultáneamente. Por lo general, la estrategia de bloqueo optimista se utiliza en las situaciones siguientes:

- Cuando `BackingMap` se configura con o sin un cargador y la información de creación de versiones está disponible.
- Cuando `BackingMap` tiene mayoritariamente transacciones que realizan operaciones de lectura. En `BackingMap`, las operaciones `insert`, `update` o `remove` no se producen con frecuencia en las entradas de correlaciones.
- Cuando una correlación `BackingMap` se inserta, actualiza o elimina con más frecuencia de lo que se lee, pero las transacciones rara vez colisionan en la misma entrada de correlación.

Al igual que en la estrategia de bloqueo pesimista, los métodos de la interfaz `ObjectMap` determinan cómo `eXtreme Scale` intenta automáticamente adquirir una modalidad de bloqueo para una entrada de correlación a la que se accede. No obstante, las diferencias entre las estrategias optimistas y pesimistas son:

- Al igual que la estrategia de bloqueo pesimista, los métodos `get` y `getAll` adquieren una modalidad de bloqueo `S` cuando se invoca el método. Sin embargo, con el bloqueo optimista, la modalidad de bloqueo `S` no se mantiene hasta que finaliza la transacción, sino que se libera antes de que el método vuelva a la aplicación. El propósito de adquirir la modalidad de bloqueo es que `eXtreme Scale` pueda garantizar que sólo los datos confirmados de otras transacciones sean visibles a la transacción actual. Después de que `eXtreme Scale` haya comprobado que los datos se han confirmado, la modalidad de bloqueo `S`

se libera. Durante el ciclo de confirmación, se realiza una comprobación de la creación de versiones optimista para garantizar que ninguna otra transacción haya modificado la entrada de correlación después de que la transacción actual haya liberado su modalidad de bloqueo S. Si no se capta una entrada de la correlación antes de que se actualice, invalide o suprima, el tiempo de ejecución de eXtreme Scale capta de forma implícita la entrada de la correlación. Esta operación get implícita se realiza para obtener el valor actual en el momento en que la entrada se solicitó para modificarse.

- A diferencia de la estrategia de bloqueo pesimista, los métodos `getForUpdate` y `getAllForUpdate` se manejan exactamente igual que los métodos `get` y `getAll` cuando se utiliza la estrategia de bloqueo optimista. Es decir, se adquiere una modalidad de bloqueo S al inicio del método y se libera la modalidad de bloqueo S antes de que se devuelva a la aplicación.

Todos los otros métodos `ObjectMap` se manejan exactamente igual que si se manejaran para la estrategia de bloqueo pesimista. Es decir, cuando se invoca el método `commit`, se obtiene una modalidad de bloqueo X para cualquier entrada de correlación que se inserte, actualice, elimine, manipule o invalide, y la modalidad de bloqueo X se mantiene hasta que la transacción complete el proceso de confirmación.

En la estrategia de bloqueo optimista, se presupone que ninguna transacción que se ejecute simultáneamente con otra intentará actualizar la misma entrada de correlación. Por este motivo, la modalidad de bloqueo no necesita mantenerse durante toda la vida de la transacción ya que es improbable que más de una transacción actualice la entrada de correlación simultáneamente. Sin embargo, como no se ha mantenido la modalidad de bloqueo, otra transacción simultánea podría actualizar de forma potencial la entrada de la correlación, después de que la transacción actual haya liberado su modalidad de bloqueo S.

Para manejar esta posibilidad, eXtreme Scale obtiene un bloqueo X durante el ciclo de confirmación y realiza una comprobación de la creación de versiones optimista para verificar que ninguna otra transacción haya modificado la entrada de correlación después de que la transacción actual haya leído la entrada de correlación en `BackingMap`. Si otra transacción ha modificado la entrada de correlación, se produce una anomalía en la comprobación de versiones y se muestra una excepción `OptimisticCollisionException`. Esta excepción obliga a la transacción actual retrotraerse y la aplicación debe volver a intentar toda la transacción. La estrategia de bloqueo optimista es muy útil cuando se producen sobre todo lecturas de una correlación y rara vez se producen actualizaciones de la misma entrada de correlación.

Sin bloqueo

Cuando un objeto `BackingMap` se configura para que no use ninguna estrategia de bloqueo, no se obtiene ningún bloqueo de transacción para una entrada de correlación.

No usar ninguna estrategia de bloqueo es útil cuando una aplicación es un gestor de persistencia como, por ejemplo, un contenedor EJB (Enterprise JavaBeans) o cuando una aplicación utiliza Hibernate para obtener los datos persistentes. En este escenario, `BackingMap` se configura sin cargador y el gestor de persistencia utiliza `BackingMap` como memoria caché de datos. En este escenario, el gestor de persistencia proporciona control de simultaneidad entre las transacciones que están accediendo a las mismas entradas de correlación.

WebSphere eXtreme Scale no necesita obtener ningún bloqueo de transacción para el control de simultaneidad. Esta situación presupone que el gestor de persistencia no libera sus bloqueos de transacción antes de actualizar la correlación de ObjectGrid con los cambios confirmados. Si el gestor de persistencia libera sus bloqueos, debe utilizarse una estrategia de bloqueo optimista o pesimista. Por ejemplo, suponga que el gestor de persistencia de un contenedor EJB está actualizando una correlación de ObjectGrid con los datos que se confirmaron en la transacción gestionada por el contenedor EJB. Si la actualización de la correlación de ObjectGrid se produce antes de que se liberen los bloqueos de transacción del gestor de persistencia, podrá utilizar la estrategia sin bloqueos. Si la actualización de la correlación de ObjectGrid se produce después de que se liberen los bloqueos de transacción del gestor de persistencia, debe utilizar la estrategia de bloqueo optimista o pesimista.

Otro escenario en el que se puede utilizar una estrategia sin bloqueos es cuando la aplicación utiliza BackingMap directamente y se ha configurado un cargador para la correlación. En este escenario, el cargador utiliza el soporte de control de simultaneidad proporcionado por un sistema de gestión de bases de datos relacionales (RDBMS) mediante el uso de Java Database Connectivity (JDBC) o Hibernate para acceder a los datos de la base de datos relacional. La implementación de cargador puede utilizar un acercamiento optimista o pesimista. Un cargador que utiliza un bloqueo optimista o un procedimiento de creación de versiones favorece un alto nivel de simultaneidad y rendimiento. Si desea más información sobre cómo implementar un enfoque de bloqueo optimista, consulte la sección OptimisticCallback en la información sobre las consideraciones de cargador en *Guía de administración*. Si utiliza un cargador que utiliza el soporte de bloqueo pesimista de un 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 140.

Procedimientos recomendados para optimizar el rendimiento de bloqueos

Los valores de las estrategias de bloqueo y del aislamiento de transacciones afectan el rendimiento de las aplicaciones.

Recuperar una instancia almacenada en memoria caché

Consulte la información sobre el bloqueo de entrada de correlación en *Guía de administración* si desea más información.

Estrategia de bloqueo pesimista

Utilice la estrategia de bloqueo pesimista en operaciones de correlación de lectura y grabación donde las claves suelen colisionar. La estrategia de bloqueo pesimista tiene un gran impacto sobre el rendimiento.

Aislamiento de transacciones de lectura confirmada y no confirmada

Si utiliza la estrategia de bloqueo pesimista, establezca el nivel de aislamiento de transacción a través del método Session.setTransactionIsolation. Para el aislamiento

confirmado de lectura o no confirmado de lectura, utilice los argumentos `Session.TRANSACTION_READ_COMMITTED` o `Session.TRANSACTION_READ_UNCOMMITTED` en función del aislamiento. Para restablecer el nivel de aislamiento en el comportamiento de bloqueo pesimista predeterminado, utilice el método `Session.setTransactionIsolation` con el argumento `Session.REPEATABLE_READ`.

El aislamiento de lectura confirmada reduce la duración de los bloqueos compartidos, que pueden mejorar la simultaneidad y reducir la probabilidad de puntos muertos. Este nivel de aislamiento debe utilizarse cuando una transacción no necesita la constatación de que los valores de lectura permanecen sin modificar durante la transacción.

Utilice una lectura no confirmada cuando la transacción no necesita ver los datos confirmados.

Estrategia de bloqueo optimista

El bloqueo optimista es la configuración predeterminada. Esta estrategia mejora el rendimiento y la escalabilidad en comparación con la estrategia pesimista. Utilice esta estrategia cuando las aplicaciones puedan tolerar anomalías de actualización optimista y el rendimiento siga siendo mejor que el de la estrategia pesimista. Esta estrategia es excelente en operaciones de lectura y aplicaciones con actualizaciones poco frecuentes.

Plug-in OptimisticCallback

La estrategia de bloqueo optimista realiza una copia de las entradas de la memoria caché y las compara como sea preciso. Esta operación puede resultar costosa porque la copia de la entrada podría implicar la clonación o serialización. Para conseguir el rendimiento más rápido posible, implemente el plug-in personalizado para las correlaciones que no son de entidad.

Consulte si desea más información. Consulte la información sobre el plug-in `OptimisticCallback` en *Visión general del producto* si desea más información.

Uso de campos de versión para entidades

Cuando utilice el bloqueo optimista con entidades, utilice la anotación `@Version` o el atributo equivalente en el archivo de descriptor de metadatos de entidad. La anotación de versión proporciona a `ObjectGrid` una forma muy eficiente de realizar el seguimiento de la versión de un objeto. Si la entidad no tiene un campo de versión y se utiliza un bloqueo optimista para la entidad, debe copiarse y compararse toda la entidad.

Estrategia de ningún bloqueo

Utilice esta estrategia en aplicaciones de sólo lectura. La estrategia de ningún bloqueo no obtiene ningún bloqueo ni usa un gestor de bloqueos. Por lo tanto, esta estrategia ofrece el rendimiento y la escalabilidad con más concurrencia.

Bloqueos de entrada de correlación con consultas e índices

Este tema describe cómo las API de consulta de eXtreme Scale y el plug-in de indexación `MapRangeIndex` interactúan con bloqueos y algunos procedimientos recomendados para aumentar la simultaneidad y reducir los puntos muertos al utilizar la estrategia de bloqueo pesimista para correlaciones.

Visión general

La API de consulta de ObjectGrid permite consultas SELECT en entidades y objetos de la memoria caché ObjectMap. Cuando se ejecuta una consulta, el motor de consultas utiliza un índice MapRangeIndex, siempre que es posible, para buscar claves coincidentes con los valores de la cláusula WHERE de la consulta o para servir de puente de las relaciones. Si no hay ningún índice disponible, el motor de consultas explorará cada entrada en una o más correlaciones para buscar las entradas correspondientes. El motor de consultas y los plug-ins de índices adquirirán bloqueos para comprobar los datos coherentes, en función de la estrategia de bloqueo, el nivel de aislamiento y el estado de la transacción.

Bloqueo con el plug-in HashIndex

El plug-in HashIndex de eXtreme Scale permite encontrar claves basadas en un único atributo almacenado en el valor de la entrada de la memoria caché. El índice almacena el valor indizado en una estructura de datos independiente de la correlación de memoria caché. El índice valida las claves respecto a las entradas de correlación antes de volver al usuario para intentar conseguir un conjunto de resultados precisos. Cuando se utiliza la estrategia de bloqueo pesimista y se usa el índice en una instancia local de ObjectMap (versus un ObjectMap de cliente/servidor), el índice adquirirá bloqueos para cada entrada coincidente. Si utiliza un bloqueo optimista o un objeto ObjectMap remoto, los bloqueos se liberan de forma inmediata.

El tipo de bloqueo que se adquiere depende del argumento forUpdate pasado al método ObjectMap.getIndex. El argumento forUpdate especifica el tipo de bloqueo que debe adquirir el índice. Si es false, se adquiere un bloqueo compatible (S) y si es true, se adquiere un bloqueo actualizable (U).

Si el bloqueo es de tipo compatible, se aplica el valor del aislamiento de la transacción de la sesión y afecta a la duración del bloqueo. Consulte el tema que trata sobre el aislamiento de transacciones para obtener más detalles sobre cómo se utiliza el aislamiento de transacciones para añadir simultaneidad a las aplicaciones.

Bloqueos compartidos con consulta

El motor de consultas de eXtreme Scale adquiere los bloqueos S cuando se necesita realizar una introspección de las entradas de la memoria caché para descubrir si cumplen los criterios del filtro de la consulta. Si se utiliza el aislamiento de transacciones de lectura repetible con bloqueo pesimista, los bloqueos compartibles S sólo se retienen en los elementos incluidos en el resultado de la consulta y se liberan en todas las entradas que no se incluyen en el resultado. Si utiliza un nivel de aislamiento de transacciones más bajo u optimista, los bloqueos S no se retienen.

Bloqueos compartidos con una consulta de cliente a servidor

Al utilizar la consulta de eXtreme Scale de un cliente, normalmente, la consulta se ejecuta en el servidor, a menos que todas las correlaciones o entidades a las que se hace referencia en la consulta son locales respecto al cliente (por ejemplo: una correlación replicada por el cliente o una entidad de resultado de consulta). Todas las consultas que se ejecutan en una transacción de lectura/grabación retendrán bloqueos S, como se ha descrito en el apartado anterior. Si la transacción no es una transacción de lectura/grabación, una sesión no se retiene en el servidor y los bloqueos S se liberan.

Una transacción de lectura/grabación sólo se direcciona a una partición primaria, y una sesión se mantiene en el servidor para la sesión de cliente. Una transacción puede promocionarse a lectura/grabación de acuerdo con las condiciones siguientes:

1. Se accede a cualquier correlación configurada para usar un bloqueo pesimista mediante los métodos de API `get` y `getAll` de `ObjectMap` o los métodos `EntityManager.find`.
2. La transacción se vacía, lo que ocasiona que se envíen actualizaciones al servidor.
3. Se accede a cualquier correlación configurada para usar un bloqueo optimista mediante los métodos `ObjectMap.getForUpdate` o `EntityManager.findForUpdate`.

Bloqueos actualizables con consulta

Los bloqueos compartidos son útiles cuando es importante la coherencia y simultaneidad. Garantizan que el valor de la entrada no cambie durante la vida de la transacción. Ninguna otra transacción podrá cambiar el valor mientras se mantengan otros bloqueos S, y sólo una transacción puede intentar actualizar la entrada. Para obtener más información sobre las modalidades de bloqueo S, U y X, consulte el tema sobre la modalidad de bloqueo pesimista.

Los bloqueos actualizables se utilizan para identificar el intento de actualizar una entrada de la memoria caché cuando se usa la estrategia de bloqueo pesimista. Permite la sincronización entre las transacciones que desean modificar una entrada de la memoria caché. Las transacciones pueden ver la entrada mediante un bloqueo S, pero otras transacciones no pueden adquirir un bloqueo U o un bloqueo X. En numerosos escenarios, es necesario adquirir un bloqueo U sin adquirir primero un bloqueo S para evitar situaciones de punto muerto. Consulte el tema sobre la modalidad de bloqueo pesimista para obtener ejemplos de situaciones comunes de punto muerto.

Las interfaces `ObjectQuery` y `EntityManager Query` proporcionan el método `setForUpdate` para identificar el uso previsto para el resultado de la consulta. De forma específica, el motor de consultas adquiere bloqueos U en lugar de bloqueos S para cada entrada de correlación del resultado de la consulta:

```
ObjectMap orderMap = session.getMap("Order");
ObjectQuery q = session.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
session.begin();
// Ejecutar la consulta. Cada orden tiene un bloqueo U
Iterator result = q.getResultIterator();
// Para cada orden, actualice el estado.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
    orderMap.update(o.getId(), o);
}
// Cuando se confirma,
session.commit();

Query q = em.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
emTran.begin();
// Ejecutar la consulta. Cada orden tiene un bloqueo U
Iterator result = q.getResultIterator();
// Para cada orden, actualice el estado.
```

```

while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
}
tmTran.commit();

```

Cuando se habilita el atributo **setForUpdate**, la transacción se convierte automáticamente en una transacción de lectura/grabación y los bloqueos se mantienen en el servidor, como se esperaba. Si la consulta no puede utilizar índices, la correlación debe explorarse, lo cual resultará en bloqueos U temporales para las entradas de correlación que no satisfagan el resultado de la consulta, y mantendrá bloqueos U para las entradas incluidas en el resultado.

Aislamiento de transacciones

Para las transacciones, puede configurar cada configuración de correlación de respaldo con una de las tres estrategias de bloqueo: pesimista, optimista o ninguno. Si utiliza el bloqueo pesimista y optimista, eXtreme Scale utiliza bloqueos compartidos (S), actualizables (U) y exclusivos (X) para mantener la coherencia. Este comportamiento de bloqueo es más notable cuando se utiliza el bloqueo pesimista, porque los bloqueos optimistas no se conservan. Puede utilizar uno de los tres niveles de aislamiento de transacción para ajustar la semántica del bloqueo que utiliza eXtreme Scale para mantener la coherencia en cada correlación de memoria caché: lectura repetible, lectura confirmada y lectura no confirmada.

Visión general del aislamiento de transacciones

El aislamiento de transacciones define cómo los cambios realizados por una operación se vuelven visibles para otras operaciones simultáneas.

WebSphere eXtreme Scale soporta tres niveles de aislamiento de transacción con las que puede ajustar de forma adicional la semántica del bloqueo que utiliza eXtreme Scale para mantener la coherencia en cada correlación de memoria caché: lectura repetible, lectura confirmada y lectura no confirmada. El nivel de aislamiento de transacción se establece en la interfaz Session utilizando el método `setTransactionIsolation`. El aislamiento de transacción se puede modificar en cualquier momento durante el ciclo de vida de la sesión, si una transacción no está actualmente en progreso.

El producto aplica las distintas semánticas de aislamiento de transacción ajustando la forma en la que se solicitan y conservan los bloqueos compartidos (S). El aislamiento de transacciones no tiene ningún efecto en las correlaciones configuradas para usar las estrategias de bloqueo optimista o ningún bloqueo o cuando se adquieren bloqueos actualizables (U).

Lectura repetible con bloqueo pesimista

El nivel de aislamiento de transacción de lectura repetible es el valor predeterminado. Este nivel de aislamiento impide lecturas sucias y lecturas no repetibles, pero no impide las lecturas fantasma. Una lectura sucia es una operación de lectura que se produce en datos que han sido modificados por una transacción, pero no han sido confirmados. Una lectura no repetible se puede producir cuando los bloqueos de lectura no se adquieren al realizar una operación de lectura. Una lectura fantasma se puede producir cuando se realizan dos operaciones de lectura idénticas, pero se devuelven dos conjuntos distintos de resultados porque se ha producido una actualización de los datos entre las operaciones de lectura. El producto consigue una lectura repetible manteniendo los

bloqueos S hasta que se complete la transacción que posee el bloqueo. Como un bloqueo X no se otorga hasta haberse liberado todos los bloqueos S, todas las transacciones que contienen el bloqueo S ven el mismo valor cuando se vuelven a leer.

```
map = session.getMap("Order");
session.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session.begin();

// Se solicita y se mantiene un bloqueo S y el valor se copia en
// la memoria caché transaccional.
Order order = (Order) map.get("100");
// La entrada se desaloja de la memoria caché transaccional.
map.invalidate("100", false);

// Se vuelve a solicitar el mismo valor. Ya contiene el
// bloqueo, por lo que se recupera el mismo valor y se copia en la
// memoria caché transaccional.
Order order2 (Order) = map.get("100");

// Se liberan todos los bloqueos después de sincronizar la transacción
// con la correlación de la memoria caché.
session.commit();
```

Las lecturas fantasmas son posibles si se utilizan consultas o índices, porque los bloqueos no se adquieren para los rangos de datos, sólo para las entradas de memoria caché que coinciden con los criterios de índice o consulta. Por ejemplo:

```
session1.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session1.begin();

// Se ejecuta una consulta que selecciona un intervalo de valores.
ObjectQuery query = session1.createObjectQuery
    ("SELECT o FROM Order o WHERE o.itemName='Widget'");

// En este caso, sólo un pedido coincide con el filtro de consultas.
// El pedido tiene una clave de "100".
// El motor de consultas adquiere automáticamente un bloqueo S para el pedido
// "100".
Iterator result = query.getResultIterator();

// Una segunda transacción inserta un pedido que también coincide con la consulta.
Map orderMap = session2.getMap("Order");
orderMap.insert("101", new Order("101", "Widget"));

// Cuando se vuelve a ejecutar la consulta en la transacción actual, el
// nuevo pedido es visible y devolverá los pedidos "100" y "101".
result = query.getResultIterator();

// Se liberan todos los bloqueos después de sincronizar la transacción
// con la correlación de la memoria caché.
session.commit();
```

Lectura confirmada con bloqueo pesimista

El nivel de aislamiento de la transacción de lectura confirmada se puede utilizar con eXtreme Scale, que impide las lecturas sucias, pero no impide lecturas no repetibles ni lecturas fantasma, de forma que eXtreme Scale sigue utilizando los bloqueos S para leer los datos de la correlación de memoria caché, pero libera inmediatamente los bloqueos.

```
map1 = session1.getMap("Order");
session1.setTransactionIsolation(Session.TRANSACTION_READ_COMMITTED);
session1.begin();

// Se solicita un bloqueo S pero se libera inmediatamente y el
```

```

//valor se copia en la memoria caché transaccional.

Order order = (Order) map1.get("100");

// La entrada se desaloja de la memoria caché transaccional.
map1.invalidate("100", false);

// Una segunda transacción actualiza el mismo pedido.
// Adquiere un bloqueo U, actualiza el valor y lo confirma.
// ObjectGrid adquiere correctamente el bloqueo X durante
// la confirmación puesto que la primera transacción utiliza el aislamiento
// de lectura confirmada.

Map orderMap2 = session2.getMap("Order");
session2.begin();
order2 = (Order) orderMap2.getForUpdate("100");
order2.quantity=2;
orderMap2.update("100", order2);
session2.commit();

// Se vuelve a solicitar el mismo valor. Esta vez, se desea
// actualizar el valor, pero ahora refleja
// el nuevo valor
Order order1Copy (Order) = map1.getForUpdate("100");

```

Lectura no confirmada con bloqueo pesimista

El nivel de aislamiento de la transacción de lectura no confirmada se puede utilizar con eXtreme Scale, que es un nivel que permite las lecturas sucias, las lecturas no repetibles y las lecturas fantasma.

Excepción de colisión optimista

Puede recibir una `OptimisticCollisionException` directamente, o recibirla con una excepción `ObjectGridException`.

El código siguiente es un ejemplo de cómo obtener la excepción y mostrar después su mensaje:

```

try {
...
} catch (ObjectGridException oe) {
    System.out.println(oe);
}

```

Causa de la excepción

La excepción `OptimisticCollisionException` se crea en una situación en la que dos clientes diferentes intentan actualizar la misma entrada de correlación prácticamente al mismo tiempo. Por ejemplo, si un cliente intenta confirmar una sesión y actualizar la entrada de correlación después de que otro cliente hay leído los datos antes de la confirmación, los datos no son correctos. La excepción se crea cuando el otro cliente intenta confirmar los datos incorrectos.

Recuperación de la clave que desencadenó la excepción

Podría resultar de utilizad, al resolver dicha excepción, recuperar la clave que corresponde a la entrada que desencadenó la excepción. La ventaja de la excepción `OptimisticCollisionException` es que contiene el método `getKey`, que devuelve el objeto que representa esa clave. El ejemplo siguiente muestra cómo recuperar e imprimir la clave al obtener `OptimisticCollisionException`:

```

try {
...
} catch (OptimisticCollisionException oce) {
    System.out.println(oce.getKey());
}

```

ObjectGridException ocasiona una excepción OptimisticCollisionException

La excepción OptimisticCollisionException podría ser la causa de que se muestre ObjectGridException. Si es así, puede utilizar el código siguiente para determinar el tipo de excepción e imprimir la clave. El siguiente código utiliza el método del programa de utilidad de findRootCause tal como se describe en la siguiente sección.

```

try {
...
}
catch (ObjectGridException oe) {
    Throwable root = findRootCause( oe );
    if (root instanceof OptimisticCollisionException) {
        OptimisticCollisionException oce = (OptimisticCollisionException)root;
        System.out.println(oce.getKey());
    }
}

```

Técnica de manejo de excepciones general

Conocer la causa raíz de un objeto Throwable es útil para aislar el origen de un error. El ejemplo siguiente muestra cómo un manejador de excepciones utiliza un método de programa de utilidad para buscar la causa raíz de un objeto Throwable.

Ejemplo:

```

static public Throwable findRootCause( Throwable t )
{
    // Iniciar con Throwable que se produjo como raíz.
    Throwable root = t;

    // Seguir cadena de causa hasta encontrar el último objeto Throwable
    // en la cadena.
    Throwable cause = root.getCause();
    while ( cause != null )
    {
        root = cause;
        cause = root.getCause();
    }

    // Devolver el último objeto Throwable en la cadena como causa raíz.
    return root;
}

```

Configuración de clientes con WebSphere eXtreme Scale

Puede configurar un cliente de eXtreme Scale basándose en sus necesidades, como por ejemplo la necesidad de alterar temporalmente valores.

Configuración del cliente con XML

Un archivo XML ObjectGrid se puede utilizar para alterar los valores en el lado del cliente. Para cambiar los valores en un cliente de eXtreme Scale, debe crear un archivo XML de ObjectGrid que sea similar en estructura al archivo que se utilizó para el servidor eXtreme Scale.

Presuponga que el siguiente archivo XML se emparejó con un archivo XML de política de despliegue, y que estos archivos se utilizaron para iniciar un servidor eXtreme Scale.

companyGridServerSide.xml

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

  <objectGrids>
    <objectGrid name="CompanyGrid">
      <bean id="TransactionCallback"
        className="com.company.MyTxCallback" />
      <bean id="ObjectGridEventListener"
        className="com.company.MyOgEventListener" />
      <backingMap name="Customer"
        pluginCollectionRef="customerPlugins" />
      <backingMap name="Item" />
      <backingMap name="OrderLine" numberOfBuckets="1049"
        timeToLive="1600" ttlEvictorType="LAST_ACCESS_TIME" />
      <backingMap name="Order" lockStrategy="PESSIMISTIC"
        pluginCollectionRef="orderPlugins" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="customerPlugins">
      <bean id="Evictor"
        className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
      <bean id="MapEventListener"
        className="com.company.MyMapEventListener" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="orderPlugins">
      <bean id="MapIndexPlugin"
        className="com.company.MyMapIndexPlugin" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

En un servidor eXtreme Scale, la instancia de ObjectGrid denominada CompanyGrid se comporta según se haya definido en el archivo companyGridServerSide.xml. De forma predeterminada, el cliente de CompanyGrid tiene los mismos valores que los de la instancia CompanyGrid que se ejecuta en el servidor. No obstante, algunos de los valores se pueden alterar temporalmente en el cliente, tal como se indica a continuación:

1. Cree una instancia de ObjectGrid específica del cliente.
2. Copie el archivo XML de ObjectGrid que se utilizó para abrir el servidor.
3. Edite el archivo nuevo que se debe personalizar para el lado del cliente.
 - Para establecer o actualizar uno de los atributos en el cliente, especifique un valor nuevo existente cambiar el valor existente.
 - Para eliminar un plug-in del cliente, utilice la serie vacía como el valor para el atributo className.
 - Para cambiar un plug-in existente, especifique un nuevo valor para el atributo className.
 - También puede añadir cualquier plug-in soportado para una alteración temporal de cliente: TRANSACTION_CALLBACK, OBJECTGRID_EVENT_LISTENER, EVICTOR, MAP_EVENT_LISTENER.

4. Cree un cliente con el archivo XML de alteración temporal del cliente recién creado.

El siguiente archivo XML de ObjectGrid se puede utilizar para especificar algunos de los atributos y plug-ins en el cliente de CompanyGrid.

companyGridClientSide.xml

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

  <objectGrids>
    <objectGrid name="CompanyGrid">
      <bean id="TransactionCallback"
        className="com.company.MyClientTxCallback" />
      <bean id="ObjectGridEventListener" className="" />
      <backingMap name="Customer" numberOfBuckets="1429"
        pluginCollectionRef="customerPlugins" />
      <backingMap name="Item" />
      <backingMap name="OrderLine" numberOfBuckets="701"
        timeToLive="800" ttlEvictorType="LAST_ACCESS_TIME" />
      <backingMap name="Order" lockStrategy="PESSIMISTIC"
        pluginCollectionRef="orderPlugins" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="customerPlugins">
      <bean id="Evictor"
        className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
      <bean id="MapEventListener" className="" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="orderPlugins">
      <bean id="MapIndexPlugin"
        className="com.company.MyMapIndexPlugin" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

- La TransactionCallback en el cliente es com.company.MyClientTxCallback en lugar del valor del lado del servidor de com.company.MyTxCallback.
- El cliente no tiene un ObjectGridEventListener porque el valor de className es la serie vacía.
- El cliente define 1429 como numberOfBuckets para la backingMap del cliente, retiene su plug-in Evictor y elimina el plug-in MapEventListener.
- Los atributos numberOfBuckets y timeToLive de la backingMap OrderLine han cambiado
- Aunque se especifique un atributo de lockStrategy diferente, no hay ningún efecto porque el atributo lockStrategy no se soporta para una alteración temporal de cliente.

Para crear el cliente de CompanyGrid utilizando el archivo companyGridClientSide.xml, pase el archivo XML de ObjectGrid como un URL a uno de los métodos connect en ObjectGridManager.

Creación del cliente para XML

```
ObjectGridManager ogManager =
  ObjectGridManagerFactory.ObjectGridManager();
ClientClusterContext clientClusterContext =
  ogManager.connect("MyServer1.company.com:2809", null, new URL(
    "file:xml/companyGridClientSide.xml"));
```

Configuración del cliente mediante programa

También puede alterar temporalmente los valores de ObjectGrid del lado del cliente mediante programa. Cree un objeto ObjectGridConfiguration que sea

similar en estructura a la instancia de ObjectGrid del lado del servidor. El código siguiente crea una instancia de ObjectGrid del lado del cliente funcionalmente equivalente a la alteración temporal del cliente en la sección anterior que utiliza un archivo XML.

alteración temporal del lado del cliente mediante programa

```
ObjectGridConfiguration companyGridConfig = ObjectGridConfigFactory
    .createObjectGridConfiguration("CompanyGrid");
Plugin txCallbackPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.TRANSACTION_CALLBACK, "com.company.MyClientTxCallback");
companyGridConfig.addPlugin(txCallbackPlugin);

Plugin ogEventListenerPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.OBJECTGRID_EVENT_LISTENER, "");
companyGridConfig.addPlugin(ogEventListenerPlugin);

BackingMapConfiguration customerMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("Customer");
customerMapConfig.setNumberOfBuckets(1429);
Plugin evictorPlugin = ObjectGridConfigFactory.createPlugin(PluginType.EVICTOR,
    "com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor");
customerMapConfig.addPlugin(evictorPlugin);

companyGridConfig.addBackingMapConfiguration(customerMapConfig);

BackingMapConfiguration orderLineMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("OrderLine");
orderLineMapConfig.setNumberOfBuckets(701);
orderLineMapConfig.setTimeToLive(800);
orderLineMapConfig.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);

companyGridConfig.addBackingMapConfiguration(orderLineMapConfig);

List ogConfigs = new ArrayList();
ogConfigs.add(companyGridConfig);

Map overrideMap = new HashMap();
overrideMap.put(CatalogServerProperties.DEFAULT_DOMAIN, ogConfigs);

ogManager.setOverrideObjectGridConfigurations(overrideMap);
ClientClusterContext client = ogManager.connect(catalogServerAddresses, null, null);
ObjectGrid companyGrid = ogManager.getObjectGrid(client, objectGridName);
```

La instancia de ogManager de la interfaz ObjectGridManager comprueba si hay alteraciones temporales sólo en los objetos ObjectGridConfiguration y BackingMapConfiguration incluidos en la correlación overrideMap. Por ejemplo, el código anterior altera temporalmente el número de grupos de la correlación OrderLine. No obstante, la correlación Order permanece inalterada en el lado del cliente porque no se incluye ninguna configuración para dicha correlación.

Configuración del cliente en la infraestructura Spring

Los valores de ObjectGrid del lado del cliente también se pueden alterar temporalmente utilizando la infraestructura Spring. El siguiente archivo XML de ejemplo muestra cómo crear un elemento ObjectGridConfiguration y utilizarlo para alterar temporalmente algunos valores del lado del cliente. Este ejemplo llama a las mismas API que se han demostrado en la configuración mediante programa. El ejemplo también es funcionalmente equivalente al ejemplo de la configuración del XML de ObjectGrid.

configuración del cliente con Spring

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
    "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="companyGrid" factory-bean="manager" factory-method="getObjectGrid"
    singleton="true">
    <constructor-arg type="com.ibm.websphere.objectgrid.ClientClusterContext">
      <ref bean="client" />
    </constructor-arg>
    <constructor-arg type="java.lang.String" value="CompanyGrid" />
  </bean>
```

```

<bean id="manager" class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
  factory-method="getObjectGridManager" singleton="true">
  <property name="overrideObjectGridConfigurations">
    <map>
      <entry key="DefaultDomain">
        <list>
          <ref bean="ogConfig" />
        </list>
      </entry>
    </map>
  </property>
</bean>

<bean id="ogConfig"
  class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
  factory-method="createObjectGridConfiguration">
  <constructor-arg type="java.lang.String">
    <value>CompanyGrid</value>
  </constructor-arg>
  <property name="plugins">
    <list>
      <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
        factory-method="createPlugin">
        <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
          value="TRANSACTION_CALLBACK" />
        <constructor-arg type="java.lang.String"
          value="com.company.MyClientTxCallback" />
        </bean>
      <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
        factory-method="createPlugin">
        <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
          value="OBJECTGRID_EVENT_LISTENER" />
        <constructor-arg type="java.lang.String" value="" />
        </bean>
    </list>
  </property>
  <property name="backingMapConfigurations">
    <list>
      <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
        factory-method="createBackingMapConfiguration">
        <constructor-arg type="java.lang.String" value="Customer" />
        <property name="plugins">
          <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
            factory-method="createPlugin">
            <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
              value="EVICTOR" />
            <constructor-arg type="java.lang.String"
              value="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
            </bean>
          </property>
          <property name="numberOfBuckets" value="1429" />
        </bean>
        <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
          factory-method="createBackingMapConfiguration">
          <constructor-arg type="java.lang.String" value="OrderLine" />
          <property name="numberOfBuckets" value="701" />
        </bean>
      <property name="timeToLive" value="800" />
      <property name="ttlEvictorType">
        <value type="com.ibm.websphere.objectgrid.
          TTLType">LAST_ACCESS_TIME</value>
      </property>
    </list>
  </property>
</bean>

  <bean id="client" factory-bean="manager" factory-method="connect"
    singleton="true">
    <constructor-arg type="java.lang.String">
      <value>localhost:2809</value>
    </constructor-arg>
    <constructor-arg
      type="com.ibm.websphere.objectgrid.security.
        config.ClientSecurityConfiguration">
      <null />
    </constructor-arg>
    <constructor-arg type="java.net.URL">

```

```

        <null />
        </constructor-arg>
    </bean>
</beans>

```

Tras crear el archivo XML, cargue el archivo y cree el ObjectGrid con el siguiente fragmento de código.

```

BeanFactory beanFactory = new XmlBeanFactory(new
UrlResource("file:test/companyGridSpring.xml"));

ObjectGrid companyGrid = (ObjectGrid) beanFactory.getBean("companyGrid");

```

Lea sobre la Visión general de integración en la infraestructura de Spring para obtener más información sobre cómo crear un archivo de descriptor XML.

Inhabilitación de la memoria caché cercana del cliente

La memoria caché cercana se habilita de manera predeterminada al configurar el bloqueo como optimista o ninguno. Los clientes no mantienen una memoria caché cercana cuando el valor de bloqueo se configura como pesimista. Para inhabilitar la memoria caché cercana, debe establecer el atributo de numberOfBuckets en 0 en el archivo descriptor de ObjectGrid de alteración temporal del cliente.

Seguimiento de las actualizaciones de correlación de una aplicación

Cuando una aplicación está realizando cambios en una Correlación durante una transacción, un objeto LogSequence rastrea estos cambios. Si la aplicación cambia una entrada en la correlación, un objeto LogElement correspondiente proporciona los detalles del cambio.

Se proporciona a los cargadores un objeto LogSequence para una correlación particular siempre que una aplicación llama a un método para desechar o confirmar la transacción. El cargador se repite en los objetos LogElement dentro del objeto LogSequence y aplica cada objeto LogElement al programa de fondo.

Los receptores de ObjectGridEventListener que se han registrado con un ObjectGrid también utilizan objetos LogSequence. Se proporciona a estos receptores un objeto LogSequence para cada correlación en una transacción confirmada. Las aplicaciones pueden utilizar estos receptores para esperar a que cambien determinadas entradas, como un desencadenante en una base de datos convencional.

Las siguientes interfaces o clases relacionadas con el registro son proporcionadas por la infraestructura de eXtreme Scale:

- com.ibm.websphere.objectgrid.plugins.LogElement
- com.ibm.websphere.objectgrid.plugins.LogSequence
- com.ibm.websphere.objectgrid.plugins.LogSequenceFilter
- com.ibm.websphere.objectgrid.plugins.LogSequenceTransformer

Interfaz LogElement

Un LogElement representa una operación o una entrada durante una transacción. Un objeto LogElement tiene varios métodos para obtener sus distintos atributos.

Los atributos utilizados de forma más habitual son los atributos de valor de tipo y actual capturados por `getType()` y `getCurrentValue()`.

El tipo está representado por una de las constantes definidas en la interfaz `LogElement`: `INSERT`, `UPDATE`, `DELETE`, `EVICT`, `FETCH` o `TOUCH`.

El valor actual representa el nuevo valor para la operación si es `INSERT`, `UPDATE` o `FETCH`. Si la operación es `TOUCH`, `DELETE` o `EVICT`, el valor actual es nulo. Este valor se puede convertir a `ValueProxyInfo` cuando se utiliza `ValueInterface`.

Consulte la documentación de la API si desea más detalles sobre la interfaz `LogElement`.

Interfaz `LogSequence`

En la mayoría de las transacciones, las operaciones que se producen en más de una entrada de la correlación, así pues se crean varios objetos `LogElement`. Debe crear un objeto que se comporte como un compuesto de varios objetos `LogElement`. La interfaz `LogSequence` debe servir a este propósito incluyendo una lista de objetos `LogElement`.

Consulte la documentación de la API si desea más detalles sobre la interfaz `LogSequence`.

Utilización de `LogElement` y `LogSequence`

`LogElement` y `LogSequence` se utilizan ampliamente en eXtreme Scale y por los plug-ins de `ObjectGrid` que se han escrito por usuarios cuando se propagan las operaciones de un componente o servidor a otro componente o servidor. Por ejemplo, un objeto `LogSequence` puede ser utilizado por la función de propagación de transacción de `ObjectGrid` distribuido para propagar los cambios en otros servidores, o el cargador lo puede aplicar en el almacén de persistencia. `LogSequence` es utilizado principalmente por las siguientes interfaces.

- `com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener`
- `com.ibm.websphere.objectgrid.plugins.Loader`
- `com.ibm.websphere.objectgrid.plugins.Evictor`
- `com.ibm.websphere.objectgrid.Session`

Ejemplo de cargador

Esta sección demuestra cómo se utilizan los objetos `LogSequence` y `LogElement` en un cargador. Un cargador se utiliza para cargar datos y persistirlos en un almacén persistente. El método `batchUpdate` de la interfaz `Loader` utiliza el objeto `LogSequence`:

```
void batchUpdate(Txid txid, LogSequence sequence) throws  
    LoaderException, OptimisticCollisionException;
```

Se llama al método `batchUpdate` cuando un `ObjectGrid` debe aplicar todos los cambios actuales a `Loader`. Se proporciona a `Loader` una lista de objetos `LogElement` para la correlación, encapsulados en un objeto `LogSequence`. La implementación del método `batchUpdate` debe repetir los cambios y aplicarlos en el programa de fondo. El siguiente fragmento de código demuestra cómo `Loader` utiliza un objeto `LogSequence`. El fragmento de código se repite en el conjunto de cambios y genera tres sentencias de proceso por lotes JDBC (Java DataBase Connectivity): inserts, updates y deletes:

```

public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException {
// Obtener una conexión SQL que vaya a utilizarse.
    Connection conn = getConnection(tx);
    try
    {
        // Procesar la lista de cambios y crear un conjunto de
        // sentencias preparadas para ejecutar una
        // operación SQL. Las sentencias se almacenan en la memoria caché
        // en stmtCache.
        Iterator iter = sequence.getPendingChanges();
        while(iter.hasNext())
        {
            LogElement logElement = (LogElement) iter.next();
            Object key = logElement.getCacheEntry().getKey();
            Object value = logElement.getCurrentValue();
            switch ( logElement.getType().getCode() )
            {
                case LogElement.CODE_INSERT:
                    buildBatchSQLInsert( key, value, conn );
                    break;
                case LogElement.CODE_UPDATE:
                    buildBatchSQLUpdate( key, value, conn );
                    break;
                case LogElement.CODE_DELETE:
                    buildBatchSQLDelete( key, conn );
                    break;
            }
        }
        // Ejecute las sentencias de proceso por lotes que se crearon mediante
        // el bucle anterior.
        Collection statements = getPreparedStatementCollection(tx, conn);
        iter = statements.iterator();
        while(iter.hasNext())
        {
            PreparedStatement pstmt = (PreparedStatement) iter.next();
            pstmt.executeBatch();
        }
    } catch (SQLException e) {
        LoaderException ex = new LoaderException(e);
        throw ex;
    }
}

```

El ejemplo anterior ilustra la lógica de alto nivel de proceso del argumento `LogSequence`. Sin embargo, el ejemplo no ilustra los detalles sobre cómo se crean una sentencia SQL insert, update o delete. Se llama al método `getPendingChanges` en el argumento `LogSequence` para obtener un iterador de objetos `LogElement` que un Loader debe procesar y se utiliza el método `LogElement.getType().getCode()` para determinar si un `LogElement` es para una operación SQL insert, update o delete.

Ejemplo de desalojador

También puede utilizar los objetos `LogSequence` y `LogElement` con un Evictor. Un Evictor se utiliza para desalojar las entradas de correlación de la correlación de respaldo basándose en determinados criterios. El método `apply` de la interfaz `Evictor` utiliza `LogSequence`.

```

/**
 * Se llama a éste durante la confirmación de la memoria caché para permitir
 * al desalojador rastrear el uso de objetos en una correlación de respaldo.
 * También se informará sobre las entradas que se hayan desalojado
 * correctamente.

```

```
*  
* @param sequence LogSequence of changes to the map  
*/  
void apply(LogSequence sequence);
```

Interfaces LogSequenceFilter y LogSequenceTransformer

A veces, es necesario filtrar los objetos LogElement de forma que sólo se aceptan los objetos LogElement con determinados criterios y se rechazan los otros objetos. Por ejemplo, es posible que desee serializar un LogElement determinado basándose en algún criterio.

LogSequenceFilter resuelve este problema con el siguiente método.

```
public boolean accept (LogElement logElement);
```

Este método devuelve el valor true si el LogElement determinado se debe utilizar en la operación, y devuelve el valor false si no se debe utilizar el LogElement proporcionado.

LogSequenceTransformer es una clase que utiliza la función LogSequenceFilter. Utiliza el LogSequenceFilter para filtrar algunos objetos LogElement y, a continuación, serializa los objetos LogElement aceptados. Esta clase tiene dos métodos. El primer método es el siguiente.

```
public static void serialize(Collection logSequences, ObjectOutputStream stream,  
    LogSequenceFilter filter, DistributionMode mode) throws IOException
```

Este método permite al interlocutor proporcionar un filtro para determinar qué LogElements incluir en el proceso de serialización. El parámetro DistributionMode permite al interlocutor controlar el proceso de serialización. Por ejemplo, si la modalidad de distribución sólo es la invalidación, no es necesario serializar el valor. El segundo método de esta clase es el método inflate, del modo siguiente.

```
public static Collection inflate(ObjectInputStream stream, ObjectGrid  
    objectGrid) throws IOException, ClassNotFoundException
```

El método inflate lee el formulario serializado de la secuencia de registro, que se ha sido creado por el método serialize, desde la corriente de datos de entrada de objeto proporcionada.

Habilitación de la réplica en el cliente

Puede habilitar también la réplica de correlaciones en el cliente para conseguir una disponibilidad de datos más rápida.

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.

Ejemplo de la API de DataGrid

Las API de DataGrid admiten dos patrones de programación de cuadrícula comunes: correlación paralela y reducción paralela.

Correlación paralela

La correlación paralela permite que las entradas de un conjunto de claves se procesen y devuelve un resultado para cada entrada procesada. La aplicación efectúa un listado de claves y recibe una correlación de pares de clave/resultado después de invocar la operación de correlación. El resultado es la consecuencia de aplicar una función a la entrada de cada clave. La función la suministra la aplicación.

Flujo de llamadas MapGridAgent

Cuando se invoca el método `AgentManager.callMapAgent` con una colección de claves, la instancia `MapGridAgent` se serializa y se envía a cada partición primaria en la que se resuelven las claves. Esto significa que los datos de la instancia almacenados en el agente pueden enviarse al servidor. Por consiguiente, cada partición primaria tiene una instancia del agente. El método de proceso se invoca para cada instancia una vez por cada clave que se resuelve en la partición. El resultado de cada método de proceso se vuelve a serializar en el cliente y se devuelve al llamante en una instancia de correlación, donde el resultado se representa como el valor en la correlación.

Cuando se invoca el método `AgentManager.callMapAgent` sin una colección de claves, la instancia `MapGridAgent` se serializa y se envía a todas las particiones primarias. Esto significa que los datos de la instancia almacenados en el agente pueden enviarse al servidor. Por consiguiente, cada partición primaria tiene una instancia (partición) del agente. El método `processAllEntries` se invoca para cada partición. El resultado de cada método `processAllEntries` se vuelve a serializar en el cliente y se devuelve al llamante en una instancia de correlación. En el siguiente ejemplo se presupone que hay una entidad `Person` con la forma siguiente:

```
import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;
@Entity
public class Person
{
    @Id String ssn;
    String firstName;
    String surname;
    int age;
}
```

La función proporcionada por la aplicación está escrita como una clase que implementa la interfaz `MapAgentGrid`. A continuación se ofrece un agente de ejemplo que muestra una función que devuelve la edad de una persona (`Person`) multiplicada por dos.

```
public class DoublePersonAgeAgent implements MapGridAgent, EntityAgentMixin
{
    private static final long serialVersionUID = -2006093916067992974L;

    int lowAge;
    int highAge;
```

```

public Object process(Session s, ObjectMap map, Object key)
{
    Person p = (Person)key;
    return new Integer(p.age * 2);
}

public Map processAllEntries(Session s, ObjectMap map)
{
    EntityManager em = s.getEntityManager();
    Query q = em.createQuery("select p from Person p where p.age > ?1 and p.age < ?2");
    q.setParameter(1, lowAge);
    q.setParameter(2, highAge);
    Iterator iter = q.getResultIterator();
    Map<Person, Integer> rc = new HashMap<Person, Integer>();
    while(iter.hasNext())
    {
        Person p = (Person)iter.next();
        rc.put(p, (Integer)process(s, map, p));
    }
    return rc;
}

public Class getClassForEntity()
{
    return Person.class;
}
}

```

Este ejemplo muestra el agente de correlación con la edad doblada de una persona. Observemos primero los métodos de proceso. El primer método de proceso proporciona la persona con la que trabajar y devuelve el doble de la edad de esa entrada. El segundo método de proceso se llama para cada partición y se buscan todos los objetos Person con una edad comprendida entre un valor lowAge y un valor highAge, y se devuelve el doble de las edades.

```

Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

DoublePersonAgeAgent agent = new DoublePersonAgeAgent();

// efectúa una lista de las claves
ArrayList<Person> keyList = new ArrayList<Person>();
Person p = new Person();
p.ssn = "1";
keyList.add(p);
p = new Person ();
p.ssn = "2";
keyList.add(p);

// obtiene los resultados de las entradas
Map<Tuple, Object> = amgr.callMapAgent(agent, keyList);

```

Este ejemplo muestra un cliente que obtiene un elemento Session y una referencia a la correlación de persona. La operación del agente se realiza en una correlación específica. La interfaz AgentManager se recupera de dicha correlación. Se crea una instancia del agente que se va a invocar y se añade cualquier estado necesario al objeto mediante el establecimiento de atributos (no hay ninguno, en este caso). Se crea una lista de las claves. Se devuelve una correlación con los valores para la persona 1 doblados y los mismos valores para la persona 2.

Se invoca el agente para ese conjunto de claves. El método de proceso del agente se invoca en cada partición con algunas de las claves especificadas en la cuadrícula en paralelo. Se devuelve una correlación con los resultados fusionados de la clave especificada. En este caso, se devuelve una correlación con los valores que contienen la edad doblada de la persona 1 y lo mismo para la persona 2.

Aunque la clave no exista, se invoca el agente. De esta manera, el agente tiene la oportunidad de crear la entrada de correlación. Si se usa EntityAgentMixin, la clave que se procesará no será la entidad, sino el valor clave del tuple real de la

entidad. Si las claves no se conocen, es posible solicitar a todas las particiones que busquen los objetos Person de una forma determinada y que devuelvan el doble de la edad. Observe el ejemplo siguiente:

```
Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

DoublePersonAgeAgent agent = new DoublePersonAgeAgent();
agent.lowAge = 20;
agent.highAge = 9999;

Map m = amgr.callMapAgent(agent);
```

El ejemplo anterior muestra el AgentManager que se obtiene para la correlación Person (persona) y el agente construido e inicializado con las edades bajas y altas para las personas de interés. A continuación se invoca el agente mediante el método callMapAgent. Observe que no se proporciona ninguna clave. Esto hace que ObjectGrid invoque el agente en todas las particiones de la cuadrícula en paralelo y que devuelva los resultados fusionados al cliente. De esta manera, se buscarán todos los objetos Person en la cuadrícula con una edad comprendida entre los valores bajos y altos y se calculará el doble de la edad de los objetos Person. Esto muestra cómo las API de la cuadrícula pueden utilizarse para ejecutar una consulta que busque entidades que coincidan con una consulta determinada. ObjectGrid serializa y transporta el agente a las particiones con las entradas necesarias. Los resultados se serializan de forma similar y se transportan al cliente. Las API de correlación deben tratarse con cuidado. Si ObjectGrid contuviera terabytes de objetos y se ejecutara en muchos servidores, posiblemente se saturarían los sistemas más grandes que se ejecutasen en el cliente. Utilice esta API para procesar un subconjunto pequeño. Si necesita procesar un subconjunto de gran tamaño, se recomienda usar un agente de reducción para realizar el proceso en la cuadrícula en lugar de en el cliente.

Reducción paralela o agentes de agregación

Este estilo de programación procesa un subconjunto de entradas y calcula un resultado único para el grupo de entradas. Ejemplos de dicho resultado son:

- Valor mínimo
- Valor máximo
- Alguna otra función específica de empresa

Un agente de reducción está codificado y se invoca de una manera similar a la de los agentes de correlación.

Flujo de llamadas ReduceGridAgent

Cuando se invoca el método AgentManager.callReduceAgent con una colección de claves, la instancia ReduceGridAgent se serializa y se envía a cada partición primaria en la que se resuelven las claves. Esto significa que los datos de la instancia almacenados en el agente pueden enviarse al servidor. Por consiguiente, cada partición primaria tiene una instancia del agente. El método reduce(Session s, ObjectMap map, Collection keys) se invoca una vez por instancia (partición) con el subconjunto de claves que se resuelve en la partición. El resultado de cada método de reducción se vuelve a serializar en el cliente. El método reduceResults se invoca en la instancia ReduceGridAgent del cliente con la colección de cada resultado de cada invocación de reducción remota. El resultado del método reduceResults se devuelve al llamante del método callReduceAgent.

Cuando se invoca el método `AgentManager.callReduceAgent` sin una colección de claves, la instancia `ReduceGridAgent` se serializa y se envía a todas las particiones primarias. Esto significa que los datos de la instancia almacenados en el agente pueden enviarse al servidor. Por consiguiente, cada partición primaria tiene una instancia del agente. El método `reduce(Session s, ObjectMap map)` se invoca una vez por instancia (partición). El resultado de cada método de reducción se vuelve a serializar en el cliente. El método `reduceResults` se invoca en la instancia `ReduceGridAgent` del cliente con la colección de cada resultado de cada invocación de reducción remota. El resultado del método `reduceResults` se devuelve al llamante del método `callReduceAgent`. A continuación se muestra un ejemplo de un agente de reducción que simplemente añade las edades de las entradas coincidentes.

```
public class SumAgeReduceAgent implements ReduceGridAgent, EntityAgentMixin
{
    private static final long serialVersionUID = 2521080771723284899L;

    int lowAge;
    int highAge;

    public Object reduce(Session s, ObjectMap map, Collection keyList)
    {
        Iterator<Person> iter = keyList.iterator();
        int sum = 0;
        while(iter.hasNext())
        {
            Person p = iter.next();
            sum += p.age;
        }
        return new Integer(sum);
    }

    public Object reduce(Session s, ObjectMap map)
    {
        EntityManager em = s.getEntityManager ();
        Query q = em.createQuery("select p from Person p where p.age > ?1 and p.age < ?2");
        q.setParameter(1, lowAge);
        q.setParameter(2, highAge);
        Iterator<Person> iter = q.getResultIterator();
        int sum = 0;
        while(iter.hasNext())
        {
            sum += iter.next().age;
        }
        return new Integer(sum);
    }

    public Class getClassForEntity()
    {
        return Person.class;
    }
}
```

El ejemplo anterior muestra el agente. El agente tiene tres partes importantes. La primera permite que un conjunto específico de entradas se procesen sin una consulta. Itera sobre el conjunto de entradas al añadir las edades. La suma se devuelve desde el método. La segunda parte utiliza una consulta para seleccionar las entradas que se agregarán. A continuación, se suman todas las edades de `Person` coincidentes. El tercer método se utiliza para agregar los resultados de cada partición a un único resultado. `ObjectGrid` realiza la agregación de entradas en paralelo en la cuadrícula. Cada partición produce un resultado intermedio que debe agregarse con otros resultados intermedios de particiones. Este tercer método realiza dicha tarea. En el ejemplo siguiente, se invoca el agente, y se agregan las edades de todas las personas comprendidas entre 10 y 20 exclusivamente:

```
Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

SumAgeReduceAgent agent = new SumAgeReduceAgent();

Person p = new Person();
p.ssn = "1";
ArrayList<Person> list = new ArrayList<Person>();
```

```
list.add(p);
p = new Person ();
p.ssn = "2";
list.add(p);
Integer v = (Integer)amgr.callReduceAgent(agent, list);
```

Funciones del agente

El agente puede llevar a cabo cualquier operación de ObjectMap o EntityManager dentro del fragmento local donde se ejecuta. El agente recibe un objeto Session y puede añadir, actualizar, consultar, leer o eliminar datos de la partición que representa el objeto Session. Algunas aplicaciones sólo consultarán los datos de la cuadrícula, pero también podrá escribir un agente para aumentar en 1 todas las edades de las entidades Person que coincidan con una consulta determinada. Existe una transacción en el objeto Session cuando se llama al agente, y se confirma cuando se devuelve el agente, a menos que se lance una excepción.

Manejo de errores

Si se invoca un agente de correlación con una clave desconocida, el valor devuelto es un objeto de error que implementa la interfaz EntryErrorValue.

Transacciones

Un agente de correlación se ejecuta en una transacción independiente del cliente. Las invocaciones de agente se pueden agrupar en una única transacción. Si se produce una anomalía en un agente (emite una excepción), la transacción se retrotrae. Cualquier agente que se haya ejecutado correctamente en una transacción se retrotraerá con el agente anómalo. AgentManager volverá a ejecutar los agentes retrotraídos que se ejecutaron correctamente en una nueva transacción.

Si desea más información, consulte la documentación de la API DataGrid.

Capítulo 4. Acceso a los datos con el servicio de datos REST

Desarrolle las aplicaciones que realizan operaciones con los protocolos del servicio de datos REST.

Operaciones con el servicio de datos REST

Después de iniciar el servicio de datos REST de eXtreme Scale, puede utilizar cualquier cliente HTTP para interactuar con él. Se puede utilizar un navegador web, un cliente PHP, un cliente Java o un cliente de WCF Data Services para emitir cualquiera de las operaciones de solicitud soportadas.

El servicio REST implementa un subconjunto de la especificación Microsoft Atom Publishing Protocol: Data Services URI and Payload Extensions, Versión 1.0, que forma parte del protocolo OData. Este capítulo describe qué características de la especificación se soportan y cómo se correlacionan con eXtreme Scale.

URI de la raíz de servicio

Microsoft WCF Data Services define normalmente un servicio por origen de datos o modelo de entidad. El servicio de datos REST de eXtreme Scale define un servicio por cada ObjectGrid definida. Cada ObjectGrid definida en el archivo XML de sustitución de cliente ObjectGrid de eXtreme Scale se expone automáticamente como raíz del servicio REST independiente.

El URI de la raíz de servicio es:

```
http://host:puerto/raízcontexto/restservice/nombrecuadrícula
```

Donde:

- *raízcontexto* se define al desplegar la aplicación de servicio de datos REST y depende del servidor de aplicaciones
- *nombrecuadrícula* es el nombre del ObjectGrid

Tipos de solicitud

La lista siguiente describe los tipos de solicitud de Microsoft WCF Data Services que el servicio de datos REST de eXtreme Scale soporta. Para obtener información detallada sobre cada tipo de solicitud que WCF Data Services soporta, consulte MSDN: Request Types

Tipos de solicitudes de inserción

Los clientes pueden insertar recursos utilizando el verbo HTTP POST con las limitaciones siguientes:

- Solicitud InsertEntity: Soportada.
- Solicitud InsertLink: Soportada.
- Solicitud InsertMediaResource: No soportada debido a la restricción de soporte de recursos de medios.

Para obtener información adicional, consulte: MSDN: Insert Request Types.

Tipos de solicitudes de actualización

Los clientes pueden actualizar recursos utilizando los verbos HTTP PUT y MERGE con las limitaciones siguientes:

- Solicitud UpdateEntity: Soportada.
- Solicitud UpdateComplexType: No soportada debido a la restricción de tipo complejo.
- Solicitud UpdatePrimitiveProperty: Soportada.
- Solicitud UpdateValue: Soportada.
- Solicitud UpdateLink: Soportada.
- Solicitud UpdateMediaResource: No soportada debido a la restricción de soporte de recursos de medios.

Para obtener información adicional, consulte: MSDN: Insert Request types.

Tipos de solicitudes de supresión

Los clientes pueden suprimir recursos utilizando el verbo HTTP DELETE con las limitaciones siguientes:

- Solicitud DeleteEntity: Soportada.
- Solicitud DeleteLink: Soportada.
- Solicitud DeleteValue: Soportada.

Para obtener información adicional, consulte: MSDN: Delete Request Types.

Tipos de solicitudes de recuperación

Los clientes pueden recuperar recursos utilizando el verbo HTTP GET con las limitaciones siguientes:

- Solicitud RetrieveEntitySet: Soportada.
- Solicitud RetrieveEntity: Soportada.
- Solicitud RetrieveComplexType: No soportada debido a la restricción de tipo complejo.
- Solicitud RetrievePrimitiveProperty: Soportada.
- Solicitud RetrieveValue: Soportada.
- Solicitud RetrieveServiceMetadata: Soportada.
- Solicitud RetrieveServiceDocument: Soportada.
- Solicitud RetrieveLink: Soportada.
- Solicitud Retrieve que contiene una correlación de canal de información personalizable: No soportada
- RetrieveMediaResource: No soportada debido a la restricción de recursos de medios.

Para obtener información adicional, consulte: MSDN: Retrieve Request Types.

Opciones de consulta del sistema

Hay consultas soportadas que permiten que los clientes identifiquen una colección de entidades o una sola entidad. Las opciones de consulta del sistema se especifican en un URI de servicio de datos y están soportadas con las limitaciones siguientes:

- \$expand: Soportada.
- \$filter: Soportada.
- \$orderby: Soportada.
- \$format: No soportada. El formato aceptable se identifica en la cabecera de solicitud HTTP Accept.

- \$skip: Soportada.
- \$top: Soportada.

Para obtener información adicional, consulte: MSDN: System Query Options.

Direccionamiento de particiones

El direccionamiento de particiones se basa en la entidad raíz. Un URI de solicitud deduce una entidad raíz si su vía de acceso de recurso empieza con una entidad raíz o con una entidad que tenga una asociación directa o indirecta con la entidad. En un entorno particionado, cualquier solicitud que no pueda deducir una entidad raíz se rechazará. Cualquier solicitud que deduzca una entidad raíz se direccionará a la partición correcta.

Para obtener información adicional sobre la definición de un esquema con asociaciones y entidades raíz, consulte los apartados Modelo de datos escalable de eXtreme Scale y Particionamiento.

Solicitud de invocación

Las solicitudes de invocación no están soportadas. Para obtener información adicional, consulte: MSDN: Invoke RequestTypes.

Solicitud por lotes

Los clientes pueden realizar por lotes varios conjuntos de cambios u operaciones de consulta en una sola solicitud. Esto puede reducir el número de transmisiones de ida y vuelta al servidor y permite la participación de varias solicitudes en una sola transacción. Para obtener información adicional, consulte: MSDN: Batch Request.

Solicitudes a través de túnel

Las solicitudes a través de túnel no están soportadas. Para obtener información adicional, consulte: MSDN: Tunneled Requests.

Protocolos de solicitud para el servicio de datos REST

En general, los protocolos para interactuar con los servicios REST son los mismos que se describen en el protocolo WCF Data Services AtomPub. No obstante, eXtreme Scale proporciona detalles adicionales, de la perspectiva de modelo de entidad de eXtreme Scale. Se espera que los usuarios estén familiarizados con los protocolos de WCF Data Services antes de leer esta sección. Como alternativa, los usuarios pueden leer esta sección con la sección del protocolo WCF Data Services.

Se proporcionan ejemplos para ilustrar la solicitud y la respuesta. Estos ejemplos se aplican tanto al servicio de datos REST de eXtreme Scale como a WCF Data Services. Puesto que los navegadores web sólo pueden recuperar datos, las operaciones CRUD (crear, actualizar y suprimir) debe realizarlas otro cliente como, por ejemplo, Java, JavaScript™, RUBY o PHP.

Solicitudes de recuperación con el servicio de datos REST

Un cliente utiliza una solicitud RetrieveEntity para recuperar una entidad de eXtreme Scale. La carga útil de respuesta contiene los datos de la entidad en formato AtomPub o JSON. Además, se puede utilizar el operador del sistema \$expand para expandir las relaciones. Las relaciones se representan en la línea

dentro de la respuesta del servicio de datos como un documento de canal de información Atom (relación a muchos) o un documento de entrada Atom (relación a uno).

Consejo: Para obtener información detallada sobre el protocolo RetrieveEntity definido en WCF Data Services, consulte: MSDN: RetrieveEntity Request.

Recuperación de una entidad

El ejemplo siguiente de RetrieveEntity recupera una entidad Customer con clave.

AtomPub

- Método

GET

- URI de la solicitud:

http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Customer('ACME')

- Cabecera de la solicitud:

Accept: application/atom+xml

- Carga útil de la solicitud:

Ninguna

- Cabecera de la respuesta:

Content-Type: application/atom+xml

- Carga útil de la respuesta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/
restservice" xmlns:d= "http://schemas.microsoft.com/ado/2007/
08/dataservices" xmlns:m = "http://schemas.microsoft.com/ado/2007/
08/dataservices/metadata" xmlns = "http://www.w3.org/2005/Atom">

<category term = "NorthwindGridModel.Customer" scheme = "http://
schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
<id>http://localhost:8080/wxsrestservice/restservice/
NorthwindGrid/Customer('ACME')</id>
<title type = "text"/>
<updated>2009-12-16T19:52:10.593Z</updated>
<author>
  <name/>
</author>
<link rel = "edit" title = "Customer" href = "Customer('ACME')"/>
<link rel = "http://schemas.microsoft.com/ado/2007/08/
dataservices/related/
orders" type = "application/atom+xml;type=feed" title =
"orders" href = "Customer('ACME')/orders"/>
<content type = "application/xml">
  <m:properties>
    <d:customerId>ACME</d:customerId>
    <d:city m:null = "true"/>
    <d:companyName>RoaderRunner</d:companyName>
    <d:contactName>ACME</d:contactName>
    <d:country m:null = "true"/>
    <d:version m:type = "Edm.Int32">3</d:version>
  </m:properties>
</content>
</entry>
```

- Código de respuesta:

200 OK

JSON

- Método
GET
- URI de la solicitud:
`http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/`
`Customer('ACME')`
- Cabecera de la solicitud:
Accept: application/json
- Carga útil de la solicitud:
Ninguna
- Cabecera de la respuesta:
Content-Type: application/json
- Carga útil de la respuesta:

```
{ "d": { "__metadata": { "uri": "http://localhost:8080/wxsrestservice/
restservice/NorthwindGrid/Customer('ACME')",
"type": "NorthwindGridModel.Customer" },
"customerId": "ACME",
"city": null,
"companyName": "RoaderRunner",
"contactName": "ACME",
"country": null,
"version": 3,
"orders": { "__deferred": { "uri": "http://localhost:8080/wxsrestservice/restservice/
NorthwindGrid/Customer('ACME')/orders" } } }
```
- Código de respuesta:
200 OK

Consultas

También se puede utilizar una consulta con una solicitud `RetrieveEntitySet` o `RetrieveEntity`. Una consulta se especifica mediante el operador `$filter` del sistema.

Para obtener información detallada sobre el operador `$filter`, consulte: MSDN: [Filter System Query Option \(\\$filter\)](#)

El protocolo OData soporta varias expresiones comunes. El servicio de datos REST de eXtreme Scale soporta un subconjunto de las expresiones definidas en la especificación:

- Expresiones booleanas:
 - eq, ne, lt, le, gt, ge
 - negate
 - not
 - parenthesis
 - and, or
- Expresiones aritméticas:
 - add
 - sub
 - mul
 - div
- Literales primitivos
 - String

- date-time
- decimal
- single
- double
- int16
- int32
- int64
- binary
- null
- byte

Las expresiones siguientes *no* están disponibles:

- Expresiones booleanas:
 - isof
 - cast
- Expresiones de llamada de método
- Expresiones aritméticas:
 - mod
- Literales primitivos:
 - Guid
- Expresiones de miembro

Para obtener una lista completa y una descripción de las expresiones que están disponibles en Microsoft WCF Data Services, consulte la sección 2.2.3.6.1.1 :Common Expression Syntax.

El ejemplo siguiente ofrece una demostración de una solicitud RetrieveEntity con una consulta. En este ejemplo, se recuperan todos los clientes cuyo nombre de contacto es "RoadRunner". El único cliente que coincide con este filtro es Customer('ACME'), tal como se muestra en la carga útil de la respuesta.

Restricción: Esta consulta sólo funcionará para entidades no particionadas. Si la entidad Customer está particionada, se necesitará la clave perteneciente al cliente.

AtomPub

- Método: GET
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?$filter=contactName eq 'RoadRunner'`
- Cabecera de la solicitud: `Accept: application/atom+xml`
- Carga útil de entrada: Ninguna
- Cabecera de la respuesta: `Content-Type: application/atom+xml`
- Carga útil de la respuesta:


```
<?xml version="1.0" encoding="iso-8859-1"?>
<feed
  xml:base="http://localhost:8080/wxsrestservice/restservice"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Customer</title>
  <id>http://localhost:8080/wxsrestservice/restservice/
    NorthwindGrid/Customer </id>
```

```

<updated>2009-09-16T04:59:28.656Z</updated>
<link rel="self" title="Customer" href="Customer" />
<entry>
  <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <id>
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/ Customer('ACME')</id>
  <title type="text" />
  <updated>2009-09-16T04:59:28.656Z</updated>
  <author>
  <name />
</author>
  <link rel="edit" title="Customer" href="Customer('ACME')" />
  <link
  rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/orders"
  type="application/atom+xml;type=feed" title="orders"
  href="Customer('ACME')/orders" />
  <content type="application/xml">
    <m:properties>
      <d:customerId>ACME</d:customerId>
      <d:city m:null = "true"/>
      <d:companyName>RoadRunner</d:companyName>
      <d:contactName>ACME</d:contactName>
      <d:country m:null = "true"/>
      <d:version m:type = "Edm.Int32">3</d:version>
    </m:properties>
  </content>
</entry>
</feed>

```

- Código de respuesta: 200 OK

JSON

- Método: GET
- URI de la solicitud:


```
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Customer?$filter=contactName eq 'RoadRunner'
```
- Cabecera de la solicitud: Accept: application/json
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/json
- Carga útil de la respuesta:


```
{
  "d": [ {
    "__metadata": {
      "uri": "http://localhost:8080/wxsrestservice/
restservice/NorthwindGrid/Customer('ACME')",
      "type": "NorthwindGridModel.Customer"
    },
    "customerId": "ACME",
    "city": null,
    "companyName": "RoadRunner",
    "contactName": "ACME",
    "country": null,
    "version": 3,
    "orders": {
      "__deferred": {
        "uri": "http://localhost:8080/
wxsrestservice/restservice/NorthwindGrid/
Customer('ACME')/orders"
      }
    }
  } ]
}
```
- Código de respuesta: 200 OK

Operador del sistema \$expand

El operador del sistema \$expand se puede utilizar para expandir asociaciones. Las asociaciones se representan en línea en la respuesta del servicio de datos. Las asociaciones con varios valores (a muchos) se representan como un documento de

canal de información Atom o una matriz JSON. Las asociaciones con un solo valor (a uno) se representan como un documento de entrada Atom o un objeto JSON.

Para obtener información detallada sobre el operador del sistema \$expand, consulte Expand System Query Option (\$expand) (Opción de consulta del sistema expand (\$expand)).

A continuación se ofrece un ejemplo sobre cómo utilizar al operador del sistema \$expand. En este ejemplo, recuperamos la entidad Customer('IBM'), que tiene asociados los pedidos 5000, 5001 y otros. La cláusula de \$expand se define como "orders", de modo que la colección de pedidos se expanda como en línea en la carga útil de la respuesta. Aquí sólo se muestran los pedidos 5000 y 5001.

AtomPub

- Método: GET
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders`
- Cabecera de la solicitud: `Accept: application/atom+xml`
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: `Content-Type: application/atom+xml`
- Carga útil de la respuesta:

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/restservice"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m = "http://schemas.microsoft.com/ado/2007/08/dataservices/
  metadata" xmlns = "http://www.w3.org/2005/Atom">
<category term = "NorthwindGridModel.Customer" scheme = "http://schemas.
microsoft.com/ado/2007/08/dataservices/scheme"/>
  <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Customer('IBM')</id>
  <title type = "text"/>
  <updated>2009-12-16T22:50:18.156Z</updated>
  <author>
    <name/>
  </author><link rel = "edit" title = "Customer" href = "Customer('IBM')"/>
<link rel = "http://schemas.microsoft.com/ado/2007/08/dataservices/
related/orders" type = "application/atom+xml;type=feed" title =
"orders" href = "Customer('IBM')/orders">
  <m:inline>
    <feed>
      <title type = "text">orders</title>
      <id>http://localhost:8080/wxsrestservice/restservice/
NorthwindGrid/Customer('IBM')/orders</id>
      <updated>2009-12-16T22:50:18.156Z</updated>
      <link rel = "self" title = "orders" href = "Customer('IBM')/orders"/>
    </feed>
  </m:inline>
</entry>
  <category term = "NorthwindGridModel.Order" scheme =
"http://schemas.microsoft.com/ado/2007/08/
dataservices/scheme"/>
  <id>http://localhost:8080/wxsrestservice/restservice/
NorthwindGrid/Order(orderId=5000,customer_customerId=
'IBM')</id>
  <title type = "text"/>
  <updated>2009-12-16T22:50:18.156Z</updated>
  <author>
    <name/>
  </author>
  <link rel = "edit" title = "Order" href =
"Order(orderId=5000,customer_customerId='IBM')"/>
  <link rel = "http://schemas.microsoft.com/ado/2007/08/
```

```

dataservices/related/customer" type = "application/
atom+xml;type=entry" title = "customer" href =
"Order(orderId=5000,customer_customerId='IBM')/customer"/>
  <link rel = "http://schemas.microsoft.com/ado/2007/08/
dataservices/related/orderDetails" type = "application/
atom+xml;type=feed" title = "orderDetails" href =
"Order(orderId=5000,customer_customerId='IBM')/orderDetails"/>
  <content type = "application/xml">
    <m:properties>
      <d:orderId m:type = "Edm.Int32">5000</d:orderId>
      <d:customer_customerId>IBM</d:customer_customerId>
      <d:orderDate m:type = "Edm.DateTime">
2009-12-16T19:46:29.562</d:orderDate>
      <d:shipCity>Rochester</d:shipCity>
      <d:shipCountry m:null = "true"/>
      <d:version m:type = "Edm.Int32">0</d:version>
    </m:properties>
  </content>
</entry>
<entry>
  <category term = "NorthwindGridModel.Order" scheme =
"http://schemas.microsoft.com/ado/2007/08/
dataservices/scheme"/>
  <id>http://localhost:8080/wxsrestservice/restservice/
NorthwindGrid/Order(orderId=5001,customer_customerId=
'IBM')</id>
  <title type = "text"/>
  <updated>2009-12-16T22:50:18.156Z</updated>
  <author>
    <name/></author>
  <link rel = "edit" title = "Order" href = "Order(
orderId=5001,customer_customerId='IBM')"/>
  <link rel = "http://schemas.microsoft.com/ado/2007/
08/dataservices/related/customer" type =
"application/atom+xml;type=entry" title =
"customer" href = "Order(orderId=5001,customer_customerId=
'IBM')/customer"/>
  <link rel = "http://schemas.microsoft.com/ado/2007/08/
dataservices/related/orderDetails" type = "application/atom+xml;type=feed" title =
"orderDetails" href = "Order(orderId=5001,customer_customerId='IBM')/orderDetails"/>
  <content type = "application/xml">
    <m:properties>
      <d:orderId m:type = "Edm.Int32">5001</d:orderId>
      <d:customer_customerId>IBM</d:customer_customerId>
      <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:
50:11.125</d:orderDate>
      <d:shipCity>Rochester</d:shipCity>
      <d:shipCountry m:null = "true"/>
      <d:version m:type = "Edm.Int32">0</d:version>
    </m:properties>
  </content>
</entry>
</feed>
</m:inline>
</link>
<content type = "application/xml">
  <m:properties>
    <d:customerId>IBM</d:customerId>
    <d:city m:null = "true"/>
    <d:companyName>IBM Corporation</d:companyName>
    <d:contactName>John Doe</d:contactName>
    <d:country m:null = "true"/>
    <d:version m:type = "Edm.Int32">4</d:version>
  </m:properties>
</content>
</entry>

```

- Código de respuesta: 200 OK

JSON

- Método: GET
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders`
- Cabecera de la solicitud: `Accept: application/json`
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: `Content-Type: application/json`
- Carga útil de la respuesta:

```
{ "d": { "__metadata": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')", "type": "NorthwindGridModel.Customer" }, "customerId": "IBM", "city": null, "companyName": "IBM Corporation", "contactName": "John Doe", "country": null, "version": 4, "orders": [ { "__metadata": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')", "type": "NorthwindGridModel.Order" }, "orderId": 5000, "customer_customerId": "IBM", "orderDate": "\\Date(1260992789562)\\", "shipCity": "Rochester", "shipCountry": null, "version": 0, "customer": { "__deferred": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/customer" } }, "orderDetails": { "__deferred": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/orderDetails" } } }, { "__metadata": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')", "type": "NorthwindGridModel.Order" }, "orderId": 5001, "customer_customerId": "IBM", "orderDate": "\\Date(1260993011125)\\", "shipCity": "Rochester", "shipCountry": null, "version": 0, "customer": { "__deferred": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')/customer" } }, "orderDetails": { "__deferred": { "uri": "http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')/orderDetails" } } } ] }
```
- Código de respuesta: 200 OK

Recuperación de elementos que no sean entidades con los servicios de datos REST

El servicio de datos REST permite recuperar no sólo entidades, sino también elementos como colecciones de entidades y propiedades.

Recuperación de una colección de entidades

Un cliente puede utilizar una solicitud RetrieveEntitySet para recuperar un conjunto de entidades de eXtreme Scale. Las entidades se representan como un documento de canal de información Atom o una matriz JSON en la carga útil de la respuesta. Para obtener información detallada sobre el protocolo RetrieveEntitySet definido en WCF Data Services, consulte: MSDN: RetrieveEntitySet Request.

El siguiente ejemplo de solicitud RetrieveEntitySet recupera todas las entidades Order asociadas con la entidad Customer('IBM'). Aquí sólo se muestran los pedidos 5000 y 5001.

AtomPub

- Método: GET
- URI de la solicitud: [http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer\('IBM'\)/orders](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders)
- Cabecera de la solicitud: Accept: application/atom+xml
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/atom+xml
- Carga útil de la respuesta:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xml:base = "http://localhost:8080/wxsrestservice/restservice"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m = "http://schemas.microsoft.com/ado/2007/08/dataservices/
  metadata" xmlns = "http://www.w3.org/2005/Atom">
  <title type = "text">Order</title>
  <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Order</id>
  <updated>2009-12-16T22:53:09.062Z</updated>
  <link rel = "self" title = "Order" href = "Order"/>
  <entry>
    <category term = "NorthwindGridModel.Order" scheme = "http://
    schemas.microsoft.com/
    ado/2007/08/dataservices/scheme"/>
    <id>http://localhost:8080/wxsrestservice/restservice/
    NorthwindGrid/Order(orderId=5000,customer_customerId=
    'IBM')</id>
    <title type = "text"/>
    <updated>2009-12-16T22:53:09.062Z</updated>
    <author>
      <name/>
    </author>
    <link rel = "edit" title = "Order" href = "Order(orderId=5000,
    customer_customerId='IBM')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/
    dataservices/related/customer"
    type = "application/atom+xml;type=entry"
    title = "customer" href = "Order(orderId=5000,
    customer_customerId='IBM')/customer"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/
    dataservices/related/orderDetails"
    type = "application/atom+xml;type=feed"
    title = "orderDetails" href = "Order(orderId=5000,customer_customerId='IBM')/
    orderDetails"/>
    <content type = "application/xml">
      <m:properties>
        <d:orderId m:type = "Edm.Int32">5000</d:orderId>
        <d:customer_customerId>IBM</d:customer_customerId>
        <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:
        46:29.562</d:orderDate>
        <d:shipCity>Rochester</d:shipCity>
      </m:properties>
    </content>
  </entry>
</feed>
```

```

        <d:shipCountry m:null = "true"/>
        <d:version m:type = "Edm.Int32">0</d:version>
    </m:properties>
</content>
</entry>
<entry>
    <category term = "NorthwindGridModel.Order" scheme = "http://
schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
    <id>http://localhost:8080/wxsrestservice/restservice/
NorthwindGrid/Order(orderId=5001, customer_customerId='IBM')
</id>
    <title type = "text"/>
    <updated>2009-12-16T22:53:09.062Z</updated>
    <author>
        <name/>
    </author>
    <link rel = "edit" title = "Order" href = "Order(orderId=5001,
customer_customerId='IBM')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/
dataservices/related/customer"
type = "application/atom+xml;type=entry"
title = "customer" href = "Order(orderId=5001,
customer_customerId='IBM')/customer"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/
dataservices/related/orderDetails"
type = "application/atom+xml;type=feed"
title = "orderDetails" href = "Order(orderId=5001,
customer_customerId='IBM')/orderDetails"/>
    <content type = "application/xml">
        <m:properties>
            <d:orderId m:type = "Edm.Int32">5001</d:orderId>
            <d:customer_customerId>IBM</d:customer_customerId>
            <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:50:
11.125</d:orderDate>
            <d:shipCity>Rochester</d:shipCity>
            <d:shipCountry m:null = "true"/>
            <d:version m:type = "Edm.Int32">0</d:version>
        </m:properties>
    </content>
</entry>
</feed>

```

- Código de respuesta: 200 OK

JSON

- Método: GET
- URI de la solicitud: [http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order\(orderId=5000,customer_customerId='IBM'\)](http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM'))
- Cabecera de la solicitud: Accept: application/json
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/json
- Carga útil de la respuesta:

```

{"d":[{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
restservice/NorthwindGrid/Order(orderId=5000,
customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
"orderId":5000,
"customer_customerId":"IBM",
"orderDate":"\\/Date(1260992789562)\\/","
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
wxsrestservice/restservice/NorthwindGrid/Order(orderId=

```

```

    5000,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
  wxsrestservice/restservice/NorthwindGrid/Order(orderId=
  5000,customer_customerId='IBM')/orderDetails"}}},
{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Order(orderId=5001,
  customer_customerId='IBM')"},
"type":"NorthwindGridModel.Order"},
"orderId":5001,
"customer_customerId":"IBM",
"orderDate":"\\/Date(1260993011125)\\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
  wxsrestservice/restservice/NorthwindGrid/Order(orderId=
  5001,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
  wxsrestservice/restservice/NorthwindGrid/Order(orderId=
  5001,customer_customerId='IBM')/orderDetails"}}}]}}

```

- Código de respuesta: 200 OK

Recuperación de una propiedad

Se puede utilizar una solicitud `RetrievePrimitiveProperty` para obtener el valor de una propiedad de una instancia de entidad de eXtreme Scale. El valor de la propiedad se representa en formato XML para las solicitudes AtomPub y como un objeto JSON para las solicitudes JSON en la carga útil de la respuesta. Si desea más detalles sobre la solicitud `RetrievePrimitiveProperty`, consulte: MSDN: `RetrievePrimitiveProperty Request`.

El siguiente ejemplo de solicitud `RetrievePrimitiveProperty` recupera la propiedad `contactName` de la entidad `Customer('IBM')`.

AtomPub

- Método: GET
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName`
- Cabecera de la solicitud: `Accept: application/xml`
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: `Content-Type: application/atom+xml`
- Carga útil de la respuesta:


```

<contactName xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
  John Doe
</contactName>

```
- Código de respuesta: 200 OK

JSON

- Método: GET
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName`
- Cabecera de la solicitud: `Accept: application/json`
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: `Content-Type: application/json`
- Carga útil de la respuesta: `{"d":{"contactName":"John Doe"}}`
- Código de respuesta: 200 OK

Recuperación del valor de una propiedad

Se puede utilizar una solicitud RetrieveValue para obtener el valor en bruto de una propiedad de una instancia de entidad de eXtreme Scale. El valor de la propiedad se representa como un valor en bruto en la carga útil de la respuesta. Si el tipo de entidad es uno de los siguientes, el tipo de medio de la respuesta será "text/plain". De lo contrario, el tipo de medio de la respuesta será "application/octet-stream". Estos tipos son los siguientes:

- Tipos primitivos Java y sus respectivos derivadores
- java.lang.String
- byte[]
- Byte[]
- char[]
- Character[]
- enums
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

Si desea más detalles sobre la solicitud RetrieveValue, consulte: MSDN: RetrieveValue Request.

El siguiente ejemplo de solicitud RetrieveValue recupera el valor en bruto de la propiedad contactName de la entidad Customer('IBM').

- Método de solicitud: GET
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName/$value`
- Cabecera de la solicitud: `Accept: text/plain`
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: `Content-Type: text/plain`
- Carga útil de la respuesta: John Doe
- Código de respuesta: 200 OK

Recuperación de un enlace

Se puede utilizar una solicitud RetrieveLink para obtener el enlace o los enlaces que representan una asociación a uno o una asociación a muchos. Para la asociación a uno, el enlace es de una instancia de entidad de eXtreme Scale a otra y el enlace se representa en la carga útil de la respuesta. Para la asociación a muchos, los enlaces son de una instancia de entidad de eXtreme Scale a todas las demás de una colección de entidades de eXtreme Scale especificada y la respuesta se representa como un conjunto de enlaces en la carga útil de la respuesta. Si desea más detalles sobre la solicitud RetrieveLink, consulte: MSDN: RetrieveLink Request.

A continuación se ofrece un ejemplo de solicitud RetrieveLink. En este ejemplo, recuperamos la asociación entre la entidad Order(orderId=5000,customer_customerId='IBM') y su cliente. La respuesta muestra el URI de la entidad Customer.

AtomPub

- Método: GET
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer`
- Cabecera de la solicitud: `Accept: application/xml`
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: `Content-Type: application/xml`
- Carga útil de la respuesta:

```
<?xml version="1.0" encoding="utf-8"?>
<uri>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
Customer('IBM')</uri>
```
- Código de respuesta: 200 OK

JSON

- Método: GET
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer`
- Cabecera de la solicitud: `Accept: application/json`
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: `Content-Type: application/json`
- Carga útil de la respuesta: `{"d":{"uri":"http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')"}}`

Recuperación de metadatos de servicio

Se puede utilizar una solicitud RetrieveServiceMetadata para obtener el documento de lenguaje de definición de esquema conceptual (CSDL), que describe el modelo de datos asociado al servicio de datos REST de eXtreme Scale. Si desea más detalles sobre la solicitud RetrieveServiceMetadata, consulte: MSDN: RetrieveServiceMetadata Request.

Recuperación de un documento de servicio

Se puede utilizar una solicitud RetrieveServiceDocument para recuperar el documento de servicio que describe la colección de recursos expuestos por el servicio de datos REST de eXtreme Scale. Si desea más detalles sobre la solicitud RetrieveServiceDocument, consulte: MSDN: RetrieveServiceDocument Request.

Solicitudes de inserción con los servicios de datos REST

Se puede utilizar una solicitud InsertEntity para insertar una nueva instancia de entidad de eXtreme Scale, potencialmente con entidades relacionadas nuevas, en el servicio de datos REST de eXtreme Scale.

Solicitud de inserción de entidad

Se puede utilizar una solicitud `InsertEntity` para insertar una nueva instancia de entidad de eXtreme Scale, potencialmente con entidades relacionadas nuevas, en el servicio de datos REST de eXtreme Scale. Al insertar una entidad, el cliente puede especificar si el recurso o la entidad se deben enlazar automáticamente a otras entidades existentes en el servicio de datos.

El cliente debe incluir la información de enlace necesaria en la representación de la relación asociada en la carga útil de la solicitud.

Además de soportar la inserción de una instancia de `EntityType` nueva (E1), la solicitud `InsertEntity` también permite insertar entidades nuevas relacionadas con E1 (descritas por una relación de entidad) en una sola solicitud. Por ejemplo, al insertar una entidad `Customer('IBM')`, podemos insertar todos los pedidos con `Customer('IBM')`. Esta forma de solicitud `InsertEntity` también se conoce como una *inserción profunda*. Con una inserción profunda, las entidades relacionadas se deben representar utilizando la representación en línea de la relación asociada con E1 que identifica el enlace a las entidades relacionadas (que se deben insertar).

Las propiedades de la entidad que se deben insertar se especifican en la carga útil de la solicitud. El servicio de datos REST analiza las propiedades y luego se define la propiedad correspondiente para éstas en la instancia de la entidad. Para el formato AtomPub, la propiedad se especifica como un elemento XML `<d:PROPERTY_NAME>`. Para JSON, la propiedad se especifica como una propiedad de un objeto JSON.

Si falta una propiedad en la carga útil de la solicitud, el servicio de datos REST define como valor de la propiedad de entidad el valor predeterminado java. No obstante, el programa de fondo de base de datos puede rechazar este valor predeterminado, por ejemplo, si la columna no puede tener un valor nulo en la base de datos. Entonces, se devolverá un código de respuesta 500 para indicar un error interno del servidor.

Si se especifican propiedades duplicadas en la carga útil, se utilizará la última propiedad. El servicio de datos REST pasa por alto todos los valores anteriores para el mismo nombre de propiedad.

Si la carga útil contiene una propiedad no existente, el servicio de datos REST devuelve un código de respuesta 400 (Bad Request) para indicar que la solicitud enviada por el cliente era sintácticamente incorrecta.

Si faltan las propiedades clave, el servicio de datos REST devuelve un código de respuesta 400 (Bad Request) para indicar que falta una propiedad clave.

Si la carga útil contiene un enlace a una entidad relacionada con una clave inexistente, el servicio de datos REST devuelve un código de respuesta 404 (Not Found) para indicar que la entidad enlazada no se encuentra.

Si la carga útil contiene un enlace a una entidad relacionada con un nombre de asociación incorrecto, el servicio de datos REST devuelve un código de respuesta 400 (Bad Request) para indicar que el enlace no se encuentra.

Si la carga útil contiene más de un enlace a una relación a uno, se utilizará el último enlace. Todos los enlaces anteriores correspondientes a la misma asociación se pasan por alto.

Si desea más detalles sobre la solicitud InsertEntity, consulte MSDN Library: InsertEntity Request (Biblioteca MSDN: solicitud InsertEntity).

Una solicitud InsertEntity inserta una entidad Customer con la clave 'IBM'.

AtomPub

- Método: POST
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')`
- Cabecera de la solicitud: `Accept: application/atom+xml Content-Type: application/atom+xml`
- Carga útil de la solicitud:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns="http://www.w3.org/2005/Atom">
  <category term="NorthwindGridModel.Customer"
    scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <content type="application/xml">
    <m:properties>
      <d:customerId>Rational</d:customerId>
      <d:city>Rochester</d:city>
      <d:companyName>Rational</d:companyName>
      <d:contactName>John Doe</d:contactName>
      <d:country>USA</d:country>
    </m:properties>
  </content>
</entry>
```

- Cabecera de la respuesta: `Content-Type: application/atom+xml`
- Carga útil de la respuesta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns="http://www.w3.org/2005/Atom">
  <category term="NorthwindGridModel.Customer"
    scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <content type="application/xml">
    <m:properties>
      <d:customerId>Rational</d:customerId>
      <d:city>Rochester</d:city>
      <d:companyName>Rational</d:companyName>
      <d:contactName>John Doe</d:contactName>
      <d:country>USA</d:country>
    </m:properties>
  </content>
</entry>
```

Cabecera de la respuesta:
`Content-Type: application/atom+xml`

Carga útil de la respuesta:

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/restservice" xmlns:d =
  "http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m =
  "http://schemas.microsoft.com/
  ado/2007/08/dataservices/metadata" xmlns = "http://www.w3.org/2005/Atom">
  <category term = "NorthwindGridModel.Customer" scheme = "http://schemas.
  microsoft.com/ado/2007/08/dataservices/scheme"/>
  <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Customer('Rational')</id>
  <title type = "text"/>
  <updated>2009-12-16T23:25:50.875Z</updated>
  <author>
    <name/>
```

```

</author>
<link rel = "edit" title = "Customer" href = "Customer('Rational')"/>
<link rel = "http://schemas.microsoft.com/ado/2007/08/dataservices/related/
orders" type = "application/atom+xml;type=feed"
title = "orders" href = "Customer('Rational')/orders"/>
<content type = "application/xml">
  <m:properties>
    <d:customerId>Rational</d:customerId>
    <d:city>Rochester</d:city>
    <d:companyName>Rational</d:companyName>
    <d:contactName>John Doe</d:contactName>
    <d:country>USA</d:country>
    <d:version m:type = "Edm.Int32">0</d:version>
  </m:properties>
</content>
</entry>

```

- Código de respuesta: 201 Created

JSON

- Método: POST
- URI de la solicitud: <http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer>
- Cabecera de la solicitud: Accept: application/json Content-Type: application/json

- Carga útil de la solicitud:

```

{"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country": "USA",}

```

- Cabecera de la respuesta: Content-Type: application/json

- Carga útil de la respuesta:

```

{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/restservice/
NorthwindGrid/Customer('Rational')",
"type":"NorthwindGridModel.Customer"},
"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country":"USA",
"version":0,
"orders":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/
NorthwindGrid/Customer('Rational')/orders"}}}}

```

- Código de respuesta: 201 Created

Solicitud de inserción de enlace

Se puede utilizar una solicitud InsertLink para crear un enlace nuevo entre dos instancias de entidad de eXtreme Scale. El URI de la solicitud se debe resolver como una asociación de eXtreme Scale a muchos. La carga útil de la solicitud contiene un solo enlace que apunta a la entidad de destino de la asociación a muchos.

Si el URI de la solicitud InsertLink representa una asociación a uno, el servicio de datos REST devuelve una respuesta 400 (Bad Request).

Si el URI de la solicitud InsertLink apunta a una asociación que no existe, el servicio de datos REST devuelve una respuesta 404 (Not Found) para indicar que el enlace no se encuentra.

Si la carga útil contiene un enlace con una clave que no existe, el servicio de datos REST devuelve una respuesta 404 (Not Found) para indicar que la entidad enlazada no se encuentra.

Si la carga útil contiene más de un enlace, el servicio de datos Rest de eXtreme Scale analizará el primer enlace. Los enlaces restantes se pasan por alto.

Si desea más detalles sobre la solicitud InsertLink, consulte: MSDN Library: InsertLink Request (Biblioteca MSDN: solicitud InsertLink).

El siguiente ejemplo de solicitud InsertLink crea un enlace de Customer('IBM') a Order(orderId=5000,customer_customerId='IBM').

AtomPub

- Método: POST
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$link/orders`
- Cabecera de la solicitud: Content-Type: application/xml
- Carga útil de la solicitud:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId=
5000,customer_customerId='IBM')</uri>
```
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

JSON

- Método: POST
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$links/orders`
- Cabecera de la solicitud: Content-Type: application/json
- Carga útil de la solicitud:

```
{"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId
=5000,customer_customerId='IBM')"}}
```
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Solicitudes de actualización con los servicios de datos REST

El servicio de datos REST de WebSphere eXtreme Scale soporta solicitudes de actualización de entidades, propiedades primitivas de entidades, etc.

Actualización de una entidad

Se puede utilizar una solicitud UpdateEntity para actualizar una entidad existente de eXtreme Scale. El cliente puede utilizar un método HTTP PUT para sustituir una entidad existente de eXtreme Scale o utilizar un método HTTP MERGE para fusionar los cambios en una entidad existente de eXtreme Scale.

Al actualizar la entidad, el cliente puede especificar si la entidad (además de actualizarse) se debe enlazar automáticamente a otras entidades existentes en el servicio de datos relacionadas mediante asociaciones con un solo valor (a uno).

La propiedad de la entidad que se debe actualizar se encuentra en la carga útil de la solicitud. El servicio de datos REST analiza la propiedad y luego se define la

propiedad correspondiente para ésta en la entidad. Para el formato AtomPub, la propiedad se especifica como un elemento XML <d:PROPERTY_NAME>. Para JSON, la propiedad se especifica como una propiedad de un objeto JSON.

Si falta una propiedad en la carga útil de la solicitud, el servicio de datos REST define como valor de la propiedad de entidad el valor predeterminado java para el método HTTP PUT. No obstante, el programa de fondo de base de datos puede rechazar este valor predeterminado si, por ejemplo, la columna no puede tener un valor nulo en la base de datos. Entonces, se devolverá un código de respuesta 500 (Internal Server Error) para indicar un error interno del servidor. Si falta una propiedad en la carga útil de la solicitud HTTP MERGE, el servicio de datos REST no cambiará el valor de la propiedad existente.

Si se especifican propiedades duplicadas en la carga útil, se utilizará la última propiedad. El servicio de datos REST pasa por alto todos los valores anteriores con el mismo nombre de propiedad.

Si la carga útil contiene una propiedad no existente, el servicio de datos REST devuelve un código de respuesta 400 (Bad Request) para indicar que la solicitud enviada por el cliente era sintácticamente incorrecta.

Como parte de la serialización de un recurso, si la carga útil de una solicitud de actualización contiene cualquiera de las propiedades clave para la entidad, el servicio de datos REST pasa por alto dichos valores clave porque las claves de entidad son inmutables.

Si desea detalles sobre la solicitud UpdateEntity, consulte: MSDN Library: UpdateEntity Request (Biblioteca MSDN: solicitud UpdateEntity).

Una solicitud UpdateEntity actualiza el nombre de ciudad de Customer('IBM') a 'Raleigh'.

AtomPub

- Método: PUT
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')`
- Cabecera de la solicitud: `Content-Type: application/atom+xml`
- Carga útil de la solicitud:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns="http://www.w3.org/2005/Atom">
  <category term="NorthwindGridModel.Customer"
    scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <title />
  <updated>2009-07-28T21:17:50.609Z</updated>
  <author>
    <name />
  </author>
  <id />
  <content type="application/xml">
    <m:properties>
      <d:customerId>IBM</d:customerId>
      <d:city>Raleigh</d:city>
      <d:companyName>IBM Corporation</d:companyName>
      <d:contactName>Big Blue</d:contactName>
    </m:properties>
  </content>
</entry>
```

```
<d:country>USA</d:country>
</m:properties>
</content>
</entry>
```

- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

JSON

- Método: PUT
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')`
- Cabecera de la solicitud: `Content-Type: application/json`
- Carga útil de la solicitud:

```
{ "customerId": "IBM",
  "city": "Raleigh",
  "companyName": "IBM Corporation",
  "contactName": "Big Blue",
  "country": "USA", }
```
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Actualización de una propiedad primitiva de entidad

La solicitud `UpdatePrimitiveProperty` puede actualizar un valor de propiedad de una entidad de eXtreme Scale. La propiedad y el valor que se deben actualizar están en la carga útil de la solicitud. La propiedad no puede ser una propiedad clave porque eXtreme Scale no permite que los clientes cambien claves de entidad.

Si desea más detalles sobre la solicitud `UpdatePrimitiveProperty`, consulte: MSDN Library: `UpdatePrimitiveProperty Request` (Biblioteca MSDN: solicitud `UpdatePrimitiveProperty`).

A continuación se ofrece un ejemplo de solicitud `UpdatePrimitiveProperty`. En este ejemplo, actualizamos el nombre de la ciudad de `Customer('IBM')` a 'Raleigh'.

AtomPub

- Método: PUT
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city`
- Cabecera de la solicitud: `Content-Type: application/xml`
- Carga útil de la solicitud:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<city xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
  Raleigh
</city>
```
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

JSON

- Método: PUT
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city`
- Cabecera de la solicitud: `Content-Type: application/json`

- Carga útil de la solicitud: {"city":"Raleigh"}
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Actualización de un valor de propiedad primitiva de entidad

La solicitud UpdateValue puede actualizar un valor de propiedad si procesar de una entidad de eXtreme Scale. El valor que se debe actualizar se representa como un valor en bruto en la carga útil de la solicitud. La propiedad no puede ser una propiedad clave porque eXtreme Scale no permite que los clientes cambien claves de entidad.

El tipo de contenido de la solicitud puede ser "text/plain" o "application/octet-stream", según el tipo de propiedad. Consulte la sección 6.3.1.4 si desea más detalles.

Si desea más detalles sobre la solicitud UpdateValue, consulte: MSDN Library: UpdateValue Request (Biblioteca MSDN: solicitud UpdateValue).

A continuación se ofrece un ejemplo de solicitud UpdateValue. En este ejemplo, actualizamos el nombre de la ciudad de Customer('IBM') a 'Raleigh'.

- Método: PUT
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city/\$value
- Cabecera de la solicitud: Content-Type: text/plain
- Carga útil de la solicitud: Raleigh
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Actualización de un enlace

La solicitud UpdateLink se puede utilizar para establecer una asociación entre dos instancias de entidad de eXtreme Scale. La asociación puede ser una relación con un solo valor (a uno) o con varios valores (a muchos).

La actualización de un enlace entre dos instancias de entidad de eXtreme Scale no sólo puede establecer asociaciones, sino también eliminar asociaciones. Por ejemplo, si el cliente establece un asociación a uno entre una entidad Order(orderId=5000,customer_customerId='IBM') y una instancia de Customer('ALFKI'), debe disociar la entidad Order(orderId=5000,customer_customerId='IBM') y la entidad de su instancia de Customer actualmente asociada.

Si cualquiera de las instancias de entidad especificadas en la solicitud UpdateLink no se encuentra, el servicio de datos REST devuelve una respuesta 404 (Not Found).

Si el URI de la solicitud UpdateLink especifica una asociación inexistente, el servicio de datos REST devuelve una respuesta 404 (Not Found) para indicar que el enlace no se encuentra.

Si el URI especificado en la carga útil de la solicitud UpdateLink no se resuelve en la misma entidad o en la misma clave especificada en el URI, si existe, entonces el servicio de datos REST de eXtreme Scale devuelve una respuesta 400 (Bad Request).

Si la carga útil de la solicitud UpdateLink contiene varios enlaces, entonces el servicio de datos REST sólo analizará el primer enlace. Los enlaces restantes se pasan por alto.

Si desea más detalles sobre la solicitud UpdateLink, consulte: MSDN Library: UpdateLink Request (Biblioteca MSDN: solicitud UpdateLink).

A continuación se ofrece un ejemplo de solicitud UpdateLink. En este ejemplo, actualizamos la relación de cliente de la entidad Order(orderId=5000,customer_customerId='IBM') y de Customer('IBM') a Customer('IBM').

Recuerde: El ejemplo anterior sólo es ilustrativo. Como todas las asociaciones son normalmente asociaciones de claves para una cuadrícula particionada, el enlace no se puede cambiar.

AtomPub

- Método: PUT
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer`
- Cabecera de la solicitud: Content-Type: application/xml
- Carga útil de la solicitud:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>
  http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
</uri>
```
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

JSON

- Método: PUT
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer`
- Cabecera de la solicitud: Content-Type: application/xml
- Carga útil de la solicitud: `{"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')"}`
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Solicitudes de supresión con los servicios de datos REST

El servicio de datos REST de WebSphere eXtreme Scale suprime entidades, valores de propiedad y enlaces.

Supresión de una entidad

La solicitud DeleteEntity suprime entidades de eXtreme Scale del servicio de datos REST.

Si cualquier relación con la entidad que se debe suprimir tiene definida la supresión en cascada, el servicio de datos REST de eXtreme Scale suprimirá la entidad o las entidades relacionadas. Si desea más detalles sobre la solicitud DeleteEntity, consulte: MSDN Library: DeleteEntity Request (Biblioteca MSDN: solicitud DeleteEntity).

La siguiente solicitud DeleteEntity suprime el cliente con la clave 'IBM'.

- Método: DELETE
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')`
- Carga útil de la solicitud: Ninguna
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Supresión del valor de una propiedad

La solicitud DeleteValue define una propiedad de entidad de eXtreme Scale como nula.

Cualquier propiedad de una entidad de eXtreme Scale se puede definir como nula con una solicitud DeleteValue. Para definir una propiedad como nula, asegúrese de todo lo siguiente:

- Para cualquier tipo de número primitivo y su derivador, BigInteger, o BigDecimal, el valor de la propiedad se define como 0.
- Para el tipo Boolean o boolean, el valor de la propiedad se define como false.
- Para el tipo char o Character, el valor de la propiedad se define con el carácter #X1 (NIL).
- Para el tipo enum, el valor de la propiedad se define con el valor enum con el ordinal 0.
- Para todos los demás tipos, el valor de la propiedad se define como nulo.

No obstante, el programa de fondo de base de datos puede rechazar una solicitud de supresión de este tipo si, por ejemplo, la propiedad no puede tener un valor nulo en la base de datos. En este caso, el servicio de datos REST devuelve una respuesta 500 (Internal Server Error). Si desea más detalles sobre la solicitud DeleteValue, consulte: MSDN Library: DeleteValue Request (Biblioteca MSDN: solicitud DeleteValue).

A continuación se ofrece un ejemplo de solicitud DeleteValue. En este ejemplo, definimos como nulo el nombre de contacto de Customer('IBM').

- Método: DELETE
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName`
- Carga útil de la solicitud: Ninguna
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Supresión de un enlace

La solicitud DeleteLink puede eliminar una asociación entre dos instancias de entidad de eXtreme Scale. La asociación puede ser una relación a uno o una relación a muchos. No obstante, el programa de fondo de base de datos puede

rechazar una solicitud de supresión de este tipo si, por ejemplo, la restricción de clave foránea está definida. En este caso, el servicio de datos REST devuelve una respuesta 500 (Internal Server Error). Si desea más detalles sobre la solicitud DeleteLink, consulte: MSDN Library: DeleteLink Request (Biblioteca MSDN: solicitud DeleteLink).

La siguiente solicitud DeleteLink elimina la asociación entre Order(101) y su Customer asociado.

- Método: DELETE
- URI de la solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer`
- Carga útil de la solicitud: Ninguna
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Simultaneidad optimista

El servicio de datos REST de eXtreme Scale sigue un modelo de bloqueo optimista utilizando cabeceras HTTP nativas: If-Match, If-None-Match y ETag. Estas cabeceras se envían en mensajes de solicitud y respuesta para transmitir la información de versión de una entidad del servidor al cliente y del cliente al servidor.

Para obtener información más detallada sobre la simultaneidad optimista, consulte MSDN Library: Optimistic Concurrency (ADO.NET) (Biblioteca MSDN: concurrencia optimista (ADO.NET)).

El servicio de datos REST de eXtreme Scale habilita la simultaneidad optimista de una entidad si hay definido un atributo de versión en el esquema de dicha entidad. Una propiedad de versión se puede definir en el esquema de entidad mediante una anotación `@Version` para clases Java o un atributo `<version/>` para entidades definidas utilizando un archivo XML descriptor de entidad. El servicio de datos REST de eXtreme Scale propaga automáticamente el valor de la propiedad `version` al cliente en la cabecera ETag para respuestas de una sola entidad utilizando un atributo `m:etag` en la carga útil para respuestas de XML de entidad múltiple y un atributo `etag` en la carga útil para respuestas de JSON de entidad múltiple.

Para obtener detalles sobre cómo definir un esquema de entidad eXtreme Scale, consulte "Definición de un esquema de entidad" en la página 62.

Capítulo 5. programación con sistema plug-ins y API del sistema

Un plug-in es un componente que proporciona una función a los componentes que se pueden conectar, que incluyen ObjectGrid y BackingMap. Para utilizar eXtreme Scale de forma más eficaz como una cuadrícula de datos en memoria o un espacio de proceso de base de datos, debe determinar con atención cómo puede maximizar mejor el rendimiento con los plug-ins disponibles.

Introducción a los plug-ins

Un plug-in de WebSphere eXtreme Scale es un componente que proporciona un determinado tipo de función a los componentes conectables que incluyen ObjectGrid y BackingMap. WebSphere eXtreme Scale proporciona varios puntos de conexión para permitir a las aplicaciones y a los proveedores de memoria caché integrarse con distintos almacenes de datos, las API de cliente alternativas y para mejorar el rendimiento general de la memoria caché. El producto se entrega con varios plug-ins predeterminados y preincorporados, pero también puede crear plug-ins personalizados con la aplicación.

Todos los plug-ins son clases concretas que implementan una o más interfaces de plug-in de eXtreme Scale. ObjectGrid crea instancias de estas clases y las invoca cuando conviene. ObjectGrid y BackingMaps permiten que se registren plug-ins personalizados.

Plug-ins ObjectGrid

Los siguientes plug-ins están disponibles para una instancia de ObjectGrid:

- **TransactionCallback:** un plug-in TransactionCallback proporciona sucesos de ciclo de vida de transacción.
- **ObjectGridEventListener:** un plug-in ObjectGridEventListener proporciona sucesos de ciclo de vida de ObjectGrid para ObjectGrid, los fragmentos y las transacciones.
- **SubjectSource, ObjectGridAuthorization, SubjectValidation:** eXtreme Scale proporciona varios puntos finales de seguridad para permitir que se integren mecanismos de autenticación personalizados con eXtreme Scale. (Sólo en el servidor)
- **MapAuthorization** (sólo en el servidor)
- **JPATxCallback** (sólo en el servidor)
- **Subclases de JPATxCallback**

Requisitos comunes de los plug-ins de ObjectGrid

ObjectGrid crea instancia de plug-in y las inicializa mediante los convenios de JavaBeans. Todas las implementaciones de plug-in anteriores tienen estos requisitos:

- La clase de plug-in debe ser una clase pública de nivel superior.
- La clase de plug-in debe proporcionar un constructor público sin argumentos.
- La clase de plug-in debe estar disponible en la vía de acceso de clase para servidores y clientes (como convenga).

- Los atributos se deben establecer utilizando los métodos de propiedad del estilo JavaBeans.
- Los plug-ins, a menos que se especifique lo contrario, se registran antes de que se inicialice ObjectGrid y no se pueden modificar una vez inicializado ObjectGrid.

Plug-ins de BackingMap

Los plug-ins siguientes están disponibles para un objeto BackingMap:

- **Evictor:** un plug-in de desalojador es un mecanismo predeterminado proporcionado para desalojar entradas de memoria caché y un plug-in para crear desalojadores personalizados.
- **ObjectTransformer:** un plug-in ObjectTransformer le permite serializar, deserializar y copiar objetos en la memoria caché.
- **OptimisticCallback:** un plug-in OptimisticCallback le permite personalizar las operaciones de mantenimiento de versiones y comparación de objetos de memoria caché al utilizar la estrategia de bloqueo optimista.
- **MapEventListener:** un plug-in MapEventListener proporciona notificaciones de devolución de llamada y cambios de estado de memoria caché significativos que se producen para BackingMap.
- **Indexing:** utilice la característica de indexación, representada por el plug-in MapIndexplug-in, para generar un índice o varios índices en una correlación de BackingMap para soportar el acceso de datos que no son clave.
- **Loader:** un plug-in Loader en una correlación de ObjectGrid actúa como una memoria caché para los datos que normalmente se conservan en un almacén persistente o en el mismo sistema o en algún otro sistema. (Sólo en el servidor)

Ciclos de vida de plug-in

La mayoría de plug-ins tienen ambos métodos, initialize y destroy, o métodos equivalentes, además de los métodos para los que se han diseñado para funcionar. Estos métodos especializados de cada plug-in están disponibles para ser invocados en puntos operativos designados. Ambos métodos, initialize y destroy, definen el ciclo de vida de los plug-ins, que están controlados por sus objetos "propietario". Un objeto propietario es el objeto que utiliza realmente el plug-in determinado. Un propietario puede ser un cliente de cuadrícula, un servidor o una correlación de respaldo.

Cuando se inicializan los objetos propietario, invocarán el método initialize de los plug-ins que poseen. Durante el ciclo de destrucción de los objetos propietario, también se invocará el método destroy de los plug-ins de forma coherente. Si desea ver los detalles sobre los valores específicos de los métodos initialize y destroy, junto con otros métodos con funciones con cada plug-in, consulte los temas relevantes a cada plug-in.

Como ejemplo, considere un entorno distribuido. Tanto los ObjectGrids del lado del cliente, como los ObjectGrids del lado del servidor pueden tener sus propios plug-ins. El ciclo de vida de un ObjectGrid del lado del cliente y, por lo tanto, sus instancias de plug-in son independientes de todas las instancias de plug-in y del ObjectGrid del lado del servidor.

En dicha topología distribuida, tiene un ObjectGrid llamado "myGrid" definido en el archivo objectGrid.xml y configurado con un ObjectGridEventListener personalizado llamado myObjectGridEventListener. El archivo

objectGridDeployment.xml define la política de despliegue para el ObjectGrid myGrid. Se utilizan tanto objectGrid.xml como objectGridDeployment.xml para iniciar los servidores de contenedor. Durante el inicio del servidor de contenedor, la instancia del ObjectGrid myGrid del lado del servidor se inicializará y se invocará el método initialize de la instancia de myObjectGridEventListener que es propiedad de la instancia de myObjectGrid. Después de que se inicie el servidor del contenedor, la aplicación se puede conectar a la instancia de la ObjectGrid myGrid del lado del servidor y obtener una instancia del lado del cliente.

Al obtener la instancia del ObjectGrid myGrid del lado del cliente, la instancia de myGrid del cliente pasará su propio ciclo de inicialización e invocará el método initialize de su propia instancia de myObjectGridEventListener del lado del cliente. Esta instancia de myObjectGridEventListener del lado del cliente es independiente de la instancia de myObjectGridEventListener del lado de servidor. Su ciclo de vida está controlado por su propietario, que es la instancia del ObjectGrid myGrid del lado del cliente.

Si la aplicación se desconecta o destruye la instancia del ObjectGrid myGrid del lado del cliente, se invocará automáticamente el método destroy de la instancia propietaria de myObjectGridEventListener del lado del cliente. Sin embargo, esto no tiene impacto en la instancia de myObjectGridEventListener del lado del servidor. Sólo se invocará el método destroy de la instancia de myObjectGridEventListener del lado del servidor durante el ciclo de destrucción de la instancia del ObjectGrid myGrid del lado del servidor al detener un servidor de contenedor. Es decir, cuando se detiene un servidor de contenedor, las instancias incluidas de ObjectGrid se destruirán y se invocará el método destroy de todos los plug-ins que posea.

Aunque el ejemplo anterior se aplica específicamente al caso de una instancia de cliente y servidor de un ObjectGrid, el propietario de un plug-in también puede ser un BackingMap y debe tener cuidado al determinar las configuraciones para los plug-ins que puede escribir basándose en estas consideraciones de ciclo de vida.

Plug-ins para desalojar los objetos de memoria caché

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.

Propiedad TimeToLive

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, que tiene los tipos siguientes.

Ninguno (NONE)

Especifica que las entradas nunca caducan y, por lo tanto, nunca se eliminan de la correlación.

Hora de creación (CREATION_TIME)

Especifica que las entradas se desalojan en función de cuándo se crearon.

Si utiliza el ttlType CREATION_TIME , el deslojador desaloja una entrada cuando su hora de creación es igual a su valor de atributo TimeToLive, que se establece en milisegundos en la configuración de la aplicación. Si estableció el valor del atributo TimeToLive en 10 segundos, la entrada se desalojará automáticamente 10 segundos después de su inserción.

Es importante establecer con cuidado este valor de CREATION_TIME. El deslojador 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.

El ttlType CREATION_TIME resulta de utilidad en casos como renovar la cotización de bolsa cada 20 minutos o menos. Suponga que una aplicación web 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.

Hora del último acceso (LAST_ACCESED_TIME)

Especifica que las entradas se desalojan en función de la última vez que se accedió a ellas, ya sea porque se leyeron o se actualizaron.

Última actualización (LAST_UPDATE_TIME)

Especifica que las entradas se desalojan en función de cuándo se actualizaron por última vez.

Si utiliza LAST_UPDATE_TIME ttlType, establezca el valor de TimeToLive en un número más bajo que si estuviera utilizando CREATION_TIME, porque el valor de 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.

Los ttlType LAST_ACCESS_TIME o LAST_UPDATE_TIME resultan de utilidad en casos como mantener los datos de sesión de un cliente, con una correlación ObjectGrid. 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 TTL de LAST_ACCESS_TIME o LAST_UPDATE_TIME con el atributo TimeToLive establecido en 30 minutos resulta adecuado para esta aplicación.

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

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("SomeMap");
bMap.setTtlEvictorType(TTLType.LAST_ACCESS_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*.

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 programación*.

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.

Desalojador TimeToLive (TTL)

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 entradas en cada instancia de BackingMap.

Habilitación del desalojador TTL mediante programa

Los desalojadores TTL están asociados a las instancias BackingMap. El desalojador predeterminado utiliza una política de desalojo de tiempo de vida (TTL) para cada instancia de BackingMap. Si proporciona un mecanismo de desalojador conectable, generalmente utiliza una política de desalojo basada en el número de entradas en lugar del tiempo.

El siguiente fragmento de código utiliza la interfaz BackingMap para definir la hora de caducidad para cada entrada a 10 minutos después de la creación de la entrada.

```
desalojador de tiempo de vida mediante programa  
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;  
import com.ibm.websphere.objectgrid.ObjectGridManager;  
import com.ibm.websphere.objectgrid.ObjectGrid;  
import com.ibm.websphere.objectgrid.BackingMap;  
import com.ibm.websphere.objectgrid.TTLType;
```

```

ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "myMap" );
bm.setTtlEvictorType( TTLType.CREATION_TIME );
bm.setTimeToLive( 600 );

```

El argumento del método `setTimeToLive` es 600 porque indica que el tiempo de vida está en segundos. El código anterior se debe ejecutar antes de que se invoque al método `initialize` en la instancia de `ObjectGrid`. Estos atributos `BackingMap` no se pueden modificar después de que se inicialice la instancia de `ObjectGrid`. Después de ejecutar el código, cualquier entrada que se inserte en la `BackingMap` `myMap` tiene una hora de caducidad. Una vez que se ha alcanzado la hora de caducidad, el desalojador TTL elimina la entrada.

Para definir como hora de caducidad la hora del último acceso más 10 minutos, cambie el argumento que se pasa al método `setTtlEvictorType` de `TTLType.CREATION_TIME` a `TTLType.LAST_ACCESS_TIME`. Con este valor, la hora de caducidad se calcula como la hora del último acceso más diez minutos. Cuando se crea una entrada por primera vez, la hora del último acceso es la hora de creación. Para basar la hora de caducidad en la última *actualización*, en lugar de solo en el último *acceso* (tanto si interviene o no una actualización), sustituya el valor `TTLType.LAST_UPDATE_TIME` por el valor `TTLType.LAST_ACCESS_TIME`.

Cuando se utiliza el valor `TTLType.LAST_ACCESS_TIME`, o `TTLType.LAST_UPDATE_TIME` se pueden utilizar las interfaces `ObjectMap` y `JavaMap` para sustituir el valor de tiempo de vida de `BackingMap`. Este mecanismo permite que una aplicación utilice un valor de tiempo de vida distinto para cada entrada que se cree. Supongamos que el fragmento de código anterior ha definido el atributo de `ttlType` como `LAST_ACCESS_TIME` y ha definido 10 minutos como valor de tiempo de vida. Una aplicación entonces puede alterar temporalmente el valor del tiempo de vida de todas las entradas ejecutando el siguiente código antes de crear o modificar una entrada:

```

import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.ObjectMap;
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
int oldTimeToLive1 = om.setTimeToLive( 1800 );
om.insert("key1", "value1" );
int oldTimeToLive2 = om.setTimeToLive( 1200 );
om.insert("key2", "value2" );

```

En el fragmento de código anterior, la entrada con la clave `key1` tiene una fecha de caducidad del tiempo de inserción más 30 minutos como resultado de la invocación del método `setTimeToLive(1800)` en la instancia de `ObjectMap`. La variable `oldTimeToLive1` se establece en 600 ya que el valor de tiempo de vida de `BackingMap` se utiliza como valor predeterminado si no se ha llamado previamente al método `setTimeToLive` en la instancia de `ObjectMap`.

La entrada con la clave `key2` tiene una hora de caducidad del tiempo de inserción más 20 minutos como resultado de la invocación del método `setTimeToLive(1200)` en la instancia de `ObjectMap`. La variable `oldTimeToLive2` se establece en 1800 ya que el valor de tiempo de vida de la invocación anterior del método `ObjectMap.setTimeToLive` ha establecido el tiempo de vida en 1800.

El ejemplo anterior muestra la inserción de dos entradas de correlación en la correlación `myMap` para las claves `key1` y `key2`. Más adelante, la aplicación puede seguir actualizando estas entradas de correlación y mantener a la vez los valores de tiempo de vida que se utilizan en el momento de la inserción para cada entrada

de correlación. El siguiente ejemplo muestra cómo mantener los valores de tiempo de vida utilizando una constante definida en la interfaz ObjectMap:

```
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
om.setTimeToLive( ObjectMap.USE_DEFAULT );
session.begin();
om.update("key1", "updated value1" );
om.update("key2", "updated value2" );
om.insert("key3", "value3" );
session.commit();
```

Dado que el valor especial de ObjectMap.USE_DEFAULT se utiliza en la llamada al método setTimeToLive, la clave key1 mantiene su valor de tiempo de vida de 1800 segundos y la clave key2 retiene su valor de tiempo de vida de 1200 segundos porque estos valores se utilizaron cuando la transacción anterior insertó estas entradas de correlaciones.

El ejemplo anterior también muestra una nueva entrada de correlación para la inserción de la clave key3insert. En este caso, el valor especial USE_DEFAULT indica que se debe utilizar el valor predeterminado del tiempo de vida para esta correlación. El valor predeterminado lo define el atributo BackingMap del tiempo de vida. Consulte Consulte los atributos de la interfaz BackingMap si desea más información sobre cómo se define el atributo tiempo de vida en la instancia de BackingMap.

Consulte la documentación de la API para el método setTimeToLive en las interfaces ObjectMap y JavaMap. La documentación explica que se genera una excepción IllegalStateException si el método BackingMap.getTtlEvictorType devuelve cualquier cosa distinta al valor TTLType.LAST_ACCESS_TIME o TTLType.LAST_UPDATE_TIME. Las interfaces ObjectMap y JavaMap pueden sustituir el valor de tiempo de vida sólo cuando se utiliza el valor LAST_ACCESS_TIME o TTLType.LAST_UPDATE_TIM para el tipo de desalojador TTL. El método setTimeToLive no se puede utilizar para sustituir el valor de tiempo de vida cuando se utiliza el valor de tipo de desalojador CREATION_TIME o NONE.

Habilitación del desalojador utilizando la configuración XML

En lugar de utilizar la interfaz BackingMap para establecer mediante programa los atributos de BackingMap que pueden ser utilizados por el desalojador TTL, puede utilizar un archivo XML para configurar cada instancia de BackingMap. El siguiente código demuestra cómo establecer estos atributos para tres correlaciones de BackingMap distintas:

habilitación del desalojador de tiempo de vida mediante XML

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
  <objectGrid name="grid1">
    <backingMap name="map1" ttlEvictorType="NONE" />
    <backingMap name="map2" ttlEvictorType="LAST_ACCESS_TIME|LAST_UPDATE_TIME"
      timeToLive="1800" />
    <backingMap name="map3" ttlEvictorType="CREATION_TIME" timeToLive="1200" />
  </objectGrid>
</objectGrids>
```

El ejemplo anterior muestra que la instancia de BackingMap map1 utiliza un tipo de desalojo NONE TTL. La instancia de BackingMap map2 utiliza el tipo de desalojador TTL LAST_ACCESS_TIME TTL o LAST_UPDATE_TIME (especifique solo uno o el otro de estos valores) y tiene un valor de tiempo de vida de 1800 segundos, o 30 minutos. La instancia de BackingMap map3 está definida para que utilice un tipo de desalojador TTL CREATION_TIME y tiene un valor de tiempo de vida de 1200 segundos o 20 minutos.

Conexión de un desalojador conectable

Dado que los desalojadores están asociados a BackingMaps, utilice la interfaz BackingMap para especificar el desalojador conectable.

Desalojadores conectables 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 elige cuándo desalojar una o más entradas de la instancia de BackingMap.

Una instancia de 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.

Utilización de desalojadores conectables opcionales

Para añadir desalojadores conectables opcionales a la configuración de BackingMap puede utilizar la configuración programática o la configuración XML.

Conexión de un desalojador conectable mediante programación

Dado que los desalojadores están asociados a BackingMaps, utilice la interfaz BackingMap para especificar el desalojador conectable. El siguiente fragmento de código es un ejemplo para especificar un desalojador LRUEvictor para la BackingMap map1 y un desalojador LFUEvictor para la instancia del BackingMap map2:

conexión de un desalojador a través de programa

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor;
import com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
LRUEvictor evictor = new LRUEvictor();
evictor.setMaxSize(1000);
evictor.setSleepTime( 15 );
evictor.setNumberOfLRUQueues( 53 );
bm.setEvictor(evictor);
bm = og.defineMap( "map2" );
LFUEvictor evictor2 = new LFUEvictor();
evictor2.setMaxSize(2000);
evictor2.setSleepTime( 15 );
evictor2.setNumberOfHeaps( 211 );
bm.setEvictor(evictor2);
```

El fragmento anterior muestra un desalojador LRUEvictor utilizado para la BackingMap map1 con un número máximo de entradas aproximado de 53.000 ($53 * 1000$). El desalojador LFUEvictor se utiliza para la BackingMap map2 con un número máximo aproximado de entradas de 422.000 ($211 * 2000$). Los desalojadores LRU y LFU tienen una propiedad de tiempo de inactividad que indica durante cuánto tiempo está inactivo el desalojador antes de entrar en actividad y comprobar si es necesario desalojar alguna entrada. El tiempo de inactividad se especifica en segundos. Un valor de 15 segundos constituye un buen término medio entre el impacto en el rendimiento y cómo evitar que BackingMap crezca demasiado. El objetivo es utilizar el mayor tiempo de inactividad posible sin provocar que BackingMap aumente excesivamente de tamaño.

El método `setNumberOfLRUQueues` establece la propiedad de LRUEvictor que indica cuántas colas LRU utiliza el desalojador para gestionar la información LRU. Una colección de colas se utiliza de modo que cada entrada no mantenga la información LRU en la misma cola. Este enfoque puede mejorar el rendimiento al minimizar el número de entradas de correlación que necesiten sincronizarse en el mismo objeto de cola. Un buen modo de minimizar el impacto que el desalojador LRU puede tener en el rendimiento consiste en aumentar el número de colas. Un buen punto de partida es utilizar el diez por ciento del número máximo de entradas como número de colas. Generalmente es mejor utilizar un número primo que uno que no lo sea. El método `setMaxSize` indica cuántas entradas se permiten en cada cola. Cuando una cola alcanza su número máximo de entradas, la entrada o las entradas menos utilizadas recientemente en esa cola se desalojarán la próxima vez que el desalojador compruebe si hay alguna entrada que es necesario desalojar.

El método `setNumberOfHeaps` establece la propiedad LFUEvictor para establecer cuántos objetos de almacenamiento dinámico binario utiliza LFUEvictor para gestionar información de LFU. En este caso también se utiliza una colección para mejorar el rendimiento. La utilización del diez por ciento del número máximo de

entradas es un buen punto de partida, y un número primo es generalmente mejor que uno que no lo sea. El método `setMaxSize` indica cuántas entradas se permiten en cada almacenamiento dinámico. Cuando un almacenamiento dinámico alcanza su número máximo de entradas, la entrada o las entradas menos utilizadas frecuentemente en ese almacenamiento dinámico se desalojarán la próxima vez que el desalojador compruebe si hay alguna entrada que es necesario desalojar.

Configuración del XML para conectar un desalojador conectable

En lugar de utilizar varias API para conectar un desalojador y establecer sus propiedades, se puede utilizar un archivo XML para configurar todas las `BackingMap` como se ilustra en el siguiente ejemplo:

conexión de un desalojador mediante XML

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
  <objectGrid name="grid">
    <backingMap name="map1" ttlEvictorType="NONE" pluginCollectionRef="LRU" />
    <backingMap name="map2" ttlEvictorType="NONE" pluginCollectionRef="LFU" />
  </objectGrid>
</objectGrids>
<backingMapPluginCollections>
  <backingMapPluginCollection id="LRU">
    <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
      <property name="maxSize" type="int" value="1000" description="set max size for each LRU queue" />
    </bean>
    <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
    <property name="numberOfLRUQueues" type="int" value="53" description="set number of LRU queues" />
  </backingMapPluginCollection>
  <backingMapPluginCollection id="LFU">
    <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
      <property name="maxSize" type="int" value="2000" description="set max size for each LFU heap" />
      <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
      <property name="numberOfHeaps" type="int" value="211" description="set number of LFU heaps" />
    </bean>
  </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Desalojo basado en memoria

Todos los desalojadores incorporados dan soporte al desalojo basado en memoria que puede habilitarse en la interfaz `BackingMap` estableciendo el atributo `evictionTriggers` de `BackingMap` en "MEMORY_USAGE_THRESHOLD". Para obtener más información sobre cómo establecer el atributo `evictionTriggers` en `BackingMap`, véala consulta de configuración de la interfaz `BackingMap` y `eXtreme Scale`.

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.

Para cambiar el porcentaje del umbral de uso predeterminado, establezca la propiedad `memoryThresholdPercentage` en el archivo de propiedades del contenedor y servidor para el proceso de servidor `eXtreme Scale`. Para establecer el umbral de uso de destino en un proceso de cliente de `eXtreme Scale`, puede utilizar `MemoryPoolMXBean`. Consulte también el archivo `containerServer.props` y el inicio de procesos de servidor de `Xtreme Scale`.

Durante el tiempo de ejecución si el uso de memoria excede el umbral del uso del destino, los desalojadores basados en memoria empezarán a desalojar entradas e intentarán mantener el uso de la memoria por debajo del umbral de uso del

destino. Sin embargo, no hay ninguna garantía de que la velocidad de desalojo sea lo suficientemente rápida como para evitar un error de falta de memoria posible si el tiempo de ejecución del sistema sigue consumiendo rápidamente la memoria.

Cómo escribir un desalojador personalizado

WebSphere eXtreme Scale permite escribir una implementación de desalojo personalizada.

Debe crear un desalojador personalizado que implemente la interfaz de desalojador y seguir los convenios comunes de plug-in de eXtreme Scale. La interfaz es la siguiente:

```
public interface Evictor
{
    void initialize(BackingMap map, EvictionEventCallback callback);
    void activate();
    void apply(LogSequence sequence);
    void deactivate();
    void destroy();
}
```

- Se invoca el método `initialize` durante la inicialización del objeto `BackingMap`. Este método inicializa un plug-in `Evictor` con una referencia a `BackingMap` y una referencia a un objeto que implementa la interfaz `com.ibm.websphere.objectgrid.plugins.EvictionEventCallback`.
- Se llama al método `activate` para activar el desalojador. Después de que se llame al método, el desalojador puede utilizar la interfaz `EvictionEventCallback` para desalojar las entradas de correlación. Si el desalojador intenta utilizar la interfaz `EvictionEventCallback` para desalojar entradas de la correlación antes de llamar al método de activación, se emitirá una excepción `IllegalStateException`.
- Se invoca el método `apply` cuando se confirman las transacciones que acceden a una o más entradas de `BackingMap`. A este método se le pasa una referencia a un objeto que implementa la interfaz `com.ibm.websphere.objectgrid.plugins.LogSequence`. La interfaz `LogSequence` permite que un plug-in `Evictor` determine qué entradas de `BackingMap` la transacción ha creado, modificado o eliminado. El desalojador utiliza esta información en el momento de decidir qué entradas va a desalojar.
- Se llama al método `deactivate` para desactivar el desalojador. Después de llamar a este método, el desalojador debe dejar de utilizar la interfaz `EvictionEventCallback` para desalojar entradas de la correlación. Si el desalojador continúa utilizando esta interfaz después de llamar a este método, se emitirá una excepción `IllegalStateException`.
- Se invoca el método `destroy` cuando se destruye `BackingMap`. Este método permite que un desalojador termine las hebras que pueda haber creado.

La interfaz `EvictionEventCallback` tiene los siguientes métodos:

```
public interface EvictionEventCallback
{
    void evictMapEntries(List evictorDataList) throws ObjectGridException;
    void evictEntries(List keysToEvictList) throws ObjectGridException;
    void setEvictorData(Object key, Object data);
    Object getEvictorData(Object key);
}
```

Los métodos `EvictionEventCallback` son utilizados por un plug-in `Evictor` para realizar una devolución de llamada a la infraestructura eXtreme Scale del modo siguiente:

- El método `setEvictorData` es utilizado por un desalojador para solicitar la infraestructura que se utiliza para almacenar y asociar algún objeto desalojador que se crea con la entrada indicada por el argumento de la clave. Los datos son específicos del desalojador y se determinan mediante la información que el desalojador necesita para implementar el algoritmo que utiliza. Por ejemplo, en un algoritmo de tipo utilizado con menor frecuencia, el desalojador mantiene una cuenta en el objeto de datos de desalojador para realizar un seguimiento de las veces que se invoca el método de aplicación con `LogElement` que se refiere a una entrada de una clave dada.
- El método `getEvictorData` es utilizado por un desalojador para recuperar los datos que pasa al método `setEvictorData` durante una invocación previa al método `apply`. Si no se encuentran los datos del desalojador del argumento de la clave especificada, se devuelve un objeto `KEY_NOT_FOUND` especial definido en la interfaz `EvictorCallback`.
- El método `evictMapEntries` es utilizado por un desalojador para solicitar el desalojo de una o más entradas de correlación. Cada objeto del parámetro `evictorDataList` debe implementar la interfaz `com.ibm.websphere.objectgrid.plugins.EvictorData`. Además, la misma instancia `EvictorData` que se pasa al método `setEvictorData` debe estar en el parámetro de la lista de datos de desalojador de este método. El método `getKey` de la interfaz `EvictorData` se utiliza para determinar la entrada de la correlación que se va a desalojar. La entrada de la correlación se desaloja si la entrada de memoria caché contiene actualmente la misma instancia `EvictorData` que se encuentra en la lista de datos de desalojador para esta entrada de memoria caché.
- El método `evictEntries` es utilizado por un desalojador para solicitar el desalojo de una o más entradas de correlación. Este método sólo se utiliza si el objeto que se pasa al método `setEvictorData` no implementa la interfaz `com.ibm.websphere.objectgrid.plugins.EvictorData`.

Después de que se complete una transacción, `eXtreme Scale` llama al método de aplicación de la interfaz `Evictor`. Ya no se conserva ningún bloqueo de transacción que se adquirieron al completarse la transacción. Varias hebras podrán potencialmente llamar al método de aplicación al mismo tiempo, y cada hebra podrá completar su propia transacción. Como los bloqueos de transacción se liberaron al completarse la transacción, el método de aplicación debe proporcionar su propia sincronización para garantizar la seguridad de las hebras en el método de aplicación.

La razón de implementar la interfaz `EvictorData` y utilizar el método `evictMapEntries` en lugar del método `evictEntries` es cerrar una ventana de sincronización potencial. Tenga en cuenta la siguiente secuencia de sucesos:

1. La transacción 1 se completa y llama al método de aplicación con `LogSequence` que elimina la entrada de correlación para la clave 1.
2. La transacción 2 se completa y llama al método de aplicación con `LogSequence` que inserta una nueva entrada de correlación para la clave 1. Es decir, la transacción 2 vuelve a crear la entrada de correlación que la transacción 1 eliminó.

Puesto que el desalojador se ejecuta de forma asíncrona en las hebras que ejecutan las transacciones, es posible que, cuando el desalojador decida desalojar la clave 1, desaloje la entrada de correlación que existió antes de la finalización de la transacción 1, o desaloje la entrada de correlación recreada por la transacción 2. Para eliminar las ventanas de sincronización y eliminar la duda sobre qué versión de la entrada de correlación de la clave 1 va a eliminar el desalojador, implemente la interfaz `EvictorData` mediante el objeto que se pasa al método `setEvictorData`.

Use la misma instancia EvictorData para la vida de una entrada de correlación. Cuando dicha entrada de correlación se elimina y después otra transacción la recrea, el desalojador debe utilizar una instancia nueva de implementación de EvictorData. Al utilizar la implementación de EvictorData y el método evictMapEntries, el desalojador puede garantizar que se desaloje la entrada de correlación sólo si la entrada de memoria caché asociada con la entrada de correlación contiene la instancia EvictorData correcta.

Las interfaces Evictor y EvictionEventCallback permiten a una aplicación conectar un desalojador que implemente un algoritmo definido por el usuario para el desalojo. El siguiente fragmento de código ilustra cómo puede implementarse el método initialize de la interfaz Evictor:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
import java.util.LinkedList;
// Variables de instancia
private BackingMap bm;
private EvictionEventCallback evictorCallback;
private LinkedList queue;
private Thread evictorThread;
public void initialize(BackingMap map, EvictionEventCallback callback)
{
    bm = map;
    evictorCallback = callback;
    queue = new LinkedList();
    // generar hebra de desalojador
    evictorThread = new Thread( this );
    String threadName = "MyEvictorForMap-" + bm.getName();
    evictorThread.setName( threadName );
    evictorThread.start();
}
```

Este código guarda las referencias a los objetos de devolución de llamada y correlación en variables de instancia de modo que estén disponibles para los métodos de aplicación y destrucción. En este ejemplo, se utiliza una lista enlazada que se usa como cola FIFO (primero en entrar, primero en salir) para implementar un algoritmo LRU (menos utilizado recientemente). Se genera una hebra y se mantiene una referencia a la hebra como variable de instancia. Al mantener esta referencia, el método de destrucción puede interrumpir y terminar la hebra generada.

Ignore los requisitos de sincronización para garantizar la seguridad de las hebras del código. El siguiente fragmento de código ilustra cómo puede implementarse el método de aplicación de la interfaz Evictor:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.EvictorData;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;

public void apply(LogSequence sequence)
{
    Iterator iter = sequence.getAllChanges();
    while(iter.hasNext())
    {
        LogElement elem = (LogElement)iter.next();
        Object key = elem.getKey();
        LogElement.Type type = elem.getType();
        if ( type == LogElement.INSERT )
        {
            // proceso de inserción aquí. Añadir en la parte delantera de la
cola LRU.
            EvictorData data = new EvictorData(key);
            evictorCallback.setEvictorData(key, data);
            queue.addFirst( data );
        }
        else if ( type == LogElement.UPDATE || type == LogElement.FETCH || type == LogElement.TOUCH )
        {
            // proceso de actualización aquí. Mover el objeto EvictorData a
// la parte delantera de la cola.

```



```

int queueSize = queue.size();
// Desalojar entradas si el tamaño de la cola sobrepasa
// el tamaño máximo. El tamaño máximo es
// otra propiedad del desalojador.
int numToEvict = queueSize - maxSize;
if ( numToEvict > 0 )
{
    // Eliminar de la parte final de la cola ya que es la
    // entrada menos utilizada recientemente.
    List evictList = new ArrayList( numToEvict );
    while( queueSize > ivMaxSize )
    {
        EvictorData data = null;
        try
        {
            EvictorData data = (EvictorData) queue.removeLast();
            evictList.add( data );
            queueSize = queue.size();
        }
        catch ( NoSuchElementException nse )
        {
            // La cola está vacía.
            queueSize = 0;
        }
    }
    // Solicitar desalojo si la lista de claves no está vacía.
    if ( ! evictList.isEmpty() )
    {
        evictorCallback.evictMapEntries( evictList );
    }
}
}
catch ( InterruptedException e )
{
    continueToRun = false;
}
} // fin bucle while
} // fin de método de ejecución.

```

Interfaz RollBackEvictor opcional

Un plug-in Evictor puede implementar opcionalmente la interfaz `com.ibm.websphere.objectgrid.plugins.RollbackEvictor`. Al implementarla, puede invocarse un desalojador no sólo cuando se confirman las transacciones, sino cuando éstas se retrotraen.

```

public interface RollbackEvictor
{
    void rollingBack( LogSequence ls );
}

```

El método de aplicación sólo se llama si se confirma una transacción. Si se retrotrae una transacción y el desalojador implementa la interfaz `RollbackEvictor`, se invoca al método `rollingBack`. Si no se implementa la interfaz `RollbackEvictor` y se retrotrae la transacción, el método de aplicación y el método `rollingBack` no se llaman.

Procedimientos recomendados para optimizar el rendimiento del desalojador de plug-in

Si utiliza desalojadores de plug-in, éstos no se activarán hasta que los cree y los asocie con una correlación de respaldo. Los procedimientos recomendados siguientes aumentarán el rendimiento para los desalojadores LFU (utilizados con menor frecuencia) y los desalojadores LRU (menos utilizados recientemente).

Desalojador LFU (utilizados con menor frecuencia)

El concepto de un desalojador LFU es eliminar entradas de la correlación que menos se utilizan. Las entradas de la correlación se expanden en un volumen establecido de almacenamientos dinámicos. A medida que aumenta el uso de una entrada en memoria caché determinada, se ordena en una posición más alta en el almacenamiento dinámico. Cuando el desalojador intenta un conjunto de desalojos,

elimina sólo las entradas de la memoria caché ubicadas por debajo de un punto específico del almacenamiento dinámico binario. Como resultado, se desalojan las entradas usadas con menor frecuencia.

Desalojador LRU (menos utilizados recientemente)

El desalojador LRU sigue los mismos conceptos que el desalojador LFU con unas pocas diferencias. La diferencia principal es que el LRU utiliza una cola FIFO (primero en entrar, primero en salir) en lugar de un conjunto de almacenamientos dinámicos binarios. Cada vez que se accede a una entrada de la memoria caché, ésta se mueve a la cabeza de la cola. En consecuencia, la parte inicial de la cola contiene las entradas de correlación utilizadas más recientemente y la parte final empieza con las entradas de correlación menos utilizadas recientemente. Por ejemplo, la entrada de la memoria caché A se utiliza 50 veces, y la entrada de la memoria caché B se utiliza sólo una después de la entrada de la memoria caché A. En esta situación, la entrada de la memoria caché B se coloca en la parte delantera de la cola porque se ha utilizado recientemente, mientras que la entrada de la memoria caché A se coloca al final de la cola. El desalojador LRU desaloja las entradas de la memoria caché que están en la parte final de la cola, que son las entradas de correlación menos usadas recientemente.

Propiedades de LFU y LRU y procedimientos recomendados para mejorar el rendimiento

Número de almacenamientos dinámicos

Al utilizar el desalojador LFU, todas las entradas de la memoria caché de una correlación determinada se ordenan según el número de almacenamientos dinámicos que se especifique, lo que mejora drásticamente el rendimiento e impide que se sincronicen los desalojos en un almacenamiento dinámico binario que contenga todas las ordenaciones de la correlación. A mayor número de almacenamientos dinámicos menor es el tiempo necesario para reordenarlos porque cada uno de ellos tiene menos entradas. Establezca el número de almacenamientos dinámicos en 10% del número de entradas en BaseMap.

Número de colas

Al utilizar el desalojador LRU, todas las entradas de la memoria caché de una correlación determinada se ordenan según el número de colas LRU que se especifique, lo que mejora drásticamente el rendimiento e impide que se sincronicen los desalojos en una cola que contenga todas las ordenaciones de la correlación. Establezca el número de colas en 10% del número de entradas en BaseMap.

Propiedad MaxSize

Cuando un desalojador LFU o LRU empieza a desalojar entradas, utiliza la propiedad de desalojador MaxSize para determinar cuántos almacenamientos dinámicos binarios o elementos de cola LRU va a desalojar. Por ejemplo, presuponga que establece el número de almacenamientos dinámicos o colas en unas 10 entradas de correlación en cada cola de correlación. Si la propiedad MaxSize se establece en 7, el desalojador desaloja 3 entradas de cada almacenamiento dinámico u objeto de cola para que el tamaño del almacenamiento dinámico o de la cola se reduzca a 7. El desalojador sólo desaloja entradas de correlación de un almacenamiento dinámico o cola cuando en éstos hay un valor mayor que el de la propiedad MaxSize. Establezca MaxSize en 70% del tamaño de

la cola o almacenamiento dinámico. En este ejemplo, el valor se estableció en 7. Puede obtener un tamaño aproximado de cada almacenamiento dinámico o cola si divide el número de entradas BaseMap entre el número de almacenamientos dinámicos o colas utilizado.

Propiedad SleepTime

Un desalojador no elimina constantemente entradas de la correlación. Está inactivo durante un tiempo determinado, y sólo comprueba la correlación cada n número de segundos, donde n equivale a la propiedad SleepTime. Esta propiedad afecta positivamente al rendimiento: ejecutar un barrido de desalojo suele disminuir el rendimiento debido a los recursos que se necesitan para procesarlos. Sin embargo, no utilizar el desalojador con frecuencia puede generar una correlación que tiene entradas que no son necesarias. Una correlación con muchas entradas que no necesita puede afectar negativamente a los requisitos de memoria y al proceso de los recursos de la correlación. Para la mayoría de las correlaciones se recomienda establecer el intervalo de barrido de desalojo en 15 segundos. Si la correlación se graba con frecuencia y se utiliza a una velocidad de transacción alta, conviene establecer el tiempo en un valor más bajo. Si se accede a la correlación con poca frecuencia, establezca el tiempo en un valor más alto.

Ejemplo

El ejemplo siguiente define una correlación, crea un desalojador LFU nuevo, establece las propiedades del desalojador y establece la correlación que va a utilizar el desalojador:

```
//Usar
ObjectGridManager para crear/obtener ObjectGrid. Consulte el
// apartado ObjectGridManger
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");

//Establecer propiedades presuponiendo 50.000 entradas de correlación
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Usar el desalojador LRU es muy parecido a usar un desalojador LFU. A continuación se muestra un ejemplo:

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("SomeMap");

//Establecer propiedades presuponiendo 50.000 entradas de correlación
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Observe que sólo hay dos líneas distintas al del ejemplo del desalojador LFU.

Plug-ins para transformar los objetos almacenados en memoria caché

Considere transformar los objetos almacenados en memoria caché para aumentar el rendimiento de la memoria caché. Puede utilizar el plug-in de ObjectTransformer cuando el uso del procesador es elevado. Se invierte hasta un 60 o 70 por ciento del tiempo total del procesador en serializar y copiar las entradas. Mediante la

implementación del plug-in de ObjectTransformer, puede serializar y deserializar los objetos con su propia implementación. Puede utilizar un plug-in de CollisionArbiter para definir cómo se tratan las colisiones de cambio en los dominios.

Desarrollo de árbitros personalizados para la réplica con varios maestros

Se podrían producir colisiones de cambio si se pueden cambiar en dos lugares a la vez los mismos registros. En una topología de réplica con varios maestros, los dominios detectan las colisiones de forma automática. Cuando un dominio detecta una colisión, invoca un árbitro. Habitualmente, las colisiones se resuelven utilizando el árbitro de colisión predeterminado. No obstante, una aplicación puede proporcionar un árbitro de colisión personalizado.

Acerca de esta tarea

Si un dominio recibe una entrada replicada que colisiona con un registro local, el árbitro predeterminado utiliza los cambios del dominio con nombre de menor valor léxico. Por ejemplo, si el dominio A y B generan un conflicto de un registro, el cambio del dominio B se pasa por alto. El dominio A conserva su versión y el registro del dominio B cambia para coincidir con el registro del dominio A. Los nombres de dominio se convierten a mayúsculas para la comparación.

Una opción alternativa para la topología de réplica con varios maestros es llamar en un plug-in de colisión personalizado para determinar el resultado. Estas instrucciones esbozan cómo desarrollar un árbitro de colisión personalizado y configurar una topología de réplica con varios maestros para utilizarla.

Procedimiento

1. Desarrolle un árbitro de colisión personalizado e intégrele en la aplicación.

La clase debe implementar la interfaz:

```
com.ibm.websphere.objectgrid.revision.CollisionArbiter
```

Un plug-in de colisión tiene tres opciones para determinar el resultado de una colisión. Puede seleccionar la copia local o puede proporcionar una versión revisada de la entrada. Un dominio proporciona la información siguiente a un árbitro de colisión personalizado:

- La versión foránea del registro
- La versión local del registro
- Un objeto Session que se debe utilizar para crear la versión revisada de la entrada que ha entrado en colisión

El método del plug-in devuelve un objeto que indica su determinación. El método invocado por el dominio para llamar al plug-in debe devolver true o false, donde false significa pasar por alto la colisión – la versión local permanece sin cambios y eXtreme Scale olvidará que jamás vio la versión foránea. El método devuelve un valor true si el método ha utilizado la sesión proporcionada para crear una versión nueva fusionada del registro, con lo que se reconcilia el cambio.

2. En el archivo objectgrid.xml, especifique utilizar el plug-in de árbitro personalizado.

El id debe ser "CollisionArbiter."

```
<dcg:objectGrid name="revisionGrid" txTimeout="10">
  <dcg:bean className="com.you.your_application.
    CustomArbiter" id="CollisionArbiter">
```

```
<dg:property name="property" type="java.lang.String"
  value="propertyValue"/>
</dg:bean>
</dg:objectGrid>
```

Plug-in ObjectTransformer

Con el plug-in ObjectTransformer puede serializar, deserializar y copiar objetos en la memoria caché para mejorar el rendimiento.

Si observa problemas de rendimiento con el uso del procesador, añada un plug-in ObjectTransformer a cada correlación. Si no proporciona un plug-in ObjectTransformer, se emplea entre un 60 y un 70 por ciento del tiempo total de procesador en serializar y copiar entradas.

Finalidad

Con el plug-in ObjectTransformer, las aplicaciones pueden proporcionar métodos personalizados para las siguientes operaciones:

- Serializar o deserializar la clave de una entrada.
- Serializar o deserializar el valor de una entrada.
- Copiar una clave o valor de una entrada.

Si no se proporciona ningún plug-in ObjectTransformer, debe poder serializar las claves y los valores porque ObjectGrid utiliza una secuencia de serialización y deserialización para copiar los objetos. Éste es un método costoso, por lo que conviene utilizar un plug-in ObjectTransformer si el rendimiento es crítico. La operación de copia se produce cuando una aplicación busca un objeto en una transacción por primera vez. Puede evitar esta copia si establece la modalidad de copia de la correlación en NO_COPY o puede reducir la copia si establece la modalidad de copia en COPY_ON_READ. Optimice la operación de copia cuando sea necesario para la aplicación; para ello, proporcione un método de copia personalizada en este plug-in. Dicho plug-in puede reducir la sobrecarga de copia del 65 al 70 por ciento a 2/3 del porcentaje del tiempo total del procesador.

Las implementaciones predeterminadas del método copyKey y copyValue intentan, en primer lugar, utilizar el método clone, si se ha proporcionado el método. Si no se ha proporcionado ninguna implementación del método clone, la implementación toma el valor predeterminado de la serialización.

La serialización de objetos también se utiliza directamente cuando eXtreme Scale se ejecuta en la modalidad distribuida. El LogSequence utiliza el plug-in ObjectTransformer para ayudar a serializar claves y valores antes de transmitir los cambios a sus iguales en ObjectGrid. Debe actuar con detenimiento cuando proporcione un método de serialización personalizado, en lugar de utilizar la serialización del Java Developer Kit incorporada. La creación de versiones de objetos es un asunto complejo y podría tener problemas con la compatibilidad de las versiones si no se asegura de que sus métodos personalizados se han diseñado para la creación de versiones.

La siguiente lista describe cómo eXtreme Scale intenta serializar tanto las claves, como los valores:

- Si se ha escrito y conectado un plug-in ObjectTransformer personalizado, eXtreme Scale llama a los métodos de la interfaz ObjectTransformer para serializar las claves y los valores y obtener copias de claves y valores de objeto.

- Si no se utiliza un plug-in ObjectTransformer personalizado, eXtreme Scale serializa y deserializa los valores de acuerdo con el valor predeterminado. Si se utiliza el plug-in predeterminado, cada objeto se implementa como externalizable o se implementa como serializable.
 - Si el objeto soporta la interfaz Externalizable, se llama al método writeExternal. Los objetos que se implementan como externalizables obtienen un mejor rendimiento.
 - Si el objeto no soporta la interfaz Externalizable e implementa la interfaz Serializable, el objeto se guarda mediante el método ObjectOutputStream.

Utilización de la interfaz ObjectTransformer

Un objeto ObjectTransformer debe implementar la interfaz ObjectTransformer y seguir las convenciones comunes de plug-in de ObjectGrid.

Se utilizan dos enfoques, configuración mediante programa y configuración de XML, para añadir un objeto ObjectTransformer a la configuración de BackingMap tal como se indica a continuación.

Conexión de ObjectTransformer mediante la configuración del XML

Imagine que el nombre de clase de la implementación ObjectTransformer es la clase com.company.org.MyObjectTransformer. Esta clase implementa la interfaz ObjectTransformer. Una implementación de ObjectTransformer se puede configurar utilizando 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="myGrid">
      <backingMap name="myMap" pluginCollectionRef="myMap" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="myMap">
      <bean id="ObjectTransformer" className="com.company.org.MyObjectTransformer" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

Conexión mediante programa de un objeto ObjectTransformer

El siguiente fragmento de código crea el objeto ObjectTransformer personalizado y lo añade a un BackingMap:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap backingMap = myGrid.getMap("myMap");
MyObjectTransformer myObjectTransformer = new MyObjectTransformer();
backingMap.setObjectTransformer(myObjectTransformer);
```

Escenarios de uso de ObjectTransformer

Puede utilizar un plug-in ObjectTransformer en las situaciones siguientes:

- Objeto no serializable
- Objeto serializable pero mejora el rendimiento de serialización
- Copia de clave o valor

En el ejemplo siguiente, se utiliza ObjectGrid para almacenar la clase Stock:

```

/**
 * Objeto Stock para ObjectGrid
 *
 */
public class Stock implements Cloneable {
    String ticket;
    double price;
    String company;
    String description;
    int serialNumber;
    long lastTransactionTime;
    /**
     * @devuelve la descripción.
     */
    public String getDescription() {
        return description;
    }
    /**
     * @establecer el parámetro description para la descripción que se va a establecer.
     */
    public void setDescription(String description) {
        this.description = description;
    }
    /**
     * @devuelve lastTransactionTime.
     */
    public long getLastTransactionTime() {
        return lastTransactionTime;
    }
    /**
     * @establecer el parámetro lastTransactionTime, que es el valor
    lastTransactionTime que se va a establecer.
     */
    public void setLastTransactionTime(long lastTransactionTime) {
        this.lastTransactionTime = lastTransactionTime;
    }
    /**
     * @devuelve el precio.
     */
    public double getPrice() {
        return price;
    }
    /**
     * @establecer el parámetro price, el precio que se va a establecer.
     */
    public void setPrice(double price) {
        this.price = price;
    }
    /**
     * @devuelve el número de serie.
     */
    public int getSerialNumber() {
        return serialNumber;
    }
    /**
     * @establecer el parámetro serialNumber, que es el valor serialNumber que se va a establecer.
     */
    public void setSerialNumber(int serialNumber) {
        this.serialNumber = serialNumber;
    }
    /**
     * @devuelve el ticket.
     */
    public String getTicket() {
        return ticket;
    }
    /**
     * @establecer el parámetro ticket, el ticket que se va a establecer.
     */
    public void setTicket(String ticket) {
        this.ticket = ticket;
    }
    /**
     * @devuelve la empresa.
     */
    public String getCompany() {
        return company;
    }
}
/**

```

```

    * @establecer el parámetro company, la empresa que se va a establecer.
    */
    public void setCompany(String company) {
        this.company = company;
    }
    //clone
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}

```

Puede escribir una clase de ObjectTransformer para la clase Stock:

```

/**
 * Implementación personalizada de ObjectGrid ObjectTransformer para el objeto stock
 *
 */
public class MyStockObjectTransformer implements ObjectTransformer {
    /* (non-Javadoc)
    * @ever com.ibm.websphere.objectgrid.plugins.ObjectTransformer#serializeKey
    * (java.lang.Object,
    * java.io.ObjectOutputStream)
    */
    public void serializeKey(Object key, ObjectOutputStream stream) throws IOException {
        String ticket= (String) key;
        stream.writeUTF(ticket);
    }

    /* (non-Javadoc)
    * @ever com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#serializeValue(java.lang.Object,
    java.io.ObjectOutputStream)
    */
    public void serializeValue(Object value, ObjectOutputStream stream) throws IOException {
        Stock stock= (Stock) value;
        stream.writeUTF(stock.getTicket());
        stream.writeUTF(stock.getCompany());
        stream.writeUTF(stock.getDescription());
        stream.writeDouble(stock.getPrice());
        stream.writeLong(stock.getLastTransactionTime());
        stream.writeInt(stock.getSerialNumber());
    }

    /* (non-Javadoc)
    * @ever com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#inflateKey(java.io.ObjectInputStream)
    */
    public Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException {
        String ticket=stream.readUTF();
        return ticket;
    }

    /* (non-Javadoc)
    * @ever com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#inflateValue(java.io.ObjectInputStream)
    */
    public Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException {
        Stock stock=new Stock();
        stock.setTicket(stream.readUTF());
        stock.setCompany(stream.readUTF());
        stock.setDescription(stream.readUTF());
        stock.setPrice(stream.readDouble());
        stock.setLastTransactionTime(stream.readLong());
        stock.setSerialNumber(stream.readInt());
        return stock;
    }

    /* (non-Javadoc)
    * @ever com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#copyValue(java.lang.Object)
    */
    public Object copyValue(Object value) {
        Stock stock = (Stock) value;
        try {
            return stock.clone();
        }
        catch (CloneNotSupportedException e)
        {
            // mostrar mensaje de excepción }
        }
    }

    /* (non-Javadoc)
    * @ever com.ibm.websphere.objectgrid.plugins.
    ObjectTransformer#copyKey(java.lang.Object)
    */
    public Object copyKey(Object key) {

```

```

String ticket=(String) key;
String ticketCopy= new String (ticket);
return ticketCopy;
}
}

```

A continuación, conecte esta clave MyStockObjectTransformer personalizada a BackingMap:

```

ObjectGridManager ogf=ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogf.getObjectGrid("NYSE");
BackingMap bm = og.defineMap("NYSEStocks");
MyStockObjectTransformer ot = new MyStockObjectTransformer();
bm.setObjectTransformer(ot);

```

Rendimiento de serialización

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

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

Escribir un ObjectTransformer para cada BackingMap

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

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

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

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

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

Uso de entidades

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

Serialización personalizada

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

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

Matrices de bytes

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

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

Procedimientos recomendados para la interfaz ObjectTransformer

La interfaz `ObjectTransformer` utiliza devoluciones de llamadas a la aplicación para proporcionar implementaciones personalizadas de operaciones comunes y costosas, como la serialización y la copia exacta de objetos.

Visión general

Si desea detalles sobre la interfaz `ObjectTransformer`, consulte “Plug-in `ObjectTransformer`” en la página 230. Desde un punto de vista de rendimiento, y a partir de la información del método `CopyMode` que está en el tema de procedimientos recomendados del método `CopyMode`, eXtreme Scale copia claramente los valores para todos los casos excepto cuando se utiliza la modalidad `NO_COPY`. El mecanismo de copia predeterminado que se emplea en eXtreme Scale es la serialización, que se sabe que es una operación costosa. La interfaz `ObjectTransformer` se utiliza en esta situación. La interfaz `ObjectTransformer` utiliza las devoluciones de llamada a la aplicación para proporcionar una implementación personalizada de las operaciones comunes y costosas como, por ejemplo, la serialización de objeto y la copia exacta de objetos.

Una aplicación puede proporcionar una implementación de la interfaz `ObjectTransformer` en una correlación y, a continuación, eXtreme Scale delega en los métodos de este objeto y se basa en la aplicación para proporcionar una versión optimizada de cada método de la interfaz. La interfaz `ObjectTransformer` actúa del modo siguiente:

```
public interface ObjectTransformer {
    void serializeKey(Object key, ObjectOutputStream stream) throws IOException;
    void serializeValue(Object value, ObjectOutputStream stream) throws IOException;
    Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object copyValue(Object value);
    Object copyKey(Object key);
}
```

Puede asociar una interfaz `ObjectTransformer` con una `BackingMap` utilizando el siguiente código de ejemplo:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ajuste de la serialización e inflado de objetos

Normalmente la serialización de objetos es la consideración de rendimiento más importante con eXtreme Scale, que utiliza el mecanismo serializable predeterminado, si la aplicación no proporciona un plug-in `ObjectTransformer`. Una aplicación puede proporcionar implementaciones de `readObject` y `writeObject` `Serializable` o tener objetos que implementen la interfaz `Externalizable`, que es unas 10 veces más rápida. Si no se pueden modificar los objetos de la correlación, entonces una aplicación puede asociar una interfaz `ObjectTransformer` a la `ObjectMap`. Se proporcionan los métodos `serialize` e `inflate` para permitir que la aplicación proporcione código personalizado para optimizar estas operaciones, dado el gran impacto que tienen en el rendimiento del sistema. El método `serialize` serializa el objeto en la corriente de datos proporcionada. El método `inflate` proporciona la corriente de datos de entrada y espera que la aplicación cree el objeto, lo infle utilizando los datos de la corriente y devuelva el objeto. Las implementaciones de los métodos `serialize` e `inflate` deben duplicarse entre sí.

Ajuste de operaciones de copia exacta

Después de que una aplicación reciba un objeto de un `ObjectMap`, eXtreme Scale realiza una copia exacta del valor de objeto para garantizar que la copia de la correlación `BaseMap` mantiene la integridad de datos. La aplicación puede entonces modificar el valor de objeto sin riesgos. Cuando se confirma la transacción, se actualiza la copia del valor de objeto de la correlación `BaseMap` al nuevo valor modificado y se detiene la aplicación, que utilizará el valor de ahí en adelante. Podría haber vuelto a copiar el objeto en la fase de confirmación para realizar una copia privada. Sin embargo, en este caso, el coste del rendimiento de esta acción se ha compensado indicando al programador de aplicaciones que no utilice el valor después de que se confirme la transacción. El `ObjectTransformer` predeterminado intenta utilizar un clon o un par de métodos `serialize` e `inflate` para generar una copia. El par de métodos `serialize` e `inflate` es el peor caso de rendimiento. Si la creación de perfiles revela que usar los métodos `serialize` e `inflate` es un problema para la aplicación, escriba un método `clone` apropiado para crear una copia exacta. Si no puede alterar la clase, cree un plug-in `ObjectTransformer` personalizado e implemente métodos `copyValue` y `copyKey` más eficaces.

Plug-ins para el mantenimiento de versiones y la comparación de objetos de memoria caché

Utilice el plug-in `OptimisticCallback` para personalizar las operaciones de creación de versiones y comparación de los objetos de la memoria caché cuando se utiliza la estrategia de bloqueo optimista.

Puede proporcionar un objeto de devolución de llamada optimista conectable que implementa la interfaz `com.ibm.websphere.objectgrid.plugins.OptimisticCallback`. En el caso de correlaciones de entidad, se configura automáticamente un plug-in `OptimisticCallback` de alto rendimiento.

Finalidad

Utilice la interfaz `OptimisticCallback` para proporcionar operaciones de comparación optimista para los valores de una correlación. Es necesario un plug-in `OptimisticCallback` al utilizar la estrategia de bloqueo optimista. El producto proporciona una implementación de `OptimisticCallback` predeterminada. Sin embargo, por lo general, la aplicación debe conectar su propia implementación de la interfaz `OptimisticCallback`.

Implementación predeterminada

La infraestructura de eXtreme Scale proporciona una implementación predeterminada de la interfaz `OptimisticCallback` que se utiliza si la aplicación no conecta un objeto `OptimisticCallback` proporcionado para la aplicación. La implementación predeterminada siempre devuelve el valor especial de `NULL_OPTIMISTIC_VERSION` como el objeto de versión del valor y nunca actualiza el objeto de versión. Esta acción hace que la comparación optimista sea una función "sin operación". En la mayoría de los casos, conviene que no se produzca la función "sin operación" cuando utilice la estrategia de bloqueo optimista. Las aplicaciones deben implementar la interfaz y conectar sus propias implementaciones `OptimisticCallback` de modo que no se utilice la implementación predeterminada. No obstante, existe un escenario donde resulta útil la implementación `OptimisticCallback` predeterminada. Observe la situación siguiente:

- Se ha conectado un cargador para la correlación de respaldo.
- El cargador sabe cómo realizar la comparación optimista sin ayuda de un plug-in `OptimisticCallback`.

¿Cómo puede realizar el cargador una creación de versiones optimista sin la ayuda de un objeto `OptimisticCallback`? El cargador conoce el objeto de clase de valor y sabe qué campo del objeto de valor se utiliza como valor de creación de versiones optimista. Por ejemplo, imagine que se utiliza la interfaz siguiente para el objeto de valor de la correlación de empleados.

```
public interface Employee
{
    // Número de secuencia utilizado para la creación de versiones optimista.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Otros métodos get/set para otros campos del objeto Employee.
}
```

En este ejemplo, el cargador sabe que puede utilizar el método `getSequenceNumber` para obtener la información de la versión actual para un objeto de valor `Employee`. El cargador incrementa el valor devuelto para generar un nuevo número de versión antes de actualizar el almacenamiento persistente con

el nuevo valor Employee. Para un cargador JDBC (Java DataBase Connectivity), se utiliza el número de secuencia actual de la cláusula WHERE de una sentencia de SQL UPDATE sobrecualificada, y utiliza el nuevo número de secuencia generado para establecer la columna de número de secuencia en el nuevo valor de número de secuencia. Otra posibilidad es que el cargador utilice una función, que proporciona el programa de fondo, que actualiza automáticamente una columna oculta que puede utilizarse para la creación de versiones optimista.

En algunas situaciones, posiblemente se puede utilizar un procedimiento almacenado o un desencadenante que ayuda a mantener una columna que aloja información sobre la creación de versiones. Si el cargador utiliza una de estas técnicas para mantener la información de la creación de versiones optimista, la aplicación no necesita proporcionar una implementación OptimisticCallback. Se puede utilizar la implementación predeterminada de OptimisticCallback en este escenario porque el cargador puede manejar la creación de versiones optimista sin la ayuda de un objeto OptimisticCallback.

Implementación predeterminada de entidades

Las entidades se almacenan en ObjectGrid mediante objetos de tuple. El comportamiento predeterminado de la implementación OptimisticCallback es similar al comportamiento para las correlaciones sin entidades. Sin embargo, el campo de versión de la entidad se identifica a través del uso de la anotación @Version o el atributo de versión en el archivo XML de descriptor de la entidad.

El atributo de versión puede ser de uno de los tipos siguientes: int, Integer, short, Short, long, Long o java.sql.Timestamp. Una entidad sólo debe tener un atributo de versión definido. Establezca el atributo de versión sólo durante la construcción. Después de persistir la entidad, el valor del atributo de versión no se debe modificar.

Si no se configura el atributo de versión y se utiliza la estrategia de bloqueo optimista, se crea una versión implícitamente de todo el tuple mediante el estado completo del tuple, que es más costoso.

En el ejemplo siguiente, la entidad Employee tiene un atributo de versión de tipo long denominado SequenceNumber:

```
@Entity
public class Employee
{
    private long sequence;
    // Número de secuencia utilizado para la creación de versiones optimista.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Otros métodos get/set para otros campos del objeto Employee.
}
```

Escritura de un plug-in OptimisticCallback

Un plug-in OptimisticCallback debe implementar la interfaz OptimisticCallback y seguir los convenios comunes de plug-in de ObjectGrid. Consulte la interfaz OptimisticCallback en la documentación de la API si desea más información.

En la lista siguiente se proporciona una descripción o consideración de cada uno de los métodos de la interfaz `OptimisticCallback`:

NULL_OPTIMISTIC_VERSION

Este valor especial es devuelto por el método `getVersionedObjectForValue` si la implementación de `OptimisticCallback` no requiere ninguna comprobación de versiones. La implementación del plug-in incorporada de la clase `com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback` utiliza este valor porque la creación de versiones está inhabilitada cuando se especifica esta implementación de plug-in.

Método `getVersionedObjectForValue`

El método `getVersionedObjectForValue` podría devolver una copia del valor o un atributo del valor que se puede utilizar para la creación de versiones. Este método se llama siempre que se asocia un objeto con una transacción. Si no se conecta ningún cargador a la correlación de respaldo, ésta utiliza este valor durante la fase de confirmación para llevar a cabo una comparación de versiones optimista. La correlación de respaldo utiliza la comparación de versiones optimista para asegurarse de que la versión no ha cambiado desde la primera que vez que esta transacción accedió a la entrada de la correlación que fue modificada por esta transacción. Si otra transacción hubiera modificado la versión de esta entrada de correlación, se produciría una anomalía en la comparación de versiones y la correlación de respaldo mostraría una excepción `OptimisticCollisionException` que forzaría la retrotracción de la transacción. Si hay un cargador conectado, la correlación de respaldo no utiliza la información de creación de versiones optimista. En su lugar, el cargador deberá realizar una comparación de versiones optimista y actualizar la información de la creación de versiones cuando sea necesario. El cargador suelte obtener el objeto de versiones inicial del `LogElement` pasado al método `batchUpdate` del cargador, que se llama cuando se produce una operación de vaciado o se confirma una transacción.

El código siguiente muestra la implementación que utiliza el objeto `EmployeeOptimisticCallbackImpl`:

```
public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}
```

Tal como se demuestra en el ejemplo anterior, se devuelve el atributo `sequenceNumber` en un objeto `java.lang.Long` tal como espera el cargador, que implica que la misma persona que escribió el cargador, también escribió la implementación de `EmployeeOptimisticCallbackImpl`, o bien trabajó estrechamente con la persona que implementó el método `EmployeeOptimisticCallbackImpl`, por ejemplo, acordó el valor devuelto por el método `getVersionedObjectForValue`. El plug-in predeterminado `OptimisticCallback` devuelve el valor especial `NULL_OPTIMISTIC_VERSION` como el objeto de versión.

Método updateVersionedObjectForValue

Este método se llama siempre que una transacción ha actualizado un valor y se necesita un nuevo objeto de versión. Si el método `getVersionedObjectForValue` devuelve un atributo del valor, este método suele actualizar el valor de atributo con un nuevo objeto de versión. Si el método `getVersionedObjectForValue` devuelve una copia del valor, este método normalmente no completa ninguna acción. El plug-in predeterminado `OptimisticCallback` no completa ninguna acción con este método porque la implementación predeterminada de `getVersionedObjectForValue` siempre devuelve el valor especial `NULL_OPTIMISTIC_VERSION` como el objeto de la versión. El siguiente ejemplo muestra la implementación utilizada por el objeto `EmployeeOptimisticCallbackImpl` que se utiliza en la sección `OptimisticCallback`:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Tal como se demuestra en el ejemplo anterior, el atributo `sequenceNumber` se incrementa por uno, de forma que la próxima vez que se llama al método `getVersionedObjectForValue`, el valor `java.lang.Long` devuelto tiene un valor largo que es el valor del número de secuencia original más uno, por ejemplo, es el valor de la siguiente versión para esta instancia de empleado. Este ejemplo implica que la misma persona que escribió el cargador escribió `EmployeeOptimisticCallbackImpl` o bien trabajó estrechamente con la persona que implementó `EmployeeOptimisticCallbackImpl`.

Método serializeVersionedValue

Este método escribe el valor con versión en la corriente especificada. En función de la implementación, el valor con versión puede utilizarse para identificar colisiones de actualización optimista. En algunas implementaciones, el valor con versión es una copia del valor original. Otras implementaciones podrían tener un número de secuencia o algún otro objeto para indicar la versión del valor. Puesto que la implementación real se desconoce, este método se proporciona para realizar la serialización apropiada. La implementación predeterminada llama al método `writeObject`.

Método inflateVersionedValue

Este método toma la versión serializada del valor con versión y devuelve el objeto de valor con versión real. En función de la implementación, el valor con versión puede utilizarse para identificar colisiones de actualización optimista. En algunas implementaciones, el valor con versión es una copia del valor original. Otras implementaciones podrían tener un número de secuencia o algún otro objeto para indicar la versión del valor. Puesto que se desconoce la implementación real, este método se proporciona para realizar la deserialización apropiada. La implementación predeterminada llama al método `readObject`.

Uso del objeto OptimisticCallback proporcionado por la aplicación

Dispone de dos enfoques para añadir un objeto OptimisticCallback proporcionado por la aplicación en la configuración de BackingMap: configuración de XML y configuración mediante programa.

Enfoque de configuración de XML para conectar un objeto OptimisticCallback

La aplicación puede utilizar un archivo XML para conectar su objeto OptimisticCallback tal como se muestra en el siguiente ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
  <objectGrid name="grid1">
    <backingMap name="employees" pluginCollectionRef="employees" lockStrategy="OPTIMISTIC" />
  </objectGrid>
</objectGrids>

<backingMapPluginCollections>
  <backingMapPluginCollection id="employees">
    <bean id="OptimisticCallback" className="com.xyz.EmployeeOptimisticCallbackImpl" />
  </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Conectar mediante programa un objeto OptimisticCallback

El siguiente ejemplo demuestra cómo una aplicación puede conectar mediante programa un objeto OptimisticCallback para la correlación de respaldo del empleado en la instancia local del ObjectGrid grid1.

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

Plug-ins para proporcionar escuchas de sucesos

Puede utilizar los plug-ins ObjectGridEventListener y MapEventListener para configurar notificaciones para diversos sucesos en la memoria caché de eXtreme Scale. Los plug-ins de escucha se registran con una instancia de ObjectGrid o BackingMap como otros plug-ins eXtreme Scale y añaden puntos de integración y personalización para las aplicaciones y los proveedores de memoria caché.

Plug-in ObjectGridEventListener

Un plug-in ObjectGridEventListener proporciona sucesos de ciclo de vida de eXtreme Scale para la instancia, fragmentos y transacciones de ObjectGrid. Utilice el plug-in ObjectGridEventListener para recibir notificaciones cuando se producen sucesos significativos en un ObjectGrid. Estos sucesos incluyen la inicialización de ObjectGrid, el inicio de una transacción, la finalización de una transacción y la destrucción de un ObjectGrid. Para escuchar estos sucesos, cree una clase que implemente la interfaz ObjectGridEventListener y añádala a eXtreme Scale.

Para obtener más información sobre cómo grabar un plug-in `ObjectGridEventListener`, consulte “Plug-in `ObjectGridEventListener`” en la página 243. También puede hacer referencia a la documentación de la API si desea más información.

Adición y eliminación de instancias `ObjectGridEventListener`

Un `ObjectGrid` puede tener varios receptores de `ObjectGridEventListener`. Añada y elimine los escuchas utilizando los métodos `addListener`, `setEventListeners` y `removeEventListener` en la interfaz `ObjectGrid`. También puede registrar de forma declarativa los plug-ins `ObjectGridEventListener` con el archivo descriptor de `ObjectGrid`. Para obtener ejemplos, consulte “Plug-in `ObjectGridEventListener`” en la página 243.

Plug-in `MapEventListener`

Un plug-in `MapEventListener` proporciona notificaciones de devolución de llamada y cambios de estado de memoria caché significativos que se producen para una instancia `BackingMap`. Para ver detalles sobre cómo escribir un plug-in `MapEventListener`, consulte “Plug-in `MapEventListener`”. También puede hacer referencia a la documentación de la API si desea más información.

Adición y eliminación de instancias de `MapEventListener`

Un `eXtreme Scale` puede tener varios receptores de `MapEventListener`. Añada y elimine escuchas con los métodos `addMapEventListener`, `setMapEventListeners` y `removeMapEventListener` en la interfaz `BackingMap`. También puede registrar de forma declarativa los receptores de `MapEventListener` con el archivo descriptor de `ObjectGrid`. Para obtener ejemplos, consulte “Plug-in `MapEventListener`”.

Plug-in `MapEventListener`

Un plug-in `MapEventListener` proporciona notificaciones de devolución de llamada y cambios significativos de estado de memoria caché que se producen para un objeto `BackingMap`: cuando una correlación ha terminado la precarga o cuando se desaloja una entrada de la correlación. Un determinado plug-in `MapEventListener` es una clase personalizada que se escribe implementando la interfaz `MapEventListener`.

Convenios del plug-in `MapEventListener`

Cuando desarrolle un plug-in `MapEventListener`, debe seguir los convenios comunes de plug-in. Para obtener más información sobre los convenios de plug-in, consulte “Introducción a los plug-ins” en la página 211. Para otros tipos de plug-ins de escuchas, consulte “Plug-ins para proporcionar escuchas de sucesos” en la página 241.

Después de escribir una implementación de `MapEventListener`, puede conectarse a éste en la configuración de `BackingMap` a través de programa o con una configuración de XML.

Grabar una implementación de `MapEventListener`

La aplicación puede incluir una implementación del plug-in `MapEventListener`. El plug-in debe implementar la interfaz `MapEventListener` para recibir sucesos significativos sobre una correlación. Los sucesos se envían al plug-in

MapEventListener cuando una entrada se desaloja de la correlación y cuando finaliza la precarga de una correlación.

Conectar una implementación de MapEventListener utilizando XML

Se puede configurar una implementación de MapEventListener utilizando el XML. El siguiente XML debe aparecer en el archivo myGrid.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridconfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="myGrid">
      <backingMap name="myMap" pluginCollectionRef="myPlugins" />
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="myPlugins">
      <bean id="MapEventListener" className=
"com.company.org.MyMapEventListener" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

Proporcionar este archivo a la instancia de ObjectGridManager facilita la creación de esta configuración. El siguiente fragmento de código muestra cómo crear una instancia de ObjectGrid utilizando este archivo XML. La instancia de ObjectGrid recién creada tiene un MapEventListener establecido en myMap BackingMap.

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", new URL("file:etc/test/myGrid.xml"), true, false);
```

Conexión a través de programa de una implementación de MapEventListener

El nombre de clase para la clase personalizada MapEventListener es com.company.org.MyMapEventListener. Esta clase implementa la interfaz MapEventListener. El siguiente fragmento de código crea el objeto MapEventListener personalizado y lo añade a un objeto BackingMap:

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap myMap = myGrid.defineMap("myMap");
MyMapEventListener myListener = new MyMapEventListener();
myMap.addMapEventListener(myListener);
```

Plug-in ObjectGridEventListener

Un plug-in ObjectGridEventListener proporciona sucesos de ciclo de vida de WebSphere eXtreme Scale para ObjectGrid, fragmentos y transacciones. Un plug-in ObjectGridEventListener proporciona notificaciones cuando se inicializa o destruye un ObjectGrid, y cuando se inicia o finaliza una transacción. Los plug-ins ObjectGridEventListener son clases personalizadas que se escriben implementando la interfaz ObjectGridEventListener. De forma opcional, la implementación incluye las subinterfaces ObjectGridEventGroup y sigue las convenciones comunes de plug-in eXtreme Scale.

Visión general

Un plug-in `ObjectGridEventListener` es útil cuando está disponible un plug-in `Loader`, y debe inicializar las conexiones JDBC (Java Database Connectivity) o las conexiones a un programa de fondo cuando se inician y finalizan las transacciones. En general, un plug-in `ObjectGridEventListener` y un plug-in `Loader` se escriben juntos.

Escritura de un plug-in `ObjectGridEventListener`

Un plug-in `ObjectGridEventListener` debe implementar la interfaz `ObjectGridEventListener` para recibir notificaciones sobre los sucesos significativos de eXtreme Scale. Para recibir notificaciones de sucesos adicionales, puede implementar las siguientes interfaces. Estas subinterfaces se incluyen en la interfaz `ObjectGridEventGroup`:

- Interfaz `ShardEvents`
- Interfaz `ShardLifecycle`
- Interfaz `TransactionEvents`

Si desea más información sobre estas interfaces, consulte la documentación de la API.

Sucesos de fragmentos

Cuando el servicio de catálogo coloca los fragmentos del primario de la partición o de réplica en una máquina virtual Java (JVM), se crea una nueva instancia de `ObjectGrid` en dicha JVM para alojar dicho fragmento. Algunas aplicaciones que necesitan iniciar hebras en la JVM alojan la notificación necesaria primaria de estos sucesos. La interfaz `ObjectGridEventGroup.ShardEvents` declara los métodos `shardActivate` y `shardDeactivate`. Estos métodos se invocan sólo cuando un fragmento está activado como primario y cuando el fragmento se desactiva del primario. Estos dos sucesos permiten a la aplicación iniciar hebras adicionales cuando el fragmento es un primario y detenerlas cuando el fragmento vuelve a ser una réplica o se queda fuera de servicio.

Una aplicación puede determinar qué partición ha sido activada por la búsqueda de un `BackingMap` específico en la referencia de `ObjectGrid` que se proporciona al método `shardActivate` mediante el método `ObjectGrid#getMap`. La aplicación puede ver el número de partición utilizando el método `BackingMap#getPartitionId()`. Las particiones están numeradas del 0 hasta el número de particiones en el descriptor de despliegue menos una.

Sucesos de ciclo de vida de fragmento

Los sucesos de los métodos `ObjectGridEventListener.initialize` y `ObjectGridEventListener.destroy` se entregan utilizando la interfaz `ObjectGridEventGroup.ShardLifecycle`.

Sucesos de transacciones

Los métodos `ObjectGridEventListener.transactionBegin` y `ObjectGridEventListener.transactionEnd` se entregan a través de la interfaz `ObjectGridEventGroup.TransactionEvents`.

Ventajas de este enfoque

Si un plug-in `ObjectGridEventListener` implementa las interfaces `ObjectGridEventListener` y `ShardLifecycle`, los sucesos de ciclo de vida de fragmentos son los únicos sucesos que se entregan al escucha. Después de implementar cualquiera de las nuevas interfaces internas de `ObjectGridEventGroup`, eXtreme Scale sólo entrega estos sucesos específicos mediante las nuevas interfaces. Con esta implementación, el código puede ser compatible con versiones anteriores. Si utiliza las nuevas interfaces internas ahora puede recibir sólo los sucesos específicos necesarios.

Utilización del plug-in `ObjectGridEventListener`

Para utilizar un plug-in `ObjectGridEventListener` personalizado, primero cree una clase que implemente la interfaz `ObjectGridEventListener` y todas las subinterfaces `ObjectGridEventGroup` opcionales. Añada el escucha personalizado a un `ObjectGrid` para recibir una notificación de los sucesos significativos. Dispone de dos procedimientos para añadir un plug-in `ObjectGridEventListener` en la configuración de eXtreme Scale: configuración programática y configuración XML.

Configurar mediante programación un plug-in `ObjectGridEventListener`

Presuponga que el nombre de la clase del receptor de sucesos de eXtreme Scale es la clase `com.company.org.MyObjectGridEventListener`. Esta clase implementa la interfaz `ObjectGridEventListener`. El siguiente fragmento de código crea el `ObjectGridEventListener` personalizado y lo añade a un `ObjectGrid`.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
MyObjectGridEventListener myListener = new MyObjectGridEventListener();
myGrid.addEventListener(myListener);
```

Configure un plug-in `ObjectGridEventListener` con XML

También puede configurar un plug-in `ObjectGridEventListener` mediante XML. El siguiente XML crea una configuración que es equivalente al receptor de sucesos de `ObjectGrid` descrito creado mediante programa. El siguiente texto debe aparecer en el archivo `myGrid.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="myGrid">
      <bean id="ObjectGridEventListener"
        className="com.company.org.MyObjectGridEventListener" />
      <backingMap name="Book"/>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Tenga en cuenta que las declaraciones `bean` se indican antes que las declaraciones `backingMap`. Proporcione este archivo al plug-in `ObjectGridManager` para facilitar la creación de esta configuración. El siguiente fragmento de código demuestra cómo crear una instancia de `ObjectGrid` utilizando este archivo XML. La instancia de `ObjectGrid` que se crea tiene un receptor `ObjectGridEventListener` establecido en el `ObjectGrid` `myGrid`.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid",
  new URL("file:etc/test/myGrid.xml"), true, false);
```

Plug-ins para la indexación personalizada de los objetos de memoria caché

Con un plug-in o índice `MapIndexPlugin`, puede escribir estrategias personalizadas de indexación que van más allá de los índices incorporados que proporciona eXtreme Scale.

Si desea información general sobre la indexación, consulte [Indexación](#).

Si desea información sobre el uso de la indexación, consulte “Uso de índices en el acceso a los datos que no son clave” en la página 251.

Las implementaciones de `MapIndexPlugin` deben utilizar la interfaz `MapIndexPlugin` y seguir las convenciones comunes de plug-in de eXtreme Scale.

Las secciones siguientes incluyen algunos de los métodos importantes de la interfaz `index`.

Método `setProperties`

Utilice el método `setProperties` para inicializar el plug-in `index` mediante programa. El parámetro del objeto `Properties` que se pasa en el método debe contener la información de configuración para inicializar el plug-in correctamente. La implementación del método `setProperties`, junto con la implementación del método `getProperties`, son necesarias en un entorno distribuido porque la configuración del plug-in `index` se mueve entre los procesos de cliente y servidor. A continuación se muestra un ejemplo de la implementación de este método.

```
setProperties(Properties properties)

// Código de ejemplo del método setProperties
public void setProperties(Properties properties) {
    ivIndexProperties = properties;

    String ivRangeIndexString = properties.getProperty("rangeIndex");
    if (ivRangeIndexString != null && ivRangeIndexString.equals("true")) {
        setRangeIndex(true);
    }
    setName(properties.getProperty("indexName"));
    setAttributeName(properties.getProperty("attributeName"));

    String ivFieldAccessAttributeString = properties.getProperty("fieldAccessAttribute");
    if (ivFieldAccessAttributeString != null && ivFieldAccessAttributeString.equals("true")) {
        setFieldAccessAttribute(true);
    }

    String ivPOJOKeyIndexString = properties.getProperty("POJOKeyIndex");
    if (ivPOJOKeyIndexString != null && ivPOJOKeyIndexString.equals("true")) {
        setPOJOKeyIndex(true);
    }
}
```

Método `getProperties`

El método `getProperties` extrae la configuración del plug-in `index` de una instancia de `MapIndexPlugin`. Puede utilizar las propiedades extraídas para inicializar otra instancia de `MapIndexPlugin` para que tenga los mismos estados internos. Las implementaciones del método `getProperties` y del método `setProperties` son necesarias en un entorno distribuido. A continuación, aparece una implementación de ejemplo del método `getProperties`:

```
getProperties()

// Código de ejemplo del método getProperties
public Properties getProperties() {
    Properties p = new Properties();
    p.put("indexName", indexName);
}
```

```

        p.put("attributeName", attributeName);
        p.put("rangeIndex", ivRangeIndex ? "true" : "false");
        p.put("fieldAccessAttribute", ivFieldAccessAttribute ? "true" : "false");
        p.put("POJOKeyIndex", ivPOJOKeyIndex ? "true" : "false");
        return p;
    }
}

```

Método setEntityMetadata

La ejecución de WebSphere eXtreme Scale llama al método setEntityMetadata durante la inicialización para establecer el EntityMetadata del BackingMap asociado en la instancia de MapIndexPlugin. El EntityMetadata es necesario para dar soporte a la indexación de objetos tuple. Un tuple es un conjunto de datos que represente un objeto de entidad o su clave. Si la BackingMap es para una entidad, el usuario debe implementar este método.

El siguiente código de ejemplo implementa el método setEntityMetadata.

```

setEntityMetadata(EntityMetadata entityMetadata)

// Código de ejemplo del método setEntityMetadata
public void setEntityMetadata(EntityMetadata entityMetadata) {
    ivEntityMetadata = entityMetadata;
    if (ivEntityMetadata != null) {
        // es una correlación de tuples
        TupleMetadata valueMetadata = ivEntityMetadata.getValueMetadata();
        int numAttributes = valueMetadata.getNumAttributes();
        for (int i = 0; i < numAttributes; i++) {
            String tupleAttributeName = valueMetadata.getAttribute(i).getName();
            if (attributeName.equals(tupleAttributeName)) {
                ivTupleValueIndex = i;
                break;
            }
        }

        if (ivTupleValueIndex == -1) {
            // no se encontró atributo en tuple de valor, intente encontrarlo en tuple de clave
            // no se encontró en tuple de clave, implica indexación de claves en uno de los atributos de clave de tuple
            TupleMetadata keyMetadata = ivEntityMetadata.getKeyMetadata();
            numAttributes = keyMetadata.getNumAttributes();
            for (int i = 0; i < numAttributes; i++) {
                String tupleAttributeName = keyMetadata.getAttribute(i).getName();
                if (attributeName.equals(tupleAttributeName)) {
                    ivTupleValueIndex = i;
                    ivKeyTupleAttributeIndex = true;
                    break;
                }
            }
        }

        if (ivTupleValueIndex == -1) {
            // si entityMetadata no es nulo y no se ha podido encontrar el
            // attributeName en entityMetadata, se trata de un
            // error
            throw new ObjectGridRuntimeException("Invalid attributeName. Entity: " + ivEntityMetadata.getName());
        }
    }
}
}

```

Métodos de nombres de atributos

El método setAttributeName establece el nombre del atributo que se va a indexar. La clase de objeto almacenada en la memoria caché debe proporcionar el método get para el atributo indexado. Por ejemplo, si el objeto tiene un atributo employeeName o EmployeeName, el índice llama al método getEmployeeName en el objeto para extraer el valor de atributo. El nombre de atributo debe ser el mismo que el nombre que aparece en el método get, y el atributo debe implementar la interfaz Comparable. Si el atributo es del tipo booleano, también puede utilizar el patrón del método isAttributeName.

El método getAttributeName devuelve el nombre del atributo indexado.

Método getAttribute

El método `getAttribute` devuelve el valor de atributo indexado del objeto especificado. Por ejemplo, si un objeto `Employee` tiene un atributo llamado `employeeName` que está indexado, puede utilizar el método `getAttribute` para extraer el valor del atributo `employeeName` de un objeto `Employee` especificado. Este método es necesario en un entorno distribuido de WebSphere eXtreme Scale.

```
getAttribute(Object value)

// Código de ejemplo del método getAttribute
public Object getAttribute(Object value) throws ObjectGridRuntimeException {
    if (ivPOJOKeyIndex) {
        // En el caso de indexación de claves POJO, no es necesario obtener el atributo del objeto de valor.
        // La clave misma es el valor del atributo utilizado para compilar el índice.
        return null;
    }

    try {
        Object attribute = null;
        if (value != null) {
            // manejar valor de tuple si ivTupleValueIndex != -1
            if (ivTupleValueIndex == -1) {
                // valor regular
                if (ivFieldAccessAttribute) {
                    attribute = this.getAttributeField(value).get(value);
                } else {
                    attribute = getAttributeMethod(value).invoke(value, emptyArray);
                }
            } else {
                // Nombre de tuple
                attribute = extractValueFromTuple(value);
            }
        }
        return attribute;
    } catch (InvocationTargetException e) {
        throw new ObjectGridRuntimeException(
            "Caught unexpected Throwable during index update processing, index name = " + indexName + ": " + t,
            t);
    } catch (Throwable t) {
        throw new ObjectGridRuntimeException(
            "Caught unexpected Throwable during index update processing, index name = " + indexName + ": " + t,
            t);
    }
}
```

Índice compuesto HashIndex

El índice compuesto `HashIndex` mejora el rendimiento de la consulta y evita la costosa exploración de correlaciones. La característica también proporciona un método práctico para que la API `HashIndex` encuentre los objetos almacenados en memoria caché cuando los criterios de búsqueda implican muchos atributos.

Rendimiento mejorado

Un `HashIndex` compuesto proporciona una forma rápida y práctica para buscar objetos almacenados en memoria caché con varios atributos en los criterios de búsqueda de coincidencia. El índice compuesto soporta búsquedas completas de coincidencia de atributo, pero no soporta las búsquedas de rango.

Nota: Los índices compuestos no soportan el operador `BETWEEN` en el lenguaje de consulta de `ObjectGrid` porque `BETWEEN` requeriría el soporte de rango. Los condicionales mayor que (`>`) y menor que (`<`) tampoco funcionan porque requieren los índices de rango.

Un índice compuesto puede mejorar el rendimiento de las consultas si el índice compuesto apropiado está disponible para la condición `WHERE`. Esto significa que el índice compuesto tiene exactamente los mismos atributos que los implicados en la condición `WHERE` con todos los atributos coincidentes.

Una consulta podría tener muchos atributos implicados en una condición como en el ejemplo siguiente.

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND a.zipcode='55901'
```

El índice compuesto puede mejorar el rendimiento de la consulta al evitar tener que explorar la correlación o unir diversos resultados de índice de un único atributo. En el ejemplo, si un índice compuesto se define con atributos (city,state,zipcode), el motor de consultas puede utilizar el índice compuesto para encontrar la entrada con city='Rochester', state='MN' y zipcode='55901'. Sin un índice compuesto y un índice de atributo en los atributos city, state y zipcode, el motor de consultas tendrá que explorar la correlación o unir varias búsquedas de atributo único, que normalmente tienen una sobrecarga costosa. Además, la consulta del índice compuesto sólo soporta un patrón de coincidencia completa.

Configuración de un índice compuesto

Puede configurar índices compuestos de tres maneras: utilizando XML, a través de programa y (sólo para correlaciones de entidad) con anotaciones de entidad.

Mediante XML

Para poder configurar un índice compuesto con XML, incluya código como, por ejemplo, el que aparece abajo en el elemento backingMapPluginCollections del archivo de configuración.

```
Índice compuesto - Configuración mediante XML  
<bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">  
<property name="Name" type="java.lang.String" value="Address.CityStateZip"/>  
<property name="AttributeName" type="java.lang.String" value="city,state,zipcode"/>  
</bean>
```

Configuración mediante programa

El código de ejemplo programático siguiente creará el mismo índice compuesto que el XML anterior.

```
HashIndex mapIndex = new HashIndex();  
mapIndex.setName("Address.CityStateZip");  
mapIndex.setAttributeName(("city,state,zipcode"));  
mapIndex.setRangeIndex(true);  
  
BackingMap bm = objectGrid.defineMap("mymap");  
bm.addMapIndexPlugin(mapIndex);
```

Observe que la configuración de un índice compuesto es igual que la configuración de un índice ordinario con XML excepto por el valor de la propiedad attributeName. En el caso de un índice compuesto, el valor de attributeName es una lista de atributos delimitada por comas. Por ejemplo, la clase de valor Address tiene 3 atributos: city, state y zipcode. Un índice compuesto se puede definir con el valor de propiedad attributeName como "city,state,zipcode" que indica que city, state y zipcode se incluyen en el índice compuesto.

Además, tenga en cuenta que los HashIndexes compuestos no soportan las búsquedas de rango y, por lo tanto, no pueden tener la propiedad RangeIndex establecida en true.

Con anotaciones de entidad

En el caso de correlaciones de entidad, el acercamiento de anotaciones puede utilizarse para definir un índice compuesto. Puede definir una lista de

CompositeIndex dentro de una anotación CompositeIndexes en el nivel de clase de la entidad. El índice CompositeIndex tiene un nombre y una propiedad attributeNames. Cada índice CompositeIndex está asociado con una instancia HashIndex aplicada al elemento BackingMap asociado de la entidad. El índice HashIndex está configurado como un índice de no intervalo.

```
@Entity
@CompositeIndexes({
    @CompositeIndex(name=" CityStateZip ", attributeNames=" city,state,zipcode"),
    @CompositeIndex(name="lastnameBirthday", attributeNames=" lastname,birthday ")
})
public class Address {
    @Id int id;
    String street;
    String city;
    String state;
    String zipcode;
    String lastname;
    Date birthday;
}
```

La propiedad name de cada índice compuesto debe ser exclusivo dentro de la entidad y BackingMap. Si no se especifica el nombre, se utilizará un nombre generado. La propiedad attributeNames se utiliza para rellenar la propiedad attributeName de HashIndex con la lista de atributos delimitada por comas. Los nombres de atributo coinciden con los nombres de campos persistentes, cuando las entidades se configuran para utilizar el acceso a campo, o el nombre de la propiedad se haya definido para los convenios de denominación de JavaBeans para las entidades de acceso a propiedades. Por ejemplo: si el nombre de atributo es "street", el método getter de la propiedad es getStreet.

Búsquedas en índices compuestos

Después de configurar un índice compuesto, una aplicación puede utilizar el método findAll(Object) de la interfaz MapIndex para realizar búsquedas, como en el ejemplo siguiente.

```
Session sess = objectgrid.getSession();
ObjectMap map = sess.getMap("MAP_NAME");
MapIndex codeIndex = (MapIndex) map.getIndex("INDEX_NAME");
Object[] compositeValue = new Object[]{ MapIndex.EMPTY_VALUE,
    "MN", "55901"};
Iterator iter = mapIndex.findAll(compositeValue);
```

MapIndex.EMPTY_VALUE se asigna a compositeValue[0] que indica que el atributo city se excluye de la evaluación. Sólo los objetos con el atributo state igual a "MN" y el atributo zipcode igual a "55901" se incluirán en el resultado.

Las siguientes consultas se benefician de la configuración del índice compuesto anterior:

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND
a.zipcode='55901'
```

```
SELECT a FROM Address a WHERE a.state='MN' AND a.zipcode='55901'
```

El motor de consultas encontrará el índice compuesto apropiado y lo utiliza para mejorar el rendimiento de la consulta en casos de coincidencia de atributos total.

En algunos escenarios, la aplicación podría definir varios índices compuestos con atributos solapados para satisfacer todas las consultas con coincidencia total de los atributos. Un inconveniente de aumentar el número de índices es la posible sobrecarga de rendimiento en las operaciones de correlaciones.

Migración e interoperatividad

La única restricción del uso de un índice compuesto es que una aplicación no puede configurarlo en un entorno distribuido con contenedores heterogéneos. Los contenedores antiguos y nuevos no pueden mezclarse, ya que los contenedores más antiguos no reconocerán una configuración de índice compuesto. El índice compuesto es como el índice de atributos ordinario existente, sólo que el primero permite la creación de índices sobre varios atributos. Si utiliza el índice de atributos ordinario, el uso del entorno de contenedores mixto es viable.

Uso de índices en el acceso a los datos que no son clave

El uso de la indexación como una alternativa al acceso de clave a los datos puede ser más eficaz.

Pasos obligatorios

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

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

Nota: En un entorno distribuido, si el objeto de índice se obtiene de un `ObjectGrid` cliente, tendrá el objeto de índice de tipo cliente y todas las operaciones de índice se ejecutarán en un `ObjectGrid` de servidor remoto. Si la correlación tiene particiones, la operación de índice se ejecutará en cada partición de forma remota y hará que cada partición se fusione antes de devolverlas a la aplicación. El rendimiento se determinará a través del número de particiones y el tamaño del resultado devuelto por cada partición. Se podría producir un bajo rendimiento si ambos factores son altos.

Para obtener información sobre cómo configurar `HashIndex`, consulte [Configuración de HashIndex](#).

Si desea escribir su propio plug-in de índice, consulte [“Plug-ins para la indexación personalizada de los objetos de memoria caché”](#) en la página 246.

Para obtener información sobre la creación de índices, consulte los apartados [Índices](#) y [“Índice compuesto HashIndex”](#) en la página 248.

Adición de plug-ins de índices estáticos

Puede utilizar dos acercamientos para añadir plug-ins de índices estáticos a la configuración de `BackingMap`: configuración del XML o configuración mediante programa. El siguiente ejemplo ilustra el enfoque de la configuración XML.

Adición de plug-ins de índice estático: enfoque de configuración de XML

```
<backingMapPluginCollection id="person">
  <bean id="MapIndexplugin"
    className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
    <property name="Name" type="java.lang.String" value="CODE" description="index name" />
    <property name="RangeIndex" type="boolean" value="true"
      description="true for MapRangeIndex" />
    <property name="AttributeName" type="java.lang.String" value="employeeCode" description="attribute name" />
  </bean>
</backingMapPluginCollection>
```

En este ejemplo de configuración de XML, la clase HashIndex incorporada se utiliza como el plug-in de índice. HashIndex soporta las propiedades que pueden configurar los usuarios como, por ejemplo, Name, RangeIndex y AttributeName en el ejemplo anterior.

- La propiedad Name se configura como “CODE”, una serie que identifica este plug-in de índice. El valor de la propiedad Name debe ser exclusivo dentro del ámbito de la BackingMap, y se puede utilizar para recuperar el objeto de índice por el nombre de la instancia de ObjectMap para la BackingMap.
- La propiedad RangeIndex se configura como “true”, que significa que la aplicación puede difundir el objeto de índice recuperado en la interfaz MapRangeIndex. Si la propiedad RangeIndex está configurada como “false”, la aplicación sólo puede difundir el objeto de índice recuperado en la interfaz MapIndex. Un MapRangeIndex soporta las funciones para encontrar los datos utilizando las funciones de rango como, por ejemplo, mayor que, menor que, o ambos, mientras que un MapIndex sólo soporta las funciones de igual. Si el índice va a ser utilizado por una consulta, la propiedad RangeIndex se debe configurar en “true” en índices de atributo único. Para un índice de relación y un índice compuesto, la propiedad RangeIndex se debe configurar en “false”.
- La propiedad AttributeName se configura como “employeeCode”, que significa que el atributo employeeCode del objeto almacenado en memoria caché se utiliza para crear un índice de atributo único. Si una aplicación necesita buscar objetos almacenados en memoria caché con varios atributos, la propiedad AttributeName se puede establecer en una lista delimitada por comas, lo que genera un índice compuesto.

Consulte la información sobre cómo configurar HashIndex en *Guía de administración* si desea más información.

La interfaz BackingMap tiene dos métodos que puede utilizar para añadir plug-ins de índice estático: addMapIndexplugin y setMapIndexplugins. Si desea más información, consulte la documentación de API.

En el ejemplo de código siguiente se muestra la configuración mediante programa:

Adición de plug-ins de índices estáticos: configuración mediante programa

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

ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid ivObjectGrid = ogManager.createObjectGrid( "grid" );
BackingMap personBackingMap = ivObjectGrid.getMap("person");

// utilizar la clase HashIndex incorporada como clase de plug-ins de índices.
HashIndex mapIndexplugin = new HashIndex();
mapIndexplugin.setName("CODE");
mapIndexplugin.setAttributeName("EmployeeCode");
mapIndexplugin.setRangeIndex(true);
personBackingMap.addMapIndexplugin(mapIndexplugin);
```

Uso de índices estáticos

Después de que se añada un plug-in de índice estático a una configuración de BackingMap y la instancia de ObjectGrid que lo contiene se inicialice, las aplicaciones pueden recuperar el objeto de índice por nombre de la instancia de ObjectMap para la BackingMap. Difunda el objeto de índice a la interfaz de índices de aplicación. Ahora, las operaciones que soporta la interfaz de índice de aplicación se pueden ejecutar.

El siguiente código de ejemplo ilustra cómo recuperar y utilizar los índices estáticos.

Ejemplo del uso de índices estáticos

```
Session session = ivObjectGrid.getSession();
ObjectMap map = session.getMap("person ");
MapRangeIndex codeIndex = (MapRangeIndex) m.getIndex("CODE");
Iterator iter = codeIndex.findLessEqual(new Integer(15));
while (iter.hasNext()) {
    Object key = iter.next();
    Object value = map.get(key);
}
```

Adición, eliminación y uso de índices dinámicos

Puede crear y eliminar, mediante programa, índices dinámicos de una instancia BackingMap en cualquier momento. Un índice dinámico se distingue de un índice estático en que el índice dinámico puede crearse después de que se inicialice la instancia ObjectGrid que lo contiene. A diferencia de la indexación estática, la indexación dinámica es un proceso asíncrono y necesita estar en un estado preparado antes de poder utilizarlo. Este método utiliza el mismo acercamiento para recuperar y usar los índices dinámicos que los índices estáticos. Puede eliminar un índice dinámico si éste deja de utilizarse. La interfaz BackingMap tiene métodos para crear y eliminar índices dinámicos.

Consulte la API BackingMap si desea más información sobre los métodos createDynamicIndex y removeDynamicIndex.

Ejemplo del uso de índices dinámicos

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

ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid("grid");
BackingMap bm = og.getMap("person");
og.initialize();

// crear índice después de la inicialización de ObjectGrid sin DynamicIndexCallback.
bm.createDynamicIndex("CODE", true, "employeeCode", null);

try {
    // Si no se utiliza DynamicIndexCallback, debe esperar a que el índice esté preparado.
    // El tiempo de espera depende del tamaño actual de la correlación
    Thread.sleep(3000);
} catch (Throwable t) {
    // ...
}

// Cuando el índice esté preparado, las aplicaciones pueden intentar obtener
// la instancia de la interfaz de índices de aplicación.
// Las aplicaciones deben encontrar un modo de asegurarse de que el índice está preparado para el uso,
// si no se utiliza la interfaz DynamicIndexCallback.
// El ejemplo siguiente muestra el modo de esperar a que el índice esté preparado.
// Tenga en cuenta el tamaño de la correlación en el tiempo total de espera.

Session session = og.getSession();
ObjectMap m = session.getMap("person");
MapRangeIndex codeIndex = null;

int counter = 0;
int maxCounter = 10;
boolean ready = false;
while (!ready && counter < maxCounter) {
```

```

try {
    counter++;
    codeIndex = (MapRangeIndex) m.getIndex("CODE");
    ready = true;
} catch (IndexNotReadyException e) {
    // implica que el índice no está preparado,...
    System.out.println("Index is not ready. continue to wait.");
    try {
        Thread.sleep(3000);
    } catch (Throwable tt) {
        // ...
    }
} catch (Throwable t) {
    // excepción inesperada
    t.printStackTrace();
}
}

if (!ready) {
    System.out.println("Index is not ready. Need to handle this situation.");
}

// Usar el índice para realizar consultas
// Consulte la interfaz MapIndex o MapRangeIndex para obtener información sobre las operaciones admitidas.
// El atributo de objeto en el que se crea el índice es EmployeeCode.
// Presuponga que el atributo EmployeeCode es de tipo Integer: el
// parámetro que se pasa a operaciones de índices tiene este tipo de datos.

Iterator iter = codeIndex.findLessEqual(new Integer(15));

// eliminar el índice dinámico cuando ya no se necesite
bm.removeDynamicIndex("CODE");

```

Interfaz DynamicIndexCallback

La interfaz `DynamicIndexCallback` se ha diseñado para aplicaciones que desean recibir notificaciones de sucesos de índices de tipo preparado, error o destruir. `DynamicIndexCallback` es un parámetro opcional del método `createDynamicIndex` de `BackingMap`. Con una instancia registrada de `DynamicIndexCallback`, las aplicaciones pueden ejecutar lógica empresarial al recibir una notificación de un suceso de índices. Por ejemplo, el suceso preparado significa que el índice está preparado para el uso. Cuando se recibe una notificación de este suceso, una aplicación puede intentar recuperar y utilizar la instancia de la interfaz de índices de aplicación. Consulte la API `DynamicIndexCallback` y la documentación de la API si desea más información.

El ejemplo de código siguiente ilustra el uso de la interfaz `DynamicIndexCallback`:

Uso de la interfaz DynamicIndexCallback

```

BackingMap personBackingMap = ivObjectGrid.getMap("person");
DynamicIndexCallback callback = new DynamicIndexCallbackImpl();
personBackingMap.createDynamicIndex("CODE", true, "employeeCode", callback);

class DynamicIndexCallbackImpl implements DynamicIndexCallback {
    public DynamicIndexCallbackImpl() {
    }

    public void ready(String indexName) {
        System.out.println("DynamicIndexCallbackImpl.ready() -> indexName = " + indexName);

        // Simular qué debe hacer la aplicación al recibir la notificación de que el índice está preparado.
        // Normalmente, la aplicación debería esperar a que se alcance el estado preparado y después proceder
        // con una lógica de uso de índices.
        if ("CODE".equals(indexName)) {
            ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
            ObjectGrid og = ogManager.createObjectGrid("grid");
            Session session = og.getSession();
            ObjectMap map = session.getMap("person");
            MapIndex codeIndex = (MapIndex) map.getIndex("CODE");
            Iterator iter = codeIndex.findAll(codeValue);
        }
    }

    public void error(String indexName, Throwable t) {
        System.out.println("DynamicIndexCallbackImpl.error() -> indexName = " + indexName);
        t.printStackTrace();
    }

    public void destroy(String indexName) {
        System.out.println("DynamicIndexCallbackImpl.destroy() -> indexName = " + indexName);
    }
}

```

Plug-ins para la comunicación con almacenes persistentes

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

Los cargadores son plug-ins de correlaciones de respaldo que se invocan cuando se realizan cambios en la correlación de respaldo o cuando la correlación de respaldo no puede satisfacer una solicitud de datos (una falta de memoria caché).

Consulte la información sobre los conceptos del almacenamiento en memoria caché en *Visión general del producto* si desea más información.

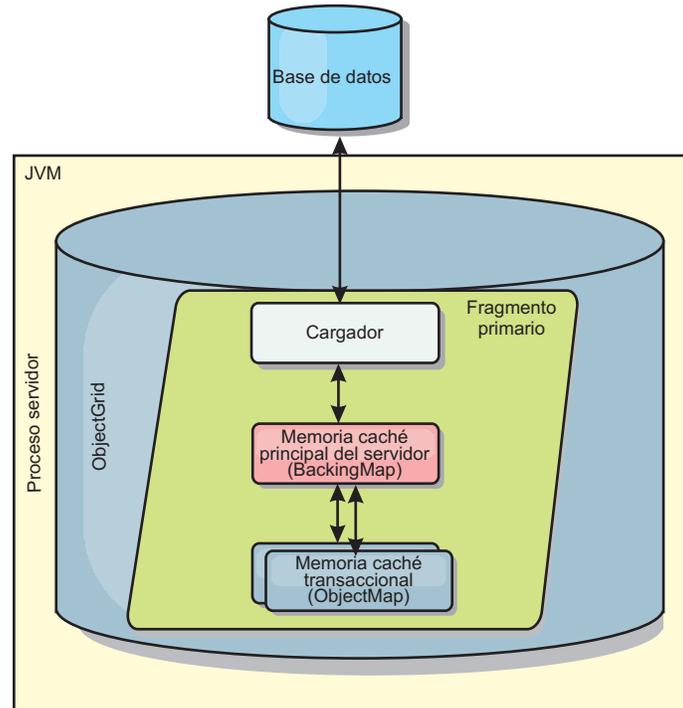


Figura 3. Cargador

WebSphere eXtreme Scale incluye dos cargadores incorporados para integrar con los programas de fondo de la base de datos relacional. Los cargadores JPA (Java Persistence API) utilizan las capacidades de correlación de objetos relacionales (ORM) de ambas implementaciones, OpenJPA e Hibernate, de la especificación JPA.

Utilización de un cargador

Para añadir un cargador a la configuración de BackingMap, puede utilizar la configuración programática o la configuración XML. Un cargador tiene la siguiente relación con una correlación de respaldo:

- Una correlación de respaldo sólo puede tener un cargador.

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

Conexión de un cargador mediante programación

El siguiente fragmento de código demuestra cómo conectar un cargador proporcionado por la aplicación a una correlación de respaldo para map1 utilizando la API de ObjectGrid:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
MyLoader loader = new MyLoader();
loader.setDataBaseName("testdb");
loader.setIsolationLevel("read committed");
bm.setLoader( loader );
```

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

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

Enfoque de configuración XML para conectar un cargador

Un cargador proporcionado por una aplicación también puede conectarse mediante la configuración de un archivo XML. El siguiente ejemplo muestra cómo conectar el cargador MyLoader a la correlación de respaldo map1 con las mismas propiedades de cargador del mismo nombre de base de datos y nivel de aislamiento:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
  <objectGrid name="grid">
    <backingMap name="map1" pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
  </objectGrid>
</objectGrids>
<backingMapPluginCollections>
  <backingMapPluginCollection id="map1">
    <bean id="Loader" className="com.myapplication.MyLoader">
      <property name="dataBaseName"
        type="java.lang.String"
        value="testdb"
        description="database name" />
      <property name="isolationLevel"
```

```

        type="java.lang.String"
        value="read committed"
        description="iso level" />
    </bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

Escribir un cargador

Puede escribir su propia implementación de plug-in de cargador en sus aplicaciones, que debe seguir los convenios de plug-in de WebSphere eXtreme Scale.

Incluir un plug-in de cargador

La definición de la interfaz Loader es la siguiente:

```

public interface Loader
{
    static final SpecialValue KEY_NOT_FOUND;
    List get(Txid txid, List keyList, boolean forUpdate) throws LoaderException;
    void batchUpdate(Txid txid, LogSequence sequence) throws LoaderException, OptimisticCollisionException;
    void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
}

```

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

Método get

La correlación de respaldo llama al método get del cargador para obtener los valores asociados a una lista de claves que se pasa como el argumento keyList. El método get es necesario para devolver una lista java.lang.util.List de valores, un valor para cada clave que aparece en la lista de claves. El primer valor que se devuelve en la lista de valores corresponde a la primera clave de la lista de claves, el segundo valor devuelto en la lista de valores corresponde a la segunda clave de la lista de claves, etc. Si un cargador no encuentra el valor de una clave en la lista de claves, se solicita al cargador que devuelva el objeto de valor especial KEY_NOT_FOUND que se define en la interfaz Loader. Puesto que se puede configurar una correlación de respaldo para permitir null como un valor válido, es muy importante para el cargador devolver el objeto especial KEY_NOT_FOUND cuando el cargador no puede encontrar la clave. Este valor especial permite a la correlación de respaldo distinguir entre un valor nulo y un valor que no existe porque no se encontró. Si una correlación de respaldo no admite valores nulos, se producirá una excepción en un cargador que devuelva un valor nulo en lugar del objeto KEY_NOT_FOUND para una clave que no exista.

El argumento forUpdate indica al cargador si la aplicación llamó a un método get en la correlación o a un método getForUpdate en la correlación. Consulte Interfaz ObjectMap en la documentación de la API si desea más información. El cargador es responsable de implementar una política de control de simultaneidad que controle los accesos simultáneos al almacén persistente. Por ejemplo, numerosos sistemas de gestión de bases de datos relacionales admiten la sintaxis FOR UPDATE de la sentencia select de SQL que se utiliza para leer los datos de una tabla relacional. El cargador puede elegir utilizar la sintaxis FOR UPDATE en la sentencia select de SQL basándose en si se ha pasado boolean true como el valor del argumento para el parámetro forUpdate de este método. Normalmente, el cargador utilizar la sintaxis FOR UPDATE sólo cuando se utiliza la política de control de simultaneidad pesimista. Para un control de simultaneidad optimista, el cargador nunca utiliza la sintaxis for update en la sentencia select de SQL. El

cargador deberá decidir si va a utilizar el argumento `forUpdate` basado en la política de control de simultaneidad que utiliza el cargador.

Si desea una explicación del parámetro `txid`, consulte “Plug-ins para gestionar los sucesos del ciclo de vida de transacciones” en la página 275.

Método `batchUpdate`

El método `batchUpdate` es importante en la interfaz `Loader`. Este método se llama siempre que eXtreme Scale necesita aplicar todos los cambios actuales al cargador. Se proporciona al cargador una lista de cambios para la correlación seleccionada. Los cambios se repiten y se aplican al programa de fondo. El método recibe el valor `TxID` actual y los cambios que se aplicarán. El siguiente ejemplo se repite en el conjunto de cambios y procesa por lotes tres sentencias JDBC (Java database connectivity), una con `insert`, otra con `update` y una con `delete`.

```
import java.util.Collection;
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;

public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException {
    // Obtener una conexión SQL que vaya a utilizarse.
    Connection conn = getConnection(tx);
    try {
        // Procesar la lista de cambios y crear un conjunto de
        // sentencias preparadas para ejecutar una
        // operación SQL de batch update, insert o delete.
        Iterator iter = sequence.getPendingChanges();
        while (iter.hasNext()) {
            LogElement logElement = (LogElement) iter.next();
            Object key = logElement.getKey();
            Object value = logElement.getCurrentValue();
            switch (logElement.getType().getCode()) {
                case LogElement.CODE_INSERT:
                    buildBatchSQLInsert(tx, key, value, conn);
                    break;
                case LogElement.CODE_UPDATE:
                    buildBatchSQLUpdate(tx, key, value, conn);
                    break;
                case LogElement.CODE_DELETE:
                    buildBatchSQLDelete(tx, key, conn);
                    break;
            }
        }
        // Ejecutar las sentencias de proceso por lotes creadas mediante el bucle anterior.
        Collection statements = getPreparedStatementCollection(tx, conn);
        iter = statements.iterator();
        while (iter.hasNext()) {
            PreparedStatement pstmt = (PreparedStatement) iter.next();
            pstmt.executeBatch();
        }
    } catch (SQLException e) {
        LoaderException ex = new LoaderException(e);
        throw ex;
    }
}
```

El ejemplo anterior ilustra la lógica de alto nivel de proceso del argumento `LogSequence`, pero no ilustra los detalles sobre cómo se crea una sentencia `insert`, `update` o `delete` de SQL. Algunos de los puntos claves que se ilustran son:

- Se llama al método `getPendingChanges` en el argumento `LogSequence` para obtener un repetidor en la lista de `LogElements` que debe procesar el cargador.
- El método `LogElement.getType().getCode()` se utiliza para determinar si el `LogElement` es para una operación `insert`, `update` o `delete` de SQL.
- Se obtiene una excepción `SQLException` que se encadena a una excepción `LoaderException` que se imprime para informar de que se ha producido una excepción durante la actualización de proceso por lotes.

- Se utiliza el soporte de actualización de proceso por lotes JDBC para minimizar el número de consultas que deben hacerse en el programa de fondo.

Método preloadMap

Durante la inicialización de eXtreme Scale, se inicializa cada correlación de respaldo definida. Si se conecta un cargador en una correlación de respaldo, esta correlación invoca el método `preloadMap` en la interfaz `Loader` para permitir al cargador que busque previamente los datos en su programa de fondo y los cargue en la correlación. En el ejemplo siguiente se presupone que las primeras 100 filas de una tabla `Employee` de empleados se leen de la base de datos y se cargan en la correlación. La clase `EmployeeRecord` es una clase proporcionada por una aplicación que aloja los datos de empleado que se leen en la tabla de empleados.

Nota: Esta muestra capta todos los datos de la base de datos y luego los inserta en la correlación base de una partición. En un caso de ejemplo de despliegue de eXtreme Scale distribuido del mundo real, los datos se deberían distribuir entre todas las particiones. Para obtener más información, consulte “Programación del programa de utilidad de precarga JPA basada en cliente” en la página 313.

```
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException

public void preloadMap(Session session, BackingMap backingMap) throws LoaderException {
    boolean tranActive = false;
    ResultSet results = null;
    Statement stmt = null;
    Connection conn = null;
    try {
        session.beginNoWriteThrough();
        tranActive = true;
        ObjectMap map = session.getMap(backingMap.getName());
        TxID tx = session.getTxID();
        // Obtener una conexión de confirmación automática que esté establecida en
        // un nivel de aislamiento de lectura confirmada.
        conn = getAutoCommitConnection(tx);
        // Precargar la correlación de empleados con objetos EmployeeRecord
        // . Leer todos los empleados de la tabla, pero
        // limitar la precarga a las primeras 100 filas.
        stmt = conn.createStatement();
        results = stmt.executeQuery(SELECT_ALL);
        int rows = 0;
        while (results.next() && rows < 100) {
            int key = results.getInt(EMPNO_INDEX);
            EmployeeRecord emp = new EmployeeRecord(key);
            emp.setLastName(results.getString(LASTNAME_INDEX));
            emp.setFirstName(results.getString(FIRSTNAME_INDEX));
            emp.setDepartmentName(results.getString(DEPTNAME_INDEX));
            emp.updateSequenceNumber(results.getLong(SEQNO_INDEX));
            emp.setManagerNumber(results.getInt(MGRNO_INDEX));
            map.put(new Integer(key), emp);
            ++rows;
        }
        // Confirmar la transacción.
        session.commit();
        tranActive = false;
    } catch (Throwable t) {
        throw new LoaderException("preload failure: " + t, t);
    } finally {
        if (tranActive) {
            try {
                session.rollback();
            } catch (Throwable t2) {
                // Tolerar anomalías de retrotracción y
                // permitir que se emitan objetos Throwable originales.
            }
        }
        // Limpie otros recursos de base de datos aquí
        // como cierre de sentencias, conjuntos de resultados, etc.
    }
}
```

Este ejemplo ilustra los puntos clave siguientes:

- La correlación de respaldo preloadMap utiliza el objeto Session que se ha pasado a aquélla como argumento de sesión.
- Se utiliza el método Session.beginNoWriteThrough para iniciar la transacción, en lugar del método begin.
- No se puede llamar al cargador para cada operación put que se produce en este método para cargar la correlación.
- El cargador puede correlacionar las columnas de la tabla de empleados con un campo en el objeto EmployeeRecord Java. El cargador obtiene todas las excepciones throwable que se producen y emite una excepción LoaderException con la excepción throwable obtenida encadenada a él.
- El bloque finally garantiza que cualquier excepción throwable que se produzca entre el momento en que se llama al método beginNoWriteThrough y el momento en que se llama al método commit provoque que el bloque finally retrotraiga la transacción activa. Esta acción es crítica para garantizar que cualquier transacción que haya sido iniciada por el método preloadMap se complete antes de devolverla al llamante. El bloque finally es un buen punto para realizar otras acciones de limpieza que podrían ser necesarias, como el cierre de la conexión JDBC (Java Database Connectivity) y otros objetos JDBC.

El ejemplo de preloadMap utiliza una sentencia select de SQL que selecciona todas las filas de la tabla. En el cargador que proporciona la aplicación, puede que necesite establecer una o más propiedades del cargador para controlar cuánta información de la tabla debe precargarse en la correlación.

Como el método preloadMap sólo se llama una vez durante la inicialización de BackingMap, es un buen momento para ejecutar el código de inicialización del cargador de una sola vez. Aunque el cargador decida no buscar previamente los datos en el programa de fondo y cargar los datos en la correlación, probablemente necesite realizar algunas operaciones de inicialización de una sola vez para hacer más eficaces otros métodos del cargador. El ejemplo siguiente ilustra el almacenamiento en memoria caché del objeto TransactionCallback y del objeto OptimisticCallback como variables de instancia del cargador. De esta manera, los otros métodos del cargador no tienen que realizar llamadas de método para obtener acceso a estos objetos. Este almacenamiento en memoria caché de los valores del plug-in ObjectGrid puede realizarse porque, después de que BackingMap se haya inicializado, los objetos TransactionCallback y OptimisticCallback no pueden cambiarse ni sustituirse. Es aceptable almacenar en memoria caché estas referencias de objeto como variables de instancia del cargador.

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.OptimisticCallback;
import com.ibm.websphere.objectgrid.plugins.TransactionCallback;

// Variables de instancia del cargador.
MyTransactionCallback ivTcb; // MyTransactionCallback

// amplía TransactionCallback
MyOptimisticCallback ivOcb; // MyOptimisticCallback

// implementa OptimisticCallback
// ...
public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
[Programación de réplica]
    // Almacenar en memoria caché los objetos TransactionCallback y OptimisticCallback
    // en variables de instancia de este cargador.
    ivTcb = (MyTransactionCallback) session.getObjectGrid().getTransactionCallback();
    ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
    // Resto de código de preloadMap (como se muestra en el ejemplo anterior).
}
```

Si desea más información sobre la precarga y la precarga recuperable en relación con la migración tras error de la réplica, consulte la información sobre la réplica en *Visión general del producto*.

Cargadores con correlaciones de entidad

Si el cargador se conecta a una correlación de entidad, el cargador debe manejar los objetos de tuple. Los objetos de tuple son un formato especial de datos de entidad. El cargador debe realizar la conversión de datos entre tuple y otros formatos de datos. Por ejemplo, el método `get` devuelve una lista de valores que corresponden al conjunto de claves que se pasan en el método. Las claves que se pasan son de tipo `Tuple`, es decir, tuplas de clave. Si se presupone que el cargador persiste la correlación con una base de datos utilizando JDBC, el método `get` debe convertir cada tuple de clave en una lista de valores de atributo que corresponden a las columnas de clave primaria de la tabla que se correlaciona con la correlación de entidad, ejecute la sentencia `SELECT` con la cláusula `WHERE` que utiliza los valores de atributo convertidos como criterios para captar datos en la base de datos y, a continuación, convertir los datos devueltos en tuplas de valor. El método `get` obtiene datos de la base de datos y los convierte en tuplas de valor para los tuplas de clave pasados y, a continuación, devuelve una lista de tuplas de valor correspondientes al conjunto de claves de tuple que se pasan en el llamante. El método `get` puede realizar una sentencia `SELECT` para captar todos los datos a la vez, o ejecutar una sentencia `SELECT` para cada tuple de clave. Si desea ver detalles de programación que muestran cómo utilizar el cargador cuando se almacenan los datos utilizando un gestor de entidades, consulte “Uso de un cargador con correlaciones de entidad y tuplas” en la página 265.

Consideraciones de programación del cargador 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. Utilice las siguientes consideraciones cuando desarrolle una aplicación que utiliza un cargador JPA.

Entidad eXtreme Scale y entidad JPA

Puede designar cualquier clase POJO como una entidad eXtreme Scale utilizando las anotaciones de entidad eXtreme Scale, la configuración XML, o ambos. También puede designar la misma clase POJO como entidad JPA mediante el uso de anotaciones de entidad JPA o de la configuración de XML.

Entidad eXtreme Scale: una entidad eXtreme Scale representa los datos persistentes almacenados en correlaciones de ObjectGrid. Un objeto de entidad se transforma en un tuple de clave y un tuple de valor, que después se almacenan como pares de clave-valor en las correlaciones. Un tuple es una matriz de atributos primitivos.

Entidad JPA: una entidad JPA representa los datos persistentes almacenados en una base de datos relacional que utiliza automáticamente la persistencia gestionada por contenedor. Los datos persisten en alguna forma de sistema de almacenamiento de datos con el formato adecuado como, por ejemplo, los tuplas de base de datos.

Cuando persiste una entidad eXtreme Scale, sus relaciones se almacenan en otras correlaciones de entidad. Por ejemplo, cuando se persiste una entidad `Consumer` con una relación de uno a muchos con una entidad `ShippingAddress`, si el valor `cascade-persist` está habilitado, la entidad `ShippingAddress` se almacena en la correlación `shippingAddress` en formato de tuple. Si persiste una entidad JPA, las entidades JPA relacionadas también se persisten en las tablas de base de datos, si el valor `cascade-persist` está habilitado. Cuando se designa una clase POJO como una

entidad eXtreme Scale y, también, una entidad JPA, los datos se pueden persistir tanto en correlaciones, como en bases de datos de la entidad ObjectGrid. Los usos comunes son los siguientes:

- **Escenario de precarga:** se carga una entidad desde una base de datos utilizando un proveedor JPA y se conserva en las correlaciones de entidad ObjectGrid.
- **Escenario de cargador:** se conecta una implementación de cargador para las correlaciones de la entidad ObjectGrid, de forma que una entidad almacenada en las correlaciones de la entidad ObjectGrid se puede conservar o cargar desde una base de datos utilizando los proveedores JPA.

También es habitual que una clase POJO se designe únicamente como entidad JPA. En ese caso, lo que se almacena en las correlaciones ObjectGrid son las instancias POJO, frente a los tuples de entidad si se tratara de entidades ObjectGrid.

Consideraciones sobre el diseño de aplicaciones en correlaciones de entidad

Cuando conecta una interfaz JPALoader, las instancias de objeto se almacenan directamente en las correlaciones de ObjectGrid.

Sin embargo, cuando se conecta un JPAEntityLoader, la clase de entidad se designa como entidad eXtreme Scale y, también como una entidad JPA. En este caso, trate a esta entidad como si tuviera dos almacenes persistentes: las correlaciones de entidad ObjectGrid y el almacén de persistencia JPA. La arquitectura es más compleja que el caso de JPALoader.

Si desea más información sobre el plug-in JPAEntityLoader y las consideraciones de diseño de aplicación, consulte la información sobre el plug-in JPAEntityLoader en *Guía de administración*. Esta información también puede ayudarle si tiene previsto implementar su propio cargador para las correlaciones de entidad.

Consideraciones sobre el rendimiento

Asegúrese de que establece el tipo Fetch en EAGER o LAZY para las relaciones. Por ejemplo, una relación Consumer bidireccional de uno a muchos con ShippingAddress, con OpenJPA para ayudar a explicar las diferencias en el rendimiento. En este ejemplo, una consulta JPA intenta select o from Consumer o where . . . para realizar una carga masiva, y también carga todos los objetos ShippingAddress relacionados. Una relación de uno a muchos se define en la clase Consumer del modo siguiente:

```
@Entity
public class Consumer implements Serializable {

    @OneToMany(mappedBy="consumer", cascade=CascadeType.ALL, fetch =FetchType.EAGER)
    ArrayList <ShippingAddress> addresses;
```

A continuación, se muestra el consumidor de una relación de muchos a uno definida en la clase ShippingAddress:

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.EAGER)
    Consumer consumer;
}
```

Si los tipos Fetch de ambas relaciones se configuran como eager, OpenJPA utiliza las consultas N+1+1 para obtener todos los objetos Consumer y objetos

ShippingAddress, donde N es el número de objetos ShippingAddress. Sin embargo, si se modifica el ShippingAddress para utilizar el tipo Fetch LAZY del modo siguiente, sólo toma dos consultas para obtener todos los datos.

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.LAZY)
    Consumer consumer;
}
```

Aunque la consulta devuelve los mismos resultados, tener un número inferior de consultas reduce de forma significativa la interacción con la base de datos, que puede aumentar el rendimiento de la aplicación.

Plug-in JPAEntityLoader

El plug-in JPAEntityLoader es una implementación de cargador incorporada que utiliza Java Persistence API (JPA) para comunicarse con la base de datos cuando se utiliza la API EntityManager. Al utilizar la API ObjectMap, utilice el cargador JPALoader.

Detalles del cargador

Utilice el plug-in JPALoader cuando almacene los datos utilizando la API ObjectMap. Utilice el plug-in JPAEntityLoader cuando almacene los datos mediante la API EntityManager.

Los cargadores proporcionan dos funciones principales:

1. **get**: en el método get, el plug-in JPAEntityLoader llama en primer lugar al método `javax.persistence.EntityManager.find(Class entityClass, Object key)` para encontrar la entidad JPA. El plug-in proyecta esta entidad JPA en los tuples de entidad. Durante la proyección, los atributos del tuple y las claves de la asociación se almacenan en el tuple de valor. Después de procesar cada clave, el método get devuelve una lista de tuples de valor de entidad.
2. **batchUpdate**: el método `batchUpdate` toma un objeto `LogSequence` que contiene una lista de objetos `LogElement`. Cada objeto `LogElement` contiene un tuple de clave y un tuple de valor. Para interactuar con el proveedor de JPA, en primer lugar, debe encontrar la entidad eXtreme Scale basada en el tuple de clave. Basándose en el tipo `LogElement`, ejecute las siguientes llamadas de JPA:
 - **insert**: `javax.persistence.EntityManager.persist(Object o)`
 - **update**: `javax.persistence.EntityManager.merge(Object o)`
 - **remove**: `javax.persistence.EntityManager.remove(Object o)`

Un `LogElement` con el tipo **update** realiza la llamada de JPAEntityLoader al método `javax.persistence.EntityManager.merge(Object o)` para fusionar la entidad. Sin embargo, un tipo **update** de `LogElement` podría ser el resultado de una llamada a `com.ibm.websphere.objectgrid.em.EntityManager.merge(object o)` o un cambio de atributo de la instancia gestionada por el EntityManager de eXtreme Scale. Consulte el siguiente ejemplo:

```
com.ibm.websphere.objectgrid.em.EntityManager em = og.getSession().getEntityManager();
em.getTransaction().begin();
Consumer c1 = (Consumer) em.find(Consumer.class, c.getConsumerId());
c1.setName("New Name");
em.getTransaction().commit();
```

En este ejemplo, un tipo `update` de `LogElement` se envía al JPAEntityLoader del consumidor de la correlación. Se llama al método

`javax.persistence.EntityManager.merge(Object o)` en el gestor de entidades JPA, en lugar de una actualización de atributo a la entidad gestionada por JPA. Debido a este cambio de comportamiento, existen algunas limitaciones con el uso de este modelo de programación.

Reglas sobre el diseño de aplicaciones

Las entidades tienen relaciones con otras entidades. Para diseñar una aplicación con relaciones y con un `JPAEntityLoader` conectado debe tenerse en cuenta una serie de consideraciones adicionales. La aplicación debe seguir cuatro reglas, que se describen en los apartados siguientes.

Soporte de profundidad de relaciones limitada

`JPAEntityLoader` sólo se admite al utilizar entidades sin ninguna relación o entidades con relaciones de un único nivel. No están soportadas las relaciones con más de un nivel como, por ejemplo, `Compañía > Departamento > Empleado`.

Un cargador por correlación

Si utiliza las relaciones de entidad `Consumer-ShippingAddress` como ejemplo, al cargar `Consumer` con el tipo `FETCH` establecido en `EAGER`, puede cargar todos los objetos `ShippingAddress` relacionados. Al persistir o fusionar un objeto `Consumer`, puede persistir o fusionar los objetos `ShippingAddress` relacionados si se ha habilitado el valor `cascade-persist` o `cascade-merge`.

No puede conectar un cargador de la correlación de entidad raíz que almacene los tuples de entidad `Consumer`. Debe configurar un cargador para cada correlación de entidad.

Mismo tipo de valor cascade para JPA y eXtreme Scale

Vuelva a considerar el escenario en el que la entidad `Consumer` tiene una relación de uno a muchos con `ShippingAddress`. Puede consultar el escenario donde se ha habilitado el valor `cascade-persist` para esta relación. Cuando se persiste un objeto `Consumer` en eXtreme Scale, el número `N` asociado de objetos `ShippingAddress` también se persistirá en eXtreme Scale.

Una llamada de persistencia del objeto `Consumer` con una relación `cascade-persist` con `ShippingAddress` se convierte en una llamada al método `javax.persistence.EntityManager.persist(consumer)` y el cargador `JPAEntityLoader` llama `N` veces al método `javax.persistence.EntityManager.persist(shippingAddress)`. Sin embargo, estas `N` llamadas de persistencia adicionales a los objetos `ShippingAddress` no son necesarias debido al valor `cascade-persist` desde el punto de vista del proveedor JPA. Para resolver este problema, eXtreme Scale proporciona un nuevo método `isCascaded` en la interfaz `LogElement`. El método `isCascaded` indica si el `LogElement` es un resultado de una operación `cascade` de eXtreme Scale `EntityManager`. En este ejemplo, el `JPAEntityLoader` de la correlación de `ShippingAddress` recibe `N` objetos `LogElement` debido a las llamadas persistencias en cascada. `JPAEntityLoader` descubre que el método `isCascaded` devuelve un valor `true` y, a continuación, los ignora sin realizar ninguna llamada de JPA. Por lo tanto, desde un punto de vista de JPA, sólo se recibe una llamada del método `javax.persistence.EntityManager.persist(consumer)`.

Este comportamiento se repite si fusiona o elimina una entidad con el valor en cascada habilitado. El plug-in JPAEntityLoader ignora todas las operaciones en cascada.

El diseño del soporte de cascade es reproducir las operaciones de EntityManager de eXtreme Scale para los proveedores JPA. Estas operaciones son persistir, fusionar y eliminar. Para habilitar el soporte de operaciones cascade, verifique que el valor de cascade para el JPA y el EntityManager de eXtreme Scale son iguales.

Use la actualización de entidad con precaución

Como se ha descrito previamente, el diseño del soporte de cascade es reproducir las operaciones EntityManager de eXtreme Scale para los proveedores JPA. Si la aplicación llama al método `ogEM.persist(consumer)` en el EntityManager de eXtreme Scale, aunque los objetos `ShippingAddress` asociados se persistan debido al valor de `cascade-persist`, y JPAEntityLoader sólo llama al método `jpAEM.persist(consumer)` en los proveedores JPA.

Sin embargo, si la aplicación actualiza una entidad gestionada, el plug-in JPAEntityLoader convertirá esta actualización en una llamada de fusión JPA. En este escenario, no está garantizado el soporte de varios niveles de relaciones y asociaciones de claves. En este caso, el procedimiento recomendado es utilizar el método `javax.persistence.EntityManager.merge(o)`, en lugar de actualizar una entidad gestionada.

Uso de un cargador con correlaciones de entidad y tuples

El gestor de entidades convierte todos los objetos de entidad en objetos de tuple antes de que se almacenen en una correlación de WebSphere eXtreme Scale. Cada entidad tiene un tuple de clave y tuple de valor. Este par de clave-valor se almacena en la correlación asociada de eXtreme Scale para la entidad. Al utilizar una correlación eXtreme Scale con un cargador, éste debe interactuar con los objetos de tuple.

Visión general

eXtreme Scale contiene plug-ins de cargador que simplifican la integración con las bases de datos relacionales. Los cargadores JPA (Java Persistence) utilizan una Java Persistence API para interactuar con la base de datos y crear los objetos de entidad. Los cargadores JPA son compatibles con las entidades de eXtreme Scale.

Tuples

Un tuple contiene información sobre los atributos y las asociaciones de una entidad. Los valores primitivos se almacenan mediante derivadores primitivos. Otros tipos de objeto admitidos se almacenan con su formato nativo. Las asociaciones a otras entidades se almacenan como una colección de objetos de tuples de clave que representan las claves de las entidades de destino.

Cada atributo o asociación se almacena mediante un índice basado en cero. Puede recuperar el índice de cada atributo utilizando los métodos `getAttributePosition` o `getAssociationPosition`. Después de que se recupere la posición, permanecerá sin cambios durante el ciclo de vida de eXtreme Scale. La posición puede cambiar cuando se reinicie eXtreme Scale. Los métodos `setAttribute`, `setAssociation` y `setAssociations` se utilizan para actualizar los elementos en el tuple.

Atención: Al crear o actualizar los objetos de tuple, actualice todos los campos primitivos con un valor que no sea nulo. Los valores primitivos como, por ejemplo, `int` no pueden ser nulos. Si no cambia el valor por un valor predeterminado, se pueden generar problemas de rendimiento bajo, que también afectan a los campos marcados con la anotación `@Version` o el atributo de versión en el archivo XML de descriptor de entidad.

El siguiente ejemplo explica de forma adicional cómo procesar tuples. Si desea más información sobre cómo definir entidades para este ejemplo, consulte la información sobre el esquema de entidad `Order` que está en la guía de aprendizaje del gestor de entidades en *Visión general del producto*. WebSphere eXtreme Scale se ha configurado para utilizar cargadores con cada una de las entidades. De forma adicional, sólo se toma la entidad `Order` y esta entidad específica tiene una relación de muchos a uno con la entidad `Customer`. El nombre de atributo es `customer`, y tiene una relación de uno a muchos con la entidad `OrderLine`.

Utilice Projector para crear automáticamente objetos `Tuple` de las entidades. La utilización de Projector puede simplificar los cargadores cuando se utiliza un programa de utilidad de correlaciones de objetos relacionales como, por ejemplo, Hibernate o JPA.

order.java

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

customer.java

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

orderLine.java

```
@Entity
public class OrderLine
{
    @Id @ManyToOne(cascade=CascadeType.PERSIST) Order order;
    @Id int lineNumber;
    @OneToOne(cascade=CascadeType.PERSIST) Item item;
    int quantity;
    double price;
}
```

Una clase `OrderLoader` que implementa la interfaz `Loader` se muestra en el siguiente código. El siguiente ejemplo presupone que se ha definido un plug-in `TransactionCallback` asociado.

orderLoader.java

```
public class OrderLoader implements com.ibm.websphere.objectgrid.plugins.Loader {
    private EntityMetadata entityMetadata;
    public void batchUpdate(TxID txid, LogSequence sequence)
        throws LoaderException, OptimisticCollisionException {
        ...
    }
}
```

```

    }
    public List get(TxID txid, List keyList, boolean forUpdate)
        throws LoaderException {
        ...
    }
    public void preloadMap(Session session, BackingMap backingMap)
        throws LoaderException {
        this.entityMetaData=backingMap.getEntityMetadata();
    }
}

```

La variable de la instancia `entityMetaData` se ha inicializado durante la llamada al método `preLoadMap` desde `eXtreme Scale`. La variable `entityMetaData` no es nula si la correlación se ha configurado para utilizar entidades. De lo contrario, el valor es nulo.

El método `batchUpdate`

El método `batchUpdate` proporciona la capacidad de saber qué acción tiene previsto realizar la aplicación. Basándose en una operación insertar, actualizar o suprimir, se puede abrir una conexión con la base de datos y el trabajo realizado. Puesto que la clave y los valores son del tipo `Tuple`, se deben transformar de forma que los valores tengan sentido en la sentencia SQL.

La tabla `ORDER` se creó con la siguiente definición DDL (lenguaje de definición de datos), tal como se muestra en el código siguiente:

```

CREATE TABLE ORDER (ORDERNUMBER VARCHAR(250) NOT NULL, DATE TIMESTAMP, CUSTOMER_ID VARCHAR(250))
ALTER TABLE ORDER ADD CONSTRAINT PK_ORDER PRIMARY KEY (ORDERNUMBER)

```

El código siguiente muestra cómo convertir un `tuple` en un objeto:

```

public void batchUpdate(TxID txid, LogSequence sequence)
    throws LoaderException, OptimisticCollisionException {
    Iterator iter = sequence.getPendingChanges();
    while (iter.hasNext()) {
        LogElement logElement = (LogElement) iter.next();
        Object key = logElement.getKey();
        Object value = logElement.getCurrentValue();

        switch (logElement.getType().getCode()) {
            case LogElement.CODE_INSERT:
                1) if (entityMetaData!=null) {
                    // El pedido sólo tiene una clave orderNumber
                2) String ORDERNUMBER=(String) getKeyAttribute("orderNumber", (Tuple) key);
                    // Obtener el valor de fecha
                3) java.util.Date unFormattedDate = (java.util.Date) getValueAttribute("date", (Tuple) value);
                    // Los valores son 2 asociaciones. Permite el proceso de clientes porque
                    // la tabla contiene customer.id como clave primaria
                4) Object[] keys= getForeignKeyForValueAssociation("customer","id", (Tuple) value);
                    //Order para Customer es M para 1. Sólo puede haber 1 clave
                    String CUSTOMER_ID=(String)keys[0];
                    // analizar variable unFormattedDate y darle formato para la base de datos como formattedDate
                6) String formattedDate = "2007-05-08-14.01.59.780272"; // formateado para DB2
                    // Por último, la sentencia SQL para insertar el registro
                7) //INSERT INTO ORDER (ORDERNUMBER, DATE, CUSTOMER_ID) VALUES(ORDERNUMBER,formattedDate, CUSTOMER_ID)
                    }
                    break;
                case LogElement.CODE_UPDATE:
                    break;
                case LogElement.CODE_DELETE:
                    break;
            }
        }
    }
}

// devuelve el valor al atributo según está almacenado en el tuple de clave
private Object getKeyAttribute(String attr, Tuple key) {
    //obtener metadatos de clave
    TupleMetadata keyMD = entityMetaData.getKeyMetadata();
    //obtener posición del atributo
    int keyAt = keyMD.getAttributePosition(attr);
    if (keyAt > -1) {
        return key.getAttribute(keyAt);
    } else { // attribute undefined
        throw new IllegalArgumentException("Invalid position index for "+attr);
    }
}

```

```

    }
    // devuelve el valor al atributo según está almacenado en el tuple de valor
    private Object getValueAttribute(String attr, Tuple value) {
        //similar a la operación anterior, excepto que se trabaja con metadatos de valor
        TupleMetadata valueMD = entityMetaData.getValueMetadata();

        int keyAt = valueMD.getAttributePosition(attr);
        if (keyAt > -1) {
            return value.getAttribute(keyAt);
        } else {
            throw new IllegalArgumentException("Invalid position index for "+attr);
        }
    }
}
// devuelve una matriz de claves que se refiere a la asociación.
private Object[] getForeignKeyForValueAssociation(String attr, String fk_attr, Tuple value) {
    TupleMetadata valueMD = entityMetaData.getValueMetadata();
    Object[] ro;

    int customerAssociation = valueMD.getAssociationPosition(attr);
    TupleAssociation tupleAssociation = valueMD.getAssociation(customerAssociation);

    EntityMetadata targetEntityMetadata = tupleAssociation.getTargetEntityMetadata();

    Tuple[] customerKeyTuple = ((Tuple) value).getAssociations(customerAssociation);

    int numberOfKeys = customerKeyTuple.length;
    ro = new Object[numberOfKeys];

    TupleMetadata keyMD = targetEntityMetadata.getKeyMetadata();
    int keyAt = keyMD.getAttributePosition(fk_attr);
    if (keyAt < 0) {
        throw new IllegalArgumentException("Invalid position index for " + attr);
    }
    for (int i = 0; i < numberOfKeys; ++i) {
        ro[i] = customerKeyTuple[i].getAttribute(keyAt);
    }

    return ro;
}
}

```

1. Asegúrese de que `entityMetaData` no es nulo, lo cual implica que las entradas de memoria caché de clave y valor son del tipo `Tuple`. En `entityMetaData`, se recupera la clave `TupleMetadata`, que refleja sólo la parte de clave de los metadatos `Order`.
2. Se procesa `KeyTuple` y se obtiene el valor del atributo de clave `orderNumber`
3. Se procesa `ValueTuple` y se obtiene el valor de la fecha de atributo
4. Se procesa `ValueTuple` y se obtiene el valor de las claves del cliente de asociación
5. Se extrae `CUSTOMER_ID`. Según la relación, un objeto `Order` sólo puede tener un cliente. Tendremos sólo una clave. Por lo tanto, el tamaño de las claves es 1. Se ha pasado por alto el análisis de la fecha para corregir el formato, para que sea más sencillo.
6. Dado que se trata de una operación insert, la sentencia SQL se pasa en la conexión de origen de datos para completar la operación insert.

La demarcación y el acceso de la transacción a la base de datos se cubre en “Escribir un cargador” en la página 257.

El método get

Si no se encuentra la clave en la memoria caché, llame al método `get` en el plug-in `Loader` para encontrar la clave.

La clave es un `Tuple`. El primer paso es convertir el `Tuple` en valores primitivos que se pueden pasar en la sentencia `SELECT` de SQL. Después de que se recuperen todos los atributos de la base de datos, debe convertirlos en `Tuples`. El siguiente código demuestra la clase `Order`.

```

public List get(Txid txid, List keyList, boolean forUpdate) throws LoaderException {
    System.out.println("OrderLoader: Get called");
    ArrayList returnList = new ArrayList();
}

```

```

1) if (entityMetaData != null) {
    int index=0;
    for (Iterator iter = keyList.iterator(); iter.hasNext();) {
2)     Tuple orderKeyTuple=(Tuple) iter.next();

// El pedido sólo tiene una clave orderNumber
3)     String ORDERNUMBERKEY = (String) getKeyAttribute("orderNumber",orderKeyTuple);
//Ejecute una consulta para obtener valores de
4)     // SELECT CUSTOMER_ID, date FROM ORDER WHERE ORDERNUMBER='ORDERNUMBERKEY'

5)     //1) Clave foránea: CUSTOMER_ID
6)     //2) fecha
// Se presupone que éstos se devuelven como
7)         String CUSTOMER_ID = "C001"; // Se presupone recuperación e inicialización
8)     java.util.Date retrievedDate = new java.util.Date();
// Se presupone que esta fecha refleja la de la base de datos

// A continuación, se deben convertir estos datos en un tuple antes de devolver

//crear un tuple de valor
9)     TupleMetadata valueMD = entityMetaData.getValueMetadata();
    Tuple valueTuple=valueMD.createTuple();

//añadir objeto retrievedDate a Tuple
    int datePosition = valueMD.getAttributePosition("date");
10)    valueTuple.setAttribute(datePosition, retrievedDate);

//A continuación se debe añadir la asociación
11)    int customerPosition=valueMD.getAssociationPosition("customer");
    TupleAssociation customerTupleAssociation =
        valueMD.getAssociation(customerPosition);
    EntityMetadata customerEMD = customerTupleAssociation.getTargetEntityMetadata();
    TupleMetadata customerTupleMDForKEY=customerEMD.getKeyMetadata();
12)    int customerKeyAt=customerTupleMDForKEY.getAttributePosition("id");

    Tuple customerKeyTuple=customerTupleMDForKEY.createTuple();
    customerKeyTuple.setAttribute(customerKeyAt, CUSTOMER_ID);
13)    valueTuple.addAssociationKeys(customerPosition, new Tuple[] {customerKeyTuple});

14)    int linesPosition = valueMD.getAssociationPosition("lines");
    TupleAssociation linesTupleAssociation = valueMD.getAssociation(linesPosition);
    EntityMetadata orderLineEMD = linesTupleAssociation.getTargetEntityMetadata();
    TupleMetadata orderLineTupleMDForKEY = orderLineEMD.getKeyMetadata();
    int lineNumberAt = orderLineTupleMDForKEY.getAttributePosition("lineNumber");
    int orderAt = orderLineTupleMDForKEY.getAssociationPosition("order");

    if (lineNumberAt < 0 || orderAt < 0) {
        throw new IllegalArgumentException(
            "Invalid position index for lineNumber or order "+
            lineNumberAt + " " + orderAt);
    }
15) // SELECT LINENUMBER FROM ORDERLINE WHERE ORDERNUMBER='ORDERNUMBERKEY'
// Se presupone que dos filas de número de línea se devuelven con los valores 1 y 2

    Tuple orderLineKeyTuple1 = orderLineTupleMDForKEY.createTuple();
    orderLineKeyTuple1.setAttribute(lineNumberAt, new Integer(1));// set Key
    orderLineKeyTuple1.addAssociationKey(orderAt, orderKeyTuple);

    Tuple orderLineKeyTuple2 = orderLineTupleMDForKEY.createTuple();
    orderLineKeyTuple2.setAttribute(lineNumberAt, new Integer(2));// Init Key
    orderLineKeyTuple2.addAssociationKey(orderAt, orderKeyTuple);

16)    valueTuple.addAssociationKeys(linesPosition, new Tuple[]
        {orderLineKeyTuple1, orderLineKeyTuple2 });

    returnList.add(index, valueTuple);

    index++;
}
} else {
// no admite tuples
}
return returnList;
}

```

1. Se llama al método get cuando la memoria caché de ObjectGrid no ha podido encontrar la clave y solicita al cargador que la capte. Pruebe el valor de entityMetaData y continúe si el valor no es nulo.
2. keyList contiene tuples.
3. Recupere el valor del atributo orderNumber.
4. Ejecute la consulta para recuperar la fecha (valor) y el ID de cliente (clave foránea).

5. CUSTOMER_ID es una clave foránea que se debe establecer en el tuple de asociación.
6. La fecha es un valor y ya debería estar definido.
7. Puesto que este ejemplo no realiza llamadas JDBC, se da por supuesto el CUSTOMER_ID.
8. Dado que este ejemplo no realiza llamadas JDBC, la fecha se da por supuesta.
9. Cree el valor de Tuple.
10. Establezca el valor de la fecha en el Tuple, basándose en su posición.
11. Order tiene dos asociaciones. Empiece con el atributo customer que se refiere a la entidad customer. Debe tener el valor del ID para establecerlo en el tuple.
12. Encuentre la posición del ID en la entidad del cliente.
13. Establezca sólo los valores de las claves de asociación.
14. Además, las líneas son una asociación que se debe configurar como un grupo de claves de asociación, de la misma forma que lo haría para la asociación de cliente.
15. Puesto que debe configurar las claves para el lineNumber asociado con este pedido, ejecute SQL para recuperar los valores de lineNumber.
16. Configure las claves de asociación en el valueTuple. Esto completa la creación del Tuple que se ha devuelto a BackingMap.

En este tema se ofrecen los pasos para crear tuples, y una descripción de la entidad Order solamente. Siga pasos similares para otras entidades, y todo el proceso unido al plug-in TransactionCallback. Consulte “Plug-ins para gestionar los sucesos del ciclo de vida de transacciones” en la página 275 si desea más detalles.

Escribir un cargador con un controlador de precarga de réplica

Un cargador con un controlador de precarga de réplica es un cargador que implementa la interfaz ReplicaPreloadController además de la interfaz del cargador.

Visión general

La interfaz ReplicaPreloadController se ha diseñado para proporcionar un modo de que la réplica que se convierte en fragmento primario sepa si el fragmento primario anterior ha completado el proceso de precarga. Si la precarga se ha completado parcialmente, se ofrece información sobre el punto en el que se quedó la correlación primaria anterior. Con la implementación de la interfaz ReplicaPreloadController, una réplica que pasa a ser la primaria continúa el proceso de precarga en el punto donde se detuvo el primario anterior y continúa hasta finalizar la operación de precarga general.

En un entorno distribuido de WebSphere eXtreme Scale, una correlación puede tener réplicas y podría precargar un gran volumen de datos durante la inicialización. La precarga es una actividad del cargador y sólo tiene lugar en la correlación primaria durante la inicialización. La operación de precarga tardará en completarse si el volumen de datos que se va a precargar es muy grande. Si la correlación primaria ha precargado una parte considerable de datos, pero se ha detenido durante la inicialización sin motivo aparente, una réplica pasa a ser la réplica primaria. En esta situación, los datos que ha precargado la correlación primaria anterior se pierden porque la nueva réplica primaria normalmente realiza una precarga incondicional. Esto quiere decir que la nueva réplica primaria inicia

el proceso de precarga desde el principio y pasa por alto los datos precargados previamente. Si desea que el nuevo primario empiece por el punto en que se detuvo el primario anterior durante el proceso de precarga, proporcione un cargador que implemente la interfaz `ReplicaPreloadController`. Si desea más información, consulte la documentación de la API.

Si desea más información sobre los cargadores, consulte la información sobre cargadores en *Visión general del producto*. Si está interesado en escribir un plug-in Loader regular, consulte “Escribir un cargador” en la página 257.

La interfaz `ReplicaPreloadController` tiene la siguiente definición:

```
public interface ReplicaPreloadController
{
    public static final class Status
    {
        static public final Status PRELOADED_ALREADY = new Status(K_PRELOADED_ALREADY);
        static public final Status FULL_PRELOAD_NEEDED = new Status(K_FULL_PRELOAD_NEEDED);
        static public final Status PARTIAL_PRELOAD_NEEDED = new Status(K_PARTIAL_PRELOAD_NEEDED);
    }

    Status checkPreloadStatus(Session session, BackingMap bmap);
}
```

Las siguientes secciones describen algunos de los métodos de la interfaz `Loader` y `ReplicaPreloadController`.

Método `checkPreloadStatus`

Cuando un cargador implementa la interfaz `ReplicaPreloadController`, se llama al método `checkPreloadStatus` antes que el método `preloadMap` durante la inicialización de la correlación. El estado devuelto de este método determina si se llama al método `preloadMap`. Si este método devuelve `Status#PRELOADED_ALREADY`, se llama al método de precarga. De lo contrario, se ejecuta el método `preload`. Debido a este comportamiento, este método debe servir como método de inicialización del cargador. Debe inicializar las propiedades del cargador en este método. Este método debe devolver el estado correcto, o la operación de precarga podría no funcionar como se espera.

```
public Status checkPreloadStatus(Session session, BackingMap backingMap) {
// Cuando un cargador implementa la interfaz ReplicaPreloadController,
// se llamará a este método antes que al método preloadMap durante la
// inicialización de la correlación. En función del estado devuelto de
// este método, se llamará o no al método preloadMap. Por ello, este
// método sirve también como el método de inicialización del cargador.
Este método
// debe devolver el estado correcto, si no, puede que la precarga no
// funcione como se espera.

    // Nota: debe inicializar esta instancia de cargador a continuación.
    ivOptimisticCallback = backingMap.getOptimisticCallback();
    ivBackingMapName = backingMap.getName();
    ivPartitionId = backingMap.getPartitionId();
    ivPartitionManager = backingMap.getPartitionManager();
    ivTransformer = backingMap.getObjectTransformer();
    preloadStatusKey = ivBackingMapName + "_" + ivPartitionId;

    try {
        // obtener preloadStatusMap para recuperar el estado de precarga
        // que otras JVM podrían haber establecido.
        ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);

        // recuperar el índice de fragmento de datos registrados por última vez.
        Integer lastPreloadedDataChunk = (Integer) preloadStatusMap.get(preloadStatusKey);
        if (lastPreloadedDataChunk == null) {
            preloadStatus = Status.FULL_PRELOAD_NEEDED;
        } else {
            preloadedLastDataChunkIndex = lastPreloadedDataChunk.intValue();
            if (preloadedLastDataChunkIndex == preloadCompleteMark) {
                preloadStatus = Status.PRELOADED_ALREADY;
            }
        }
    }
}
```

```

        } else {
            preloadStatus = Status.PARTIAL_PRELOAD_NEEDED;
        }
    }

    System.out.println("TupleHeapCacheWithReplicaPreloadControllerLoader.
    checkPreloadStatus()
-> map = " + ivBackingMapName + ", preloadStatusKey = " + preloadStatusKey
    + ", retrieved lastPreloadedDataChunk = " + lastPreloadedDataChunk + ",
    determined preloadStatus = "
+ getStatusString(preloadStatus));

    } catch (Throwable t) {
        t.printStackTrace();
    }

    return preloadStatus;
}

```

Método preloadMap

La ejecución del método `preloadMap` depende del resultado devuelto del método `checkPreloadStatus`. Si se llama al método `preloadMap`, normalmente debe recuperar la información del estado de precarga en la correlación de estado de precarga designada y determinar cómo continuar. Lo ideal es que el método `preloadMap` sepa si la precarga se ha completado parcialmente y en qué punto debe comenzar exactamente. Durante la precarga de datos, el método `preloadMap` debe actualizar el estado de precarga en la correlación designada del estado de precarga. El estado de precarga que se almacena en la correlación de estado de precarga es recuperado por el método `checkPreloadStatus` cuando necesita comprobar el estado de la precarga.

```

public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException {
    EntityMetadata emd = backingMap.getEntityMetadata();
    if (emd != null && tupleHeapPreloadData != null) {
        // El método getPreLoadData es similar a captar datos
        // de una base de datos. Estos datos se pasarán a la memoria caché
        // como proceso de precarga.
        ivPreloadData = tupleHeapPreloadData.getPreLoadData(emd);

        ivOptimisticCallback = backingMap.getOptimisticCallback();
        ivBackingMapName = backingMap.getName();
        ivPartitionId = backingMap.getPartitionId();
        ivPartitionManager = backingMap.getPartitionManager();
        ivTransformer = backingMap.getObjectTransformer();
        Map preloadMap;

        if (ivPreloadData != null) {
            try {
                ObjectMap map = session.getMap(ivBackingMapName);

                // obtener preloadStatusMap para registrar el estado de precarga.
                ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);
                // Nota: cuando se invoca este método preloadMap, se ha llamado
                // a checkPreloadStatus, se han establecido preloadStatus
                // y preloadedLastDataChunkIndex. Y
                // preloadStatus debe ser PARTIAL_PRELOAD_NEEDED
                // o FULL_PRELOAD_NEEDED que necesitarán una precarga de nuevo.

                // Si el volumen de datos que debe precargarse es muy grande, los datos
                // se suelen dividir en fragmentos y el proceso de precarga
                // procesará cada fragmento dentro de su propia transacción.. En este ejemplo sólo
                // se precargan unas pocas entradas, cada una de las cuales representa un fragmento.
                // Por tanto, la precarga procesa una entrada en una transacción para simular
                // la precarga de un fragmento.

                Set entrySet = ivPreloadData.entrySet();
                preloadMap = new HashMap();
                ivMap = preloadMap;

                // dataChunkIndex representa el fragmento de datos
                // en proceso
                int dataChunkIndex = -1;
                boolean shouldRecordPreloadStatus = false;

```

```

        int numberOfDataChunk = entrySet.size();
        System.out.println("    numberOfDataChunk to be preloaded = "
            + numberOfDataChunk);
    Iterator it = entrySet.iterator();
    int whileCounter = 0;
    while (it.hasNext()) {
        whileCounter++;
        System.out.println("preloadStatusKey = " + preloadStatusKey + " ,
            whileCounter = " + whileCounter);
    dataChunkIndex++;

        // Si dataChunkIndex <= preloadedLastDataChunkIndex
        // no es necesario realizar el proceso, porque lo ha precargado
    // antes otra JVM. Sólo se tiene que procesar dataChunkIndex
    // > preloadedLastDataChunkIndex
        if (dataChunkIndex <= preloadedLastDataChunkIndex) {
            System.out.println("ignore current dataChunkIndex =
                " + dataChunkIndex + " that has been previously
                preloaded.");
            continue;
        }

        // Nota: este ejemplo simula un fragmento de datos como entrada.
        // Cada clave representa un fragmento de datos por motivos de simplificación.
        // Si el servidor o fragmento primario se ha detenido por razones desconocidas,
    // el estado de precarga que indica el progreso
    // de la precarga debe estar disponible en preloadStatusMap. Una
    // réplica que pasa a ser primaria puede obtener el estado de precarga
    // y determinar cómo realizar la precarga de nuevo.
        // Nota: la grabación del estado de precarga debe estar en la misma
    // transacción que incluir los datos en memoria caché; por lo que si se produce una
    // retrotracción o un error de la transacción, el estado de precarga grabado es el
    // estado real.

        Map.Entry entry = (Entry) it.next();
        Object key = entry.getKey();
        Object value = entry.getValue();
        boolean tranActive = false;

        System.out.println("processing data chunk. map = " +
            this.ivBackingMapName + ", current dataChunkIndex = " +
            dataChunkIndex + ", key = " + key);

        try {
            shouldRecordPreloadStatus = false; // re-set to false
            session.beginNoWriteThrough();
            tranActive = true;

            if (ivPartitionManager.getNumOfPartitions() == 1) {
                // si sólo hay 1 partición, no es necesario realizar ninguna operación
            // con ella.
                // pase los datos a la memoria caché
                map.put(key, value);
                preloadMap.put(key, value);
                shouldRecordPreloadStatus = true;
            } else if (ivPartitionManager.getPartition(key) == ivPartitionId) {
    // si la correlación está particionada, es necesario considerar que la
            // clave de partición sólo precarga los datos que pertenecen
            // a esta partición.
                map.put(key, value);
                preloadMap.put(key, value);
                shouldRecordPreloadStatus = true;
            } else {
                // pasar por alto esta entrada porque no pertenece a
            // esta partición.
            }

            if (shouldRecordPreloadStatus) {
                System.out.println("record preload status. map = " +
                    this.ivBackingMapName + ", preloadStatusKey = " +
                    preloadStatusKey + ", current dataChunkIndex = "
                + dataChunkIndex);
                if (dataChunkIndex == numberOfDataChunk) {
                    System.out.println("record preload status. map = " +
                        this.ivBackingMapName + ", preloadStatusKey = " +
                        preloadStatusKey + ", mark complete = " +
                        preloadCompleteMark);
                    // significa que estamos en el último fragmento de datos,
                    // si la operación se confirma correctamente,
                    // el registro de la precarga se ha completado.
                }
            }
        } catch (Exception e) {
            // ...
        }
    }
}

```


correlación de estado de precarga, determinar el estado de precarga y devolver el estado al llamante. La correlación de estado de precarga debe estar en el mismo objeto mapSet que otras correlaciones que tienen cargadores de controlador de precarga de réplica.

Plug-ins para gestionar los sucesos del ciclo de vida de transacciones

Utilice el plug-in TransactionCallback para personalizar las operaciones de creación de versiones y de comparación de los objetos de la memoria caché cuando se utiliza la estrategia de bloqueo optimista.

Puede proporcionar un objeto de devolución de llamada optimista conectable que implementa la interfaz `com.ibm.websphere.objectgrid.plugins.OptimisticCallback`. Para las correlaciones de entidades, se configura automáticamente un plug-in `OptimisticCallback` de alto rendimiento.

Finalidad

Utilice la interfaz `OptimisticCallback` para proporcionar operaciones de comparación optimista para los valores de una correlación. Es necesaria una implementación `OptimisticCallback` cuando se utiliza la estrategia de bloqueo optimista. WebSphere eXtreme Scale proporciona una implementación predeterminada de `OptimisticCallback`. Sin embargo, normalmente la aplicación debe conectar su propia implementación de la interfaz `OptimisticCallback`. Consulte la información sobre las estrategias de bloqueo en *Visión general del producto* si desea más información.

Implementación predeterminada

La infraestructura de eXtreme Scale proporciona una implementación predeterminada de la interfaz `OptimisticCallback` que se utiliza si la aplicación no se conecta a un objeto `OptimisticCallback` proporcionado por la aplicación, tal como se demuestra en la sección anterior. La implementación predeterminada siempre devuelve el valor especial de `NULL_OPTIMISTIC_VERSION` como el objeto de versión del valor y nunca actualiza el objeto de versión. Esta acción realiza una comparación optimista, una operación sin función. En la mayoría de los casos, no desea que se produzca la función sin operación al utilizar la estrategia de bloqueo optimista. Las aplicaciones deben implementar la interfaz `OptimisticCallback` y conectar sus propias implementaciones de `OptimisticCallback`, de forma que no se utilice la implementación predeterminada. Sin embargo, como mínimo, existe un escenario donde resulta práctica la implementación proporcionada predeterminada de `OptimisticCallback`. Observe la situación siguiente:

- Se conecta un cargador para la correlación de respaldo.
- El cargador sabe cómo realizar la comparación optimista sin ayuda de un plug-in `OptimisticCallback`.

¿Cómo puede saber el cargador cómo tratar con la creación de versiones optimista sin la ayuda de un objeto `OptimisticCallback`? El cargador conoce el objeto de clase de valor y sabe qué campo del objeto de valor se utiliza como valor de creación de versiones optimista. Por ejemplo, imagine que se utiliza la interfaz siguiente para el objeto de valor de la correlación de empleados.

```
public interface Employee
{
    // Número de secuencia utilizado para la creación de versiones optimista.
```

```

    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Otros métodos get/set para otros campos del objeto Employee.
}

```

En este caso, el cargador sabe que puede utilizar el método `getSequenceNumber` para obtener la información de creación de versiones actual para un objeto de valor `Employee`. El cargador incrementa el valor devuelto para generar un nuevo número de versión antes de actualizar el almacenamiento persistente con el nuevo valor `Employee`. Para un cargador JDBC (Java Database Connectivity), se utiliza el número de secuencia actual en la cláusula `where` de una sentencia de actualización sobrecualificada de SQL y utiliza el nuevo número de secuencia generado para establecer la columna del número de secuencia en el nuevo valor de número de secuencia.

Otra posibilidad es que el cargador utilice una función, que proporciona el programa de fondo, que actualiza automáticamente una columna oculta que puede utilizarse para la creación de versiones optimista. En algunos casos, puede usarse un procedimiento almacenado o un desencadenante para ayudar a mantener una columna que contiene la información sobre la creación de versiones. Si el cargador utiliza una de estas técnicas para mantener la información de la creación de versiones optimista, la aplicación no necesita proporcionar una implementación `OptimisticCallback`. Puede utilizar la implementación predeterminada de `OptimisticCallback` porque el cargador puede manejar la creación de versiones optimista sin la ayuda de un objeto `OptimisticCallback`.

Implementación predeterminada de entidades

Las entidades se almacenan en `ObjectGrid` mediante objetos de tuple. La implementación predeterminada de `OptimisticCallback` se comporta del mismo modo que el comportamiento para las correlaciones sin entidad. Sin embargo, el campo de versión de la entidad se identifica a través del uso de la anotación `@Version` o el atributo de versión en el archivo XML de descriptor de la entidad.

El atributo de versión puede ser de uno de los tipos siguientes: `int`, `Integer`, `short`, `Short`, `long`, `Long` o `java.sql.Timestamp`. Una entidad debe tener sólo un atributo de versión definido. El atributo de versión sólo se debe establecer durante la construcción. Después de persistir la entidad, el valor del atributo de versión no se debe modificar.

Si no se configura el atributo de versión y se utiliza la estrategia de bloqueo optimista, se crea una versión implícitamente de todo el tuple utilizando el estado completo del tuple.

En el ejemplo siguiente, la entidad `Employee` tiene un atributo de versión de tipo `long` denominado `SequenceNumber`:

```

@Entity
public class Employee
{
    private long sequence;
    // Número de secuencia utilizado para la creación de versiones optimista.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {

```

```

        this.sequence = newSequenceNumber;
    }
    // Otros métodos get/set para otros campos del objeto Employee.
}

```

Escritura de una implementación de OptimisticCallback

Una implementación de OptimisticCallback debe implementar la interfaz OptimisticCallback y seguir los convenios comunes de plug-in de ObjectGrid

La siguiente lista proporciona una descripción o una consideración para cada uno de los métodos de la interfaz OptimisticCallback:

NULL_OPTIMISTIC_VERSION

Este valor especial es devuelto por el método getVersionedObjectForValue si se utiliza la implementación predeterminada de OptimisticCallback, en lugar de una predeterminado de OptimisticCallback proporcionada por la aplicación.

Método getVersionedObjectForValue

El método getVersionedObjectForValue podría devolver una copia del valor, o podría devolver un atributo del valor que se puede utilizar para la creación de versiones. Este método se llama siempre que se asocia un objeto con una transacción. Cuando no se establece ningún cargador en una correlación de respaldo, la correlación de respaldo utiliza este valor en la fase de confirmación para realizar una comparación de versiones optimista. La correlación de respaldo utiliza la comparación de versiones optimista para garantizar que la versión no ha cambiado desde la primera vez que esta transacción accedió a la entrada de correlación modificada por esta transacción. Si otra transacción hubiera modificado la versión de esta entrada de correlación, se produciría una anomalía en la comparación de versiones y la correlación de respaldo mostraría una excepción OptimisticCollisionException que forzaría la retrotracción de la transacción. Si hay un cargador conectado, la correlación de respaldo no utiliza la información de creación de versiones optimista. En su lugar, el cargador deberá realizar una comparación de versiones optimista y actualizar la información de la creación de versiones cuando sea necesario. Normalmente, el cargador obtiene el objeto inicial de la creación de versiones del LogElement pasado al método batchUpdate en el cargador, que se llama cuando se produce una operación de vaciado o cuando se confirma una transacción.

El código siguiente muestra la implementación que utiliza el objeto EmployeeOptimisticCallbackImpl:

```

public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}

```

Tal como se ha demostrado en el ejemplo anterior, el atributo sequenceNumber se devuelve en un objeto java.lang.Long tal como esperaba el cargador, que implica

que la misma que escribió el cargador, también escribió la implementación de `EmployeeOptimisticCallbackImpl`, o bien trabajó estrechamente con la persona que implementó la implementación de `EmployeeOptimisticCallbackImpl`. Por ejemplo, estas personas acordaron el valor devuelto por el método `getVersionedObjectForValue`. Tal como se ha descrito previamente, la implementación predeterminada de `OptimisticCallback` devuelve el valor especial `NULL_OPTIMISTIC_VERSION` como el objeto de la versión.

Método `updateVersionedObjectForValue`

Se llama al método `updateVersionedObjectForValue` cuando una transacción ha actualizado un valor y es necesario un nuevo objeto de versión. Si el método `getVersionedObjectForValue` devuelve un atributo del valor, este método suele actualizar el valor de atributo con un nuevo objeto de versión. Si el método `getVersionedObjectForValue` devuelve una copia del valor, este método normalmente no se actualiza. El `OptimisticCallback` predeterminado no se actualiza porque la implementación predeterminada del método `getVersionedObjectForValue` siempre devuelve el valor especial `NULL_OPTIMISTIC_VERSION` como el objeto de versión. El siguiente ejemplo muestra la implementación utilizada por el objeto `EmployeeOptimisticCallbackImpl` que se utiliza en la sección `OptimisticCallback`:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Como se demuestra en el ejemplo anterior, el atributo `sequenceNumber` se incrementa por uno, de forma que la próxima vez que se llame al método `getVersionedObjectForValue`, el valor `java.lang.Long` devuelto tiene un valor largo que es el valor de número de secuencia original. Más de uno, por ejemplo, es el siguiente valor de versión para esta instancia de empleado. De nuevo, este ejemplo implica que la misma persona que escribió el cargador escribió `EmployeeOptimisticCallbackImpl` o colaboró con la persona que implementó `EmployeeOptimisticCallbackImpl`.

Método `serializeVersionedValue`

Este método escribe el valor con versión en la corriente especificada. En función de la implementación, el valor con versión puede utilizarse para identificar colisiones de actualización optimista. En algunas implementaciones, el valor con versión es una copia del valor original. Otras implementaciones podrían tener un número de secuencia o algún otro objeto para indicar la versión del valor. Puesto que se desconoce la implementación real, este método se proporciona para realizar la serialización correcta. La implementación predeterminada llama al método `writeObject`.

Método `inflateVersionedValue`

Este método toma la versión serializada del valor con versión y devuelve el objeto de valor con versión real. En función de la implementación, el valor con versión puede utilizarse para identificar colisiones de actualización optimista. En algunas implementaciones, el valor con versión es una copia del valor original. Otras implementaciones podrían tener un número de secuencia o algún otro objeto para

indicar la versión del valor. Puesto que la implementación real se desconoce, se proporciona este método para realizar la deserialización adecuada. La implementación predeterminada llama al método `readObject`.

Utilización de una implementación de `OptimisticCallback` proporcionada por la aplicación

Tiene dos enfoques para añadir un `OptimisticCallback` proporcionado por la aplicación en la configuración de `BackingMap`: configuración mediante programa y configuración de XML.

Conexión mediante programa de una implementación de `OptimisticCallback`

El siguiente ejemplo demuestra cómo una aplicación puede conectar mediante programa un objeto `OptimisticCallback` para la correlación de respaldo de empleado en la instancia del `ObjectGrid` `grid1`:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

Enfoque de configuración de XML para conectar una implementación de `OptimisticCallback`

El objeto `EmployeeOptimisticCallbackImpl` del ejemplo anterior debe implementar la interfaz `OptimisticCallback`. La aplicación también puede utilizar un archivo XML para conectar su objeto `OptimisticCallback` tal como se muestra en el siguiente ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
  <objectGrid name="grid1">
    <backingMap name="employees" pluginCollectionRef="employees" lockStrategy="OPTIMISTIC" />
  </objectGrid>
</objectGrids>

<backingMapPluginCollections>
  <backingMapPluginCollection id="employees">
    <bean id="OptimisticCallback" className="com.xyz.EmployeeOptimisticCallbackImpl" />
  </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Visión general del proceso de transacciones

Introducción a las ranuras de plug-in

Una ranura de plug-in es un espacio de almacenamiento transaccional que se reserva para los plug-ins que comparten un contexto transaccional. Estas ranuras proporcionan una forma para que los plug-ins de eXtreme Scale se comuniquen entre sí, compartan el contexto transaccional y garanticen que los recursos transaccionales se utilizan de forma correcta y coherente dentro de una transacción.

Un plug-in puede almacenar el contexto transaccional como, por ejemplo, una conexión de base de datos, una conexión JMS (Java Message Service), etc. en una ranura de plug-in. El contexto transaccional almacenado puede recuperarse

mediante cualquier plug-in que conozca el número de ranura de plug-in, que sirve como clave para recuperar el contexto transaccional.

Uso de ranuras de plug-in

Las ranuras de plug-in forman parte de la interfaz TxID. Consulte la documentación de la API si desea más información sobre la interfaz. Las ranuras son entradas en una matriz ArrayList. Los plug-ins pueden reservar una entrada en la matriz ArrayList llamando al método ObjectGrid.reserveSlot e indicando que desea una ranura en todos los objetos TxID. Después de reservar las ranuras, los plug-ins pueden colocar contexto transaccional en las ranuras de cada objeto TxID y recuperar el contexto más adelante. Las operaciones put y get se coordinan a través de números de ranura devueltos por el método ObjectGrid.reserveSlot.

Normalmente, un plug-in tiene un ciclo de vida. El uso de ranuras de plug-in debe ajustarse al ciclo de vida del plug-in. Por lo general, el plug-in debe reservar las ranuras de plug-in durante la fase de inicialización y obtener los números de ranura para cada ranura. Durante la ejecución normal, el plug-in coloca el contexto transaccional en la ranura reservada en el objeto TxID en el punto apropiado. Este punto apropiado suele estar el principio de la transacción. A continuación, el plug-in u otros plug-ins pueden obtener el contexto transaccional almacenado mediante el número de ranura desde TxID dentro de la transacción.

Normalmente, el plug-in realiza una limpieza eliminando el contexto transaccional y las ranuras. El siguiente fragmento de código ilustra cómo utilizar las ranuras de plug-in en un plug-in TransactionCallback:

```
public class DatabaseTransactionCallback implements TransactionCallback {
    int connectionSlot;
    int autoCommitConnectionSlot;
    int psCacheSlot;
    Properties ivProperties = new Properties();

    public void initialize(ObjectGrid objectGrid) throws TransactionCallbackException {
        // En la fase de inicialización, reservar las ranuras de plug-in deseadas llamando al
        // método reserveSlot de ObjectGrid y
        // pasar el nombre de ranura designado, TxID.SLOT_NAME.
        // Nota: debe pasarlo en este TxID.SLOT_NAME diseñado para
        // la aplicación.
        try {
            // almacenar en memoria caché los números de ranura devueltos
            connectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
            psCacheSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
            autoCommitConnectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
        } catch (Exception e) {
        }
    }

    public void begin(TxID tx) throws TransactionCallbackException {
        // colocar contextos transaccionales en las ranuras reservadas al
        // principio de la transacción.
        try {
            Connection conn = null;
            conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
            tx.putSlot(connectionSlot, conn);
            conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
            conn.setAutoCommit(true);
            tx.putSlot(autoCommitConnectionSlot, conn);
            tx.putSlot(psCacheSlot, new HashMap());
        } catch (SQLException e) {
            SQLException ex = getLastSQLException(e);
            throw new TransactionCallbackException("unable to get connection", ex);
        }
    }

    public void commit(TxID id) throws TransactionCallbackException {
        // obtener los contextos transaccionales almacenados y utilizarlos
        // después, limpiar todos los recursos transaccionales.
        try {
            Connection conn = (Connection) id.getSlot(connectionSlot);
            conn.commit();
            cleanUpSlots(id);
        } catch (SQLException e) {
            SQLException ex = getLastSQLException(e);
            throw new TransactionCallbackException("commit failure", ex);
        }
    }
}
```

```

    }
}

void cleanUpSlots(TxID tx) throws TransactionCallbackException {
    closePreparedStatements((Map) tx.getSlot(psCacheSlot));
    closeConnection((Connection) tx.getSlot(connectionSlot));
    closeConnection((Connection) tx.getSlot(autoCommitConnectionSlot));
}

/**
 * @establecer parámetro map
 */
private void closePreparedStatements(Map psCache) {
    try {
        Collection statements = psCache.values();
        Iterator iter = statements.iterator();
        while (iter.hasNext()) {
            PreparedStatement stmt = (PreparedStatement) iter.next();
            stmt.close();
        }
    } catch (Throwable e) {
    }
}

/**
 * Cerrar conexión y aceptar cualquier objeto Throwable que se produzca.
 *
 * @establecer parámetro connection
 */
private void closeConnection(Connection connection) {
    try {
        connection.close();
    } catch (Throwable e1) {
    }
}

public void rollback(TxID id) throws TransactionCallbackException
// obtener los contextos transaccionales almacenados y utilizarlos
// después, limpiar todos los recursos transaccionales.
    try {
        Connection conn = (Connection) id.getSlot(connectionSlot);
        conn.rollback();
        cleanUpSlots(id);
    } catch (SQLException e) {
    }
}

public boolean isExternalTransactionActive(Session session) {
    return false;
}

// Métodos getter para los números de ranura, otro plug-in puede obtener los números de ranura
// de estos métodos getter.

public int getConnectionSlot() {
    return connectionSlot;
}

public int getAutoCommitConnectionSlot() {
    return autoCommitConnectionSlot;
}

public int getPreparedStatementSlot() {
    return psCacheSlot;
}
}

```

El siguiente fragmento de código ilustra cómo un cargador puede obtener el contexto transaccional almacenado colocado por un plug-in de ejemplo TransactionCallback anterior:

```

public class DatabaseLoader implements Loader
{
    DatabaseTransactionCallback tcb;
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
    {
        // El método preload es el método de inicialización de Loader.
        // Obtener el plug-in deseado de la instancia Session u ObjectGrid.
        tcb = (DatabaseTransactionCallback)session.getObjectGrid().getTransactionCallback();
    }

    public List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException
    {
        // obtener los contextos transaccionales almacenados que coloca el método begin de tcb.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implementar get aquí
        return null;
    }

    public void batchUpdate(TxID txid, LogSequence sequence) throws LoaderException, OptimisticCollisionException
    {
        // obtener los contextos transaccionales almacenados que coloca el método begin de tcb.
    }
}

```

```

        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implementar actualización de proceso por lotes aquí...
    }
}

```

Gestores de transacciones externas

Normalmente, las transacciones eXtreme Scale empiezan con el método `Session.begin` y finalizan con el método `Session.commit`. Sin embargo, cuando se incorpora un `ObjectGrid`, un coordinador de transacciones externas puede iniciar y finalizar transacciones. En este caso, no es necesario llamar a los métodos `begin` o `commit`.

Coordinación de transacciones externas

El plug-in `TransactionCallback` se amplía con el método `isExternalTransactionActive(Session session)` que asocia la sesión de eXtreme Scale con una transacción externa. La cabecera del método es la siguiente:

```
public synchronized boolean isExternalTransactionActive(Session session)
```

Por ejemplo, eXtreme Scale se puede configurar para integrarse con `WebSphere Application Server` y `WebSphere Extended Deployment`.

Además, eXtreme Scale proporciona un plug-in incorporado llamado `WebSphere "Plug-ins para gestionar los sucesos del ciclo de vida de transacciones"` en la página 275, que describe cómo generar el plug-in para los entornos de `WebSphere Application Server`, pero puede adaptar el plug-in para otras infraestructuras.

La clave de esta integración sin fisuras es la explotación de la API `ExtendedJTATransaction` en `WebSphere Application Server` versión 5.x y versión 6.x. Sin embargo, si utiliza `WebSphere Application Server` versión 6.0.2, debe aplicar el APAR PK07848 para soportar este método. Utilice el siguiente código de ejemplo para asociar a un objeto `ObjectGrid` con un ID de transacción de `WebSphere Application Server`:

```

/**
 * Este método es necesario para asociar una sesión de objectGrid con un ID de
 * transacción de WebSphere Application Server.
 */
Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
    // recuerde que este localid significa que la sesión se ha guardado para
    ser utilizada más tarde.
    localIdToSession.put(new Integer(jta.getLocalId()), session);
    return true;
}

```

Recuperación de una transacción externa

A veces, es posible que tenga que recuperar un objeto de servicio de transacción externa para que lo utilice el plug-in `TransactionCallback`. En el servidor `WebSphere Application Server`, busque el objeto `ExtendedJTATransaction` en su espacio de nombres, tal como se muestra en el ejemplo siguiente:

```

public J2EETransactionCallback() {
    super();
    localIdToSession = new HashMap();
    String lookupName="java:comp/websphere/ExtendedJTATransaction";
    try
    {
        InitialContext ic = new InitialContext();

```

```

        jta = (ExtendedJTATransaction)ic.lookup(lookupName);
        jta.registerSynchronizationCallback(this);
    }
    catch(NotSupportedException e)
    {
        throw new RuntimeException("Cannot register jta callback", e);
    }
    catch(NamingException e){
        throw new RuntimeException("Cannot get transaction object");
    }
}

```

Para otros productos, puede utilizar un acercamiento similar para recuperar el objeto de servicio de transacción.

Control de la confirmación mediante la devolución de llamada externa

El plug-in TransactionCallback debe recibir una señal externa para confirmar o retrotraer la sesión de eXtreme Scale. Para recibir esta señal externa, utilice la devolución de llamada del servicio de transacción externa. Implemente la interfaz de devolución de llamada externa y regístrela con el servicio de transacción externa. Por ejemplo, con WebSphere Application Server, implemente la interfaz SynchronizationCallback, tal como se muestra en el ejemplo siguiente:

```

public class J2EETransactionCallback implements
com.ibm.websphere.objectgrid.plugins.TransactionCallback, SynchronizationCallback {
    public J2EETransactionCallback() {
        super();
        String lookupName="java:comp/websphere/ExtendedJTATransaction";
        LocalIdToSession = new HashMap();
        try {
            InitialContext ic = new InitialContext();
            jta = (ExtendedJTATransaction)ic.lookup(lookupName);
            jta.registerSynchronizationCallback(this);
        } catch(NotSupportedException e) {
            throw new RuntimeException("Cannot register jta callback", e);
        }
        catch(NamingException e) {
            throw new RuntimeException("Cannot get transaction object");
        }
    }

    public synchronized void afterCompletion(int localId, byte[] arg1,boolean didCommit) {
        Integer lid = new Integer(localId);
        // buscar localId de Session
        Session session = (Session)localIdToSession.get(lid);
        if(session != null) {
            try {
                // si WebSphere Application Server se confirma al
                // proteger la transacción para backingMap.
                // Se realizó un vaciado en beforeCompletion
                if(didCommit) {
                    session.commit();
                } else {
                    // de lo contrario, retrotraer
                    session.rollback();
                }
            } catch(NoActiveTransactionException e) {
                // imposible en teoría
            } catch(TransactionException e) {
                // como ya se ha hecho un vaciado, no debería producirse error
            } finally {
                // siempre borra la sesión de la correlación
                localIdToSession.remove(lid);
            }
        }
    }

    public synchronized void beforeCompletion(int localId, byte[] arg1) {
        Session session = (Session)localIdToSession.get(new Integer(localId));
        if(session != null) {
            try {
                session.flush();
            } catch(TransactionException e) {
                // WebSphere Application Server no define formalmente
                // una manera de indicar que la
                // transacción ha fallado, por lo que debe hacer esto
                throw new RuntimeException("Cache flush failed", e);
            }
        }
    }
}

```

```
} } }
```

Utilice las API de eXtreme Scale con el plug-in TransactionCallback

El plug-in TransactionCallback inhabilita la confirmación automática en eXtreme Scale. El patrón de uso normal para un eXtreme Scale es el siguiente:

```
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
ogSession.begin();
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
ogSession.commit();
```

Cuando se utiliza este plug-in TransactionCallback, eXtreme Scale presupone que la aplicación utiliza eXtreme Scale cuando está presente una transacción gestionada por contenedor. El fragmento de código anterior cambia el siguiente código en este entorno:

```
public void myMethod() {
    UserTransaction tx = ...;
    tx.begin();
    Session ogSession = ...;
    ObjectMap myMap = ogSession.getMap("MyMap");
    MyObject v = myMap.get("key");
    v.setAttribute("newValue");
    myMap.update("key", v);
    tx.commit();
}
```

El método myMethod es similar a un escenario de aplicación web. La aplicación utiliza la interfaz UserTransaction normal para empezar, confirmar y retrotraer transacciones. eXtreme Scale se inicia y se confirma automáticamente cerca de la transacción de contenedor. Si el método es un método EJB (Enterprise JavaBeans) que utiliza el atributo TX_REQUIRES, elimine la referencia de UserTransaction y las llamadas para iniciar y confirmar transacciones y el método funcionan del mismo modo. En este caso, el contenedor es responsable de iniciar y terminar la transacción.

Plug-in WebSphereTransactionCallback

Cuando utilice el plug-in WebSphereTransactionCallback, las aplicaciones empresariales que se ejecutan en un entorno de WebSphere Application Server puede gestionar las transacciones de ObjectGrid.

Cuando se utiliza una sesión ObjectGrid dentro de un método que está configurado para utilizar las transacciones gestionadas por contenedor, se inicia el contenedor empresarial, confirma o retrotrae automáticamente la transacción ObjectGrid. Si utiliza objetos UserTransaction de JTA (Java Transaction API), la transacción de ObjectGrid es gestionada automáticamente por el objeto UserTransaction.

Si desea una descripción detallada de la implementación de este plug-in, consulte "Gestores de transacciones externas" en la página 282.

Nota: ObjectGrid no admite transacciones XA de dos fases. Este plug-in no lista la transacción ObjectGrid con el gestor de transacciones. Por lo tanto, si ObjectGrid no puede realizar la confirmación, todos los recursos gestionados por la transacción XA no se retrotraen.

Habilitación del plug-in WebSphereTransactionCallback

Puede habilitar WebSphereTransactionCallback en la configuración de ObjectGrid con la configuración programática o la configuración de XML.

Enfoque de configuración de XML para conectarse al objeto WebSphereTransactionCallback

La siguiente configuración de XML crea el objeto WebSphereTransactionCallback y lo añade a un ObjectGrid. El siguiente texto debe aparecer en el archivo myGrid.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="myGrid">
      <bean id="TransactionCallback" className=
        "com.ibm.websphere.objectgrid.plugins.builtins.WebSphereTransactionCallback" />
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Conexión a través de programas del objeto WebSphereTransactionCallback

El siguiente fragmento de código crea el objeto WebSphereTransactionCallback y lo añade a un ObjectGrid:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
WebSphereTransactionCallback wsTxCallback= new WebSphereTransactionCallback ();
myGrid.setTransactionCallback(wsTxCallback);
```

Capítulo 6. Programación de las tareas administrativas

Además de interfaces de programación de aplicaciones (API) del sistema, WebSphere eXtreme Scale también incluye interfaces de programación de aplicaciones de administración que permiten que las aplicaciones supervisen y administren servidores y clientes.

API de servidor incorporado

WebSphere eXtreme Scale incluye interfaces de programación de aplicaciones (API) e interfaces de programación del sistema para incorporar servidores y clientes de eXtreme Scale a sus aplicaciones Java existentes. El tema siguiente describe las API de servidor incorporado disponibles.

Creación de instancias del servidor eXtreme Scale

Puede utilizar diversas propiedades para configurar la instancia del servidor eXtreme Scale, que puede recuperar del método `ServerFactory.getServerProperties`. El objeto `ServerProperties` es un singleton, de modo que cada llamada al método `getServerProperties` recupera la misma instancia.

Puede crear un servidor nuevo con el código siguiente.

```
Server server = ServerFactory.getInstance();
```

Todas las propiedades definidas antes de la primera invocación de `getInstance` se utilizan para inicializar el servidor.

Establecimiento de las propiedades de servidor

Puede establecer las propiedades de servidor hasta que se llame a `ServerFactory.getInstance` por primera vez. La primera llamada del método `getInstance` crea una instancia del servidor eXtreme Scale y lee todas las propiedades configuradas. Establecer las propiedades después de la creación no tiene ningún efecto. El ejemplo siguiente muestra cómo definir propiedades antes de crear una instancia de `Server`.

```
// Obtener las propiedades de servidor asociadas con este proceso.
ServerProperties serverProperties = ServerFactory.getServerProperties();

// Establecer el nombre del servidor para este proceso.
serverProperties.setServerName("EmbeddedServerA");

// Establecer el nombre de la zona donde está contenido este proceso.
serverProperties.setZoneName("EmbeddedZone1");

// Establecer la información de punto final necesaria para crear una rutina de
carga para el servicio de catálogo.
serverProperties.setCatalogServiceBootstrap("localhost:2809");

// Establecer el nombre de host de escucha ORB que se va a utilizar en los enlaces.
serverProperties.setListenerHost("host.local.domain");

// Establecer el puerto de escucha ORB que se va a utilizar en los enlaces.
serverProperties.setListenerPort(9010);

// Desactivar todos los MBeans de este proceso.
```

```
serverProperties.setMBeansEnabled(false);  
Server server = ServerFactory.getInstance();
```

Incorporación del servicio de catálogo

Cualquier valor de JVM señalado por el método `CatalogServerProperties.setCatalogServer` puede albergar el servicio de catálogo de eXtreme Scale. Este método indica al tiempo de ejecución del servidor eXtreme Scale que cree una instancia del servicio de catálogo cuando se inicie el servidor. El código siguiente muestra cómo crear la instancia del servidor de catálogo de eXtreme Scale:

```
CatalogServerProperties catalogServerProperties =  
    ServerFactory.getCatalogProperties();  
catalogServerProperties.setCatalogServer(true);  
  
Server server = ServerFactory.getInstance();
```

Incorporación del contenedor de eXtreme Scale

Emita el método `Server.createContainer` de cualquier JVM para albergar varios contenedores de eXtreme Scale. El código siguiente muestra cómo crear la instancia de un contenedor de eXtreme Scale:

```
Server server = ServerFactory.getInstance();  
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(  
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),  
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL());  
Container container = server.createContainer(policy);
```

Proceso de servidor autónomo

Puede iniciar todos los servicios de forma conjunta, que es práctico para el desarrollo y, también, práctico para la producción. Al iniciar los servicios de forma conjunta, un único proceso realiza todo lo siguiente: inicia el servicio de catálogo, inicia un conjunto de contenedores y ejecutar la lógica de conexión de cliente. Iniciar los servidores de esta forma clasifica los problemas de programación antes del despliegue en un entorno distribuido. El código siguiente muestra como crear la instancia de un servidor eXtreme Scale autónomo:

```
CatalogServerProperties catalogServerProperties =  
    ServerFactory.getCatalogProperties();  
catalogServerProperties.setCatalogServer(true);  
  
Server server = ServerFactory.getInstance();  
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(  
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),  
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL());  
Container container = server.createContainer(policy);
```

Incorporación de eXtreme Scale in WebSphere Application Server

La configuración de eXtreme Scale se realiza automáticamente al instalar WebSphere Extended Deployment DataGrid en un entorno de WebSphere Application Server. No es necesario establecer ninguna propiedad antes de acceder al servidor para crear un contenedor. El código siguiente muestra cómo crear instancias de un servidor eXtreme Scale en WebSphere Application Server:

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL());
Container container = server.createContainer(policy);
```

Para ver un ejemplo paso a paso de cómo iniciar un servicio de catálogo y contenedor incorporado mediante programa, consulte el apartado “Uso de la API de servidor incorporado”.

Uso de la API de servidor incorporado

Con WebSphere eXtreme Scale, puede utilizar una API programática para gestionar el ciclo de vida de servidores y contenedores incorporados. Puede configurar a través de programas el servidor con cualquiera de las opciones que también puede configurar con las opciones de la línea de mandatos o las propiedades de servidor basadas en archivo. Puede configurar el servidor incorporado para que sea un servidor de contenedor, un servicio de catálogo, o ambos.

Antes de empezar

Debe tener un método para ejecutar el código desde una Máquina virtual Java ya existente. Las clases de eXtreme Scale deben estar disponibles a través del árbol del cargador de clases.

Acerca de esta tarea

Puede ejecutar muchas tareas de administración con la API Administration. Un uso común de la API es su uso como servidor interno para almacenar el estado de la aplicación web. El servidor web puede iniciar un servidor WebSphere eXtreme Scale incorporado, realizar informes del servidor de contenedor en el servicio de catálogo y después añadirlo como un miembro de una cuadrícula distribuida mayor. Este uso puede proporcionar escalabilidad y alta disponibilidad en un almacén de datos que de lo contrario es volátil.

Puede controlar mediante programa el ciclo de vida completo de un servidor eXtreme Scale incorporado. Los ejemplos son lo más genéricos posibles y sólo muestran códigos de ejemplo de código directo para los pasos descritos.

Procedimiento

1. Obtenga el objeto `ServerProperties` desde la clase `ServerFactory` y configure las opciones necesarias.

Cada servidor eXtreme Scale tiene un conjunto de propiedades configurables. Cuando un servidor se inicia desde la línea de mandatos, estas propiedades toman los valores predeterminado, pero puede alterar temporalmente varias propiedades proporcionando un origen o archivo externo. En el ámbito incorporado, puede establecer directamente las propiedades con un objeto `ServerProperties`. Debe establecer estas propiedades antes de obtener una instancia de servidor desde la clase `ServerFactory`. El siguiente fragmento de código obtiene un objeto `ServerProperties`, establece el campo `CatalogServiceBootstrap` e inicializa varios valores de servidor opcionales. Consulte la documentación de la API para ver una lista de los valores configurables.

```

ServerProperties props = ServerFactory.getServerProperties();
props.setCatalogServiceBootstrap("host:port"); // necesario para conectarse a un servicio
// de catálogo específico
props.setServerName("ServerOne"); // name server
props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // Establece la espec. de rastreo

```

2. Si desea que el servidor sera un servicio de catálogo, obtenga el objeto `CatalogServerProperties`.

Todos los servidores incorporados pueden ser un servicio de catálogo, un servidor de contenedor, o ambos, un servidor de contenedor y un servicio de catálogo. El siguiente ejemplo obtiene el objeto `CatalogServerProperties`, habilita la opción del servicio de catálogo y configura distintos valores de servicio de catálogo.

```

CatalogServerProperties catalogProps = ServerFactory.getCatalogProperties();
catalogProps.setCatalogServer(true); // false de forma predeterminada, es necesario establecerlo
// como un servicio de catálogo
catalogProps.setQuorum(true); // habilitar / inhabilitar el quórum

```

3. Obtenga una instancia de `Server` desde la clase `ServerFactory`. La instancia de `Server` es un singleton con un ámbito de proceso que es responsable de gestionar la pertenencia de la cuadrícula. Después de que se haya creado una instancia, este proceso se conecta y está muy disponible con los otros servidores de la cuadrícula. El siguiente ejemplo muestra cómo crear la instancia de `Server`:

```

Server server = ServerFactory.getInstance();

```

Mediante la revisión del ejemplo anterior, la clase `ServerFactory` proporciona un método estático que devuelve una instancia de `Server`. La clase `ServerFactory` tiene como objetivo ser la única interfaz para obtener una instancia de `Server`. Por lo tanto, la clase garantiza que la instancia es un singleton, o una instancia para cada JVM o cargador de clases aislado. El método `getInstance` inicializa la instancia de `Server`. Debe configurar todas las propiedades de servidor antes de inicializar la instancia. La clase `Server` es responsable de crear las nuevas instancias de `Container`. Puede utilizar ambas clases, `ServerFactory` y `Server`, para gestionar el ciclo de vida de la instancia de `Server` incorporada.

4. Inicie una instancia de `Container` utilizando la instancia de `Server`.

Antes de que los fragmentos se puedan colocar en un servidor incorporado, debe crear un contenedor en el servidor. La interfaz `Server` tiene un método `createContainer` que adopta un argumento `DeploymentPolicy`. El siguiente ejemplo utiliza la instancia del servidor que ha obtenido para crear un contenedor utilizando un archivo `DeploymentPolicy` creado. Tenga en cuenta que los contenedores requieren un cargador de clases que tiene los binarios de aplicaciones disponibles para la serialización. Puede hacer que estos binarios estén disponibles llamando al método `createContainer` con el cargador de clases del contexto `Thread` establecido en el cargador de clases que desee utilizar.

```

Política de despliegue = DeploymentPolicyFactory.createDeploymentPolicy(new
URL("file://urltodeployment.xml"),
new URL("file://urltoobjectgrid.xml"));
Container container = server.createContainer(policy);

```

5. Eliminar y borrar un contenedor.

Puede eliminar y borrar un servidor de contenedor utilizando el método `teardown` en ejecución en la instancia de `Container` obtenida. Ejecutar el método `teardown` en un contenedor correctamente borra el contenedor y elimina el contenedor del servidor incorporado.

El proceso de limpieza del contenedor incluye el movimiento y la destrucción de todos los fragmentos que se han colocado dentro de dicho contenedor. Cada servidor puede contener muchos contenedores y fragmentos. La limpieza de un contenedor no afecta al ciclo de vida de la instancia padre de `Server`. El siguiente ejemplo demuestra cómo ejecutar el método `teardown` en un servidor.

El método `teardown` se ha hecho disponible a través de la interfaz `ContainerMBean`. Mediante el uso de la interfaz `ContainerMBean`, si deja de tener acceso mediante programa a este contenedor, puede seguir eliminando y limpiando el contenedor con su `MBean`. También existe un método `terminate` en la interfaz `Container`, no utilice este método, a menos que sea absolutamente necesario. Este método es más potente y no coordina el movimiento y la limpieza de fragmentos apropiados.

```
container.teardown();
```

6. Detenga el servidor incorporado.

Cuando detenga un servidor incorporado, también puede detener los contenedores y los fragmentos que se ejecutan en el servidor. Cuando detenga un servidor incorporado, debe limpiar todas las conexiones abiertas o mover o destruir todos los fragmentos. El siguiente ejemplo demuestra cómo detener un servidor y cómo utilizar el método `waitFor` en la interfaz `Server` para asegurarse de que la instancia de `Server` se termina por completo. De forma similar al ejemplo del contenedor, el método `stopServer` pasa a estar disponible a través de la interfaz `ServerMBean`. Con esta interfaz, puede detener un servidor con el bean gestionado (`MBean`) correspondiente.

```
ServerFactory.stopServer(); // Utiliza la fábrica para matar el proceso singleton de Server
// o
server.stopServer(); // Utiliza directamente la instancia de Server
server.waitFor(); // Se devuelve este valor cuando el servidor ha completado correctamente
sus procedimientos de conclusión.
```

Ejemplo de código completo:

```
import java.net.MalformedURLException;
import java.net.URL;

import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.deployment.DeploymentPolicy;
import com.ibm.websphere.objectgrid.deployment.DeploymentPolicyFactory;
import com.ibm.websphere.objectgrid.server.Container;
import com.ibm.websphere.objectgrid.server.Server;
import com.ibm.websphere.objectgrid.server.ServerFactory;
import com.ibm.websphere.objectgrid.server.ServerProperties;

public class ServerFactoryTest {

    public static void main(String[] args) {

        try {

            ServerProperties props = ServerFactory.getServerProperties();
            props.setCatalogServiceBootstrap("catalogservice-hostname:catalogservice-port");
            props.setServerName("ServerOne"); // name server
            props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // TraceSpec

            /*
             * En la mayoría de los casos, el servidor actuará sólo como un servidor de contenedor
             * y se conectará a un servicio de catálogo externo. Este es un método más disponible
             * de realizar acciones. La siguiente excepción de código comentada
             * permitirá que este Server sea un servicio de catálogo.
             */
            /*
             * CatalogServerProperties catalogProps =
             * ServerFactory.getCatalogProperties();
             * catalogProps.setCatalogServer(true); // habilitar el servicio de catálogo
             * catalogProps.setQuorum(true); // habilitar quórum
             */

            Server server = ServerFactory.getInstance();

            DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy
            (new URL("url to deployment xml"), new URL("url to objectgrid xml file"));
            Container container = server.createContainer(policy);

            /*
             * Ahora el fragmento se colocará en este contenedor, si se cumplen los
             * requisitos de despliegue.
             * Esto engloba la creación del contenedor y del servidor incorporados.
             */
            /*
             * Las líneas siguientes simplemente demostrarán la llamada a métodos de limpieza
            */
        }
    }
}
```

```

        */
        container.teardown();
        server.stopServer();
        int success = server.waitFor();
    } catch (ObjectGridException e) {
        // El contenedor no se ha podido inicializar
    } catch (MalformedURLException e2) {
        // url no válido para los archivos xml
    }
}
}
}

```

Supervisión con la API de estadísticas

La API de estadísticas es la interfaz directa al árbol de estadísticas internas. De forma predeterminada las estadísticas están inhabilitadas, aunque pueden habilitarse estableciendo una interfaz StatsSpec. Una interfaz StatsSpec define cómo WebSphere eXtreme Scale debe supervisar estadísticas.

Acerca de esta tarea

Puede utilizar la API StatsAccessor local para consultar los datos y acceder a las estadísticas sobre cualquier instancia de ObjectGrid que está en la misma Máquina virtual Java (JVM) que el código de ejecución. Si desea más información sobre las interfaces específicas, consulta la documentación de la API. Utilice los pasos siguientes para habilitar la supervisión del árbol de estadísticas internas.

Procedimiento

1. Recupere el objeto StatsAccessor. La interfaz StatsAccessor sigue el patrón singleton. Por lo tanto, aparte de los problemas relacionados con el cargador de clases, debe existir una instancia de StatsAccessor para cada JVM . Esta clase hace las veces de interfaz principal para todas las operaciones de estadísticas locales. El siguiente código es un ejemplo sobre cómo recuperar la clase del descriptor de acceso. Llame a esta operación antes de cualquier otra llamada de ObjectGrid.

```

public class LocalClient {
    public static void main(String[] args) {

        // Recuperar un descriptor de contexto para StatsAccessor
        StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

    }
}

```

2. Establezca la interfaz StatsSpec de cuadrícula. Establezca esta JVM para recopilar todas las estadísticas sólo en el nivel de ObjectGrid. Debe asegurarse de que una aplicación habilite todas las estadísticas que puedan ser necesarias antes de empezar las transacciones. En el siguiente ejemplo se establece la interfaz StatsSpec utilizando un campo contante estático y una serie de especificación. El uso de un campo constante estático es más fácil porque el campo ya ha definido la especificación. No obstante, si utiliza una serie de especificación, podrá habilitar cualquier combinación de estadísticas que sea necesaria.

```

public static void main(String[] args) {

    // Recuperar un descriptor de contexto para StatsAccessor
    StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();
}

```

```

// Establecer la especificación a través del campo estático
StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
accessor.setStatsSpec(spec);

// Establecer la especificación a través de la serie de especificación
StatsSpec spec = new StatsSpec("og.all=enabled");
accessor.setStatsSpec(spec);
}

```

- Envíe las transacciones a la cuadrícula para obligar a que se recopilen los datos para la supervisión. Para recopilar datos prácticos para las estadísticas, debe enviar las transacciones a la cuadrícula. El siguiente extracto de código inserta un registro en MapA, que es un ObjectGridA. Dado que las estadísticas están en un nivel de ObjectGrid, cualquier correlación dentro de ObjectGrid genera los mismos resultados.

```

public static void main(String[] args) {

    // Recuperar un descriptor de contexto para StatsAccessor
    StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

    // Establecer la especificación a través del campo estático
    StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
    accessor.setStatsSpec(spec);

    ObjectGridManager manager =
    ObjectGridmanagerFactory.getObjectGridManager();
    ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
    Session session = grid.getSession();
    Map map = session.getMap("MapA");

    // Insertar unidad
    session.begin();
    map.insert("SomeKey", "SomeValue");
    session.commit();
}

```

- Consulte una StatsFact utilizando la API StatsAccessor. Cada vía de acceso de estadísticas está asociada a una interfaz StatsFact. La interfaz StatsFact es un marcador genérico que se utiliza para organizar y contener un objeto StatsModule. Para poder acceder al módulo de estadísticas real, se debe recuperar el objeto StatsFact.

```

public static void main(String[] args) {
// Recuperar un descriptor de contexto para StatsAccessor
    StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

    // Establecer la especificación a través del campo estático
    StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
    accessor.setStatsSpec(spec);

    ObjectGridManager manager =
    ObjectGridManagerFactory.getObjectGridManager();
    ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
    Session session = grid.getSession();
    Map map = session.getMap("MapA");

    // Insertar unidad
    session.begin();
    map.insert("SomeKey", "SomeValue");
    session.commit();

    // Recuperar StatsFact
}

```

```

        StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
StatsModule.MODULE_TYPE_OBJECT_GRID);
    }
}
5. Interactúe con el objeto StatsModule. El objeto StatsModule está incluido dentro
de la interfaz StatsFact. Puede obtener una referencia al módulo utilizando la
interfaz StatsFact. Como la interfaz StatsFact es una interfaz genérica, debe
convertir el módulo devuelto en el tipo de StatsModule esperado. Puesto que
esta tarea recopila estadísticas de eXtreme Scale, el objeto StatsModule devuelto
se convierte en el tipo OGStatsModule. Una vez que se ha convertido el
módulo, tendrá acceso a todas las estadísticas disponibles.
public static void main(String[] args) {

    // Recuperar un descriptor de contexto para StatsAccessor
    StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

    // Establecer la especificación a través del campo estático
    StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
    accessor.setStatsSpec(spec);

    ObjectGridManager manager =
ObjectGridmanagerFactory.getObjectGridManager();
    ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
    Session session = grid.getSession();
    Map map = session.getMap("MapA");

    // Insertar unidad
    session.begin();
    map.insert("SomeKey", "SomeValue");
    session.commit();

    // Recuperar StatsFact
    StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
StatsModule.MODULE_TYPE_OBJECT_GRID);

    // Recuperar módulo y hora
    OGStatsModule module = (OGStatsModule)fact.getStatsModule();
    ActiveTimeStatistic timeStat =
module.getTransactionTime("Default", true);
    double time = timeStat.getMeanTime();
}
}

```

Supervisión con PMI de WebSphere Application Server

WebSphere eXtreme Scale da soporte a PMI (Performance Monitoring Infrastructure) cuando se ejecuta en un servidor de aplicaciones WebSphere Application Server o WebSphere Extended Deployment. PMI recopila los datos de rendimiento de aplicaciones en tiempo de ejecución y proporciona interfaces que dan soporte a aplicaciones externas para supervisar datos de rendimiento. Puede utilizar la consola administrativa o la herramienta wsadmin para acceder a los datos de supervisión.

Antes de empezar

Puede utilizar PMI para supervisar el entorno cuando utiliza WebSphere eXtreme Scale junto con WebSphere Application Server.

Acerca de esta tarea

WebSphere eXtreme Scale utiliza la característica PMI personalizada de WebSphere Application Server para añadir su propia instrumentación PMI. Con este enfoque, puede habilitar e inhabilitar WebSphere eXtreme Scale PMI con la consola administrativa o con las interfaces JMX (Java Management Extensions) de la herramienta wsadmin. Además, puede acceder a las estadísticas de WebSphere eXtreme Scale con las interfaces PMI y JMX estándares que utilizan las herramientas de supervisión, incluido Tivoli Performance Viewer.

Procedimiento

1. Habilite eXtreme Scale PMI. Debe habilitar PMI para ver las estadísticas de PMI. Si desea más información, consulte “Habilitación de PMI”.
2. Recupere las estadísticas de eXtreme Scale PMI. Vea el rendimiento de las aplicaciones de eXtreme Scale con Tivoli Performance Viewer. Si desea más información, consulte “Recuperar estadísticas de PMI” en la página 297.

Qué hacer a continuación

Para obtener más información sobre la herramienta wsadmin, consulte “Acceso a MBeans mediante la herramienta wsadmin” en la página 306.

Habilitación de PMI

Puede utilizar PMI (Performance Monitoring Infrastructure) de WebSphere Application Server para habilitar o inhabilitar estadísticas a cualquier nivel. Por ejemplo, puede elegir que se habiliten las estadísticas de proporción de coincidencias de correlación para una correlación determinada pero no así las estadísticas de número de entradas ni las estadísticas de tiempo de actualización por lotes del cargador. Puede habilitar PMI en la consola de administración o con scripts.

Antes de empezar

El servidor de aplicaciones se debe haber iniciado y debe tener instalada una aplicación habilitada para eXtreme Scale. Para habilitar PMI con el uso de scripts, también debe poder iniciar la sesión y utilizar la herramienta wsadmin. Para obtener más información sobre la herramienta wsadmin, consulte el tema Herramienta wsadmin en el Information Center de WebSphere Application Server.

Acerca de esta tarea

Utilice WebSphere Application Server PMI para proporcionar un mecanismo granular con el que poder habilitar o inhabilitar estadísticas a cualquier nivel. Por ejemplo, puede elegir que se habiliten las estadísticas de proporción de coincidencias de correlación para una correlación determinada pero no así las estadísticas de número de entradas ni las estadísticas de tiempo de actualización por lotes del cargador. Esta sección muestra cómo utilizar la consola administrativa y los scripts wsadmin para habilitar PMI de ObjectGrid.

Procedimiento

- **Habilite PMI en la consola administrativa.**
 1. En la consola administrativa, pulse **Supervisión y ajuste** → **Performance Monitoring Infrastructure** → *nombre_servidor*.

2. Verifique que Performance Monitoring Infrastructure (PMI) se ha seleccionado. De forma predeterminada este valor está habilitado. Si el valor no está habilitado, seleccione el recuadro de selección y reinicie el servidor.
3. Pulse **Personalizado**. En el árbol de configuración, seleccione el módulo de correlaciones ObjectGrid y ObjectGrid. Habilite las estadísticas para cada módulo.

La categoría de tipo de transacción para estadísticas de ObjectGrid se crea en el tiempo de ejecución. Sólo puede ver las subcategorías de las estadísticas de ObjectGrid y de correlación en el separador **Tiempo de ejecución**.

- **Habilite PMI con el uso de scripts.**

1. Abra un indicador de línea de mandatos. Vaya al directorio raíz_instalación/bin. Escriba wsadmin para iniciar la herramienta de línea de mandatos wsadmin.
2. Modifique la configuración del tiempo de ejecución de PMI de eXtreme Scale. Verifique que PMI se ha habilitado para el servidor mediante los siguientes mandatos:

```
wsadmin>set s1 [$AdminConfig getid /Cell:CELL_NAME/Node:NODE_NAME/Server:
APPLICATION_SERVER_NAME/]
wsadmin>set pmi [$AdminConfig list PMIService $s1]
wsadmin>$AdminConfig show $pmi.
```

Si PMI no se ha habilitado, ejecute los mandatos siguientes para habilitar PMI:

```
wsadmin>$AdminConfig modify $pmi {{enable true}}
wsadmin>$AdminConfig save
```

Si necesita habilitar PMI, reinicie el servidor.

3. Establezca las variables para cambiar el conjunto de estadísticas por un conjunto personalizado utilizando los siguientes mandatos:


```
wsadmin>set perfName [$AdminControl completeObjectName type=Perf,
process=APPLICATION_SERVER_NAME,*]
wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
wsadmin>set params [java::new {java.lang.Object[]} 1]
wsadmin>$params set 0 [java::new java.lang.String custom]
wsadmin>set sigs [java::new {java.lang.String[]} 1]
wsadmin>$sigs set 0 java.lang.String
```
4. Establezca el conjunto de estadísticas que desea personalizar utilizando el siguiente mandato:


```
wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
```
5. Establezca las variables para habilitar las estadísticas de PMI de objectGridModule utilizando los siguientes mandatos:


```
wsadmin>set params [java::new {java.lang.Object[]} 2]
wsadmin>$params set 0 [java::new java.lang.String objectGridModule=1]
wsadmin>$params set 1 [java::new java.lang.Boolean false]
wsadmin>set sigs [java::new {java.lang.String[]} 2]
wsadmin>$sigs set 0 java.lang.String
wsadmin>$sigs set 1 java.lang.Boolean
```
6. Establezca la serie de estadísticas utilizando el siguiente mandato:


```
wsadmin>set params2 [java::new {java.lang.Object[]} 2]
wsadmin>$params2 set 0 [java::new java.lang.String mapModule=*]
wsadmin>$params2 set 1 [java::new java.lang.Boolean false]
wsadmin>set sigs2 [java::new {java.lang.String[]} 2]
wsadmin>$sigs2 set 0 java.lang.String
wsadmin>$sigs2 set 1 java.lang.Boolean
```
7. Establezca la serie de estadísticas utilizando el siguiente mandato:

```
wsadmin>$AdminControl invoke_jmx $perf0Name setCustomSetString $params2 $sigs2
```

Estos pasos habilitan el PMI de tiempo de ejecución de eXtreme Scale, pero no modifican la configuración de PMI. Si reinicia el servidor de aplicaciones, los valores de PMI se pierden excepto para la habilitación de PMI principal.

Ejemplo

Puede efectuar los siguientes pasos para habilitar las estadísticas de PMI para la aplicación de ejemplo:

1. Inicie la aplicación utilizando la dirección web `http://host:puerto/ObjectGridSample`, donde el host y el puerto son el nombre del host y el número de puerto HTTP del servidor en el que se ha instalado el ejemplo.
2. En la aplicación de ejemplo, pulse `ObjectGridCreationServlet`, y luego pulse los botones de acción 1, 2, 3, 4 y 5 para generar acciones para `ObjectGrid` y correlaciones. No cierre esta página de servlet ahora.
3. En la consola administrativa, pulse **Supervisión y ajuste** → **Performance Monitoring Infrastructure** → *nombre_servidor* Pulse la pestaña **Tiempo de ejecución**.
4. Pulse el botón de selección **Personalizado**.
5. Expanda el módulo de correlaciones de `ObjectGrid` en el árbol de tiempo de ejecución y pulse el enlace `clusterObjectGrid`. Bajo el grupo de correlaciones de `ObjectGrid`, hay una instancia de `ObjectGrid` llamada `clusterObjectGrid`, y bajo el grupo `clusterObjectGrid` existen cuatro correlaciones: contadores, empleados, oficinas y sitios. En la instancia de `ObjectGrids`, existe la instancia de `clusterObjectGrid` y bajo dicha instancia hay un tipo de transacción llamado `DEFAULT`.
6. Puede habilitar las estadísticas que desee. Por ejemplo, puede habilitar una cantidad de entradas de correlación para la correlación de empleados y un tiempo de respuesta de transacción para el tipo de transacción `DEFAULT`.

Qué hacer a continuación

Una vez que se ha habilitado PMI, puede ver las estadísticas PMI con la consola administrativa o con el uso de scripts.

Recuperar estadísticas de PMI

Al recuperar estadísticas de PMI, podrá ver el rendimiento de las aplicaciones eXtreme Scale.

Antes de empezar

- Habilite el rastreo de estadísticas de PMI para el entorno. Si desea más información, consulte “Habilitación de PMI” en la página 295.
- En las vías de acceso de esta tarea se da por supuesto que se recuperan estadísticas de la aplicación de ejemplo, aunque puede utilizar estas estadísticas para cualquier otra aplicación con pasos parecidos.
- Si utiliza la consola administrativa para recuperar estadísticas, debe poder iniciar la sesión en la consola administrativa. Si utiliza scripts, debe poder iniciar la sesión en `wsadmin`.

Acerca de esta tarea

Puede recuperar estadísticas de PMI y verlas en Tivoli Performance Viewer efectuando los pasos en la consola administrativa o con scripts.

- Pasos en la consola administrativa
- Pasos en los scripts

Para obtener más información sobre las estadísticas que pueden recuperarse, consulte “Módulos PMI” en la página 299.

Procedimiento

- Recupere estadísticas de PMI en la consola administrativa.
 1. En la consola administrativa, pulse **Supervisión y ajuste** → **Performance Viewer** → **Actividad actual**
 2. Seleccione el servidor que desee supervisar utilizando Tivoli Performance Viewer y luego habilite la supervisión.
 3. Pulse el servidor para ver la página de Performance Viewer.
 4. Expanda el árbol de configuración. Pulse **Correlaciones de ObjectGrid** → **clusterObjectGrid** seleccione **employees**. Expanda **ObjectGrids** → **clusterObjectGrid** y seleccione **DEFAULT**.
 5. En la aplicación de ejemplo de ObjectGrid, vaya al servlet ObjectGridCreationServlet, pulse el botón 1 y luego rellene las correlaciones. Puede ver las estadísticas en el visor.
- Recupere estadísticas de PMI con scripts.
 1. En el indicador de línea de mandatos, vaya al directorio raíz_instalación/bin. Escriba wsadmin para iniciar la herramienta wsadmin.
 2. Establezca las variables para el entorno utilizando los siguientes mandatos:


```
wsadmin>set perfName [$AdminControl completeObjectName type=Perf,*]
wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
wsadmin>set mySrvName [$AdminControl completeObjectName type=Server,
name=APPLICATION_SERVER_NAME,*]
```
 3. Establezca las variables para obtener estadísticas de mapModule utilizando los siguientes mandatos:


```
wsadmin>set params [java::new {java.lang.Object[]} 3]
wsadmin>$params set 0 [$AdminControl makeObjectName $mySrvName]
wsadmin>$params set 1 [java::new java.lang.String mapModule]
wsadmin>$params set 2 [java::new java.lang.Boolean true]
wsadmin>set sigs [java::new {java.lang.String[]} 3]
wsadmin>$sigs set 0 javax.management.ObjectName
wsadmin>$sigs set 1 java.lang.String
wsadmin>$sigs set 2 java.lang.Boolean
```
 4. Obtenga estadísticas de mapModule utilizando el siguiente mandato:


```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params $sigs
```
 5. Establezca las variables para obtener estadísticas de objectGridModule utilizando los siguientes mandatos:


```
wsadmin>set params2 [java::new {java.lang.Object[]} 3]
wsadmin>$params2 set 0 [$AdminControl makeObjectName $mySrvName]
wsadmin>$params2 set 1 [java::new java.lang.String objectGridModule]
wsadmin>$params2 set 2 [java::new java.lang.Boolean true]
wsadmin>set sigs2 [java::new {java.lang.String[]} 3]
wsadmin>$sigs2 set 0 javax.management.ObjectName
wsadmin>$sigs2 set 1 java.lang.String
wsadmin>$sigs2 set 2 java.lang.Boolean
```
 6. Obtenga las estadísticas de objectGridModule utilizando el siguiente mandato:


```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params2 $sigs2
```

Resultados

Puede ver las estadísticas en Tivoli Performance Viewer.

Módulos PMI

Puede supervisar el rendimiento de las aplicaciones con los módulos PMI (Performance Monitoring Infrastructure).

objectGridModule

objectGridModule contiene una estadística de tiempo: el tiempo de respuesta de la transacción. Una transacción se define como la duración entre la llamada del método Session.begin y la llamada del método Session.commit. Este intervalo de tiempo se considera el tiempo de respuesta de la transacción. El elemento raíz de objectGridModule, "root", hace las veces de punto de entrada de las estadísticas de WebSphere eXtreme Scale. Este elemento raíz tiene ObjectGrids como sus elementos hijo, que tienen tipos de transacción como elementos hijo. La estadística de tiempo de respuesta está asociado a cada tipo de transacción.

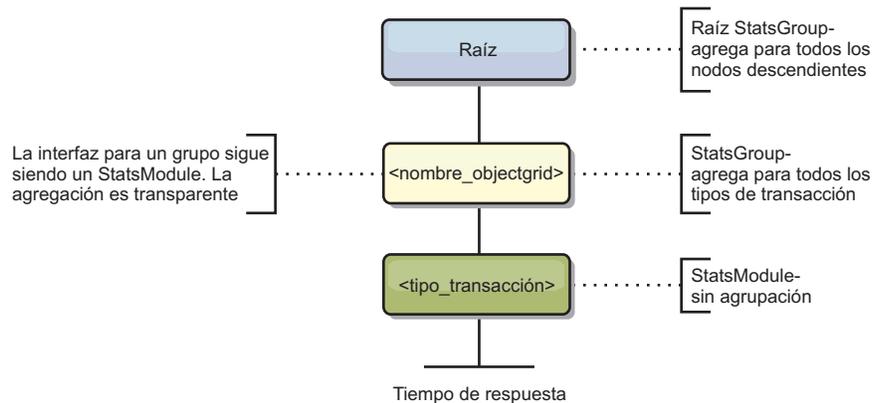


Figura 4. Estructura del módulo ObjectGridModule

El siguiente diagrama muestra un ejemplo de la estructura de ObjectGridModule. En este ejemplo, existen dos instancias de ObjectGrid en el sistema: el ObjectGrid A y el ObjectGrid B. La instancia A de ObjectGrid tiene dos tipos de transacciones: la A y la predeterminada. La instancia ObjectGrid B tiene sólo el tipo de transacción predeterminado.

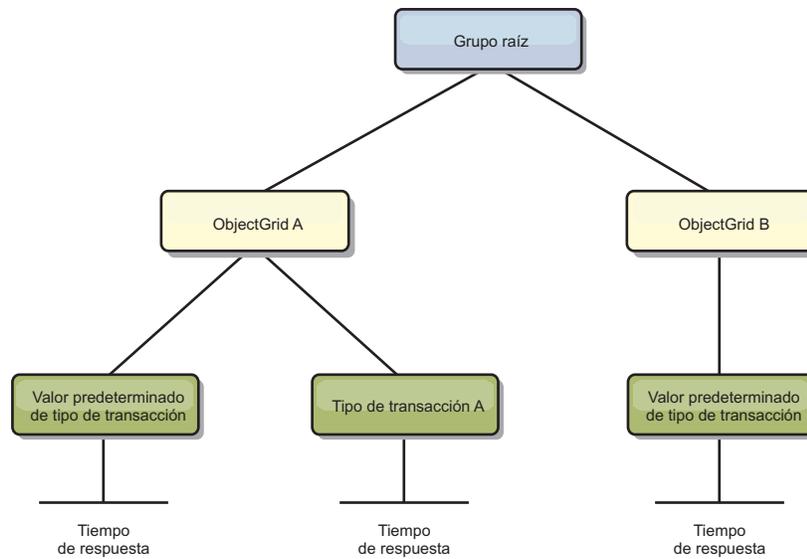


Figura 5. Ejemplo de estructura del módulo ObjectGridModule

Los tipos de transacción los definen los desarrolladores de transacciones porque conocen los tipos de transacciones que utilizan sus aplicaciones. El tipo de transacción se establece utilizando el siguiente método `Session.setTransactionType(String)`:

```

/**
 * Establece el tipo de transacción para futuras transacciones.
 *
 * Después de llamar a este método, todas las transacciones futuras tendrán el mismo
 * tipo hasta que se establezca otro tipo de transacción. Si no se establece ningún
 * tipo de transacción, se utiliza el tipo de transacción TRANSACTION_TYPE_DEFAULT
 * predeterminado.
 *
 * Los tipos de transacción se usan principalmente para fines de seguimiento de datos
 * estadísticos.
 * Los usuarios pueden definir previamente los tipos de transacciones que se
 * ejecutan en una aplicación.
La idea es clasificar las transacciones con las mismas características en una
 * categoría (tipo), de modo que una estadística de tiempo de respuesta se pueda
 * utilizar para realizar un seguimiento de cada tipo de transacción.
 *
 * Este seguimiento resulta útil cuando la aplicación tiene tipos diferentes de
 * transacciones.
 * Entre ellos, algunos tipos de transacciones, como las transacciones de
 * actualización, tardan más en procesarse que otras transacciones como, por
 * ejemplo, las de sólo lectura. Utilizando el tipo de transacción, se puede
 * realizar un seguimiento de las transacciones diferentes por estadísticas
 * diferentes, por lo que las estadísticas resultan más útiles.
 *
 * @param tranType el tipo de transacción para las transacciones futuras.
 */
void setTransactionType(String tranType);

```

El ejemplo siguiente establece el tipo de transacción en `updatePrice`:

```

// Establecer el tipo de transacción en updatePrice
// El período de tiempo entre session.begin() y session.commit() se reflejará
// en la estadística de tiempo "updatePrice".
session.setTransactionType("updatePrice");
session.begin();
map.update(stockId, new Integer(100));
session.commit();

```

La primera línea indica que el tipo de transacción subsiguiente es updatePrice. Existe una estadística updatePrice en la instancia ObjectGrid que se corresponde a la sesión del ejemplo. Utilizando las interfaces JMX (Java Management Extensions), puede obtener el tiempo de respuesta de la transacción para las transacciones updatePrice. También puede obtener la estadística agregada para todos los tipos de transacciones en la instancia ObjectGrid especificada.

mapModule

El mapModule contiene tres estadísticas que están relacionadas con las correlaciones de eXtreme Scale:

- **Proporción de coincidencias de la correlación** - *BoundedRangeStatistic*: efectúa un seguimiento de la proporción de coincidencias de una correlación. La proporción de coincidencias es un valor flotante entre 0 y 100 inclusive, que representa el porcentaje de coincidencias de una correlación en relación con las operaciones get de la correlación.
- **Número de entradas** - *CountStatistic*: efectúa un seguimiento del número de entradas de la correlación.
- **Tiempo de respuesta de la actualización por lotes del cargador** - *TimeStatistic*: efectúa un seguimiento del tiempo de respuesta que se utiliza para la operación de actualización por lotes del cargador.

El elemento raíz de mapModule, "root", hace las veces de punto de entrada para las estadísticas de la correlación de ObjectGrid. Este elemento raíz tiene ObjectGrids como elementos hijo, que tienen correlaciones como sus elementos hijo. Cada instancia de correlación tiene tres estadísticas listadas. En el diagrama siguiente se muestra la estructura de mapModule:

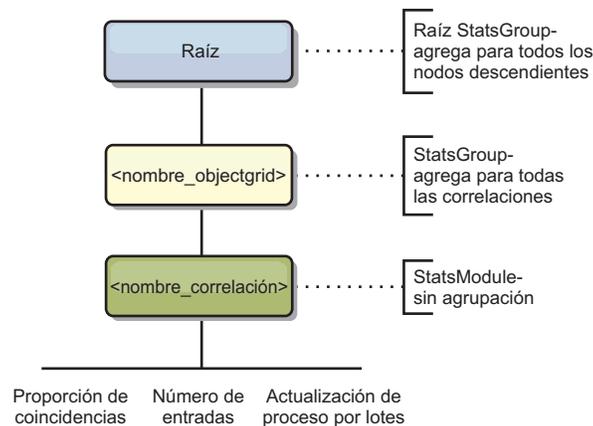
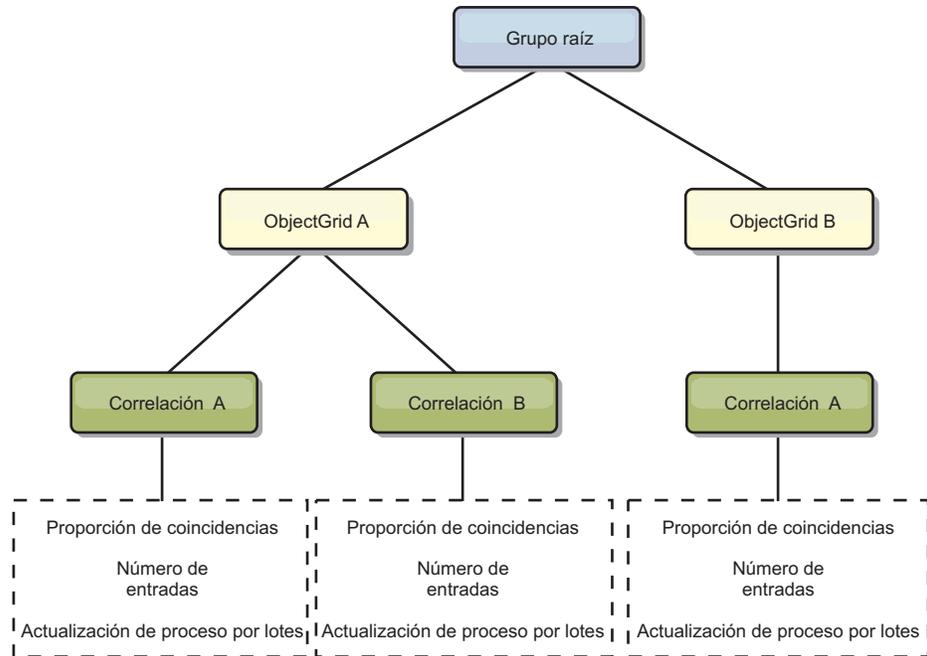


Figura 6. Estructura de mapModule

El siguiente diagrama muestra un ejemplo de la estructura de mapModule:

Figura 7. Ejemplo de la estructura del módulo mapModule



hashIndexModule

hashIndexModule contiene las siguientes estadísticas relacionadas con los índices de nivel de correlación:

- **Recuento de búsquedas** - *CountStatistic*: el número de invocaciones para la operación de búsqueda de índices.
- **Recuento de colisiones** - *CountStatistic*: el número de colisiones para la operación de búsqueda.
- **Recuento de anomalías** - *CountStatistic*: el número de anomalías para una operación de búsqueda.
- **Recuento de resultados** - *CountStatistic*: el número de claves devueltas de la operación de búsqueda.
- **Recuento de actualizaciones de proceso por lotes** - *CountStatistic*: el número de actualizaciones de proceso por lotes realizadas en relación con este índice. Cuando se cambia la correlación correspondiente en algún modo, el índice llamará a su método doBatchUpdate(). Esta estadística le indicará con que frecuencia se cambia o actualiza el índice.
- **Periodo de tiempo de la operación de búsqueda**-*TimeStatistic*: el intervalo de tiempo que la operación de búsqueda tarda en llevarse a cabo

El elemento raíz de hashIndexModule, "root", hace las veces de punto de entrada de las estadísticas de HashIndex. Este elemento raíz tiene ObjectGrids como elementos hijo, ObjectGrids tienen correlaciones como elementos hijo, que finalmente tienen HashIndexes como elementos hijo y nodos finales del árbol. Cada instancia de HashIndex tiene tres estadísticas listadas. En el diagrama siguiente se muestra la estructura de hashIndexModule:

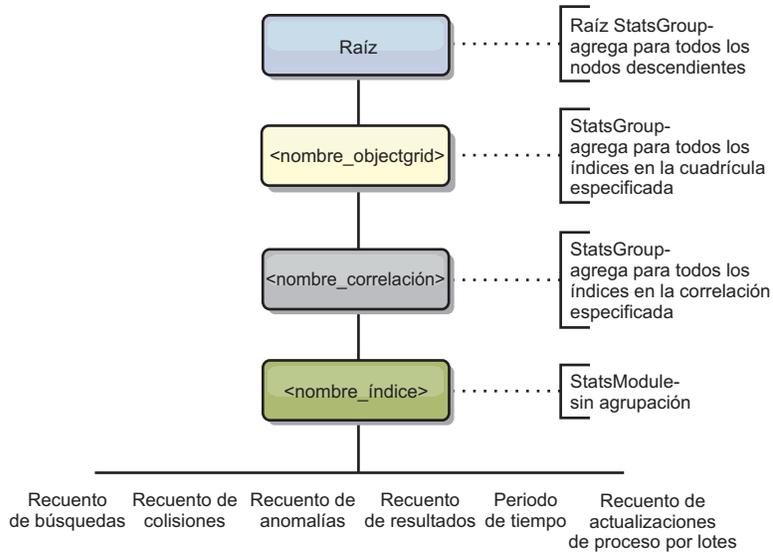


Figura 8. Estructura del módulo hashIndexModule

El siguiente diagrama muestra un ejemplo de la estructura de hashIndexModule:

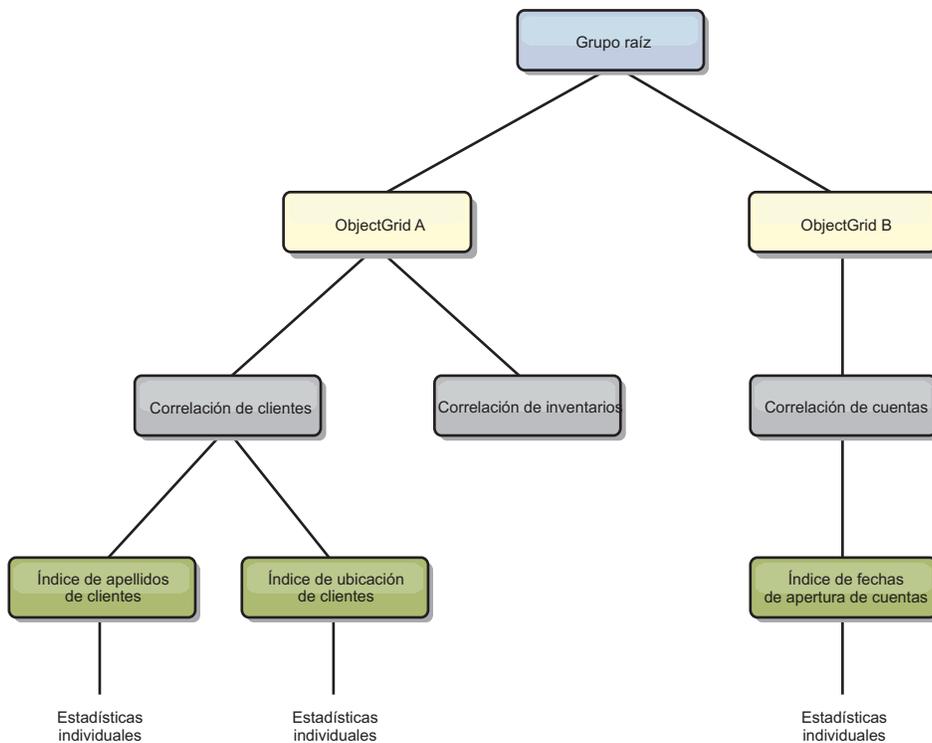


Figura 9. Ejemplo de estructura del módulo hashIndexModule

agentManagerModule

agentManagerModule contiene estadísticas relacionadas con los agentes de nivel de correlación:

- **Periodo de tiempo de reducción** - *TimeStatistic*: el intervalo de tiempo para que el agente termine la operación de reducción.

- **Periodo de tiempo total** - *TimeStatistic*: el intervalo de tiempo para que el agente complete todas las operaciones.
- **Periodo de tiempo de serialización de agente** - *TimeStatistic*: el intervalo de tiempo para serializar el agente.
- **Periodo de tiempo de inflación de agente** - *TimeStatistic*: el intervalo de tiempo que se tarda en inflar el agente en el servidor.
- **Periodo de tiempo de serialización de resultados** - *TimeStatistic*: el intervalo de tiempo para serializar los resultados de un agente.
- **Periodo de tiempo de inflación de resultados** - *TimeStatistic*: el intervalo de tiempo para inflar los resultados del agente.
- **Recuento de anomalías** - *CountStatistic*: el número de veces que el agente ha fallado.
- **Recuento de invocaciones** - *CountStatistic*: el número de veces que se ha invocado AgentManager.
- **Recuento de particiones** - *CountStatistic*: el número de particiones a las que se envía el agente.

El elemento raíz de agentManagerModule, "root", hace las veces de punto de entrada de las estadísticas de AgentManager. Este elemento raíz tiene ObjectGrids como elementos hijo, ObjectGrids tiene correlaciones como elementos hijo, que por último tienen instancias de AgentManager como elementos hijo y nodos finales del árbol. Cada instancia de AgentManager tiene las tres estadísticas listadas.

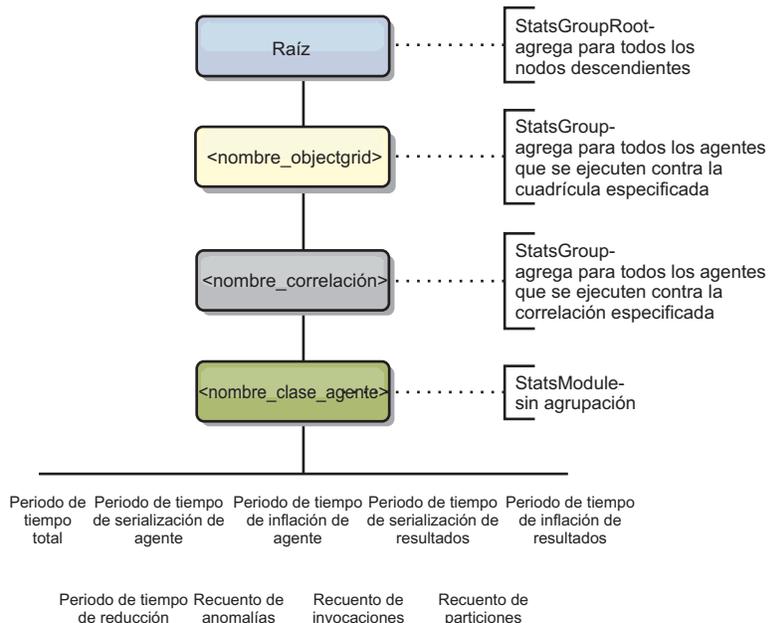


Figura 10. Estructura de agentManagerModule

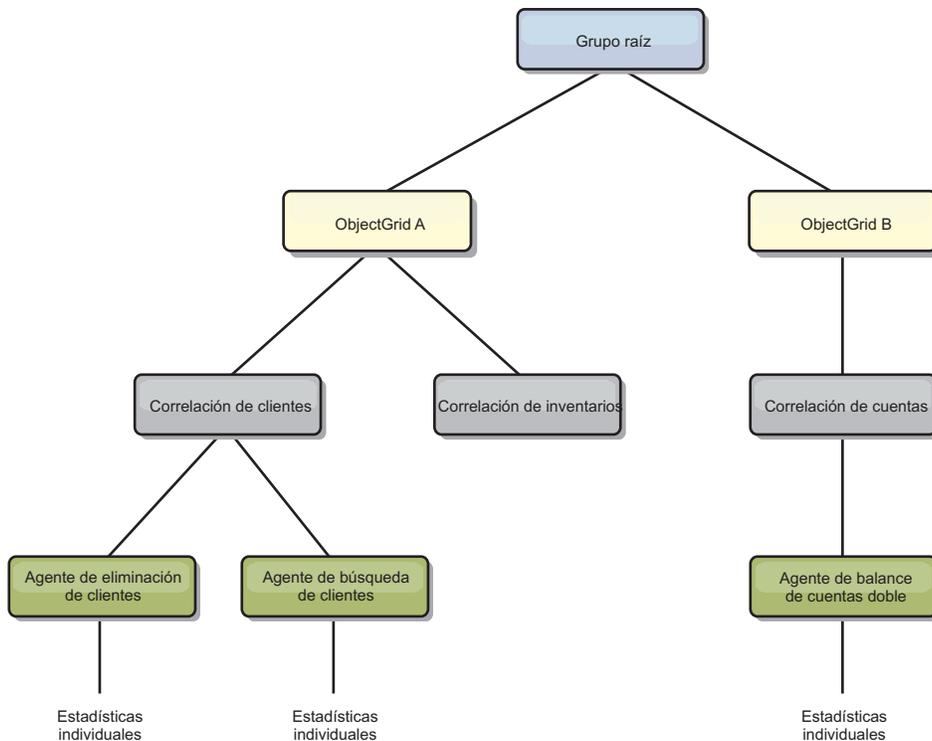


Figura 11. Ejemplo de la estructura de agentManagerModule

queryModule

queryModule contiene estadísticas relacionadas con las consultas de eXtreme Scale:

- **Tiempo de creación de plan** - *TimeStatistic*: el intervalo de tiempo para crear el plan de consulta.
- **Tiempo de ejecución** - *TimeStatistic*: el intervalo de tiempo para ejecutar la consulta.
- **Recuento de ejecuciones** - *CountStatistic*: el número de veces que se ha ejecutado la consulta.
- **Recuento de resultados** - *CountStatistic*: el recuento para cada conjunto de resultados de cada ejecución de consulta.
- **FailureCount** - *CountStatistic*: el número de veces que la consulta ha fallado.

El elemento raíz de queryModule, "root", hace las veces de punto de entrada de las estadísticas de Query. Este elemento raíz tiene ObjectGrids como elementos hijo, que tienen objetos Query como elementos hijo y nodos finales del árbol. Cada instancia de consulta tiene tres estadísticas listadas.

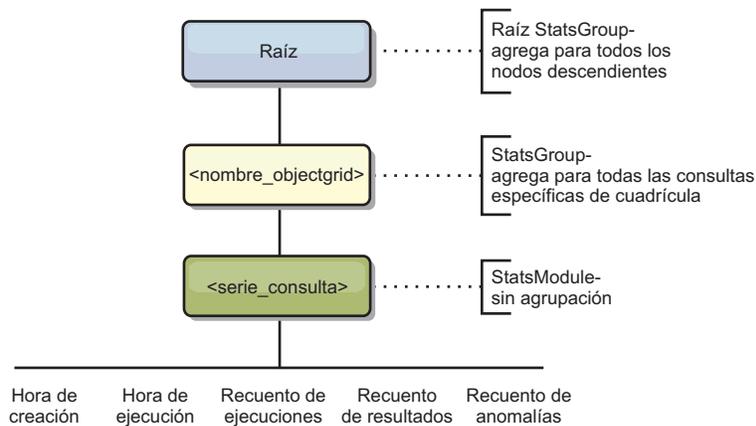


Figura 12. Estructura de queryModule

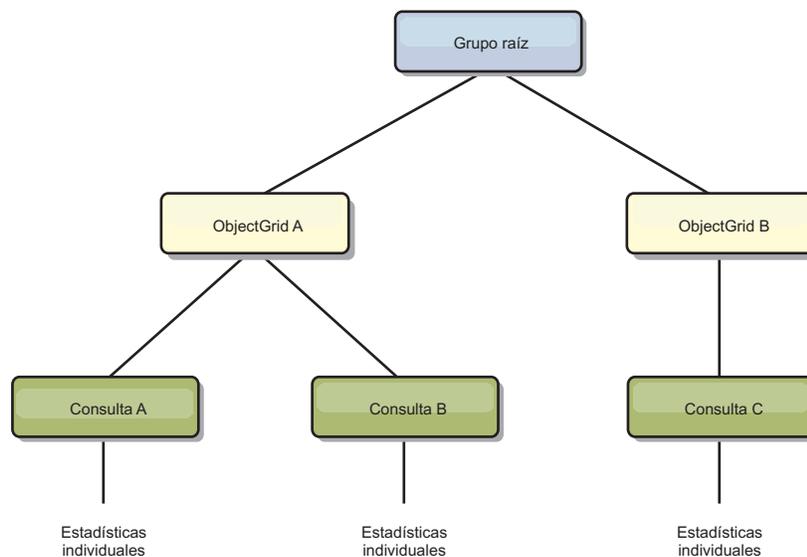


Figura 13. Ejemplo de la estructura de queryModule de QueryStats.jpg

Acceso a MBeans mediante la herramienta wsadmin

Puede utilizar el programa de utilidad wsadmin proporcionado en WebSphere Application Server para acceder a la información de MBean.

Ejecute la herramienta wsadmin desde el directorio bin de la instalación de WebSphere Application Server. En el siguiente ejemplo se recupera una vista de la colocación de fragmentos actual en un eXtreme Scale dinámico. Puede ejecutar wsadmin desde cualquier instalación donde se esté ejecutando eXtreme Scale. No tiene que ejecutar wsadmin en el servicio de catálogo.

```
$ wsadmin.sh -lang jython
wsadmin>placementService = AdminControl.queryNames
("com.ibm.websphere.objectgrid:*,type=PlacementService")
wsadmin>print AdminControl.invoke(placementService,
"listObjectGridPlacement","library ms1")

<objectGrid name="library" mapSetName="ms1">
  <container name="container-0" zoneName="DefaultDomain"
    hostName="host1.company.org" serverName="server1">
    <shard type="Primary" partitionName="0"/>
```

```
    <shard type="SynchronousReplica" partitionName="1"/>
  </container>
  <container name="container-1" zoneName="DefaultDomain"
  hostName="host2.company.org" serverName="server2">
    <shard type="SynchronousReplica" partitionName="0"/>
    <shard type="Primary" partitionName="1"/>
  </container>
  <container name="UNASSIGNED" zoneName="_ibm_SYSTEM"
  hostName="UNASSIGNED" serverName="UNNAMED">
    <shard type="SynchronousReplica" partitionName="0"/>
    <shard type="AsynchronousReplica" partitionName="0"/>
  </container>
</objectGrid>
```

Capítulo 7. Programación de la integración JPA

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

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.

Cargadores JPA

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

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

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

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

Arquitectura del cargador JPA

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

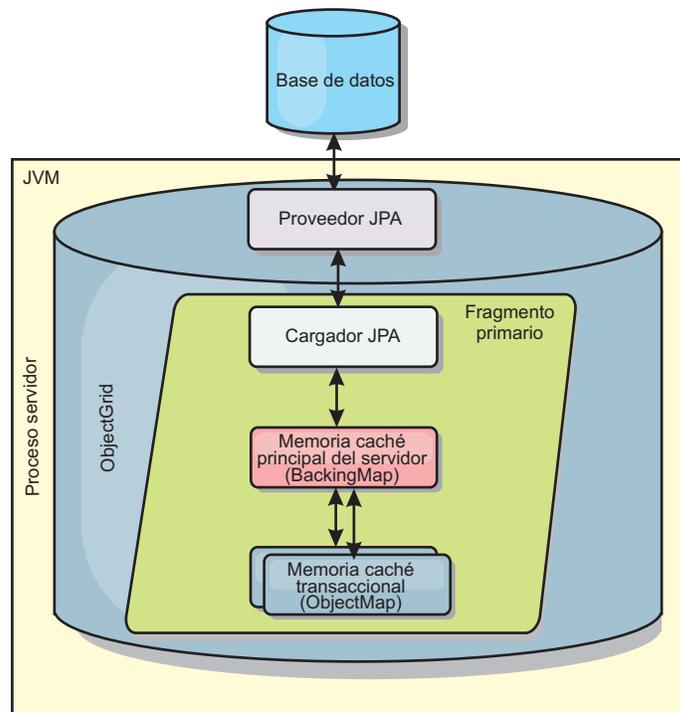


Figura 14. Arquitectura de cargador JPA

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`" en la página 263.

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 devuelta 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 $\text{value}(\text{número_total_de_particiones} - 1)$, 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.

Visión general del programa de utilidad de precarga JPA basada en cliente

El programa de utilidad de precarga de JPA (Java Persistence API) basado en cliente carga los datos en las correlaciones de respaldo de eXtreme Scale utilizando una conexión cliente con ObjectGrid.

Esta prestación puede simplificar la carga de correlaciones de eXtreme Scale cuando las consultas a la base de datos no se pueden particionar. También se puede utilizar un cargador como, por ejemplo, un cargador JPA, y es ideal cuando los datos se pueden cargar en paralelo.

El programa de utilidad de precarga JPA basado en cliente puede utilizar las implementaciones JPA de OpenJPA o Hibernate para cargar el ObjectGrid desde una base de datos. Puesto que WebSphere eXtreme Scale no interactúa directamente con la base de datos o JDBC (Java Database Connectivity), se puede utilizar cualquier base de datos que soporte OpenJPA o Hibernate para cargar el ObjectGrid.

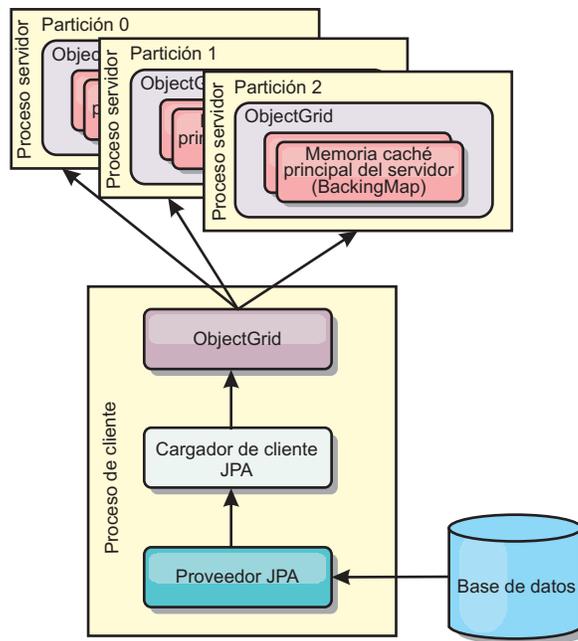


Figura 15. El cargador de cliente que utiliza la implementación JPA para cargar el ObjectGrid

Por lo general, una aplicación de usuario proporciona un nombre de unidad de persistencia, un nombre de clase de entidad y una consulta JPA al cargador de cliente. El cargador de cliente recupera el gestor de entidades JPA de acuerdo con el nombre de la unidad de persistencia, utiliza el gestor de entidades para consultar los datos de la base de datos con la clase de entidad y la consulta JPA proporcionadas, y finalmente carga los datos en las correlaciones ObjectGrid distribuidas. Cuando hay implicadas relaciones de varios niveles en la consulta, puede utilizar una serie de consulta personalizada para optimizar el rendimiento. De forma opcional, puede proporcionarse una correlación de propiedades de persistencia para alterar temporalmente las propiedades de persistencia configuradas.

Un cargador de cliente puede cargar los datos en dos modalidades distintas, tal como se muestra en la tabla siguiente:

Tabla 12. Modalidades del cargador de cliente

Modalidad	Descripción
<i>Precarga</i>	Borra todas las entradas y las carga en la correlación de respaldo. Si la correlación es una correlación de entidad, se borrarán también todas las correlaciones de entidad relacionadas si se ha habilitado la opción de ObjectGrid CascadeType.REMOVE.
<i>Recarga</i>	La consulta JPA se ejecuta en el objeto ObjectGrid para invalidar todas las entradas de la correlación que coincidan con la consulta. Si la correlación es una correlación de entidad, se borrarán también todas las correlaciones de entidad relacionadas si se ha habilitado la opción de ObjectGrid CascadeType.INVALIDATE.

En cualquier caso, una consulta JPA se utiliza para seleccionar y cargar las entidades deseadas desde la base de datos y para almacenarlas en las correlaciones ObjectGrid. Si la correlación ObjectGrid es una correlación que no es de entidad, las entidades JPA se separarán y se almacenarán directamente. Si la correlación ObjectGrid es una correlación de entidades, las entidades JPA se almacenan como tuples de entidad ObjectGrid. Puede proporcionar una consulta JPA o utilizar la consulta predeterminada `select o from EntityName o`.

Si desea más información sobre cómo configurar el programa de utilidad de precarga JPA basado en cliente, consulte la información en *Guía de programación*

Programación del programa de utilidad de precarga JPA basada en cliente

El programa de utilidad de precarga de JPA (Java Persistence API) basado en cliente carga los datos en las correlaciones de respaldo de eXtreme Scale utilizando una conexión cliente con ObjectGrid. Puede implementar la precarga y la recarga de datos en la aplicación.

A través del uso de la interfaz StateManager

Utilice el método `setObjectGridState` de la interfaz `StateManager` para establecer el estado de ObjectGrid en uno de los valores siguientes: `OFFLINE`, `ONLINE`, `QUIESCE` o `PRELOAD`. La interfaz `StateManager` impide a otros clientes acceder al ObjectGrid cuando todavía no está en línea.

Por ejemplo, establezca el estado de ObjectGrid en `PRELOAD` antes de cargar las correlaciones con los datos. Una vez que se han cargado los datos, establezca el estado de ObjectGrid en `ONLINE`. Consulte la información sobre cómo establecer la disponibilidad de un ObjectGrid en *Guía de administración* si desea más información.

Al precargar distintas correlaciones en un ObjectGrid, establezca el estado de ObjectGrid en `PRELOAD` una vez y vuelva a establecer el valor en `ONLINE` después de que todas las correlaciones acaben de cargar los datos. Esta coordinación se puede realizar mediante la interfaz `ClientLoadCallback`. Establezca el estado de ObjectGrid en `PRELOAD` después de recibir la primera notificación de `preStart` de la instancia de ObjectGrid y vuelva a establecerlo en `ONLINE` después de recibir la última notificación de `postFinish`.

Si tiene que precargar las correlaciones de distintas máquinas virtuales Java, es preciso que las coordine entre varias máquinas virtuales Java. Establezca el estado de ObjectGrid en `PRELOAD` una vez antes de que se precargue la primera correlación en cualquiera de las máquinas virtuales Java y establezca el valor de nuevo en `ONLINE` después de que todas las correlaciones finalicen la carga de datos en todas las máquinas virtuales Java.

Ejemplo de precarga basada en cliente

El flujo de la precarga de datos es el siguiente:

1. Borre la correlación que se va a precargar. En el caso de una correlación de entidad, si alguna relación se ha configurado como `cascade-remove`, las correlaciones relacionadas también se borran.
2. Ejecute la consulta en JPA para las entidades de un proceso por lotes. El tamaño del lote es de 1000.

3. Cada cada lote, cree una lista de claves y una lista de valores para cada partición.
4. Para cada partición, llame al agente de cuadrícula de datos para insertar o actualizar los datos en el lado del servidor directamente si se trata de un cliente de eXtreme Scale. Si la cuadrícula es una instancia local, inserte o actualice directamente los datos en las correlaciones ObjectGrid.

El siguiente fragmento de código de ejemplo muestra una carga sencilla de cliente.

```
// Obtener StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();

// Establecer el estado de ObjectGrid en PRELOAD antes de llamar a
ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Cargar los datos
c.load(objectGrid, "CUSTOMER", "custPU", null,
    null, null, null, true, null);

// Volver a establecer el estado de ObjectGrid en ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

En este ejemplo, la correlación CUSTOMER se configura como correlación de entidad. La clase de entidad Customer, que se configura en el archivo XML de descriptor de metadatos de entidad ObjectGrid, tiene una relación de uno a muchos con las entidades Order. La entidad Customer tiene la opción CascadeType.ALL habilitada en la relación con la entidad Order.

Antes de que se llame al método ClientLoader.load, el estado de ObjectGrid se establece en PRELOAD.

Los parámetros utilizados en el método ClientLoader.load son:

1. **objectGrid**: instancia ObjectGrid. Es una instancia ObjectGrid de cliente.
2. **"CUSTOMER"**: correlación que se va a cargar. Puesto que Customer tiene una relación de tipo cascade-all con las entidades Order, las entidades Order también se cargarán.
3. **"custPU"**: nombre de la unidad de persistencia de JPA de las entidades Customer y Order.
4. **null**: la correlación persistenceProps es nula, lo que significa que se utilizarán las propiedades de persistencia predeterminadas que se han configurado en persistence.xml.
5. **null**: la clase entityClass se configura como nula. Se establecerá en la clase de entidad configurada en el XML de descriptor de metadatos de entidad ObjectGrid para la correlación "CUSTOMER", en este caso, Customer.class.
6. **null**: loadSql es nulo, lo que significa que se utilizará la consulta predeterminada "select o from CUSTOMER o" para consultar las entidades JPA.
7. **null**: la correlación de parámetros de consulta es nula.
8. **true**: indica que la modalidad de carga de datos es de tipo precarga. Por lo tanto, las operaciones de borrado se llamarán para las correlaciones CUSTOMER y ORDER con el fin de borrar todos los datos antes de realizar la carga debido a la relación de tipo cascade-remove entre ellas.
9. **null**: ClientLoaderCallback es nulo.

Si desea más información sobre los parámetros necesarios, consulte la API ClientLoader en la documentación de la API.

Ejemplo de recarga

Recargar una correlación es lo mismo que precargar una correlación, excepto que el argumento `isPreload` se establece en `false` en el método `ClientLoader.load`.

En la modalidad de recarga, el flujo de la carga de datos es el siguiente:

1. Ejecute la consulta proporcionada en la correlación `ObjectGrid` e invalide todos los resultados. En el caso de una correlación de entidad, si se configura cualquier relación con la opción `CascadeType.INVALIDATE`, las entidades relacionadas también se invalidan en sus correlaciones.
2. Ejecute la consulta proporcionada en JPA para consultar las entidades JPA en el proceso por lotes. El tamaño del lote es de 1000.
3. Cada cada lote, cree una lista de claves y una lista de valores para cada partición.
4. Para cada partición, llame al agente de cuadrícula de datos para insertar o actualizar los datos en el servidor directamente si es un cliente eXtreme Scale. Si la cuadrícula es una configuración local de eXtreme Scale, inserte o actualice directamente los datos en las correlaciones `ObjectGrid`.

A continuación, un ejemplo de recarga:

```
// Obtener StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();

// Establecer el estado de ObjectGrid en PRELOAD antes de llamar a
ClientLoader.load
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Cargar los datos
String loadSql = "select c from CUSTOMER c
  where c.custId >= :startCustId and c.custId < :endCustId ";
Map<String, Long> params = new HashMap<String, Long>();
params.put("startCustId", 1000L);
params.put("endCustId", 2000L);

c.load(objectGrid, "CUSTOMER", "customerPU", null, null,
  loadSql, params, false, null);

// Volver a establecer el estado de ObjectGrid en ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Comparado con el ejemplo de precarga, la principal diferencia es que se proporcionan `loadSql` y parámetros. Este ejemplo sólo recarga los datos de `Customer` con un ID comprendido entre 1000 y 2000.

Tenga en cuenta que esta serie de consulta observa tanto la sintaxis de consulta de JPA, como la sintaxis de consulta de entidad de eXtreme Scale. Esta serie de consulta es importante porque se ejecuta dos veces, una vez en `ObjectGrid` para invalidar las entidades de `ObjectGrid` coincidentes y, a continuación, en JPA para cargar las entidades JPA coincidentes.

Llamar a un cargador de cliente en una implementación de cargador

En la interfaz Loader, hay un método preload:

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

Este método indica al cargador que puede precargar los datos en la correlación. Una implementación de cargador puede utilizar un cargador de cliente para precargar los datos en todas las particiones. Por ejemplo, el cargador JPA utiliza el cargador de cliente para precargar los datos en la correlación.

Si desea más información, consulte el tema de visión general de cargadores JPA en la *Visión general del producto*.

A continuación, se muestra un ejemplo sobre cómo precargar la correlación utilizando el cargador de clientes en el método preloadMap. El ejemplo, en primer lugar, comprueba si el número de partición actual es el mismo que el de la partición de precarga. Si el número de partición no es el mismo que el de la partición de precarga, no se produce ninguna acción. Si los números de partición coinciden, se llama al cargador de cliente para cargar los datos en las correlaciones. Es importante llamar al cargador de cliente en sólo una partición.

```
ObjectGrid og = session.getObjectGrid();
int partitionId = backingMap.getPartitionId();
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();

// Llamar al cargador de cliente para cargar datos en sólo una partición
if (partitionId == preloadPartition) {

    ClientLoader loader = ClientLoaderFactory.getClientLoader();

    // Llamar al cargador de cliente para cargar los datos
    try {

        loader.load(og, backingMap.getName(), txCallback.getPersistenceUnitName(),
            null, entityClass, null, null, true, null);
    } catch (ObjectGridException e) {
        LoaderException le = new LoaderException("Exception caught in ObjectMap "
            + ogName + "." + mapName);
        le.initCause(e);
        throw le;
    }
}
```

Nota: Configure el atributo de backingMap "preloadMode" en true, para que el método preload se ejecute de forma asíncrona. De lo contrario, dicho método bloqueará la instancia de ObjectGrid y no podrá activarse.

Carga manual de cliente

El método ClientLoader.load proporciona una función de carga de cliente que satisface la mayoría de los escenarios. Sin embargo, si desea cargar los datos sin el método ClientLoader.load, puede implementar su propia precarga.

Una plantilla de una carga de cliente manual es así:

```
// Obtener StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();
```

```
// Establecer el estado de ObjectGrid en PRELOAD antes de llamar a
ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);
```

```
// Cargar los datos
...
```

```
// Volver a establecer el estado de ObjectGrid en ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Si carga los datos del lado del cliente, la carga de los datos mediante un agente DataGrid podría aumentar el rendimiento. Al utilizar un agente DataGrid, todas las lecturas y escrituras de datos se producen en el proceso de servidor. También puede diseñar la aplicación para garantizar que los agentes DataGrid se ejecuten en paralelo en varias particiones para aumentar el rendimiento de forma adicional.

Para implementar la precarga de datos con un agente DataGrid, consulte el siguiente ejemplo.

Después de crear la implementación de precarga de datos, puede crear un cargador genérico para completar las tareas siguientes:

1. Consulte los datos de la base de datos en lotes.
2. Cree una lista de claves y una lista de valores para cada partición.
3. Para cada partición, llame al método `agentMgr.callReduceAgent(agent, aKey)` para ejecutar el agente en el servidor de una hebra. Al realizar la ejecución en una hebra, puede ejecutar agentes de forma simultánea en varias particiones.

Ejemplo: la precarga de datos con un agente DataGrid

Si carga los datos del lado del cliente, la carga de los datos mediante un agente DataGrid podría aumentar el rendimiento. Al utilizar un agente DataGrid, todas las lecturas y escrituras de datos se producen en el proceso de servidor. También puede diseñar la aplicación para garantizar que los agentes DataGrid se ejecuten en paralelo en varias particiones para aumentar el rendimiento de forma adicional.

A continuación, aparece un ejemplo de cómo cargar los datos con un agente DataGrid:

```
import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

import com.ibm.websphere.objectgrid.NoActiveTransactionException;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TransactionException;
import com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent;
import com.ibm.websphere.objectgrid.em.EntityManager;

public class InsertAgent implements ReduceGridAgent, Externalizable {

    private static final long serialVersionUID = 6568906743945108310L;
```

```

private List keys = null;

private List vals = null;

protected boolean isEntityMap;

public InsertAgent() {
}

public InsertAgent(boolean entityMap) {
    isEntityMap = entityMap;
}

public Object reduce(Session sess, ObjectMap map) {
    throw new UnsupportedOperationException(
        "ReduceGridAgent.reduce(Session, ObjectMap)");
}

public Object reduce(Session sess, ObjectMap map, Collection arg2) {
    Session s = null;
    try {
        s = sess.getObjectGrid().getSession();
        ObjectMap m = s.getMap(map.getName());
        s.beginNoWriteThrough();
        Object ret = process(s, m);
        s.commit();
        return ret;
    } catch (ObjectGridRuntimeException e) {
        if (s.isTransactionActive()) {
            try {
                s.rollback();
            } catch (TransactionException e1) {
            } catch (NoActiveTransactionException e1) {
            }
        }
        throw e;
    } catch (Throwable t) {
        if (s.isTransactionActive()) {
            try {
                s.rollback();
            } catch (TransactionException e1) {
            } catch (NoActiveTransactionException e1) {
            }
        }
        throw new ObjectGridRuntimeException(t);
    }
}

public Object process(Session s, ObjectMap m) {
    try {

        if (!isEntityMap) {
            // En el caso POJO, es una operación directa,
            // se coloca todo en la
            // correlación mediante la operación de insertar
            insert(m);
        } else {
            // 2. Caso Entity.
            // En el caso Entity, pueden persistirse las entidades
            EntityManager em = s.getEntityManager();
            persistEntities(em);
        }

        return Boolean.TRUE;
    } catch (ObjectGridRuntimeException e) {
        throw e;
    }
}

```

```

        } catch (ObjectGridException e) {
            throw new ObjectGridRuntimeException(e);
        } catch (Throwable t) {
            throw new ObjectGridRuntimeException(t);
        }
    }

    /**
     * Básicamente se trata de una nueva carga.
     * @establecer parámetro s
     * @establecer parámetro m
     * @emite ObjectGridException
     */
    protected void insert(ObjectMap m) throws ObjectGridException {

        int size = keys.size();

        for (int i = 0; i < size; i++) {
            m.insert(keys.get(i), vals.get(i));
        }
    }

    protected void persistEntities(EntityManager em) {
        Iterator<Object> iter = vals.iterator();

        while (iter.hasNext()) {
            Object value = iter.next();
            em.persist(value);
        }
    }

    public Object reduceResults(Collection arg0) {
        return arg0;
    }

    public void readExternal(ObjectInput in)
        throws IOException, ClassNotFoundException {
        int v = in.readByte();
        isEntityMap = in.readBoolean();
        vals = readList(in);
        if (!isEntityMap) {
            keys = readList(in);
        }
    }

    public void writeExternal(ObjectOutput out) throws IOException {
        out.write(1);
        out.writeBoolean(isEntityMap);

        writeList(out, vals);
        if (!isEntityMap) {
            writeList(out, keys);
        }
    }

    public void setData(List ks, List vs) {
        vals = vs;
        if (!isEntityMap) {
            keys = ks;
        }
    }
    /**

```

```

    * @devuelve isEntityMap.
    */
    public boolean isEntityMap() {
        return isEntityMap;
    }

    static public void writeList(ObjectOutput oo, Collection l)
        throws IOException {
        int size = l == null ? -1 : l.size();
        oo.writeInt(size);
        if (size > 0) {
            Iterator iter = l.iterator();
            while (iter.hasNext()) {
                Object o = iter.next();
                oo.writeObject(o);
            }
        }
    }

    public static List readList(ObjectInput oi)
        throws IOException, ClassNotFoundException {
        int size = oi.readInt();
        if (size == -1) {
            return null;
        }

        ArrayList list = new ArrayList(size);
        for (int i = 0; i < size; ++i) {
            Object o = oi.readObject();
            list.add(o);
        }
        return list;
    }
}

```

Actualizador de datos basado en la hora de JPA

Un actualizador de base de datos basado en la hora de JPA (Java Persistence API) actualiza las correlaciones de ObjectGrid con los últimos cambios de la base de datos.

Cuando los cambios se realizan directamente en una base de datos que es atendida por WebSphere eXtreme Scale, estos cambios no se reflejan de forma simultánea en la cuadrícula de eXtreme Scale. Para implementar correctamente eXtreme Scale como un espacio de proceso de base de datos en memoria, tenga en cuenta que la cuadrícula puede perder la sincronización con la base de datos. El actualizador de base de datos basado en la hora utiliza la capacidad SCN (System Change Number) en Oracle 10g la indicación de fecha y hora de cambio de fila en DB2 9.5 para supervisar los cambios en la base de datos para la invalidación y la actualización. El actualizador también permite a las aplicaciones tener un campo definido por el usuario con el mismo propósito.

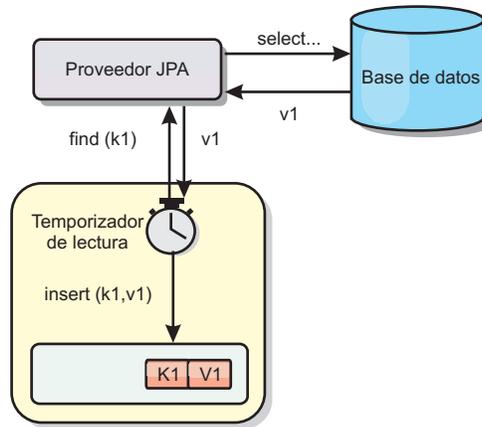


Figura 16. Renovación periódica

El actualizador de la base de datos basado en la hora consulta periódicamente la base de datos utilizando interfaces JPA para obtener las entidades JPA que representan los registros recién insertados y actualizados en la base de datos. Para actualizar de forma periódica los registros, cada registro de la base de datos debe tener una indicación de fecha y hora para identificar la hora o secuencia en la que se actualizó o insertó el registro por última vez. No es necesario que la indicación de fecha y hora esté en un formato de indicación de fecha y hora. El valor de indicación de fecha y hora puede ser tener un formato de entero o largo, si genera un valor único creciente.

Esta prestación la proporcionan varias bases de datos comerciales.

Por ejemplo, en DB2 9.5, puede definir una columna utilizando el formato ROW CHANGE TIMESTAMP del modo siguiente:

```
ROWCHGTS TIMESTAMP NOT NULL
GENERATED ALWAYS
FOR EACH ROW ON UPDATE AS
ROW CHANGE TIMESTAMP
```

En Oracle, puede utilizar la pseudo-columna **ora_rowscn**, que representa el número de cambio de sistema del registro.

El actualizador de base de datos basado en la hora actualiza las correlaciones ObjectGrid de tres maneras diferentes:

1. **INVALIDATE_ONLY**. Invalida las entradas de la correlación ObjectGrid si han cambiado los registros correspondientes de la base de datos.
2. **UPDATE_ONLY**. Actualiza las entradas de la correlación ObjectGrid si han cambiado los registros correspondientes de la base de datos. No obstante, todos los registros recién insertados en la base de datos se pasan por alto.
3. **INSERT_UPDATE**. Actualiza las entradas existentes en la correlación ObjectGrid con los valores más recientes de la base de datos. Además, todos los registros recién insertados en la base de datos se insertan en la correlación ObjectGrid.

Si desea más información sobre cómo configurar el actualizador de datos basado en la hora de JPA, consulte la información de *Guía de administración*.

Inicio del actualizador basado en la hora de JPA

Cuando inicie el actualizador basado en la hora de JPA (Java Persistence API), las correlaciones de ObjectGrid se actualizan con los últimos cambios de la base de datos.

Antes de empezar

Configure el actualizador basado en la hora. Consulte la información sobre cómo configurar un actualizador de datos basado en la hora de JPA en *Guía de administración*.

Acerca de esta tarea

Si desea más información sobre cómo funciona el actualizador de datos basado en la hora de Java Persistence API (JPA), consulte “Actualizador de datos basado en la hora de JPA” en la página 320.

Procedimiento

- Inicie un actualizador de base de datos basado en la hora.
 - **De forma automática para el eXtreme Scale distribuido:**

Si crea la configuración de `timeBasedDBUUpdate` para la correlación de respaldo, el actualizador de la base de datos basado en la hora se inicia automáticamente cuando se activa un fragmento distribuido de ObjectGrid. Para un ObjectGrid que tiene varias particiones, el actualizador de la base de datos basado en la hora sólo se inicia en la partición 0.
 - **De forma automática para el eXtreme Scale local:**

Si crea la configuración de `timeBasedDBUUpdate` para la correlación de respaldo, el actualizador de la base de datos basado en la hora se inicia automáticamente cuando se activa la correlación local.
 - **Manualmente:**

También puede iniciar o detener manualmente un actualizador de base de datos basado en la hora mediante la API `TimeBasedDBUUpdater`.

```
public synchronized void startDBUUpdate(ObjectGrid objectGrid, String mapName,
    String punitName, Class entityClass, String timestampField, DBUUpdateMode mode) {
```

 1. **ObjectGrid:** instancia ObjectGrid (local o cliente).
 2. **mapName:** nombre de la correlación de respaldo para la cual se inicia el actualizador de base de datos basado en la hora.
 3. **punitName:** el nombre de la unidad de persistencia JPA para crear una fábrica del gestor de entidades JPA; el valor predeterminado es el nombre de la primera unidad de persistencia definida en el archivo `persistence.xml`.
 4. **entityClass:** el nombre de la clase de entidad utilizado para interactuar con el proveedor de JPA (Java Persistence API); el nombre de la clase de entidad se utiliza para obtener entidades JPA utilizando las consultas de entidad.
 5. **timestampField:** un campo de indicación de fecha y hora de la clase de entidad para identificar la hora o la secuencia cuando se actualizó o insertó por última vez un registro de programa de fondo de una base de datos.
 6. **mode:** modalidad de actualización de la base de datos basada en tiempo; un tipo `INVALIDATE_ONLY` invalida las entradas en la correlación ObjectGrid si se han modificado los registros correspondientes en la base

de datos; un tipo UPDATE_ONLY indica sólo la actualización de las entradas existentes en la correlación ObjectGrid con los valores más recientes de la base de datos; no obstante, todos los registros recientemente insertados en la base de datos se pasan por alto; un tipo INSERT_UPDATE indica la actualización de las entradas existentes en la correlación ObjectGrid con los últimos valores de la base de datos; además, todos los registros recién insertados en la base de datos se insertan en la correlación ObjectGrid.

Si desea detener el actualizador de base de datos basado en la hora, puede llamar al siguiente método para detener el actualizador:

```
public synchronized void stopDBUpdate(ObjectGrid objectGrid, String mapName)
```

Los parámetros ObjectGrid y mapName deben ser los mismos que los que se pasan en el método startDBUpdate.

- Cree el campo de indicación de fecha y hora en la base de datos.

– DB2

Como parte de las características de bloqueo optimista, DB2 9.5 proporciona una función de indicación de fecha y hora de cambio de fila. Puede definir una columna ROWCHGTS mediante el formato ROW CHANGE TIMESTAMP de la siguiente manera:

```
ROWCHGTS TIMESTAMP NOT NULL
GENERATED ALWAYS
FOR EACH ROW ON UPDATE AS
ROW CHANGE TIMESTAMP
```

A continuación, puede indicar el campo de entidad que corresponde a esta columna como campo de indicación de hora mediante una anotación o configuración. A continuación se muestra un ejemplo:

```
@Entity(name = "USER_DB2")
@Table(name = "USER1")
public class User_DB2 implements Serializable {

    private static final long serialVersionUID = 1L;

    public User_DB2() {
    }

    public User_DB2(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ROWCHGTS", updatable = false, insertable = false)
    public Timestamp rowChgTs;
}
```

– Oracle

En Oracle, hay una pseudocolumna ora_rowscn para el número de cambio del sistema del registro. Puede utilizar esta columna para el mismo propósito. A

continuación, aparece un ejemplo de la entidad que utiliza el campo `ora_rowscn` como el campo de indicación de fecha y hora de actualización de la base de datos basada en tiempo:

```
@Entity(name = "USER_ORA")
@Table(name = "USER1")
public class User_ORA implements Serializable {

    private static final long serialVersionUID = 1L;

    public User_ORA() {
    }

    public User_ORA(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ora_rowscn", updatable = false, insertable = false)
    public long rowChgTs;
}
```

– Otras bases de datos

Para otros tipos de bases de datos, puede crear una columna de tabla para realizar un seguimiento de los cambios. Los valores de la columna debe gestionarlos manualmente la aplicación que actualiza la tabla.

Tome una base de datos Apache Derby como un ejemplo: puede crear una columna "ROWCHGTS" para rastrear los números de cambio. Además, se realiza un seguimiento del último número de cambio en esta tabla. Cada vez que se inserta o actualiza un registro, se aumenta el número de cambio más reciente en la tabla, y el valor de la columna ROWCHGTS para el registro se actualiza con este número incrementado.

```
@Entity(name = "USER_DER")
@Table(name = "USER1")
public class User_DER implements Serializable {

    private static final long serialVersionUID = 1L;

    public User_DER() {
    }

    public User_DER(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;
}
```

```
@Column(name = "LASTNAME")
public String lastName;

@com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
@Column(name = "ROWCHGTS", updatable = true, insertable = true)
public long rowChgTs;
}
```

Capítulo 8. Programación de la integración de Spring

Obtenga información sobre cómo integrar las aplicaciones de eXtreme Scale con la conocida infraestructura Spring.

Visión general de integración en la infraestructura de Spring

Spring es una infraestructura popular para desarrollar las aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para permitir a Spring gestionar las transacciones de eXtreme Scale y configurar los clientes y servidores que conforman la cuadrícula de datos en memoria desplegada.

Transacciones nativas gestionadas de Spring

Spring proporciona transacciones gestionadas por contenedor que son similares al servidor de aplicaciones Java Platform, Enterprise Edition. Sin embargo, el mecanismo Spring se puede conectar a distintas implementaciones. WebSphere eXtreme Scale proporciona una integración del gestor de transacciones que permite a Spring gestionar los ciclos de vida de transacción de ObjectGrid. Consulte la información sobre las transacciones nativas en *Guía de programación*, para buscar detalles.

Beans de ampliación gestionados de Spring y soporte de espacio de nombres

Además, eXtreme Scale se integra con Spring para habilitar a los beans de estilo Spring definidos para los puntos o plug-ins de ampliación. Esta característica proporciona configuraciones más sofisticadas y más flexibilidad para configurar los puntos de ampliación.

Además de los beans de ampliación gestionados de Spring, eXtreme Scale proporciona un espacio de nombres Spring denominado "objectgrid". Los beans y las implementaciones incorporadas están definidos previamente en este espacio de nombres, que hace que sea más fácil para los usuarios configurar eXtreme Scale. Consulte Beans de ampliación de Spring y soporte de espacio de nombres, si desea más detalles sobre estos temas y un ejemplo sobre cómo iniciar un servidor de contenedor de eXtreme Scale utilizando las configuraciones de Spring.

Soporte de ámbito de fragmento

Con la configuración de Spring de estilo tradicional, un bean ObjectGrid puede ser un tipo singleton o un tipo de prototipo. Además, ObjectGrid soporta un nuevo ámbito denominado el ámbito de "fragmento". Si un bean está definido como ámbito de fragmento, sólo se crea un bean por fragmento. Todas las solicitudes para los beans con un ID o IDs que coinciden con dicha definición de bean en el mismo fragmento generarán que una instancia de bean específica sea devuelta por el contenedor Spring.

El siguiente ejemplo muestra que un bean `com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl` está definido con el ámbito establecido en `shard` (fragmento). Por lo tanto, sólo se crea una instancia de la clase `JPAPropFactoryImpl` por fragmento.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Flujo web de Spring

El flujo web de Spring almacena su estado de sesión en la sesión HTTP de forma predeterminada. Si una aplicación web se configura para utilizar eXtreme Scale para la gestión de sesiones, Spring lo utiliza automáticamente para almacenar su estado y se convierte en tolerante a errores del mismo modo que la sesión.

Empaquetado

Las extensiones Spring de eXtreme Scale están en el archivo `ogspring.jar`. Este archivo Java (JAR) debe estar en la classpath para trabajar con el soporte de Spring. Si una aplicación JEE se está ejecutando en un WebSphere Extended Deployment aumentado WebSphere Application Server Network Deployment, la aplicación deberá colocar el archivo `ogspring.jar` y sus archivos asociados en los módulos archivadores empresariales (EAR). También debe colocar el archivo `ogspring.jar` en la misma ubicación.

Transacciones gestionadas de Spring

Spring es una infraestructura popular para el desarrollo de aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para que Spring pueda gestionar transacciones de eXtreme Scale y configurar clientes y servidores de eXtreme Scale.

Transacciones nativas con WebSphere eXtreme Scale

Spring proporciona transacciones gestionadas por contenedor junto con el estilo de un servidor de aplicaciones Java Platform, Enterprise Edition, pero tiene la ventaja de que el mecanismo Spring pueden tener distintas implementaciones conectadas. Este tema describe un gestor de transacciones de la plataforma eXtreme Scale que se puede utilizar con Spring. Esto permite a los programadores anotar sus POJO (Plain Old Java Object) y, a continuación, hacer que Spring adquiera automáticamente los objetos Sessions de eXtreme Scale, así como empezar, confirmar, retrotraer, suspender y reanudar transacciones eXtreme Scale. Las transacciones Spring se describen de forma más completa en el Capítulo 10 de la documentación de referencia oficial de Spring. A continuación se explica cómo crear un gestor de transacciones eXtreme Scale y utilizarlo con los POJO anotados. También explica cómo utilizar este enfoque con eXtreme Scale local o cliente así como una aplicación de estilo Data Grid con ubicación compartida.

Creación de un gestor de transacciones

eXtreme Scale proporciona una implementación de un Spring PlatformTransactionManager. Este gestor puede proporcionar sesiones de eXtreme Scale gestionadas a los POJO gestionados por Spring. A través del uso de anotaciones, Spring gestiona estas sesiones para los POJO en términos de ciclo de vida de transacción. El siguiente fragmento de código XML muestra cómo crear un gestor de transacciones:

```
<aop:aspectj-autoproxy/>
<tx:annotation-driven transaction-manager="transactionManager"/>

<bean id="ObjectGridManager"
      class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
      factory-method="getObjectGridManager"/>

<bean id="ObjectGrid"
      factory-bean="ObjectGridManager"
      factory-method="createObjectGrid"/>

<bean id="transactionManager"
      class="com.ibm.websphere.objectgrid.spring.ObjectGridSpringFactory"
      factory-method="getLocalPlatformTransactionManager"/>
```

```

</bean>

<bean id="Service" class="com.ibm.websphere.objectgrid.spring.test.TestService">
  <property name="txManager" ref="transactionManager"/>
</bean>

```

Esto muestra el bean transactionManager declarándose y conectándose al bean Service que utilizará transacciones Spring. Los demostraremos utilizando anotaciones y por este motivo existe la cláusula tx:annotation al principio.

Obtención de una Session de ObjectGrid para la transacción Spring actual

Un POJO que tiene métodos gestionados por Spring ahora puede obtener la Session de ObjectGrid para la transacción actual utilizando

```
Session s = txManager.getSession();
```

Esto devuelve la sesión para que lo utilice POJO. Los beans que participan en la misma transacción recibirán la misma sesión cuando llame a este método. Spring manejará automáticamente el inicio de Session y también invocará automáticamente la confirmación o la retrotracción cuando sea necesario. Puede obtener un EntityManager de ObjectGrid llamando simplemente a getEntityManager desde el objeto Session.

Un POJO de ejemplo utilizando anotaciones

A continuación se muestra un POJO que utiliza anotaciones para declarar sus intenciones transaccionales a Spring. Puede ver que la clase tiene una anotación de nivel de clase que indica que todos los métodos predeterminados deben utilizar la semántica de transacción REQUIRED. La clase implementa una interfaz con un método para todos los métodos en la clase. Esto es necesario para que Spring AOP funcione cuando no puede efectuar la trama de código de bytes. La clase tiene una variable de instancia txManager que conectamos al gestor de transacciones de ObjectGrid utilizando el archivo XML de Spring. Cada método simplemente llama al método txManager.getSession para obtener la Session para utilizar para el método. El método queryNewTx se anota para iniciar una semántica REQUIRES_NEW. Esto significa que cualquier transacción existente se suspenderá y se creará una nueva transacción independiente para ese método.

```

@Transactional(propagation=Propagation.REQUIRED)
public class TestService implements ITestService
{
    SpringLocalTxManager txManager;

    public TestService()
    {
    }

    public void initialize()
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        ObjectMap m = s.getMap("TEST");
        m.insert("Hello", "Billy");
    }

    public void update(String updatedValue)
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        System.out.println("Update using " + s);
        ObjectMap m = s.getMap("TEST");
        String v = (String)m.get("Hello");
        m.update("Hello", updatedValue);
    }

    public String query()
        throws ObjectGridException
    {
        Session s = txManager.getSession();

```

```

        System.out.println("Query using " + s);
        ObjectMap m = s.getMap("TEST");
        return (String)m.get("Hello");
    }

    @Transactional(propagation=Propagation.REQUIRES_NEW)
    public String queryNewTx()
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        System.out.println("QueryTX using " + s);
        ObjectMap m = s.getMap("TEST");
        return (String)m.get("Hello");
    }

    public void testRequiresNew(ITestService bean)
        throws ObjectGridException
    {
        update("1");
        String txValue = bean.query();
        if(!txValue.equals("1"))
        {
            System.out.println("Requires didnt work");
            throw new IllegalStateException("requires didn't work");
        }
        String committedValue = bean.queryNewTx();
        if(committedValue.equals("1"))
        {
            System.out.println("Requires new didnt work");
            throw new IllegalStateException("requires new didn't work");
        }
    }

    public SpringLocalTxManager getTxManager() {
        return txManager;
    }

    public void setTxManager(SpringLocalTxManager txManager) {
        this.txManager = txManager;
    }
}

```

Establecer la instancia de ObjectGrid para una hebra

Una única máquina virtual Java (JVM) puede alojar muchas instancias ObjectGrid. Cada fragmento primario colocado en una JVM tiene su propia instancia de ObjectGrid. Una JVM que funcione como cliente para un ObjectGrid remoto utiliza una instancia de ObjectGrid devuelta de ClientClusterContext del método de conexión para interactuar con Grid. Antes de invocar un método en un POJO que utilicen transacciones Spring para ObjectGrid, la hebra debe prepararse con la instancia de ObjectGrid a utilizar. La instancia TransactionManager tiene un método que permite especificar una instancia de ObjectGrid concreta. Una vez que se ha especificado, todas las llamadas a txManager.getSession posteriores devolverán Sessions para esa instancia de ObjectGrid.

Programa de arranque simple para la prueba

El siguiente ejemplo muestra un ejemplo para utilizar esta prestación:

```

ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext(new String[]
    {"applicationContext.xml"});
SpringLocalTxManager txManager = (SpringLocalTxManager)ctx.getBean("transactionManager");
txManager.setObjectGridForThread(og);

ITestService s = (ITestService)ctx.getBean("Service");
s.initialize();
assertEquals(s.query(), "Billy");
s.update("Bobby");
assertEquals(s.query(), "Bobby");
System.out.println("Requires new test");
s.testRequiresNew(s);
assertEquals(s.query(), "1");

```

Aquí utilizamos un Spring ApplicationContext. ApplicationContext se utiliza para obtener una referencia para txManager y especificar un ObjectGrid que se va a utilizar con esta hebra. A continuación, el código obtiene una referencia para el servicio e invoca métodos del mismo. Cada llamada al método de este nivel hace

que Spring cree una Session y realice llamadas de inicio/confirmación alrededor de la llamada al método. Todas las excepciones causarán una retrotracción.

Interfaz SpringLocalTxManager

La interfaz SpringLocalTxManager la implementa el gestor de transacciones de la plataforma de ObjectGrid y tiene todas las interfaces públicas. Los métodos de esta interfaz sirven para seleccionar la instancia de ObjectGrid para utilizar en una hebra y obtener una Session para la hebra. Todos los POJO que utilizan las transacciones locales de ObjectGrid deben incluirse con una referencia a esta instancia del gestor y sólo se debe crear una única instancia, es decir, su ámbito debe ser singleton. Esta instancia se crea utilizando un método estático en ObjectGridSpringFactory. getLocalPlatformTransactionManager().

eXtreme Scale para las transacciones JTA y globales

eXtreme Scale no soporta a JTA ni la confirmación de 2 fases por varias razones, principalmente, por la escalabilidad. Por ello, excepto en el último participante de una única fase, ObjectGrid no interactúa con transacciones globales de tipo XA o JTA. Este gestor de plataformas está pensado para hacer que la utilización de transacciones de ObjectGrid locales sea lo más fácil posible para los desarrolladores de Spring.

Beans de ampliación gestionados de Spring

Puede declarar los POJO para utilizarlos como puntos de ampliación en el archivo objectgrid.xml. Si denomina los beans y luego especifica el nombre de clase, eXtreme Scale suele crear instancias de la clase especificada y utiliza esas instancias como plug-in. Ahora, eXtreme Scale ObjectGrid puede delegar en Spring para actuar como la fábrica de beans para obtener instancias de estos objetos de plug-in.

Si una aplicación utiliza Spring, en general, será necesario que los POJO se conecten al resto de la aplicación.

Una aplicación puede registrar una fábrica de beans de Spring para utilizarlos para un ObjectGrid con nombre específico. La aplicación debe crear una instancia de BeanFactory o un contexto de aplicación de Spring y, a continuación, registrarla con ObjectGrid utilizando el siguiente método estático:

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object springBeanFactory)
```

Este método especifica que si ObjectGrid encuentra un bean de ampliación (como ObjectTransformer, Loader, TransactionCallback, etc.) cuyo className empieza con el prefijo {spring}, utilice el resto del nombre de un bean Spring y obtenga la instancia del bean mediante la fábrica de beans Spring. ObjectGrid también puede crear una fábrica de beans de Spring desde un archivo de configuración XML de Spring predeterminado. Si no se ha registrado ninguna fábrica de beans para un ObjectGrid dado, ObjectGrid intentará encontrar un archivo xml llamado 'ObjectGridName'_spring.xml. Por ejemplo, si la cuadrícula se llama GRID, el archivo xml se llama '/GRID_spring.xml' y debe estar en la vía de acceso de clases en el paquete raíz. Si este archivo existe, ObjectGrid construye un ApplicationContext utilizando dicho archivo y construye beans desde esa fábrica de beans. Como ejemplo, el nombre de clase sería:

```
"{spring}MyLoaderBean"
```

Esto podría causar que ObjectGrid solicitara a Spring un bean llamado "MyLoaderBean". Este enfoque puede utilizarse para especificar POJO gestionados

por Spring para cualquier punto de ampliación en ObjectGrid siempre que la fábrica de beans se haya registrado por anticipado. Las ampliaciones de Spring de ObjectGrid están en el archivo ogspring.jar. Este archivo jar debe estar en la classpath para que funcione el soporte a Spring. Si una aplicación JavaEE que se ejecuta en un ND aumentado por XD, la aplicación debe colocar el archivo spring.jar y sus archivos asociados en los módulos EAR. El archivo ogspring.jar también debe colocarse en la misma ubicación.

Beans de ampliación de Spring y soporte de espacio de nombres

WebSphere eXtreme Scale proporciona una característica para declarar objetos POJO (Plain Old Java Object) para utilizarlos como puntos de ampliación en el archivo objectgrid.xml y un método para denominar los beans y, a continuación, especificar el nombre de la clase. Normalmente, se crean las instancias de la clase especificada y estos objetos se utilizan como los plug-ins. Ahora, eXtreme Scale puede delegar en Spring para obtener las instancias de estos objetos de plug-in. Si una aplicación utiliza Spring en general será necesario que los POJO se conecten al resto de la aplicación.

En algunos casos, debe utilizar Spring para configurar determinados objetos de plug-in. Tome la siguiente configuración como ejemplo:

```
<objectGrid name="Grid">
  <bean id="TransactionCallback" className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
    <property name="persistenceUnitName" type="java.lang.String" value="employeePU" />
  </bean>
  ...
</objectGrid>
```

La implementación de TransactionCallback incorporada, la clase com.ibm.websphere.objectgrid.jpa.JPATxCallback, se configura como la clase TransactionCallback. Esta clase se configura con la propiedad persistenceUnitName tal como se muestra en el ejemplo anterior. La clase JPATxCallback también tiene el atributo JPAPropertyFactory, que es del tipo java.lang.Object. La configuración XML de ObjectGrid no puede soportar este tipo de configuración.

La integración de Spring eXtreme Scale resuelve este problema delegando la creación de bean en la infraestructura Spring. La configuración revisada es la siguiente:

```
<objectGrid name="Grid">
  <bean id="TransactionCallback" className="{spring}jpaTxCallback"/>
  ...
</objectGrid>
```

El archivo spring para el objeto "Grid" contiene la siguiente información:

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
  <property name="persistenceUnitName" value="employeeEMPU"/>
  <property name="JPAPropertyFactory" ref="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.
JPAPropFactoryImpl" scope="shard">
</bean>
```

Aquí, TransactionCallback se especifica como {spring}jpaTxCallback, y los beans jpaTxCallback y jpaPropFactory se configuran en el archivo spring tal como se indica en el ejemplo anterior. La configuración de Spring hace posible configurar un bean JPAPropertyFactory como un parámetro del objeto JPATxCallback.

Fábrica de beans Spring predeterminada

Cuando eXtreme Scale encuentra un plug-in o un bean de ampliación (como ObjectTransformer, Loader, TransactionCallback, etc.) con un valor de classname que empieza con el prefijo {spring}, eXtreme Scale utiliza el resto del nombre como un nombre de bean Spring y obtenga la instancia del bean mediante la fábrica de beans de Spring.

De forma predeterminada, si no se registró ninguna fábrica de beans para un ObjectGrid determinado, intenta encontrar un archivo ObjectGridName_spring.xml. Por ejemplo, si la cuadrícula se llama "Grid", el archivo XML se denomina /Grid_spring.xml. Este archivo debe estar en la classpath o en un directorio META-INF que está en la classpath. Si no se encuentra este archivo, eXtreme Scale construye un ApplicationContext utilizando dicho archivo y construye beans desde esa fábrica de beans.

Fábrica de beans Spring personalizada

WebSphere eXtreme Scale también proporciona una API ObjectGridSpringFactory para registrar una instancia de fábrica de beans Spring para utilizar para un ObjectGrid con un nombre específico. Esta API registra una instancia de BeanFactory con eXtreme Scale utilizando el siguiente método estático:

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object
springBeanFactory)
```

Soporte de espacio de nombres

Desde la versión 2.0, Spring tiene un mecanismo para las ampliaciones basadas en esquema del formato XML de Spring básico y para definir y configurar beans. ObjectGrid utiliza esta nueva características para definir y configurar beans ObjectGrid. Con la ampliación del esquema XML de Spring, algunas de las implementaciones incorporadas de los plug-ins eXtreme Scale y algunos beans ObjectGrid están definidos previamente en el espacio de nombres "objectgrid". Al escribir los archivos de configuración de Spring, no tiene que especificar el nombre de clase completo de las implementaciones incorporadas. En lugar de esto, puede hacer referencia a los beans predefinidos.

Además, con los atributos de los beans definidos en el esquema XML, es menos probable que proporcione un nombre de atributo erróneo. La validación XML basada en el esquema XML puede capturar antes los errores de este tipo en el ciclo de desarrollo.

Estos beans definidos en las ampliaciones del esquema XML son:

- transactionManager
- register
- server
- catalog
- catalogServerProperties
- container
- JPALoader
- JPATxCallback
- JPAEntityLoader
- LRUEvictor
- LFUEvictor

- HashIndex

Estos beans están definidos en el esquema XML objectgrid.xsd. Este archivo XSD se suministra como un archivo com/ibm/ws/objectgrid/spring/namespace/objectgrid.xsd en el archivo ogspring.jar. Si desea descripciones detalladas del archivo XSD y los beans definidos en el archivo XSD, consulte la información sobre el archivo descriptor de Spring en *Guía de administración*.

Siga utilizando el ejemplo JPATxCallback de la sección anterior. En la sección anterior, se configura el bean JPATxCallback del modo siguiente:

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
  <property name="persistenceUnitName" value="employeeEMPU"/>
  <property name="JPAPropertyFactory" ref="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard">
</bean>
```

Mediante esta característica del espacio de nombres, la configuración XML de spring se puede escribir del modo siguiente:

```
<objectgrid:JPATxCallback id="jpaTxCallback" persistenceUnitName="employeeEMPU"
  jpaPropertyFactory="jpaPropFactory" />

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl"
  scope="shard">
</bean>
```

Tenga en cuenta aquí que en lugar de especificar la clase "com.ibm.websphere.objectgrid.jpa.JPATxCallback" como en el ejemplo anterior, directamente se utiliza el bean "objectgrid:JPATxCallback" definido previamente. Como puede ver, esta configuración es menos verbosa y más apta para la comprobación de errores.

Inicio del servidor de contenedor con beans de ampliación Spring

En este ejemplo, se muestra cómo iniciar un servidor ObjectGrid utilizando los beans de ampliación gestionados Spring de ObjectGrid y el soporte de espacio de nombres.

Archivo XML ObjectGrid

En primer lugar, defina un archivo XML ObjectGrid muy sencillo que contenga una "Grid" de ObjectGrid "Grid" y una correlación "Test". ObjectGrid tiene un plug-in ObjectGridEventListener llamado "partitionListener", y la correlación "Test" tiene un desalojador conectado llamado "testLRUEvictor". Tenga en cuenta que el plug-in ObjectGridEventListener y el plug-in Evictor se han configurado ambos utilizando Spring ya que sus nombres contienen "{spring}".

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="Grid">
      <bean id="ObjectGridEventListener" className="{spring}partitionListener" />
      <backingMap name="Test" pluginCollectionRef="test" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="test">
```

```

        <bean id="Evictor" className="{spring}testLRUEvictor"/>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>

```

Archivo XML de despliegue ObjectGrid

Ahora, cree un archivo XML de despliegue de ObjectGrid sencillo del modo siguiente. Divida ObjectGrid en 5 particiones, no es necesaria ninguna réplica.

```

<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
  xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="Grid">
    <mapSet name="mapSet" numInitialContainers="1" numberOfPartitions="5" minSyncReplicas="0"
      maxSyncReplicas="1" maxAsyncReplicas="0">
      <map ref="Test"/>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>

```

Archivo XML Spring de ObjectGrid

Ahora se utilizarán ambas características, los beans de ampliación gestionados Spring de ObjectGrid y el soporte de espacio de nombres, para configurar los beans ObjectGrid. El archivo XML spring se llama "Grid_spring.xml". Tenga en cuenta que se incluyen dos esquemas en el archivo XML: spring-beans-2.0.xsd se utiliza para los beans gestionados Spring y objectgrid.xsd se utiliza para los beans predefinidos en el espacio de nombres objectgrid.

```

<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
  xsi:schemaLocation="
    http://www.ibm.com/schema/objectgrid
    http://www.ibm.com/schema/objectgrid/objectgrid.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

  <objectgrid:register id="ogregister" gridname="Grid"/>

  <objectgrid:server id="server" isCatalog="true" name="server">
    <objectgrid:catalog host="localhost" port="2809"/>
  </objectgrid:server>

  <objectgrid:container id="container"
    objectgridxml="com/ibm/ws/objectgrid/test/springshard/objectgrid.xml"
    deploymentxml="com/ibm/ws/objectgrid/test/springshard/deployment.xml"
    server="server"/>

  <objectgrid:LRUEvictor id="testLRUEvictor" numberOfLRUQueues="31"/>

  <bean id="partitionListener"
    class="com.ibm.websphere.objectgrid.springshard.ShardListener" scope="shard"/>
</beans>

```

Había 6 beans definidos en este archivo XML spring:

1. *objectgrid:register*: registra la fábrica de beans predeterminada para la "Grid" de ObjectGrid.
2. *objectgrid:server*: define un servidor ObjectGrid con el nombre "server". Este servidor también proporcionará un servicio de catálogos puesto que tiene un bean *objectgrid:catalog* que está anidado ahí.

3. *objectgrid:catalog*: define un punto final de servicio de catálogos ObjectGrid, que se establece en "localhost:2809".
4. *objectgrid:container*: define un contenedor ObjectGrid con un archivo XML *objectgrid* especificado y un archivo XML de despliegue, tal como se indicó antes. La propiedad de servidor especifica en qué servidor está alojado este contenedor.
5. *objectgrid:LRUEvictor*: define un LRUEvictor con el número de colas LRU para utilizar establecido en 31.
6. *bean partitionListener*: define un plug-in ShardListener. Debe proporcionar una implementación de este plug-in, de este modo no puede utilizar los beans predefinidos. Además, este ámbito del bean está establecido en "shard", que indica que sólo hay una instancia de este ShardListener por fragmento de ObjectGrid.

Inicio del servidor

El fragmento siguiente inicia el servidor ObjectGrid, que aloja tanto el servicio de contenedor, como el servicio de catálogos. Como se puede ver, el único método que se necesita llamar para iniciar el servidor es obtener un "container" de la fábrica de beans. Así se simplifica la complejidad de la programación moviendo la mayoría de la lógica a la configuración de Spring.

```
public class ShardServer extends TestCase
{
    Container container;
    org.springframework.beans.factory.BeanFactory bf;

    public void startServer(String cep)
    {
        try
        {
            bf = new org.springframework.context.support.ClassPathXmlApplicationContext(
                "/com/ibm/ws/objectgrid/test/springshard/Grid_spring.xml", ShardServer.class);
            container = (Container)bf.getBean("container");
        }
        catch (Exception e) {
            throw new ObjectGridRuntimeException("Cannot start OG container", e);
        }
    }

    public void stopServer()
    {
        if(container != null)
            container.teardown();
    }
}
```

Capítulo 9. Programación de la seguridad

Utilice las interfaces de programación para gestionar los distintos aspectos de seguridad de un entorno eXtremeScale.

API de seguridad

WebSphere eXtreme Scale adopta una arquitectura de seguridad abierta. Proporciona una infraestructura de seguridad básica para la autenticación, autorización y la seguridad de transporte y solicita a los clientes que implementen los plug-ins para completar la infraestructura de seguridad.

La siguiente imagen muestra el flujo básico de la autenticación y autorización de cliente para un servidor eXtreme Scale.

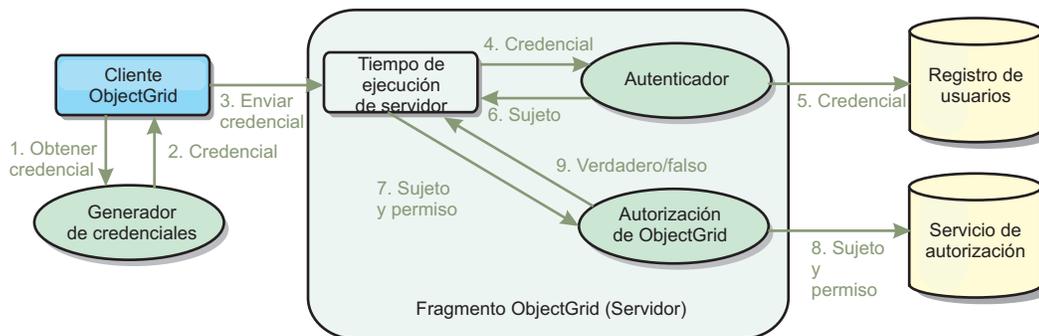


Figura 17. Flujo de autenticación y autorización de cliente

El flujo de autenticación y el flujo de autorización son los siguientes.

Flujo de autenticación

1. El flujo de autenticación se inicia con un cliente eXtreme Scale que obtiene una credencial. Esto se realiza a través del plug-in `com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator`.
2. Un objeto `CredentialGenerator` sabe como generar una credencial de cliente válida, por ejemplo, un par de ID de usuario y una contraseña, un ticket de Kerberos, etc. Esta credencial generada se vuelve a enviar al cliente.
3. Después de que el cliente recupere el objeto `Credential` utilizando el objeto `CredentialGenerator`, este objeto `Credential` se envía junto a la solicitud de eXtreme Scale al servidor eXtreme Scale.
4. El servidor eXtreme Scale autentica el objeto `Credential` antes de procesar la solicitud eXtreme Scale. A continuación, el servidor utiliza el plug-in `Authenticator` para autenticar el objeto `Credential`.
5. El plug-in `Authenticator` representa una interfaz para el registro de usuarios, por ejemplo, un servidor LDAP (Lightweight Directory Access Protocol) o un registro de usuarios del sistema operativo. El `Authenticator` consulta el registro de usuarios y toma decisiones de autenticación.
6. Si la autenticación se realiza correctamente, se devuelve un objeto `Subject` para representar este cliente.

Flujo de autorización

WebSphere eXtreme Scale adopta un mecanismo de autorización basado en los permisos y tiene distintas categorías de permiso representadas por diferentes clases de permiso. Por ejemplo, un objeto `com.ibm.websphere.objectgrid.security.MapPermission` representa los permisos para leer, escribir, insertar, invalidar y eliminar las entradas de datos en un `ObjectMap`. Puesto que WebSphere eXtreme Scale soporta la autorización JAAS (Java Authentication and Authorization Service) directamente, puede utilizar JAAS para manejar la autorización proporcionando políticas de autorización. Además, eXtreme Scale soporta las autorizaciones personalizadas. Las autorizaciones personalizadas se conectan mediante el `com.ibm.websphere.objectgrid.security.plugins.ObjectGridAuthorization`. El flujo de la autorización del cliente es la siguiente.

7. El tiempo de ejecución del servidor envía el objeto `Subject` y el permiso necesario al plug-in de autorización.
8. El plug-in de autorización consulta al servicio de autorización y toma una decisión sobre autorización. Si se otorga el permiso para este objeto `Subject`, se devuelve un valor de `true`, de lo contrario, se devuelve `false`.
9. Esta decisión de autorización, `true` o `false`, se devuelve al tiempo de ejecución del servidor.

Implementación de seguridad

Los temas de esta sección tratan cómo programar un despliegue seguro de WebSphere eXtreme Scale y cómo programar las implementaciones de plug-in. La sección se organiza basándose en las distintas características de seguridad. En cada subtema, obtendrá información sobre los plug-ins relevantes y cómo implementar los plug-ins. En la sección de autenticación, verá cómo conectarse a un entorno de despliegue seguro de WebSphere eXtreme Scale.

Autenticación de cliente: el tema de autenticación de cliente describe cómo un cliente WebSphere eXtreme Scale obtiene una credencial y cómo un servidor autentica el cliente. También tratará cómo un cliente de WebSphere eXtreme Scale se conecta a un servidor WebSphere eXtreme Scale seguro.

Autorización: el tema de autorización explica cómo utilizar `ObjectGridAuthorization` para realizar la autorización de cliente, además de la autorización JAAS.

Autenticación de cuadrícula: el tema de autenticación de cuadrícula trata cómo puede utilizar `SecureTokenManager` para transportar de forma segura los secretos de servidor.

Programación de Java Management Extensions (JMX): cuando el servidor WebSphere eXtreme Scale está protegido, el cliente JMX podría necesitar enviar una credencial JMX al servidor.

Programación de la autenticación de cliente

Para la autenticación, WebSphere eXtreme Scale proporciona un tiempo de ejecución para enviar la credencial del cliente al servidor y, a continuación, llama al plug-in autenticador para autenticar los usuarios.

WebSphere eXtreme Scale requiere que implemente los siguientes plug-ins para completar la autenticación.

- **Credencial:** una `Credential` representa una credencial de cliente como, por ejemplo, un par de ID de usuario y contraseña.

- **CredentialGenerator**: un **CredentialGenerator** representa una fábrica de credenciales para generar la credencial.
- **Authenticator**: un **Authenticator** autentica la credencial de cliente y recupera la información de cliente.

Plug-ins Credential y CredentialGenerator

Cuando un cliente de eXtreme Scale se conecta a un servidor que requiere la autenticación, es necesario que el cliente proporcione una credencial de 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 información sobre la API de credencial en la documentación de la API si desea más detalles. Esta interfaz define explícitamente 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. **WebSphere 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 es práctico si la credencial puede caducar. En este caso, se llama al método `getCredential` para renovar una credencial.

La interfaz **Credential** define explícitamente 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.

También puede utilizar el plug-in proporcionado para generar una credencial. Este plug-in se representa mediante la interfaz `com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator`, y es práctico si la credencial puede caducar. En este caso, se llama al método `getCredential` para renovar una credencial. Consulte la documentación de la API si desea más detalles.

Hay tres implementaciones predeterminadas proporcionadas para las interfaces **Credential**:

- La implementación `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential`, que contiene un par de ID de usuario y contraseña.
- La implementación `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential`, que contiene las señales de autenticación y autorización específicas de **WebSphere Application Server**. Estas señales pueden usarse para propagar los atributos de seguridad en los servidores de aplicaciones del mismo dominio de seguridad.

WebSphere 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.WebSphere`. **WebSphere eXtreme Scale** proporciona dos implementaciones incorporadas predeterminadas:

- El constructor `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator` toma un ID de usuario y una contraseña. Cuando se llama al método `getCredential`, éste devuelve un objeto `UserPasswordCredential` que contiene el ID de usuario y una contraseña.

- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator` representa un generador de credenciales (señal de seguridad) al ejecutarse en WebSphere Application Server. Cuando se llama al método `getCredential`, se recupera el sujeto (Subject) asociado a la hebra actual. A continuación, la información de seguridad de este objeto Subject se convierte en un objeto `WSTokenCredential`. Puede especificar si va a recuperar un sujeto `runAs` o un sujeto caller de la hebra mediante la constante `WSTokenCredentialGenerator.RUN_AS_SUBJECT` o `WSTokenCredentialGenerator.CALLER_SUBJECT`.

UserPasswordCredential y UserPasswordCredentialGenerator

Con finalidades de pruebas, WebSphere eXtreme Scale proporciona las siguientes implementaciones de plug-in:

1. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential`
2. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator`

La credencial de contraseña de usuario almacena un ID de usuario y una contraseña. El generador de credenciales de contraseñas de usuarios contendrá este ID de usuario y contraseña.

El siguiente código de ejemplo muestra cómo implementar estos dos plug-ins.

```

UserPasswordCredential.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,
// para uso interno del cliente o para su redistribución por su parte, integrado en una
// aplicación de ese tipo, en los productos propios del cliente.
// Material bajo licencia - Propiedad de IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import com.ibm.websphere.objectgrid.security.plugins.Credential;

/**
 * Esta clase representa una credencial que contiene un ID de usuario y una contraseña.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Credential
 * @see UserPasswordCredentialGenerator#getCredential()
 */
public class UserPasswordCredential implements Credential {

    private static final long serialVersionUID = 1409044825541007228L;

    private String ivUserName;

    private String ivPassword;

    /**
     * Crea una UserPasswordCredential con el nombre de usuario y contraseña
     * específicos.
     *
     * @param userName el nombre de usuario para esta credencial
     * @param password la contraseña para esta credencial
     *
     * @throws IllegalArgumentException si userName o password es <code>null</code>
     */
    public UserPasswordCredential(String userName, String password) {
        super();
        if (userName == null || password == null) {
            throw new IllegalArgumentException("El nombre y la contraseña no pueden ser nulos.");
        }
        this.ivUserName = userName;
        this.ivPassword = password;
    }

    /**
     * Obtiene el nombre de usuario para esta credencial.
     *
     * @return el argumento del nombre de usuario que se ha pasado al constructor

```

```

*         o <code>setUserName(String)</code>
*         de esta clase
*
* @see #setUserName(String)
*/
public String getUserName() {
    return ivUserName;
}

/**
 * Establece el nombre de usuario para esta credencial.
 *
 * @param userName el nombre de usuario a establecer.
 *
 * @throws IllegalArgumentException si userName es <code>>null</code>
 */
public void setUserName(String userName) {
    if (userName == null) {
        throw new IllegalArgumentException("El nombre de usuario no puede ser nulo.");
    }
    this.ivUserName = userName;
}

/**
 * Obtiene la contraseña para esta credencial.
 *
 * @return el argumento de contraseña que se ha pasado al constructor
 *         o el método <code>setPassword(String)</code>
 *         de esta clase
 *
 * @see #setPassword(String)
 */
public String getPassword() {
    return ivPassword;
}

/**
 * Establece la contraseña para esta credencial.
 *
 * @param password la contraseña a establecer.
 *
 * @throws IllegalArgumentException si password es <code>>null</code>
 */
public void setPassword(String password) {
    if (password == null) {
        throw new IllegalArgumentException("La contraseña no puede ser nula.");
    }
    this.ivPassword = password;
}

/**
 * Comprueba si dos objetos UserPasswordCredential son iguales.
 *
 * <p>
 * Dos objetos UserPasswordCredential son iguales si y sólo si sus nombres de usuario
 * y contraseñas son iguales.
 *
 * @param o el objeto que se está comprobando que sea igual que este objeto.
 *
 * @return <code>>true</code> si los dos objetos UserPasswordCredential son equivalentes.
 *
 * @see Credential#equals(Object)
 */
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o instanceof UserPasswordCredential) {
        UserPasswordCredential other = (UserPasswordCredential) o;
        return other.ivPassword.equals(ivPassword) && other.ivUserName.equals(ivUserName);
    }
    return false;
}

/**
 * Devuelve el código hash del objeto UserPasswordCredential.
 *
 * @return el código hash de este objeto
 *
 * @see Credential#hashCode()
 */
public int hashCode() {
    return ivUserName.hashCode() + ivPassword.hashCode();
}
}

```

UserPasswordCredentialGenerator.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,
// para uso interno del cliente o para su redistribución por su parte, integrado en una

```

```

// aplicación de ese tipo, en los productos propios del cliente.
// Material bajo licencia - Propiedad de IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.util.StringTokenizer;

import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator;

/**
 * Este generador de credenciales crea objetos <code>UserPasswordCredential</code>.
 * <p>
 * UserPasswordCredentialGenerator tiene una relación de uno a uno con
 * UserPasswordCredential porque sólo puede crear una UserPasswordCredential
 * que represente una identidad.
 *
 * @since WAS XD 6.0.1
 * @ibm-api
 *
 * @see CredentialGenerator
 * @see UserPasswordCredential
 */
public class UserPasswordCredentialGenerator implements CredentialGenerator {

    private String ivUser;

    private String ivPwd;

    /**
     * Crea un UserPasswordCredentialGenerator sin nombre de usuario o contraseña.
     *
     * @see #setProperties(String)
     */
    public UserPasswordCredentialGenerator() {
        super();
    }

    /**
     * Crea un UserPasswordCredentialGenerator con un nombre de usuario y contraseña
     * específicos
     *
     * @param user el nombre de usuario
     * @param pwd la contraseña
     */
    public UserPasswordCredentialGenerator(String user, String pwd) {
        ivUser = user;
        ivPwd = pwd;
    }

    /**
     * Crea un nuevo objeto <code>UserPasswordCredential</code> utilizando el
     * nombre de usuario y la contraseña de este objeto.
     *
     * @return una nueva instancia de <code>UserPasswordCredential</code>
     *
     * @see CredentialGenerator#getCredential()
     * @see UserPasswordCredential
     */
    public Credential getCredential() {
        return new UserPasswordCredential(ivUser, ivPwd);
    }

    /**
     * Obtiene la contraseña para este generador de credenciales.
     *
     * @return el argumento de contraseña que se ha pasado al constructor
     */
    public String getPassword() {
        return ivPwd;
    }

    /**
     * Obtiene el nombre de usuario para esta credencial.
     *
     * @return el argumento de usuario que se ha pasado al constructor
     * de esta clase
     */
    public String.getUserName() {
        return ivUser;
    }

    /**
     * Establece propiedades adicionales, en concreto, un nombre de usuario y una contraseña.
     *
     * @param properties una serie de propiedades con un nombre de usuario y
     * una contraseña separados por un blanco.
     *
     * @throws IllegalArgumentException si el formato no es válido
     */
    public void setProperties(String properties) {
        StringTokenizer token = new StringTokenizer(properties, " ");

```

```

        if (token.countTokens() != 2) {
            throw new IllegalArgumentException(
                "Las propiedades deben tener un nombre de usuario y una contraseña separados por un blanco.");
        }

        ivUser = token.nextToken();
        ivPwd = token.nextToken();
    }
    /**
     * Comprueba si dos objetos UserPasswordCredentialGenerator son iguales.
     * <p>
     * Dos objetos UserPasswordCredentialGenerator son iguales si y sólo si
     * sus nombres de usuario y contraseñas son iguales.
     *
     * @param obj el objeto que se está comprobando que sea igual que este objeto.
     *
     * @return <code>true</code> si los dos objetos UserPasswordCredentialGenerator
     * son equivalentes.
     */
    public boolean equals(Object obj) {
        if (obj == this) {
            return true;
        }

        if (obj != null && obj instanceof UserPasswordCredentialGenerator) {
            UserPasswordCredentialGenerator other = (UserPasswordCredentialGenerator) obj;

            boolean bothUserNull = false;
            boolean bothPwdNull = false;

            if (ivUser == null) {
                if (other.ivUser == null) {
                    bothUserNull = true;
                } else {
                    return false;
                }
            }

            if (ivPwd == null) {
                if (other.ivPwd == null) {
                    bothPwdNull = true;
                } else {
                    return false;
                }
            }

            return (bothUserNull || ivUser.equals(other.ivUser)) && (bothPwdNull || ivPwd.equals(other.ivPwd));
        }

        return false;
    }

    /**
     * Devuelve el código hash del objeto UserPasswordCredentialGenerator.
     *
     * @return el código hash de este objeto
     */
    public int hashCode() {
        return ivUser.hashCode() + ivPwd.hashCode();
    }
}

```

La clase `UserPasswordCredential` contiene dos atributos: nombre de usuario y contraseña. `UserPasswordCredentialGenerator` actúa como una fábrica que contiene los objetos `UserPasswordCredential`.

WSTokenCredential y WSTokenCredentialGenerator

Cuando los clientes y los servidores de WebSphere eXtreme Scale están desplegados en WebSphere Application Server, la aplicación cliente puede utilizar estas dos implementaciones incorporadas cuando se cumplen las siguientes condiciones:

1. La seguridad global de WebSphere Application Server está activa.
2. Todos los clientes y servidores WebSphere eXtreme Scale se ejecutan en WebSphere Application Server Máquinas virtuales Java.
3. Los servidores de aplicaciones están en el mismo dominio de seguridad.
4. El cliente ya ha sido autenticado en WebSphere Application Server.

En esta situación, el cliente puede utilizar la clase `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator` para generar una credencial. El servidor utiliza la clase de implementación `WSAuthenticator` para autenticar la credencial.

Este escenario se aprovecha del hecho de que el cliente eXtreme Scale ya se ha autenticado. Puesto que los servidores de aplicaciones que tienen los servidores en el mismo dominio de seguridad que los servidores de aplicaciones que alojan los clientes, las señales de seguridad se pueden propagar del cliente al servidor, de forma que no es necesario que se vuelva a autenticar el mismo registro de usuarios.

Nota: No dé por sentado que un `CredentialGenerator` siempre genera la misma credencial. Para una credencial que puede caducar y se puede renovar, `CredentialGenerator` debe poder generar la última credencial válida para garantizar el éxito de la autenticación. Un ejemplo es utilizar el ticket de Kerberos como un objeto `Credential`. Cuando se renueva el ticket Kerberos, `CredentialGenerator` deberá recuperar el ticket renovado cuando se llame a `CredentialGenerator.getCredential`.

Plug-in de autenticador

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 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 información sobre el Autenticador en la documentación de la API si desea más detalles.

El plug-in `Authenticator` es donde el tiempo de ejecución de eXtreme Scale autentica el objeto `Credential` del registro de usuarios del cliente, por ejemplo, un servidor LDAP (Lightweight Directory Access Protocol).

WebSphere eXtreme Scale no proporciona una configuración de registro de usuarios disponible inmediatamente. La configuración y la gestión del registro de usuarios se deja fuera de WebSphere eXtreme Scale con fines de simplicidad y flexibilidad. Este plug-in implementa la conexión y la autenticación al registro de usuarios. Por ejemplo, una implementación de autenticador extrae el ID de usuario y la contraseña de la credencial, los utiliza para conectarse a un servidor LDAP y validarlo, y crea un objeto `Subject` como resultado de la autenticación. La implementación podría utilizar los módulos de inicio de sesión JAAS. Como resultado de la autenticación, se devuelve un objeto `Subject`.

Tenga en cuenta que este método crea dos excepciones: `InvalidCredentialException` y `ExpiredCredentialException`. La excepción `InvalidCredentialException` indica que

la credencial no es válida. La excepción `ExpiredCredentialException` indica que la credencial ha caducado. Si se genera una de estas dos excepciones del método de autenticación, las excepciones se envían de vuelta al cliente. Sin embargo, el tiempo de ejecución del cliente maneja estas dos excepciones de forma distinta:

- Si el error es una excepción `InvalidCredentialException`, el tiempo de ejecución del cliente visualiza esta excepción. La aplicación debe tratar la excepción. Puede corregir `CredentialGenerator`, por ejemplo, y volver a repetir la operación.
- Si el error es una excepción `ExpiredCredentialException` y el recuento de reintentos no es 0, el tiempo de ejecución del cliente vuelve a llamar la método `CredentialGenerator.getCredential` y envía el nuevo objeto `Credential` al servidor. Si la autenticación de la nueva credencial es correcta, el servidor procesa la solicitud. Si, en cambio, la autenticación falla, el excepción vuelve a enviarse al cliente. Si el número de intentos de autenticación alcanza el valor soportado y el cliente sigue obteniendo una excepción `ExpiredCredentialException`, se genera la excepción `ExpiredCredentialException`. La aplicación debe tratar el error.

La interfaz `Authenticator` proporciona una gran flexibilidad. Puede implementar la interfaz `Authenticator` de su propia manera específica. Por ejemplo, puede implementar esta interfaz para soportar dos registros de usuarios distintos.

WebSphere eXtreme Scale proporciona implementaciones de plug-in de autenticador de ejemplo. Excepto para el plug-in de autenticador de WebSphere Application Server, las otras implementaciones sólo son ejemplos con finalidades de prueba.

KeyStoreLoginAuthenticator

Este ejemplo utiliza una implementación incorporada de eXtreme Scale: `KeyStoreLoginAuthenticator`, que tiene finalidades de pruebas y de ejemplo (un almacén de claves es un registro de usuarios sencillo y no se debe utilizar para un entorno de producción). De nuevo, la clase se visualiza para demostrar de forma adicional cómo implementar un autenticador.

```
KeyStoreLoginAuthenticator.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,
// para uso interno del cliente o para su redistribución por su parte, integrado en una
// aplicación de ese tipo, en los productos propios del cliente.
// Material bajo licencia - Propiedad de IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007

package com.ibm.websphere.objectgrid.security.plugins.builtins;

import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

import com.ibm.websphere.objectgrid.security.plugins.Authenticator;
import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.ExpiredCredentialException;
import com.ibm.websphere.objectgrid.security.plugins.InvalidCredentialException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.security.auth.callback.UserPasswordCallbackHandlerImpl;

/**
 * Esta clase es una implementación de la interfaz Authenticator
 * cuando un nombre de usuario y una contraseña se utilizan como credencial.
 * <p>
 * Cuando se utiliza la autenticación de ID de usuario y contraseña, la credencial pasada al
 * método authenticate(Credential) es un objeto UserPasswordCredential.
 * <p>
 * Esta implementación utilizará un KeyStoreLoginModule para autenticar
 * al usuario en 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. Consulte la clase
 * KeyStoreLoginModule para obtener más detalles sobre cómo configurar
 * el archivo de configuración de inicio de sesión JAAS.
 * <p>
```

```

* Esta clase sólo sirve de ejemplo y de comprobación rápida. Los usuarios deben
* crear su propia implementación de Authenticator que puede adaptarse mejor al
* entorno.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see Authenticator
* @see KeyStoreLoginModule
* @see UserPasswordCredential
*/
public class KeyStoreLoginAuthenticator implements Authenticator {

    /**
     * Crea un nuevo KeyStoreLoginAuthenticator.
     */
    public KeyStoreLoginAuthenticator() {
        super();
    }

    /**
     * Autentica un <code>UserPasswordCredential</code>.
     * <p>
     * Utiliza el nombre de usuario y la contraseña de la UserPasswordCredential especificada
     * para iniciar la sesión en el KeyStoreLoginModule llamado "KeyStoreLogin".
     *
     * @throws InvalidCredentialException si la credencial no es una
     *         UserPasswordCredential o se produce un error durante el proceso
     *         de la UserPasswordCredential proporcionada
     *
     * @throws ExpiredCredentialException si la credencial ha caducado. Esta excepción
     *         no la utiliza esta implementación
     *
     * @see Authenticator#authenticate(Credential)
     * @see KeyStoreLoginModule
     */
    public Subject authenticate(Credential credential) throws InvalidCredentialException,
        ExpiredCredentialException {
        if (credential == null) {
            throw new InvalidCredentialException("Supplied credential is null");
        }

        if (! (credential instanceof UserPasswordCredential) ) {
            throw new InvalidCredentialException("Supplied credential is not a UserPasswordCredential");
        }

        UserPasswordCredential cred = (UserPasswordCredential) credential;
        LoginContext lc = null;
        try {
            lc = new LoginContext("KeyStoreLogin",
                new UserPasswordCallbackHandlerImpl(cred.getUserName(), cred.getPassword().toCharArray()));

            lc.login();

            Subject subject = lc.getSubject();

            return subject;
        }
        catch (LoginException le) {
            throw new InvalidCredentialException(le);
        }
        catch (IllegalArgumentException ile) {
            throw new InvalidCredentialException(ile);
        }
    }
}

KeyStoreLoginModule.java
// Este programa de ejemplo se proporciona TAL CUAL y se puede utilizar, ejecutar, copiar y modificar
// sin que el cliente tenga que pagar derechos
// (a) para su propia formación,
// (b) para desarrollar aplicaciones diseñadas para ejecutarse con un producto IBM WebSphere,
// para uso interno del cliente o para su redistribución por su parte, integrado en una
// aplicación de ese tipo, en los productos propios del cliente.
// Material bajo licencia - Propiedad de IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.io.File;
import java.io.FileInputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.UnrecoverableKeyException;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Map;

```

```

import java.util.Set;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import javax.security.auth.x500.X500Principal;
import javax.security.auth.x500.X500PrivateCredential;

import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.util.ObjectGridUtil;

/**
 * Un KeyStoreLoginModule es un módulo de inicio de sesión de autenticación de almacén de claves
 * basado en la autenticación JAAS.
 * <p>
 * Una configuración de inicio de sesión debe proporcionar una opción
 * "<code>keyStoreFile</code>" para indicar donde está ubicado el archivo de almacén
 * de claves. Si el valor <code>keyStoreFile</code> contiene una propiedad del sistema
 * en el formato <code>${system.property}</code>, se expandirá hasta el valor de
 * la propiedad del sistema.
 * <p>
 * Si no se proporciona una opción "<code>keyStoreFile</code>", el nombre de archivo
 * del almacén de claves predeterminado es <code>${java.home}${/}.keystore</code>.
 * <p>
 * A continuación se muestra un ejemplo de configuración del módulo Login:
 * <pre><code>
 *   KeyStoreLogin {
 *     com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule required
 *     keyStoreFile="${user.dir}${/}security${/}.keystore";
 *   };
 * </code></pre>
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see LoginModule
 */
public class KeyStoreLoginModule implements LoginModule {

    private static final String CLASS_NAME = KeyStoreLoginModule.class.getName();

    /**
     * Nombre de propiedad del archivo de almacén de claves
     */
    public static final String KEY_STORE_FILE_PROPERTY_NAME = "keyStoreFile";

    /**
     * Tipo del almacén de claves. Sólo se da soporte a JKS
     */
    public static final String KEYSTORE_TYPE = "JKS";

    /**
     * El nombre de archivo de almacén de claves predeterminado
     */
    public static final String DEFAULT_KEY_STORE_FILE = "${java.home}${/}.keystore";

    private CallbackHandler handler;

    private Subject subject;

    private boolean debug = false;

    private Set principals = new HashSet();

    private Set publicCreds = new HashSet();

    private Set privateCreds = new HashSet();

    protected KeyStore keyStore;

    /**
     * Crea un nuevo KeyStoreLoginModule.
     */
    public KeyStoreLoginModule() {
    }

    /**
     * Inicializa el módulo de inicio de sesión.
     *
     * @see LoginModule#initialize(Subject, CallbackHandler, Map, Map)
     */
    public void initialize(Subject sub, CallbackHandler callbackHandler,
        Map mapSharedState, Map mapOptions) {

        // inicializar todas las opciones configuradas

```

```

debug = "true".equalsIgnoreCase((String) mapOptions.get("debug"));

if (sub == null)
    throw new IllegalArgumentException("Subject is not specified");

if (callbackHandler == null)
    throw new IllegalArgumentException(
        "CallbackHandler no especificado");

// Obtener la vía de acceso del almacén de claves
String sKeyStorePath = (String) mapOptions
    .get(KEY_STORE_FILE_PROPERTY_NAME);

// Si no hay ninguna vía de acceso de almacén de claves, el valor predeterminado es
// el archivo .keystore en el directorio inicial de java
if (sKeyStorePath == null) {
    sKeyStorePath = DEFAULT_KEY_STORE_FILE;
}

// Sustituir la variable de entorno del sistema
sKeyStorePath = ObjectGridUtil.replaceVar(sKeyStorePath);

File fileKeyStore = new File(sKeyStorePath);

try {
    KeyStore store = KeyStore.getInstance("JKS");
    store.load(new FileInputStream(fileKeyStore), null);

    // Guardar el almacén de claves
    keyStore = store;

    if (debug) {
        System.out.println("Inicializar [KeyStoreLoginModule]: almacén de claves cargado satisfactoriamente");
    }
}
catch (Exception e) {
    ObjectGridRuntimeException re = new ObjectGridRuntimeException(
        "Failed to load keystore: " + fileKeyStore.getAbsolutePath());
    re.initCause(e);
    if (debug) {
        System.out.println("[KeyStoreLoginModule] initialize: Key store loading failed with exception "
            + e.getMessage());
    }
}

this.subject = sub;
this.handler = callbackHandler;
}

/**
 * Autentica un usuario basándose en el archivo de almacén de claves.
 *
 * @see LoginModule#login()
 */
public boolean login() throws LoginException {

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: entry");
    }

    String name = null;
    char pwd[] = null;

    if (keyStore == null || subject == null || handler == null) {
        throw new LoginException("Module initialization failed");
    }

    NameCallback nameCallback = new NameCallback("Username:");
    PasswordCallback pwdCallback = new PasswordCallback("Password:", false);

    try {
        handler.handle(new Callback[] { nameCallback, pwdCallback });
    }
    catch (Exception e) {
        throw new LoginException("Callback failed: " + e);
    }

    name = nameCallback.getName();
    char[] tempPwd = pwdCallback.getPassword();

    if (tempPwd == null) {
        // tratar una contraseña NULL como una contraseña vacía
        tempPwd = new char[0];
    }
    pwd = new char[tempPwd.length];
    System.arraycopy(tempPwd, 0, pwd, 0, tempPwd.length);

    pwdCallback.clearPassword();

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: "

```

```

        + "user entered user name: " + name);
    }

    // Validar el nombre de usuario y la contraseña
    try {
        validate(name, pwd);
    }
    catch (SecurityException se) {
        principals.clear();
        publicCreds.clear();
        privateCreds.clear();
        LoginException le = new LoginException(
            "Exception encountered during login");
        le.initCause(se);

        throw le;
    }

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: exit");
    }
    return true;
}

/**
 * Indica si el usuario se acepta.
 * <p>
 * Este método sólo se invoca si el usuario es autenticado por todos los módulos del archivo
 * de configuración de inicio de sesión. Los objetos principales se añadirán al asunto
 * almacenado.
 *
 * @return false si por alguna razón los principales no se pueden añadir; de lo contrario,
 *         true
 *
 * @exception LoginException
 *         Se genera una LoginException si el asunto es de sólo lectura o si se
 *         encuentran excepciones no recuperables.
 *
 * @see LoginModule#commit()
 */
public boolean commit() throws LoginException {
    if (debug) {
        System.out.println("[KeyStoreLoginModule] commit: entry");
    }

    if (principals.isEmpty()) {
        throw new IllegalStateException("Commit is called out of sequence");
    }

    if (subject.isReadOnly()) {
        throw new LoginException("Subject is ReadOnly");
    }

    subject.getPrincipals().addAll(principals);
    subject.getPublicCredentials().addAll(publicCreds);
    subject.getPrivateCredentials().addAll(privateCreds);

    principals.clear();
    publicCreds.clear();
    privateCreds.clear();

    if (debug) {
        System.out.println("[KeyStoreLoginModule] commit: exit");
    }
    return true;
}

/**
 * Indica que el usuario no se acepta
 *
 * @see LoginModule#abort()
 */
public boolean abort() throws LoginException {
    boolean b = logout();
    return b;
}

/**
 * Cierra la sesión de usuario. Borrar todas las correlaciones.
 *
 * @see LoginModule#logout()
 */
public boolean logout() throws LoginException {

    // Borrar las variables de instancia
    principals.clear();
    publicCreds.clear();
    privateCreds.clear();

    // borrar correlaciones en el asunto

```

```

    if (!subject.isReadOnly()) {
        if (subject.getPrincipals() != null) {
            subject.getPrincipals().clear();
        }

        if (subject.getPublicCredentials() != null) {
            subject.getPublicCredentials().clear();
        }

        if (subject.getPrivateCredentials() != null) {
            subject.getPrivateCredentials().clear();
        }
    }
    return true;
}

/**
 * Valida el nombre de usuario y la contraseña basándose en el almacén de claves.
 *
 * @param userName nombre de usuario
 * @param password contraseña
 * @throws SecurityException si se encuentran excepciones
 */
private void validate(String userName, char password[])
    throws SecurityException {
    PrivateKey privateKey = null;

    // Obtener la clave privada del almacén de claves
    try {
        privateKey = (PrivateKey) keyStore.getKey(userName, password);
    }
    catch (NoSuchAlgorithmException nsae) {
        SecurityException se = new SecurityException();
        se.initCause(nsae);
        throw se;
    }
    catch (KeyStoreException kse) {
        SecurityException se = new SecurityException();
        se.initCause(kse);
        throw se;
    }
    catch (UnrecoverableKeyException uke) {
        SecurityException se = new SecurityException();
        se.initCause(uke);
        throw se;
    }

    if (privateKey == null) {
        throw new SecurityException("Invalid name: " + userName);
    }

    // Comprobar certificados
    Certificate certs[] = null;
    try {
        certs = keyStore.getCertificateChain(userName);
    }
    catch (KeyStoreException kse) {
        SecurityException se = new SecurityException();
        se.initCause(kse);
        throw se;
    }

    if (debug) {
        System.out.println(" Print out the certificates:");
        for (int i = 0; i < certs.length; i++) {
            System.out.println(" certificate " + i);
            System.out.println(" " + certs[i]);
        }
    }

    if (certs != null && certs.length > 0) {
        // Si el primer certificado es un X509Certificate
        if (certs[0] instanceof X509Certificate) {
            try {
                // Obtener el primer certificado que representa el usuario
                X509Certificate certX509 = (X509Certificate) certs[0];

                // Crear un principal
                X500Principal principal = new X500Principal(certX509
                    .getIssuerDN()
                    .getName());
                principals.add(principal);

                if (debug) {
                    System.out.println(" Principal added: " + principal);
                }
            }
            // Crear un objeto de vía de acceso de certificación y añadirlo al
            // conjunto de credenciales públicas
            CertificateFactory factory = CertificateFactory

```



```

/**
 * Autentica el usuario en el directorio LDAP.
 * @establecer el parámetro user para el ID de usuario, por ej.,
 * uid=xxxxxx,c=us,ou=bluepages,o=ibm.com
 * @param pwd la contraseña
 *
 * @emite NamingException
 */
public String[] authenticate(String user, String pwd)
throws NamingException {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, factoryClass);
    env.put(Context.PROVIDER_URL, providerURL);
    env.put(Context.SECURITY_PRINCIPAL, user);
    env.put(Context.SECURITY_CREDENTIALS, pwd);
    env.put(Context.SECURITY_AUTHENTICATION, "simple");

    InitialContext initialContext = new InitialContext(env);

    // Buscar el usuario
    DirContext dirCtx = (DirContext) initialContext.lookup(user);

    String uid = null;
    int iComma = user.indexOf(",");
    int iEqual = user.indexOf("=");
    if (iComma > 0 && iComma > 0) {
        uid = user.substring(iEqual + 1, iComma);
    }
    else {
        uid = user;
    }

    Attributes attributes = dirCtx.getAttributes("");

    // Comprobar el UID
    String thisUID = (String) (attributes.get(UID).get());

    String thisDept = (String) (attributes.get(HR_DEPT).get());

    if (thisUID.equals(uid)) {
        return new String[] { thisUID, thisDept };
    }
    else {
        return null;
    }
}

```

Si la autenticación es correcta, el ID de usuario y la contraseña se consideran válidos. El módulo de inicio de sesión obtiene la información de ID y de departamento a partir de este método `authenticate`. El módulo de inicio de sesión crea dos principales: `SimpleUserPrincipal` y `SimpleDeptPrincipal`. Puede usar el sujeto autenticado para autorización de grupos (en este caso, el departamento es un grupo) y para autorización individual.

El ejemplo siguiente muestra una configuración del módulo de inicio de sesión que se utiliza para iniciar sesión en el servidor LDAP:

```

LDAPLogin { com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule required
    providerURL="ldap://directory.acme.com:389/"
    factoryClass="com.sun.jndi.ldap.LdapCtxFactory";
};

```

En la configuración anterior, el servidor LDAP apunta al `ldap://directory.acme.com:389/server`. Cambie este valor por el de su servidor LDAP. Este módulo de inicio de sesión utiliza el ID y la contraseña proporcionados para conectarse al servidor LDAP. Esta implementación es sólo para realizar pruebas.

Uso del plug-in de autenticador de WebSphere Application Server

Además, eXtreme Scale proporciona la implementación incorporada `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator` para utilizar la infraestructura de seguridad de WebSphere Application Server. Esta implementación incorporada se puede utilizar cuando las siguientes condiciones son verdaderas.

1. La seguridad global de WebSphere Application Server está activa.
2. Todos los clientes y servidores de eXtreme Scale se inician en las JVM de WebSphere Application Server.
3. Estos servidores de aplicaciones están en el mismo dominio de seguridad.
4. El cliente eXtreme Scale ya ha sido autenticado en WebSphere Application Server.

El cliente puede utilizar la clase `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator` para generar una credencial. El servidor utiliza esta clase de implementación `Authenticator` para autenticar la credencial. Si la señal se autentica correctamente, se devuelve un objeto `Subject`.

Este escenario saca partido del hecho que el cliente ya ha sido autenticado. Puesto que los servidores de aplicaciones que tienen los servidores en el mismo dominio de seguridad que los servidores de aplicaciones que alojan los clientes, las señales de seguridad se pueden propagar del cliente al servidor, de forma que no es necesario que se vuelva a autenticar el mismo registro de usuarios.

Uso del plug-in de autenticador de Tivoli Access Manager

Tivoli Access Manager se utiliza ampliamente como un servidor de seguridad. También puede implementar el autenticador utilizando los módulos de inicio de sesión proporcionados de Tivoli Access Manager.

Para autenticar un usuario para Tivoli Access Manager, aplique el módulo de inicio de sesión `com.tivoli.mts.PDLoginModule`, que requiere que la aplicación que llama proporcione la siguiente información:

1. Un nombre principal, especificado como un nombre abreviado o un nombre X.500 (DN)
2. Una contraseña

El módulo de inicio de sesión autentica el principal y devuelve la credencial de Tivoli Access Manager. El módulo de inicio de sesión espera que la aplicación que llama proporcione la información siguiente:

1. El nombre de usuario, a través de un objeto `javax.security.auth.callback.NameCallback`.
2. La contraseña, a través de un objeto `javax.security.auth.callback.PasswordCallback`.

Cuando la credencial de Tivoli Access Manager se recupera correctamente, el módulo `JAAS LoginModule` crea un objeto `Subject` y `PDPrincipal`. No se proporciona ningún objeto incorporado para la autenticación de Tivoli Access Manager, porque sólo se proporciona con el módulo `PDLoginModule`. Consulte la publicación *IBM Tivoli Access Manager Authorization Java Classes Developer Reference* si desea más detalles.

Conexión a WebSphere eXtreme Scale de forma segura

Para conectar de forma segura un cliente de eXtreme Scale con un servidor, puede utilizar cualquier método connect en la interfaz ObjectGridManager que toma un objeto ClientSecurityConfiguration. A continuación, aparece un breve ejemplo:

```
public ClientClusterContext connect(String catalogServerAddresses,  
    ClientSecurityConfiguration securityProps,  
    URL overRideObjectGridXml) lanza ConnectException;
```

Este método toma un parámetro del tipo ClientSecurityConfiguration, que es una interfaz que representa una configuración de seguridad de cliente. Puede utilizar la API pública

com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory para crear una instancia con valores predeterminados, o puede crear una instancia pasando el archivo de propiedades de cliente de WebSphere eXtreme Scale. Este archivo contiene las siguientes propiedades relacionadas con la autenticación. El valor marcado con un signo más (+) es el valor predeterminado.

- securityEnabled (true, false+): esta propiedad indica si la seguridad está habilitada. Cuando un cliente se conecta a un servidor, el valor securityEnabled en el lado del cliente y del servidor deben ser ambos true o false. Por ejemplo, si la seguridad del servidor conectado está habilitada, el cliente debe tener esta propiedad establecida en true para poder conectarse al servidor.
- authenticationRetryCount (un valor entero, 0+): esta propiedad determina cuántos reintentos se realizan para el inicio de sesión cuando caduca una credencial. Si el valor es 0, no se realiza ningún reintento. El reintento de autenticación sólo se aplica en el caso de que la credencial haya caducado. Si la credencial no es válida, no se produce ningún reintento. Su aplicación es responsable de reintentar la operación.

Después de crear un objeto

com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration, establezca el objeto credentialGenerator en el cliente mediante el método siguiente:

```
/**  
 * Establezca el objeto {@link CredentialGenerator} para este cliente.  
 * @establezca el parámetro generator para el objeto CredentialGenerator  
 asociado con este cliente  
 */  
void setCredentialGenerator(CredentialGenerator generator);
```

También, puede establecer el objeto CredentialGenerator en el archivo de propiedades de cliente de WebSphere eXtreme Scale, del modo siguiente:

- credentialGeneratorClass: nombre de la implementación de clase del objeto CredentialGenerator. Debe tener un constructor predeterminado.
- credentialGeneratorProps: propiedades de la clase CredentialGenerator. Si el valor es nulo, se establece en el objeto CredentialGenerator construido mediante el método setProperties(String).

En el ejemplo siguiente se crea una instancia de ClientSecurityConfiguration, que se utiliza para conectar con el servidor.

```
/**  
 * Obtiene un ClientClusterContext seguro  
 * @devuelve un objeto ClientClusterContext seguro  
 */  
protected ClientClusterContext connect() throws ConnectException {  
    ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory  
        .getClientSecurityConfiguration("/properties/security.ogclient.props");
```

```

UserPasswordCredentialGenerator gen= new
UserPasswordCredentialGenerator("manager", "manager1");

csConfig.setCredentialGenerator(gen);

return objectGridManager.connect(csConfig, null);
}

```

Cuando se llama a la conexión, el cliente de WebSphere eXtreme Scale llama al método `CredentialGenerator.getCredential` para obtener la credencial del cliente. Esta credencial de autenticación se envía al servidor con la solicitud de conexión.

Uso de una instancia `CredentialGenerator` diferente por sesión

En algunos casos, un cliente de WebSphere eXtreme Scale representa sólo una identidad de cliente, pero en otros, podría representar varias identidades. Aquí, aparece un escenario del último caso: un cliente de WebSphere eXtreme Scale se crea y comparte en un servidor web. Todos los servlets de este servidor web utilizan este cliente de WebSphere eXtreme Scale. Puesto que cada uno de los servlets representa un cliente web diferente, utilice distintas credenciales al enviar solicitudes a servidores WebSphere eXtreme Scale.

WebSphere eXtreme Scale puede cambiar la credencial en el nivel de la sesión. Cada sesión puede utilizar un objeto `CredentialGenerator` diferente. Por lo tanto, los escenarios anteriores se pueden implementar dejando al servlet obtener una sesión con un objeto `CredentialGenerator` distinto. El ejemplo siguiente ilustra el método `ObjectGrid.getSession(CredentialGenerator)` en la interfaz `ObjectGridManager`.

```

/**
 * Obtener una sesión utilizando <code>CredentialGenerator</code>.
 * <p>
 * Este método sólo puede ser llamado por el cliente ObjectGrid en un entorno de
 * servidor cliente de ObjectGrid. Si se utiliza ObjectGrid en un modelo local, es decir,
 * dentro de la misma JVM no existe ningún cliente ni servidor, se debe utilizar el
 * plug-in <code>getSession(Subject)</code>
 * o <code>SubjectSource</code> para proteger el ObjectGrid.
 *
 * <p>Si el método <code>initialize()</code> no ha sido invocado antes de la
 * primera invocación de <code>getSession()</code>, se producirá una inicialización
 * implícita. Esto garantiza que toda la configuración se completa
 * antes de que sea necesario cualquier uso del tiempo de ejecución.</p>
 *
 * @param credGen <code>CredentialGenerator</code> para generar una credencial
 * para la sesión devuelta.
 *
 * @return Una instancia de <code>Session</code>
 *
 * @throws ObjectGridException si se produce un error durante el proceso
 * @throws TransactionCallbackException si <code>TransactionCallback</code>
 * lanza una excepción
 * @emite la excepción IllegalStateException si se llama a este método después de
 * que se llame al método <code>destroy()</code>.
 *
 * @see #destroy()
 * @see #initialize()
 * @see CredentialGenerator
 * @see Session
 * @since WAS XD 6.0.1
 */
Session getSession(CredentialGenerator credGen) throws
ObjectGridException, TransactionCallbackException;

```

A continuación se muestra un ejemplo:

```

ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();

CredentialGenerator credGenManager = new UserPasswordCredentialGenerator("manager", "xxxxxx");
CredentialGenerator credGenEmployee = new UserPasswordCredentialGenerator("employee", "xxxxxx");

ObjectGrid og = ogManager.getObjectGrid(ctx, "accounting");

// Obtiene una sesión con CredentialGenerator;
Session session = og.getSession(credGenManager );

```

```

// Obtiene la correlación de empleados
ObjectMap om = session.getMap("employee");

// inicia una transacción.
session.begin();

Object rec1 = map.get("xxxxx");

session.commit();

// Obtiene otra sesión con otro CredentialGenerator;
session = og.getSession(credGenEmployee );

// Obtiene la correlación de empleados
om = session.getMap("employee");

// inicia una transacción.
session.begin();

Object rec2 = map.get("xxxxx");

session.commit();

```

Si utiliza el método `ObjectGrid.getSession` para obtener un objeto `Session`, la sesión utiliza el objeto `CredentialGenerator` establecido en el objeto `ClientConfigurationSecurity`. El método `ObjectGrid.getSession(CredentialGenerator)` sustituye al objeto `CredentialGenerator` establecido en el objeto `ClientSecurityConfiguration`.

Si puede volver a utilizar el objeto `Session`, se favorece el rendimiento. Sin embargo, llamar al método `ObjectGrid.getSession(CredentialGenerator)` no es muy caro. La principal sobrecarga es el aumento del tiempo de recogida de basura del objeto. Asegúrese de que libera las referencias después de terminar con los objetos `Session`. Por lo general, si el objeto `Session` puede compartir la identidad, intente reutilizarlo. En caso contrario, utilice el método `ObjectGrid.getSession(CredentialGenerator)`.

Programación de autorización de cliente

WebSphere eXtreme Scale soporta la autorización JAAS (Java Authentication and Authorization Service) directamente y también soporta la autorización personalizada utilizando la interfaz `ObjectGridAuthorization`.

El plug-in `ObjectGridAuthorization` se utiliza para autorizar los accesos `ObjectGrid`, `ObjectMap` y `JavaMap` a los principales representados por un objeto `Subject` de una forma personalizada. Una implementación típica de este plug-in es recuperar los principales del objeto `Subject` y después comprobar si se han concedido los permisos especificados a los principales.

Un permiso pasado en el método `checkPermission(Subject, Permission)` puede ser uno de los permisos siguientes:

- `MapPermission`
- `ObjectGridPermission`
- `ServerMapPermission`
- `AgentPermission`

Consulte la documentación de la API `ObjectGridAuthorization` para obtener más información.

MapPermission

La clase pública `com.ibm.websphere.objectgrid.security.MapPermission` representa los permisos de los recursos de `ObjectGrid`, 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**: permiso para leer los datos de la correlación. La constante entera se define como `MapPermission.READ`.
- **grabar**: permiso para actualizar los datos de la correlación. La constante entera se define como `MapPermission.WRITE`.
- **insertar**: permiso para insertar los datos en la correlación. La constante entera se define como `MapPermission.INSERT`.
- **eliminar**: permiso para eliminar los datos de la correlación. La constante entera se define como `MapPermission.REMOVE`.
- **invalidar**: permiso para invalidar los datos de la correlación. La constante entera se define como `MapPermission.INVALIDATE`.
- **todo**: todos los permisos anteriores: leer, grabar, insertar, eliminar e invalidar. La constante entera se define como `MapPermission.ALL`.

Consulte la documentación de la API `MapPermission` para obtener más información.

Puede construir un objeto `MapPermission`; para ello, pase el nombre de la correlación `ObjectGrid` totalmente calificado (con el formato `[nombre_ObjectGrid].[nombre_ObjectMap]`) y la serie de permiso o valor entero. Una serie de permiso puede ser una serie delimitada por comas de las series de permiso anteriores como leer, insertar, o todos ellos. Un valor entero de permiso puede ser cualquier constante entera de los permisos mencionados anteriormente o un valor matemático de varias constantes enteras de permisos, como `MapPermission.READ|MapPermission.WRITE`.

La autorización se produce cuando se llama el método `ObjectMap` o `JavaMap`. El tiempo de ejecución de eXtreme Scale comprueba los distintos permisos para los métodos diferentes. Si los permisos requeridos no se conceden al cliente, se produce una excepción `AccessControlException`.

Tabla 13. Lista de métodos y `MapPermission` requeridos

Permiso	<code>ObjectMap/JavaMap</code>
leer	<code>boolean containsKey(Object)</code>
	<code>boolean equals(Object)</code>
	<code>Object get(Object)</code>
	<code>Object get(Object, Serializable)</code>
	<code>List getAll(List)</code>
	<code>List getAll(List keyList, Serializable)</code>
	<code>List getAllForUpdate(List)</code>
	<code>List getAllForUpdate(List, Serializable)</code>
	<code>Object getForUpdate(Object)</code>
	<code>Object getForUpdate(Object, Serializable)</code>
	<code>public Object getNextKey(long)</code>

Tabla 13. Lista de métodos y MapPermission requeridos (continuación)

Permiso	ObjectMap/JavaMap
grabar	Object put(Object key, Object value)
	void put(Object, Object, Serializable)
	void putAll(Map)
	void putAll(Map, Serializable)
	void update(Object, Object)
	void update(Object, Object, Serializable)
insertar	public void insert (Object, Object)
	void insert(Object, Object, Serializable)
eliminar	Object remove (Object)
	void removeAll(Collection)
	void clear()
invalidar	public void invalidate (Object, boolean)
	void invalidateAll(Collection, boolean)
	void invalidateUsingKeyword(Serializable)
	int setTimeToLive(int)

La autorización se basa únicamente en el método utilizado y no en lo que el método hace. Por ejemplo, un método put puede insertar o actualizar un registro en función de si el registro existe. No obstante, los casos de inserción o actualización no se distinguen.

Puede conseguirse un tipo de operación mediante la combinación de otros tipos. Por ejemplo, una actualización puede conseguirse mediante una operación de eliminación primero, y después una inserción. Tenga en cuenta estas combinaciones al diseñar las políticas de autorización.

ObjectGridPermission

com.ibm.websphere.objectgrid.security.ObjectGridPermission representa permisos para ObjectGrid:

- Consulta: permiso para crear una consulta de objeto o una consulta de entidad. La constante entera se define como ObjectGridPermission.QUERY.
- Correlación dinámica: permiso para crear una correlación dinámica basada en la plantilla de correlación. La constante entera se define como ObjectGridPermission.DYNAMIC_MAP.

Consulte la documentación de la API ObjectGridPermission para obtener más información.

En la siguiente tabla se resumen los métodos y los ObjectGridPermission requeridos:

Tabla 14. Lista de métodos y ObjectGridPermission requeridos

Acción de permiso	Métodos
consulta	com.ibm.websphere.objectgrid.Session.createObjectQuery(String)
consulta	com.ibm.websphere.objectgrid.em.EntityManager.createQuery(String)
dynamicmap	com.ibm.websphere.objectgrid.Session.getMap(String)

ServerMapPermission

ServerMapPermission representa permisos para ObjectMap alojado en un servidor. El nombre del permiso es el nombre completo del nombre de la correlación ObjectGrid. Tiene las acciones siguientes:

- replicar: permiso para replicar una correlación del servidor en una memoria caché cercana.
- dynamicIndex: permiso para que un cliente pueda crear o eliminar un índice dinámico en el servidor.

Consulte la documentación de la API ServerMapPermission para obtener más información. Los métodos detallados, que requieren diferentes ServerMapPermission, se listan en la siguiente tabla:

Tabla 15. Permisos para un ObjectMap alojado en un servidor

Acción de permiso	Métodos
replicar	<code>com.ibm.websphere.objectgrid.ClientReplicableMap.enableClientReplication(Mode, int[], ReplicationMapListener)</code>
dynamicIndex	<code>com.ibm.websphere.objectgrid.BackingMap.createDynamicIndex(String, boolean, String, DynamicIndexCallback)</code>
dynamicIndex	<code>com.ibm.websphere.objectgrid.BackingMap.removeDynamicIndex(String)</code>

AgentPermission

Un AgentPermission representa permisos para los agentes de datagrid. El nombre del permiso es el nombre completo de la correlación ObjectGrid, y la acción es una serie delimitada por comas de los nombres de clase o nombres de paquete de la implementación del agente.

Consulte la documentación de la API AgentPermission si desea más información.

Los siguientes métodos de la clase

`com.ibm.websphere.objectgrid.datagrid.AgentManager` requieren AgentPermission.

`com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent, Collection)`

`com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent)`

`com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)`

`com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)`

Mecanismos de autorización

WebSphere eXtreme Scale soporta dos tipos de mecanismos de autorización: la autorización JAAS (Java Authentication and Authorization Service) y la autorización personalizada. Estos mecanismos se aplican a todas las autorizaciones. La autorización JAAS aumenta las políticas de seguridad Java con controles de acceso centrados en el usuario. Los permisos se conceden no sólo en función del código que se ejecute, sino en función de quién lo ejecute. La autorización JAAS forma parte de SDK versión 1.4 y posterior.

De forma adicional, WebSphere eXtreme Scale también soporta la autorización personalizada con el siguiente plug-in:

- ObjectGridAuthorization: forma personalizada de autorizar el acceso a todos los artefactos.

Puede implementar su propio mecanismo de autorización si no desea utilizar la autorización JAAS. Mediante el uso de un mecanismo de autorización personalizado, puede utilizar la base de datos de políticas, o Tivoli Access Manager para gestionar las autorizaciones.

Puede configurar el mecanismo de autorización de dos maneras:

- configuración de XML

Puede utilizar el archivo XML ObjectGrid para definir un ObjectGrid y establecer el mecanismo de autorización en `AUTHORIZATION_MECHANISM_JAAS` o `AUTHORIZATION_MECHANISM_CUSTOM`. A continuación se muestra el archivo `secure-objectgrid-definition.xml` que se utiliza en ObjectGridSample de aplicación de empresa:

```
<objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
    <bean id="TransactionCallback"
      className="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
    ...
  </objectGrids>
```

- Configuración mediante programa

Si desea crear un ObjectGrid mediante un método `ObjectGrid.setAuthorizationMechanism(int)`, puede llamar al método siguiente para establecer el mecanismo de autorización. La llamada a este método sólo se aplica al modelo de programación de WebSphere eXtreme Scale local cuando se crea directamente una instancia de ObjectGrid:

```
/**
 * Establecer mecanismo de autorización. El valor predeterminado es
 * com.ibm.websphere.objectgrid.security.SecurityConstants.
 * AUTHORIZATION_MECHANISM_JAAS.
 * @establecer el parámetro authMechanism para el mecanismo de autorización de
 * correlación
 */
void setAuthorizationMechanism(int authMechanism);
```

Autorización JAAS

Un objeto `javax.security.auth.Subject` representa un usuario autenticado. Subject se compone de un conjunto de principales, y cada principal representa una identidad del usuario. Por ejemplo, Subject puede tener un principal de nombre, por ejemplo, Cristina López, y un principal de grupo, por ejemplo, gestor.

Si usa la política de autorización JAAS, los permisos se pueden conceder a principales específicos. WebSphere eXtreme Scale asocia el Subject con el contexto de control de accesos actual. Para cada llamada al método `ObjectMap` o `Javamap`, el tiempo de ejecución de Java determina automáticamente si la política otorga el permiso necesario sólo a un Principal específico y, de esta forma, la operación sólo está permitida si el Subject asociado al contexto de control de accesos contiene el Principal designado.

Debe estar familiarizado con la sintaxis de la política del archivo de políticas. Si desea obtener una descripción detallada de la autorización JAAS, consulte la publicación `JAAS Reference Guide`.

WebSphere eXtreme Scale tiene una base de código especial que se utiliza para comprobar la autorización JAAS para las llamadas a los métodos `ObjectMap` y `JavaMap`. Esta base de código especial es <http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction>. Utilice esta base de código al conceder permisos `ObjectMap` o `JavaMap` a los principales. Este código especial se creó porque el archivo JAR (Java Archive) para eXtreme Scale se otorga con todos los permisos.

La plantilla de la política para conceder el permiso MapPermission es:

```
grant codeBase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
  <Principal field(s)>{
    permission com.ibm.websphere.objectgrid.security.MapPermission
      "[ObjectGrid_name].[ObjectMap_name]", "action";
    ....
    permission com.ibm.websphere.objectgrid.security.MapPermission
      "[ObjectGrid_name].[ObjectMap_name]", "action";
  };
```

Un campo de principal se parece al ejemplo siguiente:

```
principal Principal_class "principal_name"
```

En esta política, sólo se conceden los permisos de insertar y leer a estas cuatro correlaciones con un principal determinado. El otro archivo de políticas, `fullAccessAuth.policy`, concede todos los permisos a estas correlaciones con un principal. Antes de ejecutar la aplicación, cambie el nombre del principal (`principal_name`) y la clase del principal por los valores correspondientes. El valor de `principal_name` depende del Registro de usuarios. Por ejemplo, si el sistema operativo local se utiliza como Registro de usuarios, el nombre de la máquina es MACH1, el ID de usuario es user1, y el `principal_name` es MACH1/user1.

La política de autorización JAAS se puede colocar directamente en el archivo de políticas Java, o se puede colocar en un archivo de autorización JAAS separado y, a continuación, establecerlo de cualquiera de estas dos maneras:

- Utilice el siguiente argumento de JVM :
-Djava.security.auth.policy=file:[JAAS_AUTH_POLICY_FILE]
- Utilice la propiedad siguiente del archivo `java.security`:
-Dauth.policy.url.x=file:[JAAS_AUTH_POLICY_FILE]

Autorización personalizada ObjectGrid

El plug-in `ObjectGridAuthorization` se utiliza para autorizar a `ObjectGrid` los accesos `ObjectMap` y `JavaMap` a los principales, representados por un objeto `Subject` de una manera personalizada. Una implementación típica de este plug-in es recuperar los principales del objeto `Subject` y después comprobar si se han concedido los permisos especificados a los principales.

Un permiso pasado al método `checkPermission(Subject, Permission)` puede ser uno de los siguientes:

- `MapPermission`
- `ObjectGridPermission`
- `AgentPermission`
- `ServerMapPermission`

Consulte la documentación de la API `ObjectGridAuthorization` para obtener más información.

El plug-in `ObjectGridAuthorization` puede configurarse de las siguientes maneras:

- configuración de XML

Puede utilizar el archivo XML `ObjectGrid` para definir un plug-in `ObjectAuthorization`. Ejemplo:

```
<objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
```

```

...
<bean id="ObjectGridAuthorization"
className="com.acme.ObjectGridAuthorizationImpl" />
</objectGrids>

```

- Configuración mediante programa

Si desea crear un ObjectGrid mediante el método de API ObjectGrid.setObjectGridAuthorization(ObjectGridAuthorization), puede llamar al método siguiente para establecer el plug-in de autorización. Este método sólo se aplica al modelo de programación eXtreme Scale local cuando cree directamente la instancia de ObjectGrid.

```

/**
 * Establece <code>ObjectGridAuthorization</code> para esta
 * instancia ObjectGrid.
 * <p>
 * Al pasar <code>null</code> a este método elimina un objeto
 * <code>ObjectGridAuthorization</code> establecido
 * anteriormente de una invocación anterior de este método
 * e indica que este <code>ObjectGrid</code> no está asociado a un
 * objeto <code>ObjectGridAuthorization</code>.
 * <p>
 * Este método sólo debe utilizarse cuando se ha habilitado la seguridad
 * ObjectGrid. Si
 * la seguridad ObjectGrid está inhabilitada, el objeto
 * <code>ObjectGridAuthorization</code> proporcionado
 * no se utilizará.
 * <p>
 * Puede utilizarse un plug-in <code>ObjectGridAuthorization</code> para autorizar
 * el acceso a ObjectGrid y correlaciones. Consulte
 * <code>ObjectGridAuthorization</code> para obtener más información.
 * <p>
 * Desde XD 6.1, <code>setMapAuthorization</code> está en desuso
 * y se recomienda el uso de <code>setObjectGridAuthorization</code>.
 * No obstante,
 * si el plug-in <code>MapAuthorization</code> y el plug-in
 * <code>ObjectGridAuthorization</code>
 * se utilizan, ObjectGrid usará el
 * <code>MapAuthorization</code> proporcionado para autorizar los
 * accesos a las correlaciones,
 * aunque esté en desuso.
 * <p>
 * Para evitar una excepción <code>IllegalStateException</code>,
 * este método
 * debe llamarse antes que al método <code>initialize</code>. Recuerde
 * también
 * que los métodos <code>getSession</code> llaman implícitamente
 * al método <code>initialize</code> si debe llamarlo la
 * aplicación.
 *
 * @establecer el parámetro ogAuthorization para el plug-in
 * <code>ObjectGridAuthorization</code>
 *
 * @emite la excepción IllegalStateException si se llama a este método
 * después de
 * llamar al método <code>initialize</code>.
 *
 * @ver #initialize()
 * @ver ObjectGridAuthorization
 * @desde WAS XD 6.1
 */
void setObjectGridAuthorization(ObjectGridAuthorization ogAuthorization);

```

Implementación de ObjectGridAuthorization

El método checkPermission(Subject subject, Permission permission) booleano de la interfaz ObjectGridAuthorization es invocado por el tiempo de ejecución de WebSphere eXtreme Scale para comprobar si el objeto Subject pasado tiene el permiso pasado. La implementación de la interfaz ObjectGridAuthorization devuelve true si el objeto tiene el permiso, y false si no lo tiene.

Una implementación típica de este plug-in es recuperar los principales del objeto Subject y comprobar si los permisos especificados se han concedido a los principales mediante la consulta de políticas específicas. Estas políticas las definen los usuarios. Por ejemplo, las políticas se pueden definir en una base de datos, en un archivo plano, o un servidor de políticas de Tivoli Access Manager.

Por ejemplo, se puede utilizar el servidor de políticas Tivoli Access Manager para gestionar la política de autorización y utilizar su API para autorizar el acceso. Si desea saber cómo utilizar las API de Tivoli Access Manager Authorization, consulte IBM Tivoli Access Manager Authorization Java Classes Developer Reference para obtener más detalles.

Esta implementación de ejemplo tiene las siguientes presunciones:

- Sólo se comprueba la autorización de MapPermission. Para los demás permisos, siempre devuelve el valor true.
- El objeto Subject contiene un principal com.tivoli.mts.PDPrincipal.
- El servidor de políticas Tivoli Access Manager ha definido los siguientes permisos para el objeto de nombre ObjectMap o JavaMap. El objeto definido en el servidor de políticas debe tener el mismo nombre que el nombre de ObjectMap o JavaMap con el formato [nombre_ObjectGrid].[nombre_ObjectMap]. El permiso es el primer carácter de las series de permisos que se definen en el permiso MapPermission. Por ejemplo, el permiso "r" definido en el servidor de políticas representa el permiso read (lectura) para la correlación ObjectMap.

El siguiente fragmento de código demuestra cómo implementar el método checkPermission:

```
/**
 * @see com.ibm.websphere.objectgrid.security.plugins.
 * MapAuthorization#checkPermission
 * (javax.security.auth.Subject, com.ibm.websphere.objectgrid.security.
 * MapPermission)
 */
public boolean checkPermission(final Subject subject,
    Permission p) {

    // Para non-MapPermission, siempre se da la autorización.
    if (!(p instanceof MapPermission)){
        return true;
    }

    MapPermission permission = (MapPermission) p;

    String[] str = permission.getParsedNames();

    StringBuffer pdPermissionStr = new StringBuffer(5);
    for (int i=0; i<str.length; i++) {
        pdPermissionStr.append(str[i].substring(0,1));
    }

    PDPermission pdPerm = new PDPermission(permission.getName(),
        pdPermissionStr.toString());

    Set principals = subject.getPrincipals();

    Iterator iter= principals.iterator();
    while(iter.hasNext()) {
        try {
            PDPrincipal principal = (PDPrincipal) iter.next();
            if (principal.implies(pdPerm)) {
                return true;
            }
        }
        catch (ClassCastException cce) {
            // Handle exception
        }
    }
    return false;
}
```

Autenticación de cuadrícula

Puede utilizar el plug-in de gestor de señales seguro para habilitar la autenticación de servidor a servidor, que requiere la implementación de la interfaz SecureTokenManager.

El método generateToken(Object) toma un objeto y, a continuación, genera una señal que los otros no pueden entender. El método verifyTokens(byte[]) realiza el proceso inverso: convierte la señal en el objeto original.

Una implementación sencilla de SecureTokenManager utiliza un algoritmo de codificación sencillo como, por ejemplo, un algoritmo XOR, para codificar el objeto en un formato serializado y, a continuación, utilizar el algoritmo de decodificación correspondiente para descifrar la señal. Esta implementación no es segura y es fácil quebrantarla.

Implementación predeterminada de WebSphere eXtreme Scale

WebSphere eXtreme Scale proporciona una implementación disponible de forma inmediata para esta interfaz. Esta implementación predeterminada utiliza un par de claves para firmar y verificar la firma y utiliza una clave secreta para cifrar el contenido. Cada servidor tiene un almacén de claves de tipo JCKES donde se almacena el par de claves, una clave privada y una clave pública, y una clave secreta. El almacén de claves tiene que ser de tipo JCKES para poder almacenar las claves secretas. Estas claves se utilizan para cifrar y firmar o verificar la serie secreta en el envío. Además, la señal se asocia con un tiempo de caducidad. En el extremo receptor, los datos se verifican, se descifran y se comparan con la serie secreta del receptor. Los protocolos de comunicación SSL (Secure Sockets Layer) no son obligatorios para la autenticación entre un par de servidores porque las claves privadas y públicas sirven para ese mismo propósito. No obstante, si la comunicación del servidor no está cifrada, los datos podrían robarse con sólo observar la comunicación. Como la señal caduca pronto, la amenaza de ataque de reproducción se minimiza. Esta posibilidad disminuye en gran medida si todos los servidores se despliegan detrás de un cortafuegos.

La desventaja de este enfoque es que los administradores de WebSphere eXtreme Scale deben generar claves y transportarlas a todos los servidores, que pueden provocar una violación de seguridad durante el transporte.

Seguridad local

WebSphere eXtreme Scale proporciona varios puntos finales de seguridad que permiten integrar mecanismos personalizados. En el modelo de programación local, la principal función de seguridad es la autorización, que no tiene soporte de autenticación. Debe autenticar fuera de WebSphere Application Server. No obstante, se proporcionan plug-ins con el fin de obtener y validar objetos Subject.

Habilitación de la seguridad

La seguridad local puede habilitarse de dos modos:

- **Configuración del XML** Puede utilizar el archivo XML ObjectGrid para definir un ObjectGrid y habilitar la seguridad para dicho objeto ObjectGrid. El archivo siguiente es el archivo secure-objectgrid-definition.xml que se utiliza en el ejemplo de aplicación de empresa ObjectGridSample. En este archivo XML, la seguridad se habilita al establecer el atributo securityEnabled en true.

```

<objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JASS">
    ...
</objectGrids>

```

- **Configuración mediante programa** Si desea crear un ObjectGrid mediante el método de la API ObjectGrid.setSecurityEnabled(), llame al método siguiente en la interfaz ObjectGrid para habilitar la seguridad.

```

/**
 * Habilitar la seguridad ObjectGrid
 */
void setSecurityEnabled();

```

Autenticación

En el modelo de programación local, eXtreme Scale no proporciona ningún mecanismo de autenticación, sino que se basa en el entorno, ya sean servidores de aplicación o aplicaciones, para realizar la autenticación. Cuando se utiliza eXtreme en WebSphere Application Server o WebSphere Extended Deployment, las aplicaciones pueden utilizar el mecanismo de autenticación de seguridad de WebSphere Application Server. Cuando eXtreme Scale se ejecuta en un entorno de Java 2 Platform, Standard Edition (J2SE), la aplicación debe gestionar las autenticaciones con la autenticación JAAS (Java Authentication and Authorization Service) u otro mecanismo de autenticación. Para obtener más información sobre cómo utilizar la autenticación JAAS, consulte la publicación JAAS Reference Guide. El contrato entre una aplicación y una instancia ObjectGrid es el objeto javax.security.auth.Subject. Una vez que el servidor de aplicaciones o la aplicación ha autenticado al cliente, la aplicación puede recuperar el objeto javax.security.auth.Subject autenticado y utilizar este objeto Subject para obtener una sesión de la instancia ObjectGrid mediante la invocación del método ObjectGrid.getSession(Subject). Este objeto Subject se utiliza para autorizar los accesos a los datos de la correlación. Este contrato se denomina mecanismo de paso de sujetos. El ejemplo siguiente ilustra la API ObjectGrid.getSession(Subject).

```

/**
 * Esta API permite que la memoria caché utilice un sujeto específico en lugar del
 * configurado en ObjectGrid para obtener una sesión.
 * @establecer parámetro subject
 * @devuelve una instancia de Session
 * @emite ObjectGridException
 * @emite TransactionCallbackException
 * @emite InvalidSubjectException, el sujeto pasado no es válido según
 * el mecanismo SubjectValidation.
 */
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;

```

El método ObjectGrid.getSession() de la interfaz ObjectGrid también puede utilizarse para obtener un objeto Session:

```

/**
 * Este método devuelve un objeto Session que puede utilizar una única hebra cada vez.
 * No se puede compartir este objeto Session entre las hebras sin colocar una
 * sección crítica a su alrededor. Mientras que la infraestructura principal
 * permite que el objeto se mueva entre hebras, TransactionCallback y Loader
 * podrían impedir este uso, especialmente en entornos J2EE. Cuando la
 * seguridad está habilitada, este método usa SubjectSource para obtener un
 * objeto Subject.
 *
 * Si el método initialize no se ha invocado antes de la primera
 * invocación de getSession, se producirá una inicialización implícita. Esta
 * inicialización garantiza que se completa toda la configuración antes
 * de que se necesite el uso de tiempo de ejecución.
 *
 * @ver #initialize()

```

```

* @devuelve una instancia de Session
* @emite ObjectGridException
* @emite TransactionCallbackException
* @emite la excepción IllegalStateException si se llama a este método después del
*     método destroy().
*/
public Session getSession()
throws ObjectGridException, TransactionCallbackException;

```

Como se especifica en la documentación de la API, al habilitar la seguridad, este método utiliza el plug-in SubjectSource para obtener un objeto Subject. El plug-in SubjectSource es uno de los plug-ins de seguridad definidos en eXtreme Scale para dar soporte a la propagación de objetos Subject. Consulte los plug-ins relacionados con la seguridad para obtener más información. El método getSession(Subject) sólo puede llamarse en la instancia ObjectGrid local. Si llama al método getSession(Subject) en un cliente en una configuración distribuida de eXtreme Scale, se genera una IllegalStateException.

Plug-ins de seguridad

WebSphere eXtreme Scale proporciona dos plug-ins de seguridad relacionados con el mecanismo de paso de asuntos: los plug-ins SubjectSource y SubjectValidation.

Plug-in SubjectSource

El plug-in SubjectSource, representado por la interfaz com.ibm.websphere.objectgrid.security.plugins.SubjectSource, es un plug-in que se utiliza para obtener un objeto Subject de un entorno de ejecución de eXtreme Scale. Este entorno puede ser una aplicación que utiliza el ObjectGrid o un servidor de aplicaciones que contiene la aplicación. Este plug-in SubjectSource es una alternativa al mecanismo de paso de sujetos. Si utiliza el mecanismo de paso de sujetos, la aplicación recupera el objeto Subject y lo utiliza para obtener el objeto de sesión ObjectGrid. Con el plug-in SubjectSource, la ejecución de eXtreme Scale recupera el objeto Subject y lo utiliza para obtener el objeto de sesión. El mecanismo de paso de sujetos otorga el control de objetos Subject a las aplicaciones, mientras que con el mecanismo de plug-in SubjectSource las aplicaciones no tienen que recuperar el objeto Subject. Puede utilizar el plug-in SubjectSource para obtener un objeto Subject que represente un cliente eXtreme Scale que se utilice para la autorización. Cuando se llama al método ObjectGrid.getSession, el Subject getSubject lanza una ObjectGridSecurityException si la seguridad está habilitada. WebSphere eXtreme Scale proporciona una implementación predeterminada de este plug-in: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl. Esta implementación se puede utilizar para recuperar un asunto llamante o un asunto RunAs de la hebra cuando una aplicación se ejecuta en WebSphere Application Server. Puede configurar esta clase en el archivo XML de descriptor de ObjectGrid como la clase de implementación de SubjectSource al utilizar eXtreme Scale en WebSphere Application Server. El fragmento de código siguiente muestra el flujo principal del método WSSubjectSourceImpl.getSubject.

```

Subject s = null;
try {
    if (finalType == RUN_AS_SUBJECT) {
        // obtener el sujeto RunAs
        s = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
    }
    else if (finalType == CALLER_SUBJECT) {
        // obtener callersubject
        s = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
    }
}

```

```

catch (WSSecurityException wse) {
    throw new ObjectGridSecurityException(wse);
}

return s;

```

Si desea obtener más detalles, consulte la documentación de la API del plug-in `SubjectSource` y la implementación `WSSubjectSourceImpl`.

Plug-in `SubjectValidation`

El plug-in `SubjectValidation`, que está representado por la interfaz `com.ibm.websphere.objectgrid.security.plugins.SubjectValidation`, es otro plug-in de seguridad. El plug-in `SubjectValidation` puede utilizarse para validar que un objeto `javax.security.auth.Subject`, pasado a `ObjectGrid` o recuperado mediante el plug-in `SubjectSource`, es un `Subject` válido que no se ha manipulado de forma indebida.

El método `SubjectValidation.validateSubject(Subject)` de la interfaz `SubjectValidation` toma un objeto `Subject` y devuelve un objeto `Subject`. El que un objeto `Subject` se considere válido y qué objeto `Subject` se va a devolver dependerá de las implementaciones. Si el objeto `Subject` no es válido, se genera una `InvalidSubjectException`.

Puede utilizar este plug-in si no confía en el objeto `Subject` pasado a este método. Esto no es habitual si se tiene en cuenta que el usuario suele confiar en los desarrolladores de las aplicaciones que desarrollan el código que recupera el objeto `Subject`.

Una implementación de este plug-in necesita el soporte del creador del objeto `Subject` porque sólo el creador sabe si el objeto `Subject` ha sido manipulado indebidamente. Puede darse el caso, no obstante, de que el creador no sepa si el objeto `Subject` se ha manipulado de forma indebida. En un caso así, este plug-in no resulta de utilidad.

WebSphere eXtreme Scale proporciona una implementación predeterminada de `SubjectValidation`:

`com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl`. Puede utilizar esta implementación para validar el asunto autenticado por WebSphere Application Server. Puede configurar esta clase como la clase de implementación de `SubjectValidation` cuando se utiliza eXtreme Scale en WebSphere Application Server. La implementación `WSSubjectValidationImpl` considera válido un objeto `Subject` sólo si la señal de credencial asociada con este objeto `Subject` no se ha manipulado de forma indebida. Puede modificar otras partes del objeto `Subject`. La implementación de `WSSubjectValidationImpl` solicita a WebSphere Application Server el `Subject` original correspondiente a la señal de credencial y devuelve el objeto `Subject` original como el objeto `Subject` validado. Por lo tanto, los cambios realizados en el contenido de `Subject` que no sea la señal de credencial, no tiene ningún efecto. El fragmento de código siguiente muestra el flujo básico de `WSSubjectValidationImpl.validateSubject(Subject)`.

```

//
// Crear un LoginContext con WSLogin de esquema y
// pasar un manejador de devolución de llamada.
LoginContext lc = new LoginContext("WSLogin",
new WSCredTokenCallbackHandlerImpl(subject));

// Cuando se llama a este método, los métodos del manejador de devolución de llamada
// se llamarán para iniciar la sesión de usuario.

```

```
lc.login();

// Obtener el objeto Subject de LoginContext
return lc.getSubject();
```

En el fragmento de código anterior, se crea un objeto de manejador de devolución de llamada de la señal de credencial, `WSCredTokenCallbackHandlerImpl`, con el objeto `Subject` para validar. Después, se crea un objeto `LoginContext` con el esquema de inicio de sesión `WSLogin`. Cuando se llama al método `lc.login`, la seguridad de WebSphere Application Server recupera la señal de credencial del objeto `Subject` y, a continuación, devuelve el `Subject` correspondiente como el objeto `Subject` validado.

Para ver otros detalles, consulte las API Java de la implementación `SubjectValidation` y `WSSubjectValidationImpl`.

Configuración de plug-in

Puede configurar el plug-in `SubjectValidation` y el plug-in `SubjectSource` de dos maneras:

- **Configuración del XML** Puede utilizar el archivo XML `ObjectGrid` para definir un `ObjectGrid` y establecer estos dos plug-ins. A continuación se muestra un ejemplo en el que la clase `WSSubjectSourceImpl` se configura como plug-in `SubjectSource` y la clase `WSSubjectValidation` se configura como plug-in `SubjectValidation`.

```
<objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
    <bean id="SubjectSource"
      className="com.ibm.websphere.objectgrid.security.plugins.builtins.
        WSSubjectSourceImpl" />
    <bean id="SubjectValidation"
      className="com.ibm.websphere.objectgrid.security.plugins.builtins.
        WSSubjectValidationImpl" />
    <bean id="TransactionCallback"
      className="com.ibm.websphere.samples.objectgrid.
        HeapTransactionCallback" />
    ...
  </objectGrids>
```

- **Configuración mediante programa** Si desea crear un `ObjectGrid` mediante API, puede llamar a los métodos siguientes para establecer los plug-ins `SubjectSource` o `SubjectValidation`.

```
**
* Establecer el plug-in SubjectValidation para esta instancia ObjectGrid. Un
* plug-in SubjectValidation puede utilizarse para validar el objeto Subject
* pasado como Subject válido. Consulte {@link SubjectValidation}
* para obtener más información.
* @establecer el parámetro subjectValidation para el plug-in SubjectValidation
*/
void setSubjectValidation(SubjectValidation subjectValidation);

/**
* Establecer el plug-in SubjectSource. Un plug-in SubjectSource puede utilizarse
* para obtener un objeto Subject del entorno para representar el
* cliente ObjectGrid.
*
* @establecer el parámetro source para el plug-in SubjectSource
*/
void setSubjectSource(SubjectSource source);
```

Escribir código de autenticación JAAS

Puede escribir su propio código de autenticación JAAS (Java Authentication and Authorization Service) para manejar la autenticación. Debe escribir los módulos de inicio de sesión y después configurarlos para el módulo de autenticación.

El módulo de inicio de sesión recibe información sobre un usuario y lo autentica. Esta información puede ser cualquier información que identifique al usuario. Por ejemplo, puede ser un ID de usuario y una contraseña, un certificado de cliente, etc. Después de recibir la información, el módulo de inicio de sesión comprueba que la información representa a un sujeto válido y crea un objeto Subject. Actualmente existen diversas implementaciones de módulos de inicio de sesión.

Una vez que se ha escrito un módulo de inicio de sesión, configure este módulo de inicio de sesión para su uso en el tiempo de ejecución. Debe configurar un módulo de inicio de sesión JAAS. Este módulo de inicio de sesión contiene el módulo de inicio de sesión y su esquema de autenticación. Por ejemplo:

```
FileLogin
{
    com.acme.auth.FileLoginModule required
};
```

El esquema de autenticación es FileLogin y el módulo de inicio de sesión es com.acme.auth.FileLoginModule. La señal requerida indica que el módulo FileLoginModule debe validar este inicio de sesión o se producirá una anomalía en todo el esquema.

Puede establecer el archivo de configuración del módulo de inicio de sesión JAAS de uno de los siguientes modos:

- Establezca el archivo de configuración del módulo de inicio de sesión JAAS en la propiedad login.config.url del archivo java.security, por ejemplo:
login.config.url.1=file:\${java.home}/lib/security/file.login
- Establezca el archivo de configuración del módulo de inicio de sesión JAAS desde la línea de mandatos utilizando los argumentos de la máquina virtual Java (JVM) **-Djava.security.auth.login.config**, por ejemplo,
-Djava.security.auth.login.config ==\$JAVA_HOME/lib/security/file.login

Si el código se está ejecutando en WebSphere Application Server, debe configurar el inicio de sesión JAAS en la consola de administración y almacenar esta configuración de inicio de sesión en la configuración del servidor de aplicaciones. Consulte la configuración del inicio de sesión para Java Authentication and Authorization Service si desea más detalles.

Capítulo 10. Consideraciones sobre el rendimiento para desarrolladores de aplicaciones

Para mejorar el rendimiento de la cuadrícula de datos en memoria o del espacio de proceso de la base de datos, puede investigar varias consideraciones como, por ejemplo, ajustar los valores de la máquina virtual Java y utilizar los procedimientos recomendados para las características del producto como el bloqueo, la serialización y el rendimiento de la consulta.

Ajuste de la JVM

Debe tener en cuenta varios aspectos específicos sobre el ajuste de la máquina virtual Java (JVM) para conseguir el mejor rendimiento posible de WebSphere eXtreme Scale.

La recomendación es entre 1 y 2 Gb de almacenamiento dinámico con una JVM por 4 núcleos. Los tamaños del almacenamiento dinámico dependen de la naturaleza de los objetos que se almacenan en los servidores, se habla de estos más adelante en este documento.

Recomendaciones de tamaño del almacenamiento dinámico y de la recogida de basura

El número de tamaño óptimo de almacenamiento dinámico depende de tres factores:

1. El número de objetos activos en el almacenamiento dinámico.
2. La complejidad de los objetos activos del almacenamiento dinámico.
3. El número de núcleos disponibles para la JVM.

Por ejemplo, una aplicación que almacena 10 KB de matrices de bytes puede ejecutar un almacenamiento dinámico mucho mayor que una aplicación que utiliza gráficos complejos de objetos POJO.

Actualmente, todas las JVM modernas utilizan algoritmos de recogida de basura paralela, lo que significa que utilizar más núcleos puede reducir las pausas en la recogida de basura. Por lo tanto, una caja de 8 núcleos realizará las recopilaciones más rápido que una caja con 4 núcleos.

Uso de memoria real versus la especificación de almacenamiento dinámico

Una JVM de almacenamiento dinámico de 1 Gb utiliza, aproximadamente 1,3 Gb de memoria real. En nuestro laboratorio, no hemos podido ejecutar 10 JVM de 1 Gb en una caja con 16 Gb de RAM. Una vez que se llenaban del todo los almacenamientos dinámicos de la JVM con más de 800 MB, la caja empezaba la paginación.

Recogida de basura

Para las JVM de IBM, utilice el recolector `avgoptpause` para los escenarios con un índice alto de actualizaciones (100% de entradas de modificación de transacciones). El recolector `gencon` funciona mucho mejor que el recolector `avgoptpause` en

escenarios donde los datos se actualizan con poca frecuencia (10% del tiempo o menos). Experimente con los dos tipos de recolectores para ver cuál funciona mejor en su escenario. Si observa un problema de rendimiento, ejecute la operación con la recogida de basura detallada activada para comprobar el porcentaje del tiempo que se emplea en la recogida de basura. Se han dado casos en los que se empleaba el 80% del tiempo en la recogida de basura hasta que se arregló el problema.

Rendimiento de la JVM

WebSphere eXtreme Scale se puede ejecutar en distintas versiones de Java 2 Platform, Standard Edition (J2SE). ObjectGrid Versión 6.1 soporta J2SE versión 1.4.2 y posterior. Para obtener un rendimiento y productividad del programador mayores, utilice J2SE 5 o posterior para aprovechar los beneficios de las anotaciones y de la recogida de basura mejorada. ObjectGrid funciona en las JVM de 32 bits o 64 bits.

Los clientes de ObjectGrid versión 6.0.2 pueden conectarse a una cuadrícula de ObjectGrid versión 6.1. Utilice los clientes de ObjectGrid versión 6.1 para J2SE versión 1.4.2 o clientes mejores. La única razón para usar un cliente de ObjectGrid versión 6.0.2 es para dar soporte a J2SE versión 1.3.

WebSphere eXtreme Scale se prueba con un subconjunto de las máquinas virtuales disponibles, sin embargo, la lista soportada no es exclusiva. Puede ejecutar WebSphere eXtreme Scale en cualquier versión 1.4.2 o superior, pero si se identifica un problema en la JVM, debe ponerse en contacto con el proveedor de la JVM para recibir soporte. Si es posible, utilice la JVM del tiempo de ejecución de WebSphere en cualquier plataforma que soporte WebSphere Application Server.

Java Platform, Standard Edition 6 es la mejor JVM. Java 2 Platform, Standard Edition, v 1.4 tiene un rendimiento bajo, especialmente para los escenarios donde el recolector gencon tiene un buen rendimiento. Java Platform Standard Edition 5 tiene un buen rendimiento, pero el rendimiento de Java Platform, Standard Edition 6 es mejor.

Ajuste de orb.properties

La recomendación es utilizar el siguiente archivo orb.properties para la producción. En nuestro laboratorio, hemos utilizado este archivo en cuadrículas de hasta 1500 JVM. El archivo orb.properties está en la carpeta lib del JRE que se está utilizando.

```
# Propiedades de IBM JDK para ORB
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton

# Interceptores de WS
org.omg.PortableInterceptor.ORBInitializerClass=com.ibm.ws.objectgrid.corba.ObjectGridInitializer

# Propiedades de plugins y ORB de WS
com.ibm.CORBA.ForceTunnel=never
com.ibm.CORBA.RequestTimeout=10
com.ibm.CORBA.ConnectTimeout=10

# Necesario cuando muchas JVM se conectan al catálogo a la vez
com.ibm.CORBA.ServerSocketQueueDepth=2048

# Los clientes y el servidor de catálogo pueden tener sockets abiertos para todas las JVM
com.ibm.CORBA.MaxOpenConnections=1016

# Agrupación de hebras para el manejo de solicitudes de entrada, aquí 200 hebras
com.ibm.CORBA.ThreadPool.IsGrowable=false
com.ibm.CORBA.ThreadPool.MaximumSize=200
com.ibm.CORBA.ThreadPool.MinimumSize=200
com.ibm.CORBA.ThreadPool.InactivityTimeout=180000

# No se dividen las peticiones/respuestas grandes en fragmentos menores
com.ibm.CORBA.FragmentSize=0
```

Número de hebras

El número de hebras depende de unos pocos factores. Existe un límite en el número de hebras que puede gestionar un solo fragmento. A más fragmentos para cada JVM, más hebras y más simultaneidad podrán existir. Cada fragmento adicional proporciona más vías de acceso simultáneas a los datos. Aunque cada fragmento es tan simultáneo como es posible, existe un límite.

Procedimientos recomendados de CopyMode

WebSphere eXtreme Scale realiza una copia del valor basado en los seis valores de CopyMode disponibles. Determine el valor que funcione mejor para sus necesidades de despliegue.

Puede utilizar el método `setCopyMode(CopyMode, valueInterfaceClass)` de la API `BackingMap` para establecer la modalidad de copia en uno de los siguientes campos estáticos finales que se definen en la clase `com.ibm.websphere.objectgrid.CopyMode`.

Cuando una aplicación utiliza la interfaz `ObjectMap` para obtener una referencia a una entrada de correlación, utilice dicha referencia sólo dentro de la transacción de WebSphere eXtreme Scale que obtuvo la referencia. El uso de la referencia en una transacción diferente puede conducir a errores. Por ejemplo, si utiliza la estrategia de bloqueo pesimista para `BackingMap`, una llamada de método `get` o `getForUpdate` adquiere un bloqueo S (compartido) o U (actualización), en función de la transacción. El método `get` devuelve la referencia al valor y el bloqueo que se obtiene se libera cuando se completa la transacción. La transacción debe llamar al método `get` o `getForUpdate` para bloquear la entrada de la correlación en una transacción diferente. Cada transacción debe obtener su propia referencia al valor llamando al método `get` o `getForUpdate`, en lugar de reutilizar la misma referencia de valor en varias transacciones.

CopyMode para correlaciones de entidad

Si se utiliza una correlación asociada con una entidad de API `EntityManager`, la correlación siempre devuelve los objetos tuple de entidad directamente sin realizar una copia, a menos que utilice la modalidad de copia `COPY_TO_BYTES`. Es importante que `CopyMode` se actualice o que se copie el objeto `Tuple` correctamente al realizar los cambios.

COPY_ON_READ_AND_COMMIT

La modalidad `COPY_ON_READ_AND_COMMIT` es la modalidad predeterminada. El argumento `valueInterfaceClass` se pasa por alto cuando se utiliza esta modalidad. Esta modalidad garantiza que una aplicación no contenga una referencia al objeto de valor que esté en la correlación `BackingMap`. En su lugar, la aplicación siempre trabaja con una copia del valor que esté en la correlación `BackingMap`. La modalidad `COPY_ON_READ_AND_COMMIT` garantiza que la aplicación nunca pueda dañar accidentalmente los datos almacenados en memoria caché en `BackingMap`. Cuando una transacción de la aplicación llama a un método `ObjectMap.get` de una clave determinada, y es el primer acceso de la entrada `ObjectMap` de esa clave, se devuelve una copia del valor. Cuando se confirma la transacción, los cambios confirmados por la aplicación se copian en `BackingMap` para garantizar que la aplicación no tenga una referencia al valor confirmado en `BackingMap`.

COPY_ON_READ

La modalidad `COPY_ON_READ` mejora el rendimiento en comparación con la modalidad `COPY_ON_READ_AND_COMMIT` al eliminar la copia que se produce cuando se confirma una transacción. El argumento `valueInterfaceClass` se pasa por alto cuando se utiliza esta modalidad. Para conservar la integridad de los datos de `BackingMap`, la aplicación garantiza que todas las referencias que tiene de una entrada se destruyan una vez confirmada la transacción. Con esta modalidad, el método `ObjectMap.get` devuelve una copia del valor en lugar de una referencia al valor para garantizar que los cambios realizados por la aplicación en el valor no afecten al valor de `BackingMap` hasta que se confirme la transacción. No obstante, cuando se confirma, no se realiza una copia de los cambios. En su lugar, se almacena en `BackingMap` la referencia a la copia devuelta por el método `ObjectMap.get`. La aplicación destruye todas las referencias de la entrada de correlación una vez que se confirma la transacción. Si la aplicación no las destruye, la aplicación podría dañar los datos almacenados en memoria caché de `BackingMap`. Si una aplicación que utiliza esta modalidad experimenta problemas, cambie a la modalidad `COPY_ON_READ_AND_COMMIT` para ver si se sigue produciendo el problema. Si desaparece, significa que la aplicación no está destruyendo todas las referencias después de la confirmación de la transacción.

COPY_ON_WRITE

La modalidad `COPY_ON_WRITE` mejora el rendimiento en comparación con la modalidad `COPY_ON_READ_AND_COMMIT` al eliminar la copia que se produce cuando una transacción llama por primera vez al método `ObjectMap.get` para una clave determinada. El método `ObjectMap.get` devuelve un proxy al valor en lugar de una referencia directa al objeto de valor. El proxy garantiza que no se realice una copia del valor a no ser que la aplicación llame a un método `set` en la interfaz de valor especificada en el argumento `valueInterfaceClass`. El proxy proporciona una copia en la implementación de grabación. Cuando se confirma una transacción, `BackingMap` examina el proxy para determinar si se realizó una copia como resultado de haber llamado a un método `set`. Si se realizó una copia, la referencia a dicha copia se almacena en `BackingMap`. La ventaja de utilizar esta modalidad es que un valor nunca se copia en una operación de lectura o de confirmación si la transacción no ha llamado a un método `set` para cambiar el valor.

Las modalidades `COPY_ON_READ_AND_COMMIT` y `COPY_ON_READ` realizan una copia profunda cuando un valor se recupera de `ObjectMap`. Si una aplicación sólo actualiza algunos de los valores recuperados en una transacción, esta modalidad no es la ideal. La modalidad `COPY_ON_WRITE` admite este comportamiento de una manera eficaz, pero requiere que la aplicación utilice un patrón sencillo. Los objetos de valor son obligatorios para admitir una interfaz. La aplicación debe utilizar los métodos de esta interfaz cuando interactúe con el valor de una sesión de eXtreme Scale. Si éste fuera el caso, eXtreme Scale crea proxies para los valores devueltos a la aplicación. El proxy tiene una referencia a un valor real. Si la aplicación sólo realiza operaciones de lectura, éstas siempre se ejecutan contra la copia real. Si la aplicación modifica un atributo en el objeto, el proxy realiza una copia del objeto real y después realiza la modificación en la copia. A continuación, el proxy utilice la copia a partir de ese punto. El uso de la copia permite que no se realice la operación de copia para los objetos que sólo lee la aplicación. Todas las operaciones de modificación deben empezar con el prefijo `set`. Normalmente, los Enterprise JavaBeans se codifican para utilizar este estilo de denominación de método para los métodos que modifican los atributos de objetos. Debe seguirse este convenio. Los objetos que se modifican se copian en el

momento en que los modifica la aplicación. Este escenario de lectura y escritura es el escenario más eficaz soportado por eXtreme Scale. Para configurar una correlación de modo que utilice la modalidad COPY_ON_WRITE, observe el ejemplo siguiente. En este ejemplo, la aplicación almacena objetos Person que utilizan el nombre en la correlación. El objeto person se representa en el siguiente fragmento de código.

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

La aplicación utiliza la interfaz IPerson sólo cuando interactúa con valores recuperados de ObjectMap. Modifique el objeto para utilizar una interfaz como en el ejemplo siguiente:

```
interface IPerson
{
    void setName(String n);
    String getName();
    void setAge(int a);
    int getAge();
}
// Modificar Person para implementar la interfaz IPerson
class Person implements IPerson {
    ...
}
```

La aplicación necesita configurar BackingMap para que utilice la modalidad COPY_ON_WRITE, como en este ejemplo:

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// usar COPY_ON_WRITE para esta correlación con
// IPerson como valueProxyInfo Class
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// La aplicación debe utilizar el siguiente
// patrón al usar la correlación PERSON.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// la aplicación difunde el valor devuelto a IPerson y no Person
IPerson p = (IPerson)person.get("Billy");
p.setAge(p.getAge()+1);
...
// hacer Person nuevo y añadirlo a la correlación
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
```

```
// el fragmento de código siguiente NO FUNCIONARÁ. Devolverá ClassCastException
sess.begin();
// el error ha sido utilizar Person en lugar de
// IPerson
Person a = (Person)person.get("Bobby");
sess.commit();
```

La primera sección muestra la aplicación que recupera un valor de nombre Billy en la correlación. La aplicación difunde el valor devuelto al objeto IPerson, no al objeto Person porque el proxy que se devuelve implementa dos interfaces:

- La interfaz especificada en la llamada al método `BackingMap.setCopyMode`.
- La interfaz `com.ibm.websphere.objectgrid.ValueProxyInfo`.

Puede difundir el proxy para dos tipos. La última parte del fragmento de código anterior muestra lo que no se permite en la modalidad `COPY_ON_WRITE`. La aplicación recupera el registro Bobby e intenta difundir el registro a un objeto Person. Esta acción produce una excepción de difusión de clase porque el proxy devuelto no es un objeto Person. El proxy devuelto implementa el objeto IPerson y ValueProxyInfo.

Interfaz ValueProxyInfo y soporte de actualización parcial: esta interfaz permite a una aplicación recuperar el valor confirmado de sólo lectura al que hace referencia el proxy o el conjunto de atributos modificado durante esta transacción.

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

El método `ibmGetRealValue` devuelve una copia de sólo lectura del objeto. La aplicación no debe modificar este valor. El método `ibmGetDirtyAttributes` devuelve una lista de series que representa los atributos modificados por la aplicación durante esta transacción. El principal caso de uso de `ibmGetDirtyAttributes` está en un cargador basado en CMP o JDBC (Java database connectivity). Sólo deben actualizarse los atributos de la lista, ya sea en la sentencia SQL o en el objeto correlacionado con la tabla; se obtiene así un SQL, generado por el cargador, más eficaz. Cuando se confirma una transacción "copy on write" y se conecta un cargador, éste puede difundir los valores de los objetos modificados a la interfaz ValueProxyInfo para obtener esta información.

Manejo del método `equals` al utilizar `COPY_ON_WRITE` o servidores proxy: por ejemplo, el código siguiente construye un objeto Person y lo inserta en un ObjectMap. A continuación, recupera el mismo objeto mediante el método ObjectMap.get. El valor se difunde a la interfaz. Si el valor se difunde a la interfaz Person, se produce una excepción ClassCastException porque el valor devuelto es un proxy que implementa la interfaz IPerson y no es un objeto Person. La comprobación de igualdad falla al utilizar la operación `==` porque no son el mismo objeto.

```
session.begin();
// objeto Person nuevo
Person p = new Person(...);
personMap.insert(p.getName(), p);
// recuperarlo de nuevo, recordar usar la interfaz para la difusión
IPerson p2 = personMap.get(p.getName());
if(p2 == p) {
    // son iguales
} else {
    // no son iguales
}
```

Otra consideración a tener en cuenta es cuando debe alterarse temporalmente el método equals. Como se ilustra en el fragmento de código siguiente, el método equals debe verificar que el argumento es un objeto que implementa la interfaz IPerson y difunde el argumento para ser IPerson. Como el argumento puede ser un proxy que implementa la interfaz IPerson, debe usar los métodos getAge y getName al comparar la igualdad de las variables de instancia.

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

Requisitos de configuración de ObjectQuery y HashIndex: cuando se utiliza COPY_ON_WRITE con el plug-in ObjectQuery o HashIndex, es importante configurar el esquema ObjectQuery y un plug-in HashIndex para acceder a los objetos a través de métodos de propiedades, que es el valor predeterminado. Si se ha configurado para usar el acceso de campos, el motor de consultas y el índice intentarán acceder a los campos en los objetos proxy, que siempre devolverán null o 0 ya que la instancia del objeto será un proxy.

NO_COPY

La modalidad NO_COPY permite que una aplicación nunca modifique un objeto de valor que se obtiene mediante el método ObjectMap.get a cambio de mejoras en el rendimiento. El argumento valueInterfaceClass se pasa por alto cuando se utiliza esta modalidad. Si se utiliza esta modalidad, no se produce nunca una copia del valor. Si la aplicación modifica los valores, los datos de BackingMap se dañarán. La modalidad NO_COPY es útil especialmente en el caso de correlaciones de sólo lectura en las que la aplicación nunca modifica los datos. Si la aplicación utiliza esta modalidad y experimenta problemas, cambie a la modalidad COPY_ON_READ_AND_COMMIT para ver si se sigue produciendo el problema. Si desaparece, significa que la aplicación está modificando el valor devuelto por el método ObjectMap.get, durante la transacción o una vez confirmada ésta. Todas las correlaciones asociadas a las entidades de la API EntityManager utilizan automáticamente esta modalidad, independientemente de lo que se ha especificado en la configuración de eXtreme Scale.

Todas las correlaciones asociadas a las entidades de la API EntityManager utilizan automáticamente esta modalidad, independientemente de lo que se ha especificado en la configuración de eXtreme Scale.

COPY_TO_BYTES

Puede almacenar los objetos en un formato serializado, en lugar del formato POJO. Mediante el uso del valor COPY_TO_BYTES, puede reducir la huella de la memoria que puede consumir un gran gráfico de objetos. Consulte “Correlaciones de matriz de bytes” en la página 40 si desea información adicional.

Uso incorrecto de CopyMode

Los errores se producen cuando la aplicación intenta mejorar el rendimiento al usar las modalidades de copia COPY_ON_READ, COPY_ON_WRITE o NO_COPY, como se ha descrito anteriormente. Los errores intermitentes no se producen al cambiar la modalidad de copia a la modalidad COPY_ON_READ_AND_COMMIT.

Problema

Los datos de la correlación ObjectGrid pueden resultar dañados como resultado de la violación por parte de la aplicación del contrato de programación de la modalidad de copia que se está utilizando. El daño en los datos puede ocasionar errores imprevisibles de forma intermitente o errores que se manifiestan de forma inexplicable o inesperada.

Solución

La aplicación debe cumplir el contrato de programación que se aplica para la modalidad de copia que se vaya a utilizar. Para las modalidades de copia COPY_ON_READ y COPY_ON_WRITE, la aplicación utiliza una referencia a un objeto de valor fuera del ámbito de la transacción del que se obtuvo la referencia del valor. Para utilizar estas modalidades, la aplicación debe eliminar la referencia al objeto de valor una vez completada la transacción, y obtener una nueva referencia al objeto de valor en cada transacción que acceda al objeto de valor. Para la modalidad de copia NO_COPY, la aplicación nunca debe modificar el objeto de valor. En este caso, escriba la aplicación de modo que no cambie el objeto de valor, o establezca la aplicación para utilizar otra modalidad de copia.

Correlaciones de matriz de bytes

Puede almacenar los pares de clave-valor en sus correlaciones en una matriz de bytes, en lugar del formato POJO, que reduce la huella de la memoria que puede consumir un gráfico grande de objetos.

Ventajas

La cantidad de memoria que se consume aumenta con el número de objetos de una gráfico de objetos. Reduciendo un gráfico complicado de objetos a una matriz de bytes, sólo se conserva un objeto en el almacenamiento dinámico, en lugar de varios objetos. Con esta reducción del número de objetos en el almacenamiento dinámico, el tiempo de ejecución Java tiene menos objetos para buscar durante la recogida de basura.

El mecanismo de copia predeterminado utilizado por WebSphere eXtreme Scale es la serialización, que es un mecanismo caro. Por ejemplo, si utiliza la modalidad de copia predeterminada de COPY_ON_READ_AND_COMMIT, se realiza una copia tanto en el momento de leer, como en el de obtener. En lugar de realizar una copia durante la lectura, con las matrices de bytes, el valor se infla a partir de los bytes y, en lugar de realizar una copia durante la confirmación, el valor se serializa en bytes. El uso de matrices de bytes genera una coherencia de datos equivalentes al valor predeterminado con una reducción de la memoria utilizada.

Si se utilizan las matrices de bytes, tenga en cuenta que tener un mecanismo de serialización optimizado es vital para ver una reducción en el consumo de memoria. Para obtener más información, consulte "Rendimiento de serialización" en la página 234.

Configuración de correlaciones de matrices de bytes

Puede habilitar las correlaciones de matrices de bytes con el archivo XML ObjectGrid modificando el atributo CopyMode utilizado por una correlación por el valor COPY_TO_BYTES, mostrado en el ejemplo siguiente:

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

Consulte el tema sobre el archivo XML de descriptor de ObjectGrid en la *Guía de administración* si desea más información.

Consideraciones

Debe considerar si va a utilizar o no las correlaciones de matrices de bytes en un escenario determinado. Aunque puede reducir el uso de la memoria, el uso del procesador puede aumentar cuando se utilizan las matrices de bytes.

La lista siguiente describe varios factores que se deben tener en cuenta antes de elegir utilizar la función de correlación de matrices de bytes.

Tipo de objeto

Comparativamente, la reducción de la memoria no es posible si se utilizan las correlaciones de matrices de bytes para algunos tipos de objeto. Por consecuencia, existen varios tipos de objeto para los que no deberá utilizar las correlaciones de matrices de bytes. Si utiliza algunos de los derivadores primitivos de Java como valores, o un POJO que no contiene referencias a ningún otro objeto (sólo almacenar campos primitivos), el número de objetos Java ya es tan bajo como sea posible, sólo hay uno. Puesto que la cantidad de memoria utilizada por el objeto ya se ha optimizado, no se recomienda el uso de una correlación de matrices de bytes para estos tipos de objetos. Las correlaciones de matrices de bytes son más idóneas para los tipos de objeto que contiene otros objetos o colecciones de objetos donde el número total de objetos POJO es mayor que uno.

Por ejemplo, si tiene un objeto Customer (Cliente) que tenía una dirección empresarial y una dirección personal, así como una colección de Orders (Pedidos), el número de objetos en el almacenamiento dinámico y el número de bytes utilizados por dichos objetos se pueden reducir mediante el uso de correlaciones de matrices de bytes.

Acceso local

Cuando se utilizan otras modalidades de copia, las aplicaciones se pueden optimizar, cuando las copias se realizan, si los objetos son Cloneable con el ObjectTransformer predeterminado o cuando se proporciona un ObjectTransformer personalizado con un método copyValue optimizado. En comparación con otras modalidades de copia, la copia en operaciones de lecturas, escrituras o confirmación tendrá un coste adicional al acceder a los objetos de forma local. Por ejemplo, si tiene una memoria caché cercana en una topología distribuida o al acceder directamente a una instancia de ObjectGrid de servidor o local, el tiempo de acceso y confirmación se aumentará si se utilizan las correlaciones de matrices de bytes debido al coste de serialización. Verá un coste similar en una topología distribuida, si utiliza los agentes de cuadrícula de datos o si accede al primario del servidor, al utilizar el plug-in ObjectGridEventGroup.ShardEvents.

Interacciones de plug-in

Con las correlaciones de matrices de bytes, los objetos no se inflan cuando se establece una comunicación entre un cliente y un servidor, a menos que el servidor necesite un formato POJO. Los plug-ins que interactúan con el valor de correlación experimentará una reducción en el rendimiento debido a la necesidad de inflar el valor.

Cualquier plug-in que utiliza `LogElement.getCacheEntry` o `LogElement.getCurrentValue` verá este coste adicional. Si desea obtener la clave, podrá utilizar `LogElement.getKey`, que impide la sobrecarga adicional asociada al método `LogElement.getCacheEntry().getKey`. En las siguientes secciones se tratan los plug-ins para clarificar el uso de las matrices de bytes.

Índices y consultas

Cuando los objetos se almacenan en el formato POJO, el coste de los índices y las consultas es mínimo, porque el objeto no necesita ser inflado. Cuando se utiliza una correlación de matrices de bytes tendrá el coste adicional de inflar el objeto. En general, si la aplicación utiliza índices o consultas, no se recomienda utilizar las correlaciones de matrices de bytes, a menos que sólo ejecute consultas sobre atributos de clave.

Bloqueo optimista

Si se utiliza la estrategia de bloqueo optimista, tendrá el coste adicional durante las actualizaciones y las operaciones invalidar. Esto procede de tener que inflar el valor en el servidor para obtener el valor de la versión para realizar una comprobación de colisión optimista. Si simplemente utiliza el bloqueo optimista para garantizar las operaciones de obtención de información y no necesita una comprobación de colisión optimista, puede utilizar `com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback` para inhabilitar la comprobación de versiones.

Cargador

Con un cargador, también tendrá el coste en el tiempo de ejecución de eXtreme Scale de inflar y serializar el valor si es utilizado por el cargador. Puede seguir utilizando las correlaciones de matrices de bytes con los cargadores, pero tenga en cuenta el coste de realizar cambios en el valor en dicho escenario. Por ejemplo, puede utilizar la característica de matriz de bytes en el contexto de una memoria caché que se lee con frecuencia. En este caso, la ventaja de tener menos objetos en el almacenamiento dinámico y utilizar menos memoria superará el coste generado del uso de las matrices de bytes en las operaciones insertar y actualizar.

ObjectGridEventListener

Cuando se utiliza el método `transactionEnd` en el plug-in `ObjectGridEventListener`, tendrá un coste adicional en el lado del servidor para las solicitudes remotas al acceder a una `CacheEntry` del `LogElement` o al valor actual. Si la implementación del método no accede a estos campos, no tendrá el coste adicional.

Procedimientos recomendados para optimizar el rendimiento del desalojador de plug-in

Si utiliza desalojadores de plug-in, éstos no se activarán hasta que los cree y los asocie con una correlación de respaldo. Los procedimientos recomendados siguientes aumentarán el rendimiento para los desalojadores LFU (utilizados con menor frecuencia) y los desalojadores LRU (menos utilizados recientemente).

Desalojador LFU (utilizados con menor frecuencia)

El concepto de un desalojador LFU es eliminar entradas de la correlación que menos se utilizan. Las entradas de la correlación se expanden en un volumen establecido de almacenamientos dinámicos. A medida que aumenta el uso de una entrada en memoria caché determinada, se ordena en una posición más alta en el almacenamiento dinámico. Cuando el desalojador intenta un conjunto de desalojos, elimina sólo las entradas de la memoria caché ubicadas por debajo de un punto específico del almacenamiento dinámico binario. Como resultado, se desalojan las entradas usadas con menor frecuencia.

Desalojador LRU (menos utilizados recientemente)

El desalojador LRU sigue los mismos conceptos que el desalojador LFU con unas pocas diferencias. La diferencia principal es que el LRU utiliza una cola FIFO (primero en entrar, primero en salir) en lugar de un conjunto de almacenamientos dinámicos binarios. Cada vez que se accede a una entrada de la memoria caché, ésta se mueve a la cabeza de la cola. En consecuencia, la parte inicial de la cola contiene las entradas de correlación utilizadas más recientemente y la parte final empieza con las entradas de correlación menos utilizadas recientemente. Por ejemplo, la entrada de la memoria caché A se utiliza 50 veces, y la entrada de la memoria caché B se utiliza sólo una después de la entrada de la memoria caché A. En esta situación, la entrada de la memoria caché B se coloca en la parte delantera de la cola porque se ha utilizado recientemente, mientras que la entrada de la memoria caché A se coloca al final de la cola. El desalojador LRU desaloja las entradas de la memoria caché que están en la parte final de la cola, que son las entradas de correlación menos usadas recientemente.

Propiedades de LFU y LRU y procedimientos recomendados para mejorar el rendimiento

Número de almacenamientos dinámicos

Al utilizar el desalojador LFU, todas las entradas de la memoria caché de una correlación determinada se ordenan según el número de almacenamientos dinámicos que se especifique, lo que mejora drásticamente el rendimiento e impide que se sincronicen los desalojos en un almacenamiento dinámico binario que contenga todas las ordenaciones de la correlación. A mayor número de almacenamientos dinámicos menor es el tiempo necesario para reordenarlos porque cada uno de ellos tiene menos entradas. Establezca el número de almacenamientos dinámicos en 10% del número de entradas en BaseMap.

Número de colas

Al utilizar el desalojador LRU, todas las entradas de la memoria caché de una correlación determinada se ordenan según el número de colas LRU que se especifique, lo que mejora drásticamente el rendimiento e impide que se sincronicen los desalojos en una cola que contenga todas las ordenaciones de la correlación. Establezca el número de colas en 10% del número de entradas en BaseMap.

Propiedad MaxSize

Cuando un desalojador LFU o LRU empieza a desalojar entradas, utiliza la propiedad de desalojador MaxSize para determinar cuántos almacenamientos dinámicos binarios o elementos de cola LRU va a desalojar. Por ejemplo,

presuponga que establece el número de almacenamientos dinámicos o colas en unas 10 entradas de correlación en cada cola de correlación. Si la propiedad `MaxSize` se establece en 7, el desalojador desaloja 3 entradas de cada almacenamiento dinámico u objeto de cola para que el tamaño del almacenamiento dinámico o de la cola se reduzca a 7. El desalojador sólo desaloja entradas de correlación de un almacenamiento dinámico o cola cuando en éstos hay un valor mayor que el de la propiedad `MaxSize`. Establezca `MaxSize` en 70% del tamaño de la cola o almacenamiento dinámico. En este ejemplo, el valor se estableció en 7. Puede obtener un tamaño aproximado de cada almacenamiento dinámico o cola si divide el número de entradas `BaseMap` entre el número de almacenamientos dinámicos o colas utilizado.

Propiedad `SleepTime`

Un desalojador no elimina constantemente entradas de la correlación. Está inactivo durante un tiempo determinado, y sólo comprueba la correlación cada `n` número de segundos, donde `n` equivale a la propiedad `SleepTime`. Esta propiedad afecta positivamente al rendimiento: ejecutar un barrido de desalojo suele disminuir el rendimiento debido a los recursos que se necesitan para procesarlos. Sin embargo, no utilizar el desalojador con frecuencia puede generar una correlación que tiene entradas que no son necesarias. Una correlación con muchas entradas que no necesita puede afectar negativamente a los requisitos de memoria y al proceso de los recursos de la correlación. Para la mayoría de las correlaciones se recomienda establecer el intervalo de barrido de desalojo en 15 segundos. Si la correlación se graba con frecuencia y se utiliza a una velocidad de transacción alta, conviene establecer el tiempo en un valor más bajo. Si se accede a la correlación con poca frecuencia, establezca el tiempo en un valor más alto.

Ejemplo

El ejemplo siguiente define una correlación, crea un desalojador LRU nuevo, establece las propiedades del desalojador y establece la correlación que va a utilizar el desalojador:

```
//Usar
ObjectGridManager para crear/obtener ObjectGrid. Consulte el
// apartado ObjectGridManger
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");

//Establecer propiedades presuponiendo 50.000 entradas de correlación
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Usar el desalojador LRU es muy parecido a usar un desalojador LRU. A continuación se muestra un ejemplo:

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("SomeMap");

//Establecer propiedades presuponiendo 50.000 entradas de correlación
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Observe que sólo hay dos líneas distintas al del ejemplo del desalojador LRU.

Procedimientos recomendados para optimizar el rendimiento de bloqueos

Los valores de las estrategias de bloqueo y del aislamiento de transacciones afectan el rendimiento de las aplicaciones.

Recuperar una instancia almacenada en memoria caché

Consulte la información sobre el bloqueo de entrada de correlación en *Guía de administración* si desea más información.

Estrategia de bloqueo pesimista

Utilice la estrategia de bloqueo pesimista en operaciones de correlación de lectura y grabación donde las claves suelen colisionar. La estrategia de bloqueo pesimista tiene un gran impacto sobre el rendimiento.

Aislamiento de transacciones de lectura confirmada y no confirmada

Si utiliza la estrategia de bloqueo pesimista, establezca el nivel de aislamiento de transacción a través del método `Session.setTransactionIsolation`. Para el aislamiento confirmado de lectura o no confirmado de lectura, utilice los argumentos `Session.TRANSACTION_READ_COMMITTED` o `Session.TRANSACTION_READ_UNCOMMITTED` en función del aislamiento. Para restablecer el nivel de aislamiento en el comportamiento de bloqueo pesimista predeterminado, utilice el método `Session.setTransactionIsolation` con el argumento `Session.REPEATABLE_READ`.

El aislamiento de lectura confirmada reduce la duración de los bloqueos compartidos, que pueden mejorar la simultaneidad y reducir la probabilidad de puntos muertos. Este nivel de aislamiento debe utilizarse cuando una transacción no necesita la constatación de que los valores de lectura permanecen sin modificar durante la transacción.

Utilice una lectura no confirmada cuando la transacción no necesita ver los datos confirmados.

Estrategia de bloqueo optimista

El bloqueo optimista es la configuración predeterminada. Esta estrategia mejora el rendimiento y la escalabilidad en comparación con la estrategia pesimista. Utilice esta estrategia cuando las aplicaciones puedan tolerar anomalías de actualización optimista y el rendimiento siga siendo mejor que el de la estrategia pesimista. Esta estrategia es excelente en operaciones de lectura y aplicaciones con actualizaciones poco frecuentes.

Plug-in OptimisticCallback

La estrategia de bloqueo optimista realiza una copia de las entradas de la memoria caché y las compara como sea preciso. Esta operación puede resultar costosa porque la copia de la entrada podría implicar la clonación o serialización. Para conseguir el rendimiento más rápido posible, implemente el plug-in personalizado para las correlaciones que no son de entidad.

Consulte si desea más información. Consulte la información sobre el plug-in OptimisticCallback en *Visión general del producto* si desea más información.

Uso de campos de versión para entidades

Cuando utilice el bloqueo optimista con entidades, utilice la anotación @Version o el atributo equivalente en el archivo de descriptor de metadatos de entidad. La anotación de versión proporciona a ObjectGrid una forma muy eficiente de realizar el seguimiento de la versión de un objeto. Si la entidad no tiene un campo de versión y se utiliza un bloqueo optimista para la entidad, debe copiarse y compararse toda la entidad.

Estrategia de ningún bloqueo

Utilice esta estrategia en aplicaciones de sólo lectura. La estrategia de ningún bloqueo no obtiene ningún bloqueo ni usa un gestor de bloqueos. Por lo tanto, esta estrategia ofrece el rendimiento y la escalabilidad con más concurrencia.

Rendimiento de serialización

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

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

Escribir un ObjectTransformer para cada BackingMap

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

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

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

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

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

Uso de entidades

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

Serialización personalizada

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

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

Matrices de bytes

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

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

Procedimientos recomendados para la interfaz ObjectTransformer

La interfaz ObjectTransformer utiliza devoluciones de llamadas a la aplicación para proporcionar implementaciones personalizadas de operaciones comunes y costosas, como la serialización y la copia exacta de objetos.

Visión general

Si desea detalles sobre la interfaz `ObjectTransformer`, consulte “Plug-in `ObjectTransformer`” en la página 230. Desde un punto de vista de rendimiento, y a partir de la información del método `CopyMode` que está en el tema de procedimientos recomendados del método `CopyMode`, eXtreme Scale copia claramente los valores para todos los casos excepto cuando se utiliza la modalidad `NO_COPY`. El mecanismo de copia predeterminado que se emplea en eXtreme Scale es la serialización, que se sabe que es una operación costosa. La interfaz `ObjectTransformer` se utiliza en esta situación. La interfaz `ObjectTransformer` utiliza las devoluciones de llamada a la aplicación para proporcionar una implementación personalizada de las operaciones comunes y costosas como, por ejemplo, la serialización de objeto y la copia exacta de objetos.

Una aplicación puede proporcionar una implementación de la interfaz `ObjectTransformer` en una correlación y, a continuación, eXtreme Scale delega en los métodos de este objeto y se basa en la aplicación para proporcionar una versión optimizada de cada método de la interfaz. La interfaz `ObjectTransformer` actúa del modo siguiente:

```
public interface ObjectTransformer {
    void serializeKey(Object key, ObjectOutputStream stream) throws IOException;
    void serializeValue(Object value, ObjectOutputStream stream) throws IOException;
    Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object copyValue(Object value);
    Object copyKey(Object key);
}
```

Puede asociar una interfaz `ObjectTransformer` con una `BackingMap` utilizando el siguiente código de ejemplo:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ajuste de la serialización e inflado de objetos

Normalmente la serialización de objetos es la consideración de rendimiento más importante con eXtreme Scale, que utiliza el mecanismo serializable predeterminado, si la aplicación no proporciona un plug-in `ObjectTransformer`. Una aplicación puede proporcionar implementaciones de `readObject` y `writeObject` `Serializable` o tener objetos que implementen la interfaz `Externalizable`, que es unas 10 veces más rápida. Si no se pueden modificar los objetos de la correlación, entonces una aplicación puede asociar una interfaz `ObjectTransformer` a la `ObjectMap`. Se proporcionan los métodos `serialize` e `inflate` para permitir que la aplicación proporcione código personalizado para optimizar estas operaciones, dado el gran impacto que tienen en el rendimiento del sistema. El método `serialize` serializa el objeto en la corriente de datos proporcionada. El método `inflate` proporciona la corriente de datos de entrada y espera que la aplicación cree el objeto, lo infle utilizando los datos de la corriente y devuelva el objeto. Las implementaciones de los métodos `serialize` e `inflate` deben duplicarse entre sí.

Ajuste de operaciones de copia exacta

Después de que una aplicación reciba un objeto de un `ObjectMap`, eXtreme Scale realiza una copia exacta del valor de objeto para garantizar que la copia de la correlación `BaseMap` mantiene la integridad de datos. La aplicación puede entonces modificar el valor de objeto sin riesgos. Cuando se confirma la transacción, se actualiza la copia del valor de objeto de la correlación `BaseMap` al

nuevo valor modificado y se detiene la aplicación, que utilizará el valor de ahí en adelante. Podría haber vuelto a copiar el objeto en la fase de confirmación para realizar una copia privada. Sin embargo, en este caso, el coste del rendimiento de esta acción se ha compensado indicando al programador de aplicaciones que no utilice el valor después de que se confirme la transacción. El ObjectTransformer predeterminado intenta utilizar un clon o un par de métodos serialize e inflate para generar una copia. El par de métodos serialize e inflate es el peor caso de rendimiento. Si la creación de perfiles revela que usar los métodos serialize e inflate es un problema para la aplicación, escriba un método clone apropiado para crear una copia exacta. Si no puede alterar la clase, cree un plug-in ObjectTransformer personalizado e implemente métodos copyValue y copyKey más eficaces.

Capítulo 11. Resolución de problemas

Además de los registros y el rastreo, los mensajes y las notas de release que se describen en este apartado, puede utilizar herramientas de supervisión para descubrir cuestiones como, por ejemplo, la ubicación de los datos en el entorno, la disponibilidad de los servidores en la cuadrícula, etc. Si está trabajando en un entorno WebSphere Application Server, podrá utilizar la infraestructura PMI (Performance Monitoring Infrastructure). Si está trabajando en un entorno autónomo, podrá utilizar una herramienta de supervisión de proveedor como, por ejemplo, CA Wily Introscope o Hyperic HQ. También puede utilizar y personalizar el programa de utilidad xsAdmin de ejemplo para mostrar la información textual sobre el entorno.

Registros y rastreo

Puede utilizar los registros y el rastreo para supervisar y resolver problemas del entorno. Los registros están en distintas ubicaciones en función de su configuración. Es posible que sea necesario proporcionar el rastreo para un servidor cuando se trabaja con el servicio de soporte IBM.

Registros con WebSphere Application Server

Consulte WebSphere Application Server Information Center para obtener más información.

Registros con WebSphere eXtreme Scale en un entorno autónomo

Con servidores de catálogo autónomo y de contenedor, establezca la ubicación de los registros y toda especificación de rastreo. Los registros del servidor de catálogo se encuentran en la ubicación en la que ejecutó el mandato de inicio de servidor.

Establecimiento de la ubicación de registro para los servidores de contenedor

De forma predeterminada, los registros de un contenedor están en el directorio en el que se ha ejecutado el mandato de servidor. Si inicia los servidores en el directorio `<inicio_extremeScale>/bin`, los registros y los archivos de rastreo se encuentran en el directorio `logs/<nombre_servidor>` del directorio `bin`. Para especificar una ubicación alternativa de los registros de un servidor de contenedor, cree un archivo de propiedades, como el archivo `server.properties`, con el siguiente contenido:

```
workingDirectory=<directorio>
traceSpec=
systemStreamToFileEnabled=true
```

La propiedad `workingDirectory` es el directorio raíz para los registros y el archivo de rastreo opcional. WebSphere eXtreme Scale crea un directorio con el nombre del servidor de contenedor con un archivo `SystemOut.log`, un archivo `SystemErr.log` y un archivo de rastreo si el rastreo se ha habilitado con la opción `traceSpec`. Para utilizar un archivo de propiedades durante el inicio del contenedor, utilice la opción `-serverProps` y proporcione la ubicación del archivo de propiedades de servidor.

Los mensajes de información común que se deben buscar en el archivo SystemOut.log son mensajes de confirmación de inicio. Si desea más información sobre un mensaje específico, consulte “Mensajes” en la página 394.

Rastreo con WebSphere Application Server

Consulte WebSphere Application Server Information Center para obtener más información.

Rastreo en el servicio de catálogo

Puede establecer el rastreo en un servicio de catálogo utilizando los parámetros **-traceSpec** y **-traceFile** durante el inicio del servicio de catálogo. Por ejemplo:

```
startOgServer.sh catalogServer -traceSpec
ObjectGridPlacement=all=enabled -traceFile
/home/user1/logs/trace.log
```

Si inicia el servicio de catálogo en el directorio `<inicio_eXtremeScale>/bin`, los registros y los archivos de rastreo estarán en un directorio `logs/<nombre_servicio_catálogo>` en el directorio `bin`. Consulte la información sobre cómo iniciar el proceso del servicio de catálogo en un entorno autónomo en *Guía de administración*.

Rastreo en un servidor de contenedor autónomo

Puede habilitar el rastreo en un servidor de contenedor de dos formas. Puede crear un archivo de propiedades de servidor tal como se explica en la sección de registros, o puede habilitar el rastreo utilizando la línea de mandatos durante el inicio. Para habilitar el rastreo de contenedor con un archivo de propiedades de servidor, actualice la propiedad **traceSpec** con la especificación de rastreo necesaria. Para habilitar el rastreo de contenedor utilizando parámetros de inicio, utilice los parámetros **-traceSpec** y **-traceFile**. Por ejemplo:

```
startOgServer.sh c0 -objectGridFile ../xml/myObjectGrid.xml
-deploymentPolicyFile ../xml/myDepPolicy.xml -catalogServiceEndpoints
server1.rchland.ibm.com:2809 -traceSpec
ObjectGridPlacement=all=enabled -traceFile /home/user1/logs/trace.log
```

Si inicia el servidor en el directorio `<inicio_eXtremeScale>/bin`, los registros y archivos de rastreo se encuentran en los directorios `logs/<nombre_servidor>` en el directorio `bin`

Consulte

Rastreo con la interfaz ObjectGridManager

Otra opción es establecer el rastreo durante la ejecución de una interfaz ObjectGridManager. Si se establece el rastreo en una interfaz ObjectGridManager, se puede utilizar para obtener el rastreo en un cliente de eXtreme Scale, mientras se conecta a eXtreme Scale y confirma transacciones. Para establecer el rastreo en una interfaz ObjectGridManager, proporcione una especificación de rastreo y un registro de rastreo.

```
ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
...
manager.setTraceEnabled(true);
manager.setTraceFileName("logs/myClient.log");
manager.setTraceSpecification("ObjectGridReplication=all=enabled");
```

Habilitación del rastreo con el programa de utilidad xsadmin

Para habilitar el rastreo con el programa de utilidad xsadmin, utilice la opción **setTraceSpec**. Utilice el programa de utilidad xsadmin para habilitar el rastreo en un entorno autónomo durante la ejecución, en lugar de durante el arranque. Puede habilitar el rastreo en todos los servidores y servicios de catálogo o puede filtrar los servidores basándose en el nombre de ObjectGrid, etc. Por ejemplo, para habilitar el rastreo de ObjectGridReplication con acceso al servidor del servicio de catálogo, ejecute:

```
<inicio_eXtremeScale>/bin>xsadmin.bat -setTraceSpec "ObjectGridReplication=all=enabled"
```

También puede inhabilitar el rastreo estableciendo la especificación de rastreo en `*=all=disabled`.

Consulte la información sobre el programa de utilidad xsAdmin en *Guía de administración* si desea más información.

Archivos y directorio ffdc

Los archivos FFDC sirven para que el servicio de soporte de IBM ayude a realizar la depuración. El servicio de soporte de IBM puede solicitar estos archivos si se produce un problema.

Estos archivos están en un directorio denominado ffdc, y contienen archivos que se parecen al siguiente:

```
server2_exception.log  
server2_20802080_07.03.05_10.52.18_0.txt
```

Opciones de rastreo

Puede habilitar el rastreo para proporcionar información sobre el entorno al servicio de soporte de IBM.

Sobre el rastreo

El rastreo de WebSphere eXtreme Scale se divide en varios componentes distintos. De forma similar al rastreo de WebSphere Application Server, puede especificar el nivel de rastreo que se debe utilizar. Los niveles comunes de rastreo son: all, debug, entryExit y event.

A continuación se muestra una serie de rastreo de ejemplo:

```
ObjectGridComponent=level=enabled
```

Puede concatenar valores de rastreo. Utilice el símbolo * (asterisco) para especificar un valor comodín como, por ejemplo, `ObjectGrid*=all=enabled`. Si necesita proporcionar un rastreo al servicio de soporte de IBM, se solicita una serie de rastreo específica. Por ejemplo, si hay un problema con la réplica, se puede solicitar la serie de rastreo `ObjectGridReplication=debug=enabled`.

Especificación de rastreo

ObjectGrid

Motor de memoria caché principal general.

ObjectGridCatalogServer

Servicio de catálogo general.

- ObjectGridChannel**
Comunicaciones de topología de despliegue estática.
- 7.1+ ObjectGridClientInfo**
Información del cliente DB2
- 7.1+ ObjectGridClientInfoUser**
Información del usuario de DB2.
- ObjectgridCORBA**
Comunicaciones de topología de despliegue dinámica.
- ObjectGridDataGrid**
API de AgentManager.
- ObjectGridDynaCache**
El proveedor de la memoria caché dinámica de WebSphere eXtreme Scale.
- ObjectGridEntityManager**
La API de EntityManager. Utilízela con la opción Projector.
- ObjectGridEvalidators**
Desalojadores incorporados de ObjectGrid.
- ObjectGridJPA**
Cargadores JPA (Java Persistence API).
- ObjectGridJPACache**
Plug-ins de memoria caché JPA.
- ObjectGridLocking**
Gestor de bloqueos de entradas de memoria caché de ObjectGrid.
- ObjectGridMBean**
Beans de gestión.
- 7.1+ ObjectGridMonitor**
Infraestructura de supervisión histórica.
- ObjectGridPlacement**
Servicio de colocación de fragmentos de servidor de catálogo.
- ObjectGridQuery**
Consulta de ObjectGrid.
- ObjectGridReplication**
Servicio de réplica.
- ObjectGridRouting**
Detalles de direccionamiento de cliente/servidor.
- ObjectGridSecurity**
Rastreo de seguridad.
- ObjectGridStats**
Estadísticas de ObjectGrid.
- ObjectGridStreamQuery**
La API de consulta de secuencia.
- ObjectGridWriteBehind**
Escritura diferida de ObjectGrid
- Projector**
El motor dentro de la API de EntityManager.

QueryEngine

El motor de consulta para la API de consulta de objetos y la API de consulta de EntityManager.

QueryEnginePlan

Diagnósticos del plan de consulta.

IBM Support Assistant para WebSphere eXtreme Scale

Puede utilizar IBM Support Assistant para recopilar los datos, analizar los síntomas y acceder a la información sobre el producto.

IBM Support Assistant Lite

IBM Support Assistant Lite para WebSphere eXtreme Scale proporciona una recopilación automática de los datos y soporte de análisis de síntomas para los casos de determinación de problemas.

IBM Support Assistant Lite reduce el tiempo que lleva reproducir un problema con los niveles de rastreo establecidos correctos de fiabilidad, disponibilidad y capacidad de servicio (la herramienta establece automáticamente los niveles de rastreo) para simplificar la determinación de problemas. Si necesita más asistencia, IBM Support Assistant Lite reduce también el esfuerzo necesario para enviar la información de registro adecuada a IBM Support.

IBM Support Assistant Lite se incluye en todas las instalaciones de WebSphere eXtreme Scale Versión 7.1.0

IBM Support Assistant

IBM® Support Assistant (ISA) proporciona un acceso rápido a los recursos del producto, formación y soporte que pueden ayudarle a contestar las preguntas y a resolver los problemas con los productos de software de IBM por sí solo, sin necesidad de ponerse en contacto con IBM Support. Distintos plug-ins específicos del producto le permiten personalizar IBM Support Assistant para los productos concretos que ha instalado. IBM Support Assistant recopila además los datos del sistema, los archivos de registro y otra información para ayudar a IBM Support a determinar la causa de un problema concreto.

IBM Support Assistant es un programa de utilidad para instalarlo en la estación de trabajo, no directamente en el sistema servidor WebSphere eXtreme Scale en sí. Los requisitos de memoria y de recursos para Assistant podrían afectar negativamente al rendimiento del sistema servidor WebSphere eXtreme Scale. Los componentes de diagnóstico portátiles incluidos están diseñados para un impacto mínimo en la operación normal de un servidor.

Puede utilizar IBM Support Assistant para que le ayude de estos modos:

- Para buscar en las fuentes de información y de conocimientos de IBM y no IBM entre varios productos de IBM para contestar una pregunta o solucionar un problema
- Para encontrar información adicional en los recursos web específicos del producto; incluidas las páginas iniciales del producto y de soporte, los foros y los grupos de noticias de clientes, las capacidades y los recursos de formación y la información sobre resolución de problemas y preguntas más frecuentes

- Para ampliar la capacidad para diagnosticar los problemas específicos del producto con herramientas de diagnóstico orientadas disponibles en Support Assistant
- Para simplificar la recopilación de datos de diagnóstico para ayudarle a usted y a IBM a resolver los problemas (recopilando datos generales o específicos del síntoma o producto)
- Para ayudarle a informar de las incidencias de problemas a IBM Support mediante una interfaz en línea personalizada para adjuntar los datos de diagnóstico mencionados anteriormente o cualquier otra información a las incidencias nuevas o existentes.

Finalmente, puede utilizar el recurso actualizador incorporado para obtener soporte de los productos y las capacidades de software adicionales a medida que están disponibles. Para configurar IBM Support Assistant para utilizarlo con WebSphere eXtreme Scale, instale en primer lugar IBM Support Assistant con los archivos proporcionados en la imagen descargada de la página web Visión general de soporte de IBM en: http://www-947.ibm.com/support/entry/portal/Overview/Software/Other_Software/IBM_Support_Assistant. A continuación, utilice IBM Support Assistant para ubicar e instalar las actualizaciones del producto. Puede elegir también instalar los plug-ins disponibles para otro software de IBM en el entorno. Hay disponible más información y la última versión de IBM Support Assistant desde la página web de IBM Support Assistant en la dirección: <http://www.ibm.com/software/support/isa/>.

Mensajes

Cuando encuentre un mensaje en un registro u otras partes de la interfaz del producto, puede buscar el mensaje por su prefijo de componente para descubrir más información.

Cómo encontrar mensajes

Cuando encuentre un mensaje en un registro, copie el número de mensaje con su prefijo de letra y el número y búsquelo en el centro de información (por ejemplo, CWOBJ15261). Cuando busque el mensaje, podrá encontrar una explicación adicional del mensaje y las posibles acciones que puede llevar a cabo para resolver el problema.

En Information Center encontrará un índice de los mensajes del producto.

Notas del release

Se proporcionan enlaces al sitio web de soporte del producto, a la documentación del producto y a las últimas actualizaciones, limitaciones y problemas conocidos.

- “Acceso a las actualizaciones, limitaciones y problemas conocidos más recientemente” en la página 395
- “Acceso a los requisitos de software y del sistema” en la página 395
- “Acceso a documentación del producto” en la página 395
- “Acceso al sitio web de soporte de producto” en la página 395
- “Cómo ponerse en contacto con el servicio de soporte de software de IBM” en la página 395

Acceso a las actualizaciones, limitaciones y problemas conocidos más recientemente

Las notas de release están disponibles en el sitio de soporte del producto como notas técnicas. Para ver una lista de todas las notas técnicas para WebSphere eXtreme Scale, vaya a la página web de soporte. Al pulsar los enlaces proporcionados aquí se realiza una búsqueda de las notas del release de interés en la página web de soporte, que se devolverán en una lista.

- **7.1+** Para ver una lista de las notas del release para la versión 7.1, visite la página web de soporte.
- Para ver una lista de las notas del release para la versión 7.0, visite la página web de soporte.
- Para ver una lista de las notas del release para la versión 6.1, vaya a la página wiki de las notas del release.

Acceso a los requisitos de software y del sistema

Los requisitos de hardware y software aparecen documentados en las páginas siguientes:

- Requisitos de sistema detallados

Acceso a documentación del producto

Para obtener todo el conjunto de información, vaya a la página de la biblioteca.

Acceso al sitio web de soporte de producto

Para localizar las últimas notas técnicas, descargas, arreglos y otra información relacionada con el soporte, vaya a la página de soporte técnico.

Cómo ponerse en contacto con el servicio de soporte de software de IBM

Si encuentra un problema con el producto, primero intente llevar a cabo las siguientes acciones:

- Siga los pasos descritos en la documentación del producto
- Busque la documentación relacionada en la ayuda en línea
- Busque los mensajes de error en la consulta de mensajes

Si no puede resolver el problema con ninguno de los métodos citados anteriormente, póngase en contacto con el servicio de IBM.

Avisos

Las referencias en esta publicación a productos, programas o servicios de IBM no implica que IBM tenga previsto ponerlos a la venta en todos los países en los que IBM opera. Cualquier referencia a un producto, programa o servicio de IBM no pretende indicar ni implica que sólo se pueda utilizar este producto, programa o servicio de IBM. En su lugar, se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ningún derecho de propiedad intelectual de IBM. La evaluación y la verificación del funcionamiento con otros productos, excepto aquellos expresamente designados por IBM, es responsabilidad del usuario.

IBM puede tener patentes o solicitudes de patentes pendientes que conciernan al tema de este documento. La posesión de este documento no le da ninguna licencia sobre estas patentes. Puede enviar preguntas acerca de licencias por escrito a:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 Estados Unidos

Los propietarios de licencias de este programa que deseen obtener información sobre el mismo con el fin de habilitar: (i) el intercambio de información entre programas creados de forma independiente y otros programas (incluido este) y (ii) el uso mutuo de la información intercambiada, se deben poner en contacto con:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
Estados Unidos
Attention: Information Requests

Esta información puede estar disponible, bajo las condiciones y los términos adecuados, incluyendo en algunos casos, el pago de una cuota.

Marcas registradas

Los siguientes términos son marcas registradas de IBM Corporation en Estados Unidos y en otros países.

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java y todas las marcas registradas basadas en Java son marcas registradas de Sun Microsystems, Inc. en Estados Unidos y/o en otros países.

LINUX es una marca registrada de Linus Torvalds en Estados Unidos y/o en otros países.

Microsoft, Windows[®], Windows NT[®] y el logotipo de Windows son marcas registradas de Microsoft Corporation en Estados Unidos y/o en otros países.

UNIX[®] es una marca registrada de The Open Group en Estados Unidos y en otros países.

Otros nombres de compañías, productos y servicios pueden ser marcas registradas o de servicio de terceros.

Índice

A

- Acceder 32
- acceso a los datos
 - consultas 32
 - datos almacenados 32
 - particiones 32
 - transacciones 32
- agente de instrumentación 88
- aislamiento
 - bloqueo pesimista 167
 - lectura repetible 167
 - para transacciones 167
- almacenamientos dinámicos 226, 381
- API 287
- API de administración 289
- API de estadísticas 15, 134, 279, 327, 337
- API del sistema 211
- API EntityManager 62
- API ObjectMap
 - API 50
 - API ObjectMap 50
- API seguridad 337
- autónomo 185
- autorización 179, 356
- autorización de cuadrícula 364

B

- bean de ampliación 328
- Beans de ampliación de Spring 332
- bloqueo
 - estrategias de 140, 160
 - optimista 140, 160
 - pesimista 140, 160
- bloqueo actualizable 150
- bloqueo compartido 150
- bloqueo exclusivo 150
- bloqueos
 - ciclo de vida 150
 - compatibilidad 150
 - tiempo de espera 150
- bloqueos de entrada de correlación
 - consulta 165
 - índices 165

C

- cargador 175
- consideraciones de programación
 - JPA 261
 - escribir 257
 - utilización con correlaciones de entidad y tuples 265
 - visión general 255
 - Visión general de JPA (Java Persistence API) 309
- ciclo de vida de la entidad 79
- colas 226, 381
- colas FIFO
 - correlaciones 58

- consulta 248
 - ajuste
 - índices 121
 - paginación 121
 - parámetros 121
 - anomalía de cliente 91
 - atributos válidos 104
 - Backus Naur 119
 - BNF 119
 - cláusulas 110
 - cola
 - entidades en un bucle 91
 - todas las particiones 91
 - colisión de claves 91
 - correlación de objetos
 - esquema 101
 - ejemplo 109
 - elementos de búsqueda 95
 - entidad
 - recuperar resultados 106
 - esquema 104
 - esquema ObjectQuery 104
 - funciones 110
 - índice 109, 126
 - métodos 95
 - obtener plan 123
 - optimización
 - relaciones 126
 - paginación 109
 - parámetros 109
 - plan de consulta 123
 - predicados 110
- CopyMode 35, 373
- correlación 15
- correlación de respaldo
 - estrategia de bloqueo 137
 - plug-ins 18
 - sesión 18
- correlaciones de entidad
 - crear 265
- correlaciones de matrices de bytes 40, 378
- correlaciones dinámicas
 - correlaciones 54
- crear índices
 - índice compuesto 248
 - índice hash 248

D

- Datos 32
- desalojador 175
- desalojador TTL 213
- desalojadores 213
 - configuración 8, 10, 13
 - desalojador TTL 216
 - plug-in 219
- detener servidor
 - mediante programa 289
- distribuir cambios
 - usar Java Message Service 143

E

- elemento de registro 175
- entidad 62
 - ciclos de vida de 77
- EntityManager 73, 83, 109
- escribir desalojadores
 - desalojador de retrotracción 222
- escucha de entidad 79, 82
- escuchas
 - introducción 241
 - ObjectGridEventListener 244
 - para eXtreme Scale 241
 - para objetos BackingMap 241
 - Plug-in MapEventListener 242
 - Plug-in ObjectGridEventListener 244
- esquema de entidad
 - entidad 62
- estadística 292

F

- FetchPlan 83
- flujo web 328

G

- gestor de entidades 83
 - guía de aprendizaje 95
- gestor de transacciones externas 282

I

- IBM Support Assistant 393
- índice
 - acceso a los datos 251
 - de no clave 251
 - devolución de llamada 251
- iniciar servidor
 - mediante programa 289
- inicio de servidores 185
- Interfaz EntityManager
 - rendimiento 87
- Interfaz EntityTransaction 95
- interfaz JavaMap 58
- Interfaz ObjectGrid 15
- interfaz ObjectGridManager
 - control del ciclo de vida con 29
 - habilitación del rastreo con 389
- interfaz ObjectMap 51

J

- Java Authentication and Authorization Service
 - JAAS 364
- Java Persistence API (JPA)
 - actualizador basado en la hora
 - inicio 322

Java Persistence API (JPA) (*continuación*)
actualizador de datos basado en la hora
visión general 320
mediante eXtreme Scale
visión general 309
Plug-in JPAEntityLoader
introducción 263
programa de utilidad de precarga
visión general 311
programa de utilidad de precarga basada en cliente
programación 313
JVM 371

L

LogElement 175
LogSequence 175

M

manejo de excepciones 169
mensajes 394
metadatos de entidad
Archivo emd.xsd 70
configuración de XML 70
Método batchUpdate 265
Método get 265
Métodos removeObjectGrid 28

N

notas del release 394

O

ObjectGridManager 23
ObjectTransformer
procedimientos recomendados para 235, 386
objetos de tuple
crear 265

P

particiones
transacciones 144
Performance Monitoring
Infraestructure 294, 295, 299
Performance Monitoring Infrastructure (PMI) 15, 134, 279, 327, 337
plug-in 15
índice 246
introducción 211
introducción a 211
OptimisticCallback 237
Plug-in ObjectTransformer 230
Plug-in
WebSphereTransactionCallback 284
ranuras de plug-in 279
TransactionCallback 275
PMI i, 299
Véase también Performance Monitoring Infrastructure

PMI (*continuación*)

MBean 15, 134, 279, 327, 337
precarga de réplica 270
procedimientos recomendados 226, 381
Programación de eXtreme Scale 7
puntos muertos
escenarios de 150

R

rastreo
opciones para configurar 391
visión general 389
receptores de sucesos 241
registros
visión general 389
rendimiento 226, 371, 381
bloqueo 163, 383
procedimientos recomendados 163, 383
resolución de problemas 389
mensajes 394
notas del release 394

S

secuencia de registro 175
seguridad 179
local 364
plug-ins 364
serialización
bloqueo 234, 384
rendimiento 234, 384
servidor de catálogo
habilitación de rastreo 389
habilitación de registros 389
servidor de contenedor
habilitación de rastreo 389
habilitación de registros 389
sesión 15
colisión 169
transacción 169
sesiones
datos de acceso
cuadrícula 42
flush 42
SessionHandle
direccionamiento 47
solicitud
direccionamiento 47
por contenedor 47
sesión
SessionHandle 47
soporte 393, 394
Spring 328
ámbito de fragmento 327
bean de ampliación 327
empaquetado 327
flujo web 327
infraestructura 327
soporte de espacio de nombres 327
transacciones nativas 327
supervisar 294
con la API de estadísticas 292

T

transacción 15, 282
transacciones
con sesiones 134
cuadrícula cruzada 144
partición única 144
ventajas de 134
visión general 134, 136
transacciones de partición 144
transacciones nativas 328



Impreso en España