WebSphere® WebSphere Extended Deployment Versión 6.0.x

Guía del modelo de programación de ObjectGrid

Nota

Antes de utilizar esta información, asegúrese de leer la información general bajo “Avisos” en la página 383.

Fecha de compilación: 27 Marzo 2006

Esta publicación es la traducción del original inglés *IBM WebSphere WebSphere Extended Deployment Version 6.0: ObjectGrid programming model guide*.

Contenido

Cómo se envían los comentarios

Sus comentarios son importantes porque pueden ayudarnos a proporcionar una información más precisa y de máxima calidad.

- Para enviar sus comentarios sobre artículos en el centro de información de WebSphere Extended Deployment, disponible en la dirección: `http://www.ibm.com/software/webservers/appserv/extend/library/`
  1. Visualice el artículo en el navegador Web y avance la página hasta el final del artículo.
  2. Rellene el enlace **Comentarios** al final del artículo y envíelo.
- Para enviar comentarios sobre éste y otros manuales en PDF, puede hacerlo por correo electrónico a la dirección: **hojacom@es.ibm.com**.

  Asegúrese de incluir el nombre y número de documento y, si es aplicable, el número de página, tabla o figura al que aluden los comentarios.

Cuando envía información a IBM, otorga a IBM el derecho no exclusivo a utilizar o distribuir la información de cualquier modo que crea que sea adecuado sin incurrir en ninguna obligación hacia usted.

Capítulo 1. Iniciación a ObjectGrid ejecutando la aplicación de ejemplo

Utilice este tema para iniciarse con ObjectGrid, una infraestructura de sistemas distribuidos que pone los objetos a disposición de un conjunto de aplicaciones.

WebSphere Extended Deployment Versión 6.0 o posterior y WebSphere Application Server Versión 6.0.2 o posterior deben instalarse en una máquina del entorno como mínimo.

Restricción: Si está utilizando ObjectGrid con WebSphere Extended Deployment Versión 6.0, se precisan modificaciones adicionales de los acuerdos de licencia para utilizar ObjectGrid en J2SE (Java 2 Platform, Standard Edition) Versión 1.4.2 o entornos superiores o en WebSphere Application Server Versión 6.0.2 o entornos superiores. Póngase en contacto con el representante de ventas para obtener información detallada.

Si desea desarrollar las aplicaciones ObjectGrid sin acceder a las máquinas de servidor que tengan instalado WebSphere Extended Deployment, puede ejecutarlas en la máquina local. La máquina local requiere la instalación de IBM Software Developer Kit (SDK) o Eclipse.

Para desarrollar aplicaciones ObjectGrid en su equipo local, copie los directorios siguientes de la instalación al equipo local:

- Si está utilizando WebSphere Extended Deployment Versión 6.0.1, copie los archivos /lib/wsobjectgrid.jar y /optionalLibraries/ObjectGrid/objectgridSamples.jar en el directorio de trabajo.
- Si ha instalado ObjectGrid mediante la instalación para un entorno de distintos servidores, copie los archivos /ObjectGrid/lib/objectgrid.jar y /ObjectGrid/samples/objectgridSamples.jar en el directorio de trabajo.

Para obtener más información sobre los archivos Java (JAR) que se instalan con ObjectGrid, consulte Paquete de ObjectGrid.

Utilice esta tarea para ejecutar las aplicaciones ObjectGrid de ejemplo y seguir los pasos correspondientes. Puede ejecutar las aplicaciones de esta tarea en un entorno de línea de mandatos Java, de Eclipse o de J2EE (Java 2 Platform, Enterprise Edition).

- Para que la aplicación ObjectGrid de ejemplo se ejecute en la línea de mandatos, consulte Ejecución de la aplicación ObjectGrid de ejemplo en la línea de mandatos.
- Para ejecutar la aplicación ObjectGrid de ejemplo en Eclipse, consulte Importación de la aplicación ObjectGrid de ejemplo a Eclipse y cómo se utiliza.
- Para ejecutar la aplicación ObjectGrid de ejemplo en WebSphere Extended Deployment, consulte Carga y ejecución de la aplicación ObjectGrid de ejemplo con WebSphere Extended Deployment

Se ha iniciado con ObjectGrid mediante la ejecución de la aplicación de ejemplo y la carga del ejemplo en el entorno de desarrollo.

Ejecución de la aplicación ObjectGrid de ejemplo en la línea de mandatos

Utilice este tema para ejecutar las aplicaciones habilitadas para ObjectGrid en una línea de mandatos Java y pruebe la configuración de ObjectGrid.

Antes de empezar esta tarea, instale el entorno de distintos servidores, incluido la ObjectGrid autónoma.

Debe tener instalado Software Development Kit (SDK). Asimismo, debe tener acceso a las aplicaciones ObjectGrid de ejemplo. Consulte Guía de iniciación con ObjectGrid para obtener más información.

Utilice esta tarea para ejecutar una aplicación que tenga ObjectGrid habilitado con rapidez.

1. Compruebe la versión de Software Development Kit (SDK). ObjectGrid requiere IBM SDK 1.4.2 o superior. Para probar el entorno Java antes de ejecutar la aplicación ObjectGrid de ejemplo, realice los pasos siguientes:
   a. Abra un indicador de línea de mandatos.
   b. Escriba el siguiente mandato:

      ```
      java -version
      ```

      Si el mandato se ejecuta correctamente, aparecerá un texto parecido al siguiente ejemplo:

      ```
      java version "1.4.2"
      Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2)
      Classic VM (build 1.4.2, J2RE 1.4.2 IBM Windows 32 build cn142-20040820
      (JIT enabled: jitc))
      ```

      Nota: También puede ejecutar estos ejemplos mediante un SDK (Software Development Kit) de J2SE (Java 2 Platform, Standard Edition) Versión 1.3.x. Para obtener más información, consulte el tema Paquete de ObjectGrid.

      Si se visualiza un error, asegúrese de que el SDK esté instalado y esté en la CLASSPATH.
2. Ejecute la aplicación ObjectGrid de ejemplo. La aplicación de ejemplo ilustra un caso sencillo que implica a empleados, oficinas y ubicaciones de trabajo. La aplicación de ejemplo crea una instancia de ObjectGrid con correlaciones para todos los tipos de objeto. Todas las correlaciones tienen entradas insertadas y manipuladas para demostrar la función de colocación en antememoria de ObjectGrid.
   a. Abra una línea de mandatos y desplácese hasta el directorio de trabajo. Copie los archivos objectgrid.jar, asm.jar y cglib.jar de la carpeta /ObjectGrid/lib en un directorio de trabajo. Copie /ObjectGrid/samples/objectgridSamples.jar en el directorio de trabajo.
   b. Emita el siguiente mandato:

      ```
      cd
      directorio_trabajo
      java -cp "objectgrid.jar;objectgridSamples.jar;asm.jar;cglib.jar"
      com.ibm.websphere.samples.objectgrid.basic.ObjectGridSample
      ```

      El sistema muestra una salida similar al siguiente texto. Esta salida se ha abreviado para su publicación.

```
Initializing ObjectGridSample ...
resourcePath: META-INF/objectgrid-definition.xml
objectgridUrl:
 jar:file:/C:/temp/objg/objectgridSample.jar!/
META-INF/objectgrid-definition.xml
EmployeeOptimisticCallback returning version object for employee
 = Perry Cheng, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Hao Lee, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Ken Huang, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Jerry Anderson, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Kevin Bockhold, version = 0
-------------------------------------------
com.ibm.websphere.samples.objectgrid.basic.ObjectGridSample status:
ivObjectGrid Name = clusterObjectGrid
ivObjectGrid = com.ibm.ws.objectgrid.ObjectGridImpl@187b81e4
ivSession = com.ibm.ws.objectgrid.SessionImpl@6b0d81e4
ivEmpMap = com.ibm.ws.objectgrid.ObjectMapImpl@6b1841e4
ivOfficeMap = com.ibm.ws.objectgrid.ObjectMapImpl@6ba081e4
ivSiteMap = com.ibm.ws.objectgrid.ObjectMapImpl@6bae01e4
ivCounterMap = com.ibm.ws.objectgrid.ObjectMapImpl@697b41e4
-------------------------------------------
interactiveMode = false
Action = populateMaps
CounterOptimisticCallback returning version object for
 counter name = Counter1, version = 0
CounterOptimisticCallback returning version object for
 counter name = Counter2, version = 0
CounterOptimisticCallback returning version object for
 counter name = Counter3, version = 0
ivCounterMap operations committed
ivOfficeMap operations committed
... ending with:
CounterOptimisticCallback returning version object for
 counter name = Counter1, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Ken Huang, version = 0
CounterOptimisticCallback returning version object for
 counter name = Counter2, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Perry Cheng, version = 0
CounterOptimisticCallback returning version object for counter name =
 Counter3, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Jerry Anderson, version = 0
CounterOptimisticCallback returning version object for
  counter name = Counter4, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Hao Lee, version = 0
EmployeeOptimisticCallback returning version object for employee =
 Kevin Bockhold, version = 1
DONE cleanup
```

3. Ejecute la aplicación ObjectGrid distribuida de ejemplo.

   El programa com.ibm.websphere.samples.objectgrid.basic.ObjectGridSample utiliza una instancia local de ObjectGrid como la antememoria de datos. Todos los objetos se almacenan en la antememoria en la máquina virtual Java (JVM) local. Para utilizar una ObjectGrid distribuida que se despliega en un clúster ObjectGrid, utilice el programa com.ibm.websphere.samples.objectgrid.distributed.DistributedObjectGridSample. El programa DistributedObjectGridSample se incluye en objectgridSamples.jar.

a. Inicie un clúster ObjectGrid. Si desea más información acerca del inicio de un clúster ObjectGrid autónomo para utilizarlo con el ejemplo de ObjectGrid distribuida, consulte Inicio del clúster ObjectGrid autónomo de ejemplo.

b. Una vez que el servidor ObjectGrid se haya iniciado, puede ejecutar la aplicación de ejemplo de ObjectGrid distribuida con el mandato siguiente:

```
java -cp "objectgrid.jar;objectgridSamples.jar;asm.jar;cglib.jar"
com.ibm.websphere.samples.objectgrid.distributed.DistributedObjectGridSample
```

Una vez que haya iniciado el clúster ObjectGrid necesario, el programa DistributedObjectGridSample tiene una salida similar al programa ObjectGridSample.

Ha ejecutado la aplicación ObjectGrid de ejemplo en una línea de mandatos Java para probar la funcionalidad de ObjectGrid.

El origen de este ejemplo se encuentra en el archivo objectgridSamples.jar, específicamente en los archivos com\ibm\websphere\samples\objectgrid\basic\ObjectGridSample.java y com\ibm\websphere\samples\objectgrid\distributed\DistributedObjectGridSample.java.

Inicio del clúster ObjectGrid autónomo de ejemplo

Para ejecutar el ejemplo de ObjectGrid distribuida, debe iniciar un clúster ObjectGrid que aloje la ObjectGrid necesaria.

Verifique que WebSphere Extended Deployment para un entorno de distintos servidores, Versión 6.0.x esté instalado.

Utilice esta tarea para iniciar un servidor ObjectGrid que esté basado en los archivos `cluster-config-1.xml` y `cluster-objectgrid-definition.xml`. Esta tarea es necesaria para ejecutar la ObjectGrid distribuida de ejemplo. Consulte Ejecución de la aplicación ObjectGrid de ejemplo en la línea de mandatos e Importación de la aplicación ObjectGrid de ejemplo a Eclipse y cómo se utiliza, si desea más información. El archivo `cluster-config-1.xml` sólo tiene una definición del servidor ObjectGrid. Este servidor ObjectGrid representa el clúster ObjectGrid de ejemplo.

1. Localice el archivo `objectgridSamples.jar` en el directorio *mse_install_root*`/ObjectGrid/samples`.
2. Extraiga el archivo `META-INF/cluster-config-1.xml` y el archivo `META-INF/cluster-objectgrid-definition.xml` del archivo `objectgridSamples.jar` al directorio *mse_install_root*`/ObjectGrid/samples`.
3. Verifique que la variable de entorno JAVA_HOME esté establecida y que la versión de Java cumple el requisito. El servidor ObjectGrid necesita un entorno J2SE (Java 2 Platform, Standard Edition) Versión 1.4.2 o posterior. Para comprobar el entorno Java, realice los pasos siguientes:
   a. Compruebe la variable de entorno JAVA_HOME. En un indicador de línea de mandatos, emita el siguiente mandato:

      ```
      echo %JAVA_HOME%
      ```

      Este mandato muestra la vía de acceso a Java. Si necesita establecer la variable de entorno JAVA_HOME, ejecute el mandato siguiente:

      ```
      set JAVA_HOME=JDK_INSTALL_ROOT
      ```

      Establezca el directorio *JDK_INSTALL_ROOT* en el directorio de instalación de Java, por ejemplo, `c:\java`.
   b. Compruebe la versión de Java. Ejecute el mandato siguiente:

```
java -version
```

Verifique que la versión es J2SE (Java 2 Platform, Standard Edition) Versión 1.4.2 o posterior.

4. Inicie el servidor ObjectGrid. En un indicador de línea de mandatos, emita los siguientes mandatos:

```
cd
mse_install_root/ObjectGrid/bin
startOgServer.bat server1 -objectgridFile mse_install_root/ObjectGrid/
samples/META-INF/cluster-objectgrid-definition.xml
-clusterFile mse_install_root/ObjectGrid/samples/META-INF/
cluster-config-1.xml
-jvmArgs -cp mse_install_root/ObjectGrid/samples/objectgridSamples.jar
```

Importante: Debe especificar el archivo `objectgridSamples.jar` en la classpath mediante la opción -jvmArgs. El archivo `objectgridSamples.jar` contiene clases que necesita el servidor ObjectGrid de ejemplo para las implementaciones de plug-in que se definen en el archivo `cluster-objectgrid-definition.xml`. Este archivo JAR se utiliza también para la serialización y deserialización de los objetos que se almacenan en correlaciones.

El sistema muestra una salida similar al siguiente texto. Esta salida se ha abreviado para su publicación.

```
************ Inicio de la visualización del entorno actual ************
[1/17/06 14:04:34:144 CST] 7daee176 Launcher
I CWOBJ2501I: Iniciando el servidor ObjectGrid server1.
:
[1/17/06 14:04:37:719 CST] 7daee176 ServerRuntime
I CWOBJ1001I: el servidor ObjectGrid server1 está preparado para procesar
peticiones.
```

Consulte Ejecución de la aplicación ObjectGrid de ejemplo en la línea de mandatos o Importación de la aplicación ObjectGrid de ejemplo a Eclipse y cómo se utiliza para ejecutar la aplicación de ejemplo de ObjectGrid distribuida. Para obtener más detalles acerca del inicio y detención del servidor ObjectGrid autónomo en la línea de mandatos, consulte Capítulo 8, “Soporte de línea de mandatos”, en la página 83.

Importación de la aplicación ObjectGrid de ejemplo a Eclipse y cómo se utiliza

Utilice esta tarea para importar la aplicación ObjectGrid de ejemplo a Eclipse y utilizarla en ese entorno.

Antes de empezar esta tarea, instale el entorno de distintos servidores, incluido la ObjectGrid autónoma.

Para esta aplicación de ejemplo, utilice Eclipse Versión 3.1 o posterior para importar y ejecutar el ejemplo. Puede obtener Eclipse a través de Application Server Toolkit, que se incluye con WebSphere Application Server, instalando Rational Application Developer o bajándolo directamente de Eclipse.org.

Si utiliza Eclipse, puede depurar fácilmente las aplicaciones. Puede seguir los distintos pasos de la aplicación de ejemplo.

1. Importe el proyecto a Eclipse:

a. Ejecute el programa Eclipse. Utilice el archivo `eclipse.exe` del directorio de instalación de Eclipse.
b. Utilizando Eclipse, cree un nuevo proyecto.
    1) Pulse **Archivo > Nuevo > Proyecto > Java > Proyecto Java**. Pulse **Siguiente**.
    2) Escriba un nombre de proyecto. Por ejemplo, escriba `ObjectGridSamples`.
    3) Seleccione **Crear proyecto nuevo en área de trabajo**.
    4) En la sección Diseño de proyecto, pulse **Configurar valor predeterminado**.
    5) Para la carpeta fuente y de salida, seleccione **Proyecto** y pulse **Aceptar**.
    6) Pulse **Siguiente**.
    7) Pulse la pestaña **Bibliotecas**.
    8) Pulse **Añadir Jars externos.**
    9) Vaya a la carpeta `/ObjectGrid/lib` y seleccione los archivos **objectgrid.jar**, **asm.jar** y **cglib.jar**. Pulse **Abrir** en el asistente para la **selección de archivos JAR**.
    10) Pulse **Finalizar**.

2. Importe el archivo `objectgridSamples.jar` al proyecto Java.
    a. Pulse con el botón derecho del ratón en el proyecto Java y seleccione **Importar**.
    b. Seleccione **Archivo zip** en **Seleccionar un origen de importación**.
    c. Pulse **Siguiente**.
    d. Pulse **Examinar** para abrir el asistente Importar de archivo zip.
    e. Abra el archivo **objectgridSamples.jar**. Vaya al directorio `/ObjectGrid/samples`. Seleccione el archivo **objectgridSamples.jar** y pulse **Abrir**.
    f. Verifique que el recuadro de selección del árbol de archivos del directorio raíz esté seleccionado.
    g. Verifique que **A la carpeta** sea el proyecto Java que haya creado en el paso anterior, por ejemplo, el proyecto `ObjectGridSamples`.
    h. Pulse **Finalizar**.
3. Compruebe las propiedades del proyecto Java.
    a. Abra la perspectiva Java. Pulse **Ventana > Abrir perspectiva > Java**.
    b. Vaya a la vista de la consola. Pulse **Ventana > Mostrar vista > Consola**.
    c. Verifique que la vista Explorador de paquetes esté disponible y seleccionada. Pulse **Ventana > Mostrar vista > Explorador de paquetes**.
    d. Pulse con el botón derecho del ratón en el proyecto Java y seleccione **Propiedades**.
    e. Pulse **Vía de construcción Java** en el panel izquierdo.
    f. Pulse la pestaña **Fuente** en el panel derecho.
    g. Verifique que la raíz del proyecto aparezca enumerada en las carpetas Fuente del panel Vía de construcción.
    h. Pulse la pestaña **Bibliotecas** en el panel derecho.
    i. Verifique que los archivos `objectgrid.jar`, `asm.jar` `cglib.jar` y una Biblioteca JRE del sistema aparezcan enumerados en las carpetas JAR y class del panel Vía de construcción.

j. Pulse **Aceptar**.

4. Ejecute la ObjectGrid de ejemplo.
   a. En la vista Explorador de paquetes, expanda el proyecto Java.
   b. Expanda el paquete com.ibm.websphere.samples.objectgrid.basic.
   c. Pulse con el botón derecho del ratón en el archivo **ObjectGridSample.java**. Pulse **Ejecutar > Aplicación Java**.
   d. La consola muestra una salida parecida a la que aparece al ejecutar la aplicación mediante la línea de mandatos Java. Para obtener un ejemplo de la salida, consulte Ejecución de la aplicación ObjectGrid de ejemplo en la línea de mandatos.
5. Ejecute la ObjectGrid distribuida de ejemplo. Para ejecutar el ejemplo de ObjectGrid distribuida, debe configurar un clúster ObjectGrid. Para ejecutar este ejemplo, puede utilizar los archivos de configuración XML que se proporcionan en el archivo objectgridSamples.jar. Consulte Inicio del clúster ObjectGrid autónomo de ejemplo si desea más información.

   Una vez que el servidor ObjectGrid se haya iniciado, puede ejecutar la aplicación de ejemplo de ObjectGrid distribuida siguiendo los pasos siguientes:
   a. En la vista Explorador de paquetes, expanda el proyecto Java.
   b. Expanda el paquete com.ibm.websphere.samples.objectgrid.distributed.
   c. Pulse con el botón derecho del ratón en el archivo **DistributedObjectGridSample.java**. Pulse **Ejecutar > Aplicación Java**.
   d. La consola muestra una salida similar al ejemplo de ObjectGrid.

Los pasos que deben seguirse para cargar y ejecutar el depurador se encuentran también en el archivo `SamplesGuide.htm`. El archivo `SamplesGuide.htm` se encuentra en el directorio `doc` del archivo `objectgridSamples.jar`.

Referencia relacionada

Paquete de ObjectGrid
Puede acceder a los paquetes de ObjectGrid de dos formas: instalando WebSphere Extended Deployment, o instalando el entorno de distintos servidores.

Carga y ejecución de la aplicación ObjectGrid de ejemplo con WebSphere Extended Deployment

Utilice esta tarea para cargar y ejecutar la aplicación ObjectGrid de J2EE (Java 2 Platform, Enterprise Edition) de ejemplo en WebSphere Extended Deployment.

WebSphere Application Server y WebSphere Extended Deployment deben estar instalados.

Utilice esta tarea para comprender y probar la integración de ObjectGrid con WebSphere Extended Deployment.Para obtener más información, consulte Capítulo 10, “Integración de ObjectGrid con WebSphere Application Server”, en la página 291.

1. Instale el archivo `ObjectGridSample.ear`. Puede instalar el archivo EAR (Enterprise Archive) en un único servidor de aplicaciones o clúster. Para instalar el archivo `ObjectGridSample.ear` en la consola administrativa, realice los pasos siguientes:
   a. En la consola administrativa, pulse **Aplicaciones > Instalar nueva aplicación**.

b. En la página **Preparación para la instalación de la aplicación**, especifique la ubicación de la aplicación ObjectGrid de ejemplo. Por ejemplo, desplácese a: `<raíz_instalación>/installableApps/ObjectGridSample.ear`. Pulse **Siguiente**.

c. En la segunda página **Preparación para la instalación de la aplicación**, tome los valores por omisión y pulse **Siguiente**.

d. En la página **Seleccionar opciones de instalación**, tome los valores por omisión y pulse **Siguiente**.

e. En la página **Correlacionar módulos con servidores**, especifique los destinos de despliegue donde desee instalar los módulos contenidos en la aplicación. Seleccione un servidor de destino o clúster de la lista **Clústeres y servidores** para cada módulo. Seleccione el recuadro de selección **Módulo** para seleccionar módulos de aplicaciones individuales o seleccionarlos todos.

f. En las siguientes páginas, utilice los valores por omisión y pulse **Finalizar**.

g. Pulse **Guardar en configuración maestra** después de completar la instalación de la aplicación.

h. Pulse la opción **Sincronizar cambios con nodos**. En la página **Aplicaciones de empresa > Guardar**, pulse **Guardar**.

i. Pulse **Aceptar**.

2. Compruebe el puerto HTTP del sistema principal por omisión de los servidores y añada un alias de sistema principal. Por omisión, los módulos Web se enlazan al nombre de sistema principal virtual del sistema principal por omisión, a menos que modifique el nombre de sistema principal durante la instalación. Si está instalando la aplicación en un clúster, debe configurar un alias de sistema principal como mínimo para el puerto HTTP del sistema principal por omisión de todos los miembros de clúster. Asimismo, debe comprobar el puerto HTTP del sistema principal por omisión de todos los miembros de clúster y añadir el correspondiente alias de sistema principal en la lista Alias de sistema principal de la consola administrativa. Para comprobar el puerto HTTP del sistema principal por omisión de un servidor, realice los pasos siguientes:

   a. En la consola administrativa, pulse **Servidores > Servidores de aplicaciones > *nombre_servidor***.

   b. Expanda los puertos de la sección Comunicación. El puerto **WC_defaulthost** es el puerto de sistema principal virtual del sistema principal por omisión.

   Para añadir un alias de sistema principal, realice los pasos siguientes:

   a. En la consola administrativa, pulse **Entorno > Sistemas principales virtuales > sistema principal por omisión > Alias de sistema principal > Nuevo**.

   b. Utilice el valor por omisión del nombre de sistema principal y especifique el puerto.

   c. Pulse **Aceptar**.

3. Inicie la aplicación ObjectGrid de ejemplo.

   - Para iniciar la aplicación en un servidor, pulse **Servidores > Servidores de aplicaciones**. Selecciones el servidor que tenga instalado el archivo `ObjectGridSample.ear`. Pulse **Iniciar**.
   - Para iniciar la aplicación en un clúster, pulse **Servidores > Clústeres**. Selecciones el clúster que tenga instalado el archivo `ObjectGridSample.ear`. Pulse **Iniciar**.

Después de iniciar la aplicación en un servidor o clúster, puede detener o iniciar la aplicación independientemente del servidor o clúster del sistema principal. Para detener o iniciar la aplicación ObjectGrid de ejemplo, realice los pasos siguientes:

a. En la consola administrativa, pulse **Aplicaciones > Aplicaciones de empresa**.
b. Seleccione la aplicación ObjectGrid de ejemplo.
c. Pulse **Iniciar** o **Detener**.

4. Acceda al ejemplo ObjectGrid. Después de instalar el archivo `ObjectGridSample.ear` en un único servidor o clúster e iniciar la aplicación, podrá acceder al ejemplo ObjectGrid en la siguiente dirección Web:

   `http://`*`nombre_sistppal`*`:`*`puerto`*`/ObjectGridSample`

   Por ejemplo, si el nombre de sistema principal es `localhost` y el valor de puerto es `9080`, utilice la dirección Web `http://localhost:9080/ObjectGridSample`.

5. Pruebe la funcionalidad de ObjectGrid distribuida en el entorno de WebSphere Application Server. El archivo `ObjectGridSample.ear` también contiene el servlet DistributedObjectGridServlet que demuestra el uso de una ObjectGrid distribuida en el entorno WebSphere Application Server. El servidor de aplicaciones que aloja el servlet DistributedObjectGridServlet también debe alojar el servidor ObjectGrid, que es un miembro del clúster ObjectGrid necesario.

   - Para obtener más información sobre la configuración de un clúster ObjectGrid para hacer que se ejecute DistributedObjectGridServlet, consulte Inicio de un clúster ObjectGrid de ejemplo en el entorno de WebSphere.
   - Para obtener más información sobre el inicio de los servidores ObjectGrid en los servidores de aplicaciones, consulte Inicio de un servidor ObjectGrid en un servidor de aplicaciones.

   Una vez que el servidor de aplicaciones con el archivo `ObjectGridSample.ear` instalado también aloja el servidor ObjectGrid necesario, el servlet DistributedObjectGridServlet se comporta de la misma forma que otros servlets. Puede acceder al servlet en la siguiente dirección Web: `http://hostname:port/ObjectGridSample/DistributedObjectGridServlet`. Por ejemplo, si el nombre de sistema principal es `localhost` y el valor de puerto es 9080, utilice la dirección Web `http://localhost:9080/ObjectGridSample/DistributedObjectGridServlet`.

   Puede habilitar el rastreo de ObjectGrid utilizando la siguiente serie de rastreo: `ObjectGrid*=all=enabled`.

Ha instalado y configurado la aplicación ObjectGrid de ejemplo y la aplicación ObjectGrid distribuida de ejemplo en un servidor WebSphere Extended Deployment.

Después de instalar la aplicación en un servidor o clúster, puede acceder a la documentación del ejemplo después de iniciar la aplicación en la siguiente dirección Web:

`http://`*`nombre_sistppal`*`:`*`puerto`*`/ObjectGridSample/docs/introduction.html`

Por ejemplo, si el nombre de sistema principal es **localhost** y el valor de puerto es **9080**, utilice la dirección Web `http://localhost:9080/ObjectGridSample/docs/introduction.html`.

Inicio de un clúster ObjectGrid de ejemplo en el entorno de WebSphere

Utilice esta tarea para iniciar un único clúster ObjectGrid para probar la funcionalidad de la ObjectGrid distribuida en el entorno de WebSphere Application Server.

WebSphere Extended Deployment debe estar instalado. Debe tener instalado el archivo ObjectGridSample.ear en el servidor de aplicaciones. Para obtener más información sobre la instalación del archivo ObjectGridSample.ear, consulte Carga y ejecución de la aplicación ObjectGrid de ejemplo con WebSphere Extended Deployment.

Utilice esta tarea para establecer un servidor de aplicaciones para alojar un servidor ObjectGrid que está basado en los archivos cluster-config-1.xml y cluster-objectgrid-definition.xml.

El archivo cluster-config-1.xml sólo tiene una definición del servidor ObjectGrid. Este servidor ObjectGrid representa el clúster ObjectGrid de ejemplo. Puede utilizar cualquiera de los servidor de aplicaciones autónomos o un clúster con un miembro de clúster para alojar el servidor ObjectGrid de ejemplo.

1. Extraiga los archivos META-INF/cluster-config-1.xml y META-INF/cluster-objectgrid-definition.xml del archivo /optionalLibraries/ObjectGrid/objectgridSamples.jar al directorio /optionalLibraries/ObjectGrid.
2. Defina los argumentos genéricos de JVM necesarios.
   a. En la consola administrativa, pulse **Servidores > Servidores de aplicaciones > *nombre_servidor* > Definición de proceso > Máquina Virtual Java**.
   b. En el panel argumentos de JVM genéricos, escriba el texto siguiente:

      ```
      -Dobjectgrid.server.name=server1
      -Dobjectgrid.xml.url=file:///<INSTALL_ROOT>\optionalLibraries\ObjectGrid\
      META-INF\cluster-objectgrid-definition.xml
      -Dobjectgrid.cluster.xml.url=file:///<INSTALL_ROOT>\optionalLibraries\
      ObjectGrid\META-INF\cluster-config-1.xml
      ```

      El directorio `INSTALL_ROOT` es el directorio raíz de instalación de WebSphere Application Server.
   c. Pulse **Guardar**.
   d. Pulse **Guardar en configuración maestra**.
   e. Seleccione la opción **Sincronizar cambios con nodos**. Pulse **Guardar**.
3. Copie el archivo /optionalLibraries/ObjectGrid/objectgridSamples.jar en el directorio /classes o `lib/ext`. El archivo objectgridSamples.jar contiene clases que necesita el servidor ObjectGrid de ejemplo para las implementaciones de plug-in que se definen en el archivo cluster-objectgrid-definition.xml. Este archivo JAR se utiliza también para la serialización y deserialización de los objetos que se almacenan en correlaciones.
4. Reinicie el servidor para que los cambios surtan efecto.

Para obtener más información sobre el inicio y la detención de los servidores ObjectGrid en los servidores de aplicaciones, consulte Inicio de un servidor ObjectGrid en un servidor de aplicaciones.

Inicio de un servidor ObjectGrid en un servidor de aplicaciones

Un servidor ObjectGrid puede configurarse para iniciarse en un servidor de aplicaciones. WebSphere Application Server detecta el componente ObjectGrid y automáticamente inicia el servidor ObjectGrid.

Puede configurar los servidores ObjectGrid en WebSphere Application Server Versión 6.0.2 y posteriores, incluido cuando se instalen complementos como WebSphere Extended Deployment o WebSphere Business Integration Server. Versiones anteriores de WebSphere Application Server, como WebSphere Application Server Versión 5.0.2, pueden tener aplicaciones que utilicen ObjectGrid como clientes, pero la función de servidor ObjectGrid no puede colocarse con las versiones anteriores del servidor de aplicaciones.

Si está utilizando configuraciones de clúster que permiten la duplicación, se necesita el gestor de alta disponibilidad. Los servidores ObjectGrid utilizan el gestor de alta disponibilidad de una forma distinta a los servidores de aplicaciones normales. Cuando el servidor ObjectGrid está en un servidor de aplicaciones, el servidor ObjectGrid no configura, inicializa ni crea el servicio de gestor de alta disponibilidad, sino que utiliza el servicio de alta disponibilidad existente en el servidor de aplicaciones. Para la duplicación entre servidores ObjectGrid, los servidores ObjectGrid deben estar ejecutándose en servidores de aplicaciones que sean miembros del mismo grupo principal.

Todas las otras funciones del servidor ObjectGrid son las mismas que cuando el servidor se ejecuta en WebSphere Application Server. Si la especificación de clúster ObjectGrid incluye tres servidores, tres servidores de aplicaciones cualesquiera en un único grupo principal pueden alojar estos servidores ObjectGrid. Los servidores de aplicaciones también pueden abarcar varios clústeres, siempre que los clústeres pertenezcan al mismo grupo principal. El paso más importante es la correlación de la información de nombre de sistema principal y de puerto TCP/IP del servidor en el archivo cluster.xml.

Utilice esta tarea para ejecutar servidores ObjectGrid en los servidores de aplicaciones del entorno WebSphere Application Server.

1. Añada las propiedades personalizadas necesarias a la Máquina Virtual Java (JVM). En la consola administrativa, pulse **Servidores > Servidores de aplicaciones > *nombre_servidor* > Java y gestión de procesos > Definición de proceso > Máquina Virtual Java > Propiedades personalizadas**. Pulse **Nuevo**. Cree las siguientes propiedades personalizadas:

Tabla 1. Propiedades personalizadas de JVM para servidores ObjectGrid

Nombre de la propiedad personalizada	Descripción	Valor de ejemplo
`objectgrid.server.name`	Especifica el nombre de servidor ObjectGrid que debe utilizarse en este servidor de aplicaciones. El nombre proporcionado debe ser uno de los nombres de servidores definidos en el archivo XML del clúster ObjectGrid.	server1
`objectgrid.xml.url`	Especifica el URL (Universal Resource Locator) del archivo XML de ObjectGrid. Esta propiedad es obligatoria.	`file:///d:/was/etc/test/objectGridMatch.xml`

Tabla 1. Propiedades personalizadas de JVM para servidores ObjectGrid (continuación)

Nombre de la propiedad personalizada	Descripción	Valor de ejemplo
`objectgrid.cluster.xml.url`	Especifica el URL del archivo XML del clúster ObjectGrid. Esta propiedad es obligatoria	`file:///d:/was/etc/test/csCluster0.xml`
`objectgrid.security.server.props`	Especifica el URL del archivo de propiedades de seguridad del servidor ObjectGrid. Esta propiedad sólo es obligatoria si se ha habilitado la seguridad en el archivo XML del clúster ObjectGrid. Para determinar si se ha habilitado la seguridad en el archivo XML del clúster, compruebe el texto siguiente: `<cluster name="cluster1" securityEnabled="true" ....` Si el atributo securityEnabled se ha establecido en `false`, no necesita definir esta propiedad: Utilice el archivo `security.ogserver.props` como plantilla. Consulte en "Seguridad de ObjectGrid" en la página 140 el significado de estas propiedades del archivo y cómo pueden utilizarse.	`file:///d:/was/optionalLibraries/ObjectGrid/properties/security.ogserver.props`

Asimismo puede definir estas propiedades JVM en el campo **Argumentos de JVM genéricos** del panel **Máquina Virtual Java** en la consola administrativa. A continuación figura un valor de ejemplo para el campo Argumentos de JVM genéricos:

```
-Dobjectgrid.server.name=server1
-Dobjectgrid.xml.url=file:///<INSTALL_ROOT>\optionalLibraries\ObjectGrid\
META-INF\cluster-objectgrid-definition.xml
-Dobjectgrid.cluster.xml.url=file:///<INSTALL_ROOT>\optionalLibraries\
ObjectGrid\META-INF\cluster-config-1.xml
```

2. Guarde los cambios y reinicie el servidor de aplicaciones. WebSphere Application Server detecta el componente ObjectGrid y automáticamente inicia el servidor ObjectGrid.

   El componente ObjectGrid del servidor de aplicaciones utiliza la infraestructura de canales para interactuar con los clientes ObjectGrid, específicamente denominada puerto de acceso de clientes. Cuando se inicia el servidor ObjectGrid, detecta la asignación en WebSphere Application Server y utiliza la infraestructura de canales que ya está ejecutándose en el servidor de aplicaciones. El servidor ObjectGrid sólo crea e inicia su propia infraestructura de canales si no se ha creado o iniciado una infraestructura de canales en el servidor de aplicaciones.

3. Detenga el servidor ObjectGrid. Detenga el servidor ObjectGrid deteniendo el servidor de aplicaciones asociado. No puede detener el servidor ObjectGrid utilizando los mandatos de gestión de sistemas de ObjectGrid.

Los servidores de aplicaciones del entorno WebSphere Application Server están ejecutando servidores ObjectGrid.

Capítulo 2. ObjectGrid

ObjectGrid es una infraestructura ampliable de colocación en antememoria de objetos transaccionales para las aplicaciones J2SE (Java 2 Platform, Standard Edition J2SE) y J2EE (Java 2 Platform, Enterprise Edition).

Puede utilizar la API de ObjectGrid al desarrollar las aplicaciones para que recuperen, almacenen, supriman o actualicen objetos en la infraestructura ObjectGrid. Asimismo, puede implementar plug-ins personalizados que supervisen las actualizaciones de la antememoria, recuperen y almacenen datos con orígenes de datos externos, gestionen el desalojo de entradas de la antememoria y manejen las funciones subordinadas de antememoria para su propio entorno de aplicaciones ObjectGrid.

API basada en correlaciones

ObjectGrid proporciona una API basada en la interfaz java.util.Map. La API se ha ampliado para dar soporte a la agrupación de operaciones en bloques transaccionales. Esta interfaz es un superconjunto de la interfaz java.util.Map y añade soporte para las operaciones por lotes, la invalidación, la asociación de palabras clave, y la inserción y actualización explícitas. La semántica de la Map de Java se ha ampliado con puntos de extensión para que se puedan implementar las siguientes mejoras:

- Desalojadores de antememoria para ajustar con precisión la duración de las entradas de antememoria
- Interfaces de retorno de llamada de transacciones para controlar con cuidado la gestión de transacciones y, de manera opcional, integrarse con el gestor de transacciones de WebSphere en entornos de J2EE
- Implementaciones de cargador que recuperan y colocan automáticamente datos desde y en una base de datos cuando el programador de aplicaciones utiliza las operaciones get y put de correlaciones de ObjectGrid.
- Interfaces de receptor que pueden proporcionar información sobre todas las transacciones comprometidas mientras ocurren y se aplican a la infraestructura ObjectGrid como un todo o se aplican a determinadas instancias de Map.
- Interfaces de transformador de objetos que permiten una copia y una serialización más eficaces de las claves y los valores.

Entorno de ObjectGrid

Puede utilizar la infraestructura de ObjectGrid instalando una de las siguientes ofertas:

- ObjectGrid está integrada con WebSphere Extended Deployment Versión 6.0.1 y forma parte de la instalación completa.
- La ObjectGrid autónoma forma parte de la instalación del entorno de distintos servidores (MSE).

En ambas ofertas, ObjectGrid da soporte a características de cliente/servidor. El tiempo de ejecución del servidor da soporte completo a la agrupación en clúster, la duplicación y la partición de antememorias de objetos distribuidos. El tiempo de ejecución del cliente da soporte al concepto de una antememoria cercana y la lógica de direccionamiento de gestión de cargas de trabajo a clústeres remotos. El tiempo de ejecución del cliente también da soporte a la creación de correlaciones de objetos locales.

El nivel de soporte varía dependiendo de si ejecuta el tiempo de ejecución del cliente, el tiempo de ejecución del servidor, la ObjectGrid integrada o la ObjectGrid autónoma.

ObjectGrid integrada con la oferta de WebSphere Extended Deployment

Tiempo de ejecución del servidor: el tiempo de ejecución del servidor está integrado. Para WebSphere Extended Deployment Versión 6.0.1, el tiempo de ejecución integrado no está soportado en la plataforma z/OS.

Tiempo de ejecución del cliente: el tiempo de ejecución del cliente está soportado en J2SE y J2EE a nivel de JDK 1.3.1 y superiores, incluidos WebSphere Application Server Versión 5.0.2 y posteriores. El tiempo de ejecución del cliente está plenamente soportado en la plataforma z/OS.

Oferta de ObjectGrid autónoma

Tiempo de ejecución del servidor: el tiempo de ejecución del servidor puede ejecutarse en máquinas virtuales Java (JVM) autónomas como un único servidor o como un clúster de servidores. El servidor autónomo está soportado en la mayoría de plataformas J2SE y J2EE a nivel de JDK 1.4.2 y superiores. El servidor autónomo está soportado en WebSphere Application Server Versión 6.0.2 y posteriores. El tiempo de ejecución del servidor autónomo no está soportado en la plataforma z/OS para WebSphere Extended Deployment Versión 6.0.1.

Tiempo de ejecución del cliente: el tiempo de ejecución del cliente está soportado en las plataformas J2SE y J2EE a nivel de JDK 1.3.1 y superiores, incluidos WebSphere Application Sever Versión 5.0.2 y posteriores.

Gestión de sesiones

Se proporciona una implementación de gestión de sesiones HTTP plenamente distribuida que almacena los objetos de sesión HTTP en la ObjectGrid.

Instalación sencilla

Puede instalar y configurar ObjectGrid mediante unos pocos pasos sencillos. Estos pasos incluyen la copia de los archivos JAR (Java Archive) en la classpath y la definición de algunas directivas de configuración.

Cambios transaccionales

Todos los cambios se realizan en el contexto de una transacción para garantizar una interfaz robusta mediante programación. La transacción puede controlarse explícitamente dentro de la aplicación o bien la aplicación puede utilizar la modalidad de programación de compromiso automático. Estos cambios transaccionales se pueden duplicar en un clúster ObjectGrid en modalidades asíncronas y síncronas para proporcionar acceso escalable y con tolerancia a errores.

Puede escalar ObjectGrid desde una cuadrícula sencilla que se ejecute en una única máquina virtual Java (JVM) hasta una cuadrícula que implique uno o varios clústeres ObjectGrid de máquinas virtuales Java. Estos servidores permiten que los datos estén disponibles a través de las API de correlación en un gran conjunto de clientes habilitados para ObjectGrid. Los clientes ObjectGrid utilizan las API de correlación Java básicas. No obstante, el desarrollador de aplicaciones no necesita desarrollar API RMI (Remote Method Invocation) y TCP/IP Java, ya que el cliente ObjectGrid puede alcanzar los otros servidores ObjectGrid que mantienen

información en la red. Si el conjunto de datos es demasiado grande para una única JVM, puede utilizar ObjectGrid para particionar los datos.

ObjectGrid también ofrece a la solución de aplicación posibilidades de alta disponibilidad añadidas. La compartición de objetos se basa en un modelo de duplicación donde existe un servidor primario, uno o varios servidores de duplicación, y uno o varios servidores en reposo. Este clúster de servidores de duplicación se conoce como grupo de duplicación. Si el acceso al grupo de duplicación es una operación de escritura, la petición se direcciona al servidor primario. Si el acceso es una operación de lectura, o si la correlación es una correlación de sólo lectura, la petición se puede direccionar a los servidores primario o de duplicación. Los servidores en reposo están definidos como servidores de duplicación potenciales si falla un servidor. Si falla un servidor primario, un servidor de duplicación se convierte en el servidor primario para minimizar las interrupciones. Este comportamiento se puede configurar y ampliar, dependiendo de sus necesidades.

Si desea utilizar un enfoque de propagación de objetos más sencillo, también hay disponible un modelo de servicio de igual a igual de menor calidad, como existía en Extended Deployment Versión 6.0. Con este soporte de transacciones distribuidas, se pueden notificar a los iguales los cambios mediante un transporte de mensajes. El transporte de mensajes está incorporado si ejecuta WebSphere Application Server Versión 6.0.2 o posterior. Si no ejecuta WebSphere Application Server Versión 6.0.2 o posterior, se debe proporcionar otro transporte de mensajes como, por ejemplo, un proveedor JMS (Java Message Service).

API compatibles con el contenedor de inyección

Configure ObjectGrid utilizando un archivo XML sencillo o mediante programación utilizando las API de Java. Las API de Java también están diseñadas para funcionar en entornos donde se estén utilizando infraestructuras basadas en la inyección para configurar las aplicaciones. Un contenedor de Inversión de control (IoC) también puede invocar a las API e interfaces de los objetos ObjectGrid, y, a continuación, pueden inyectarse referencias a los objetos ObjectGrid clave en la aplicación.

Arquitectura ampliable

Puede ampliar la mayoría de los elementos de la infraestructura ObjectGrid mediante el desarrollo de plug-ins. Puede ajustar la ObjectGrid para que permita a una aplicación tomar decisiones para lograr un equilibrio entre coherencia y rendimiento. El código personalizado para plug-ins también puede dar soporte a los siguientes comportamientos específicos de la aplicación:

- Escuchar sucesos de instancias de ObjectGrid de inicialización, inicio de transacción, finalización de transacciones y destrucción.
- Invocar retornos de llamada de transacciones para permitir el proceso específico para transacciones.
- Implementar políticas de transacción comunes específicas con las transacciones genéricas de ObjectGrid.
- Utilizar cargadores para puntos de entrada y de salida comunes y transparentes de almacenes de datos y otros depósitos de información.
- Manejar objetos no serializables de un modo específico con las interfaces ObjectTransformer.

Puede implementar todos estos comportamientos sin que se vea afectada la utilización de las interfaces básicas de API de antememoria de ObjectGrid. Con este grado de transparencia, las aplicaciones que utilicen la infraestructura de antememoria pueden experimentar modificaciones considerables de los almacenes de datos y del proceso de transacciones sin que éstas resulten alteradas.

Utilización de una API primaria o antememoria de segundo nivel

La aplicación puede utilizar directamente las API de ObjectGrid como una antememoria compartida o como una antememoria WriteThrough. En la modalidad WriteThrough, la aplicación conecta un objeto Loader para que ObjectGrid pueda aplicar cambios a la aplicación y recoger datos de ésta directamente y de forma transparente. ObjectGrid también puede utilizarse como una antememoria de segundo nivel para los correlacionadores relacionales de objetos conocidos mediante la escritura de un adaptador. La antememoria es invisible para la aplicación en esa modalidad porque la aplicación utiliza las API del correlacionador relacional de objetos como API primaria para acceder a los datos.

Capítulo 3. Visión general de ObjectGrid

ObjectGrid proporciona un modelo de acceso a datos basado en correlaciones Java y una tecnología de antememoria distribuida. Con ObjectGrid, puede configurar un entorno de clústeres de alta disponibilidad. Los clientes ObjectGrid pueden ponerse en contacto con distintos clústeres ObjectGrid de forma simultánea para obtener soluciones de integración a gran escala. ObjectGrid también proporciona una solución compleja de particiones de datos distribuidas para grandes cantidades de información normalizada con datos en más de una máquina virtual Java. Básicamente, ObjectGrid es un conjunto de API Java estandarizadas y servicios de red que permiten el almacenamiento en antememorias locales y distribuidas. La solución se escala de una sola máquina virtual Java (JVM) donde se necesita una solución de correlaciones Java más compleja, a una amplia matriz de servicios de datos escalables y distribuidos que se necesitan de varios clústeres ObjectGrid en toda la empresa.

ObjectGrid en una única máquina virtual Java (JVM)

El uso más básico de la ObjectGrid es en una única JVM.

Puede utilizar ObjectGrid para crear un conjunto de instancias de ObjectGrid. Cada instancia de ObjectGrid puede contener una o varias instancias compatibles con la correlación Java. Las instancias de correlación Java proporcionan las interfaces get y put a las que están acostumbrados los programadores Java, además de características adicionales que la interfaz de correlación Java y la funcionalidad actual no ofrecen. En el siguiente diagrama se muestra el uso más básico de ObjectGrid.

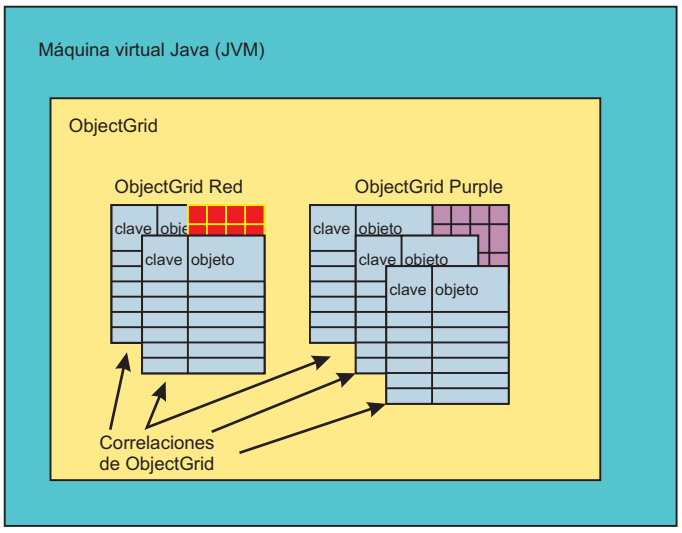


Figura 1. Uso de JVM de ObjectGrid

Una ObjectGrid y una correlación de ObjectGrid incluyen muchas características que actualmente no se proporcionan en la interfaz de correlación Java estándar. Estas características incluyen el acceso transaccional, distintos tipos de estrategias de bloqueo (Ninguno, Optimista y Pesimista), el desalojo Plug and Play, la interacción sin interrupciones con las bases de datos como efecto secundario de utilizar las API get y put, entre otras. También puede desarrollar sus propias ampliaciones de ObjectGrid. Por ejemplo, puede desarrollar un receptor de

correlaciones que proporcione resultados para cada transacción comprometida en una determinada instancia de correlación. Por ejemplo, los usuarios pueden anotar los cambios en el archivo de una ubicación de sucursal para prevenir la pérdida de transacciones, o propagar los cambios con Java Message Service (JMS) u otra infraestructura.

En el diagrama anterior, la JVM tiene dos instancias de ObjectGrid, una con dos objetos tipo correlación Java y la otra con tres objetos de correlación. Los objetos de correlación son bidimensionales, lo que permite manipular los pares de clave y objeto como una correlación Java normal. Una única instancia de ObjectGrid puede dar soporte a varias instancias de correlación específicas.

La siguiente es una configuración de ObjectGrid básica para las instancias de ObjectGrid Red y Purple:

```
<?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="Red">
<backingMap name="FirstRedMap" readOnly="false" />
<backingMap name="SecondRedMap readOnly="false" />
</objetGrid>
<objectGrid name="Purple">
<backingMap name="FirstPurpleMap readOnly="false" />
<backingMap name="SecondPurpleMap readOnly="false" />
<backingMap name="ThirdPurpleMap readOnly="false" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

ObjectGrid distribuida

Además de utilizar el archivo JAR (Java Archive) de ObjectGrid en una única JVM, puede utilizar ObjectGrid en un entorno distribuido. En este entorno, puede crear un clúster ObjectGrid. Un clúster ObjectGrid está formado por un conjunto de servidores ObjectGrid, cada uno con su propia JVM.

En el tema “ObjectGrid en una única máquina virtual Java (JVM)” en la página 19 se describe el soporte de ObjectGrid al concepto de correlación Java. Este concepto también está soportado localmente en una única JVM y en un cliente Java que se conecte a uno o varios clústeres de antememoria ObjectGrid remotos. El servidor ObjectGrid ofrece la posibilidad de distribuir la funcionalidad básica descrita anteriormente en el caso de una única JVM. Por ejemplo, varios clientes pueden compartir la misma correlación de instancia de ObjectGrid, utilizando una estrategia de bloqueo Ninguno, Optimista o Pesimista. Asimismo, un desalojador en los servidores del clúster ObjectGrid puede gestionar el desalojo de los datos de la instancia de correlación del lado del servidor. Todos los clientes pueden utilizar la semántica común de get y put, pero el desalojador que se configura en el servidor del clúster ObjectGrid realiza toda la interacción con la base de datos, en lugar de desplegar y gestionar los controladores JDBC (Java Database Connectivity) en cada cliente.

En el siguiente diagrama, la JVM tiene dos instancias de ObjectGrid: una con dos objetos tipo correlación Java y la otra con tres objetos tipo correlación Java. Las correlaciones son todas objetos bidimensionales que permiten una clave y un objeto. Una única instancia de ObjectGrid puede dar soporte a un gran número de correlaciones, dependiendo principalmente de los requisitos de la aplicación. La

diferencia en este caso es que las correlaciones se alojan dentro de un servidor del clúster ObjectGrid. Los clientes pueden ser una aplicación normal Java o servidores de aplicaciones J2EE (Java 2 Platform, Enterprise Edition).

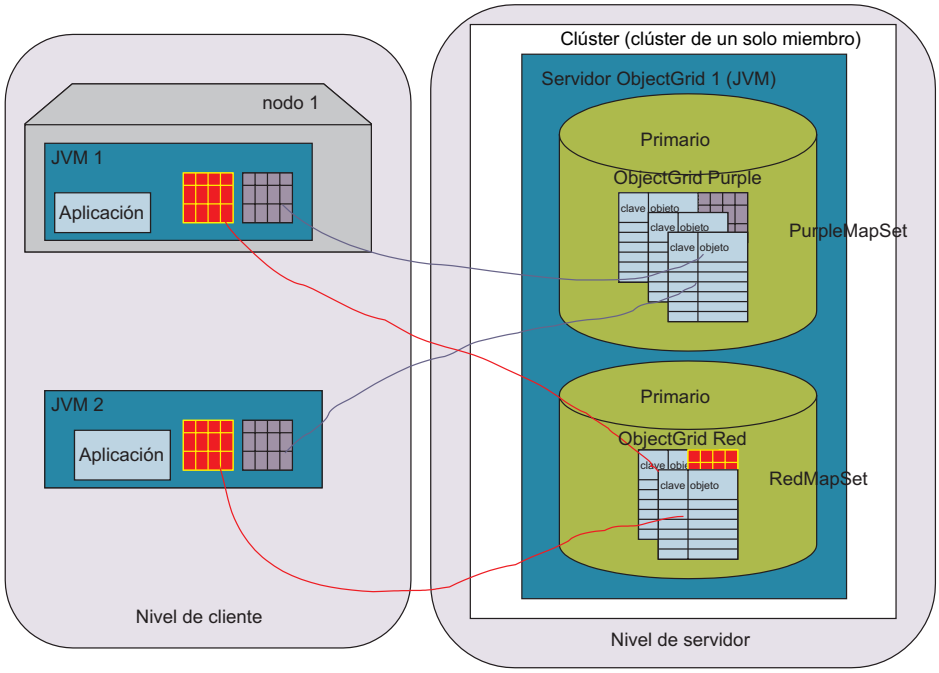


Figura 2. Topología de un solo servidor ObjectGrid distribuido (con dos MapSets)

Clientes ObjectGrid

Los clientes ObjectGrid están formados por un conjunto de API para conectarse a un clúster ObjectGrid, realizar la rutina de carga en la configuración de todo el clúster ObjectGrid y, a continuación, realizar las operaciones de correlación de ObjectGrid que estén distribuidas. Un cliente ObjectGrid es cualquier aplicación Java dentro de su propia instancia de JVM que utilice la ObjectGrid de forma distribuida. Un cliente ObjectGrid distribuido también puede utilizar la funcionalidad no distribuida en la misma máquina virtual Java. El uso del cliente ObjectGrid puede ser tan complicado como el de un servidor de aplicaciones completo con varias conexiones de ObjectGrid paralelas, cada una con la seguridad habilitada y actuando en nombre de un usuario diferente.

Para habilitar el comportamiento distribuido, se debe crear un clúster ObjectGrid (servicios del lado del servidor de la solución ObjectGrid). La configuración adicional necesaria es un archivo XML de clúster ObjectGrid, además del archivo de configuración de ObjectGrid.

A continuación, se muestra el XML de clúster ObjectGrid que configura el despliegue de red de ObjectGrid en el diagrama anterior:

```
<?xml version="1.0" encoding="UTF-8" ?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!– single server ->
```

```
<serverDefinition name="server1" host="localhost" clientAccessPort="12053"
peerAccessPort="12500" />
</cluster>
<objectGridBinding ref="Red">
<mapSet name="RedMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstRedMap" />
<map ref="SecondRedMap" />
</mapSet>
</objectGridBinding>
<objectGridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectGridBinding>
<partitionSet name="ColorMapsParitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

Esta configuración describe un clúster, ″cluster1″, que contiene el servidor server1. El servidor server1 aloja dos ObjectGrids, ″Red″ y ″Purple″. El archivo de configuración especifica también información para las particiones y las duplicaciones. El soporte de cliente-servidor ObjectGrid de ObjectGrid requiere que el programador se conecte a un servidor definido en el clúster ObjectGrid. Durante el proceso de conexión, la configuración de ObjectGrid y el clúster ObjectGrid se descarga dinámicamente en el cliente, lo que simplifica significativamente la preparación del cliente y la gestión del contenido de configuración del lado del cliente. Aparte del cliente ObjectGrid que realiza la operación de ″Conexión″, las API de programación y los conceptos necesarios para utilizar una ObjectGrid con ámbito en la JVM local y una JVM que se aloja en el clúster ObjectGrid son generalmente los mismos.

Inicialización del clúster ObjectGrid

Puede iniciar servidores ObjectGrid en un clúster con las herramientas de línea de mandatos que se proporcionan con ObjectGrid. Una aplicación ObjectGrid puede incluir un cliente ObjectGrid y estar integrada como cualquier otra biblioteca de API Java en la infraestructura de desarrollo de aplicaciones. No obstante, en ambos casos, se debe inicializar el uso de ObjectGrid.

Para trabajar con el caso de ejemplo de uso de la máquina virtual Java (JVM) local o en un clúster ObjectGrid distribuido, debe obtener una configuración válida para realizar una rutina de carga mediante un enfoque gestionable. Los clientes ObjectGrid y los servidores del clúster ObjectGrid deben utilizar una configuración uniforme. Como programador, puede empezar por una configuración muy sencilla, probablemente enlazada en una misma aplicación Java en una JVM.

A continuación, cuando se prepare para empezar las pruebas de usuarios concurrentes de varios clientes, cree su primer clúster ObjectGrid de un solo servidor. Cuando finalice la prueba inicial basada en cliente-servidor, puede trabajar con el personal de administración y experimentar con la duplicación y otros requisitos de soluciones de alta disponibilidad. Cada uno de estos progresos normales en los requisitos de desarrollo requiere un archivo de configuración más complejo.

Los cambios del archivo de configuración para habilitar cada una de estas características avanzadas en cada una de las fases de desarrollo son relativamente modestos, pero cada fase de desarrollo de soluciones requiere una versión diferente del archivo de configuración. El objetivo es que los cambios se creen sobre los anteriores. Puede realizar una prueba unitaria de una solución duplicada en una sola máquina si la cantidad de datos necesarios para desarrollar la solución no desborda un único sistema o se puede restringir artificialmente a efectos de desarrollo.

Configuración de ObjectGrid con XML

Una configuración de ObjectGrid distribuida, con uno o más clientes y uno o más servidores ObjectGrid, requiere la configuración XML. Además del archivo de configuración XML de ObjectGrid base, debe crear una descripción XML de clúster ObjectGrid.

Una descripción de la configuración XML de ObjectGrid y una descripción de la configuración XML de clúster ObjectGrid proporcionan a los clientes y los servidores de una misma ObjectGrid la información que necesitan para funcionar según lo esperado. Puede tener un número cualquiera de clústeres ObjectGrid en el entorno; no obstante, un documento XML de clúster ObjectGrid específico del clúster debe describir cada clúster.

Los archivos de configuración que se necesitan para que se inicie la ObjectGrid se pueden adquirir mediante cualquier enfoque de URL normal. Por ejemplo, los clientes y los servidores pueden adquirir los archivos XML con un archivo físico o un URL HTTP.

En un entorno de ObjectGrid distribuida como el que se describe en los siguientes diagramas, se puede configurar un conjunto inicial de servidores ObjectGrid con la línea de mandatos para recuperar su configuración mediante un URL o, si el URL del archivo es complicado, mediante un archivo del sistema de archivos. No obstante, se recomienda iniciar los siguientes servidores dentro del mismo clúster ObjectGrid ejecutando su rutina de carga desde otros servidores que ya estén operativos en el clúster. Este enfoque es mucho más gestionable, ya que los administradores no necesitan hacer un seguimiento de los archivos de configuración en cada máquina que aloje un cliente o un servidor ObjectGrid. Asimismo, un servidor que se inicia mediante una rutina de carga tiene garantizado que el XML se ha procesado satisfactoriamente, lo que reduce los errores de configuración XML.

Rutina de carga

Rutina de carga del servidor ObjectGrid

El siguiente diagrama describe la rutina de carga de un entorno de clúster ObjectGrid típico que aloja la misma configuración de ObjectGrid, pero ofrece una configuración de clústeres de duplicación más compleja. En este caso, el primer servidor ejecuta la rutina de carga a través de un URL HTTP, y el segundo y el tercer servidor se inician desde el primero. El segundo y el tercer servidor también se pueden iniciar desde el mismo URL que el primer servidor.

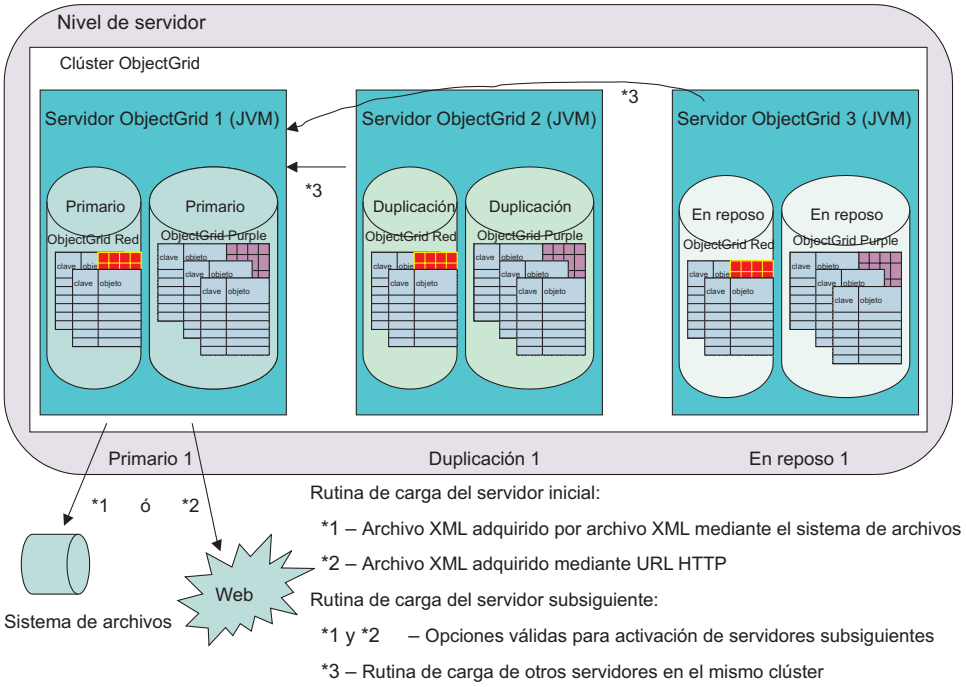


Figura 3. Rutina de carga de servidor inicial mediante la configuración del archivo XML o desde un servidor existente

Tal como se muestra en el diagrama anterior, el servidor server1 del clúster cluster1 es el servidor inicial en el que se ejecuta la rutina de carga. El servidor server1 puede ejecutar la rutina de carga mediante un archivo XML en el sistema de archivos, o mediante un URL en un archivo local, el servidor HTTP remoto u otra opción de URL válida. El servidor server2 y el servidor server3 se pueden iniciar de esta forma o eligiendo el servidor server1 como sistema principal de rutina de carga de configuración. En general, la rutina de carga de los siguientes servidores desde otros servidores garantiza que la configuración sea coherente entre los miembros del clúster.

En este caso de ejemplo concreto, si el servidor server1 falla, y server2 y server3 continúan funcionando, se puede ejecutar la rutina de carga de server1 desde server2 o server3, o se puede ejecutar otra vez mediante el archivo o los enfoques de URL. Consulte “API de conexión de cliente ObjectGrid” en la página 100 para obtener más información sobre la rutina de carga y las opciones de configuración específicas.

Rutina de carga del cliente ObjectGrid

El cliente ObjectGrid, para utilizar los servicios de los servidores miembros del clúster ObjectGrid, debe ejecutar la rutina de carga desde uno de los servidores ObjectGrid dentro del clúster. Cada cliente puede ″conectarse″ a cualquier miembro activo del clúster. Los administradores pueden configurar servidores específicos para ejecutar dicho servicio. Para los despliegues de cliente de gran tamaño, el único objetivo de los servidores configurados del clúster ObjectGrid es proporcionar soporte de rutina de carga de cliente. Este enfoque es muy útil si el número de clientes es elevado y se conectan y desconectan con frecuencia. Cuando los clientes se ″conectan″, pueden obtener una referencia distribuida a las ObjectGrids definidas en la configuración del clúster. Para obtener más información, consulte “Interfaz ObjectGridManager” en la página 93.

Los clientes adquieren la configuración estándar del clúster ObjectGrid, de forma que el administrador no tiene que gestionar el XML para la comunidad de clientes. El cliente ObjectGrid puede utilizar un URL remoto de la misma forma que los servidores del clúster ObjectGrid se activan para alterar temporalmente valores específicos que deben ser específicos del cliente.

Clientes ObjectGrid en un entorno ObjectGrid distribuido

Los clientes ObjectGrid pueden conectarse a más de un clúster ObjectGrid de forma simultánea. Una única aplicación Java en una máquina virtual Java (JVM) puede conectarse varias veces al mismo clúster remoto. La aplicación también puede conectarse a distintos clústeres remotos al mismo tiempo. Esto es importante, ya que habilita la funcionalidad del cliente para acceder a distintos recursos de información que se exportan mediante uno o varios clústeres ObjectGrid.

El primer caso, en el que el mismo cliente ObjectGrid puede ponerse en contacto con el mismo servidor ObjectGrid, es importante para los entornos seguros, donde el cliente puede ser un servidor de aplicaciones, y cada conexión del servidor de aplicaciones con el clúster ObjectGrid remoto utiliza distintas credenciales de seguridad. Otro ejemplo es un cliente ObjectGrid que necesita correlacionar datos de varios clústeres ObjectGrid diferentes con un único objetivo.

El siguiente diagrama muestra un caso de ejemplo donde el usuario del cliente basado en Web corporativo, a través de una aplicación Web, genera un informe desde tres divisiones corporativas diferentes. El motor de servlets utiliza la funcionalidad de cliente ObjectGrid del servidor de aplicaciones para ponerse en contacto con tres clústeres ObjectGrid diferentes, gestionados por cada división corporativa. En muchas corporaciones, los datos se pueden recopilar, y un objetivo clave de ObjectGrid es que la información esté disponible de forma fácil. Una vez externalizada la información, los otros usuarios que tengan interés y credenciales de seguridad podrán adquirir y utilizar la información de otra forma. Los datos se pueden proporcionar en modalidad de sólo lectura o, cuando corresponda, para casos de ejemplo de actualización de lectura-grabación.

En este caso de ejemplo, los datos se pueden adquirir de forma segura. El almacenamiento en antememoria de ObjectGrid de este caso de ejemplo no sólo habilita la compartición flexible de datos mediante programación común en cada división corporativa, sino que también habilita el acceso a datos entre divisiones para la información adquirida a través de un modelo de programación sencillo, limpio y muy seguro que varios desarrolladores Java han utilizando a menudo.

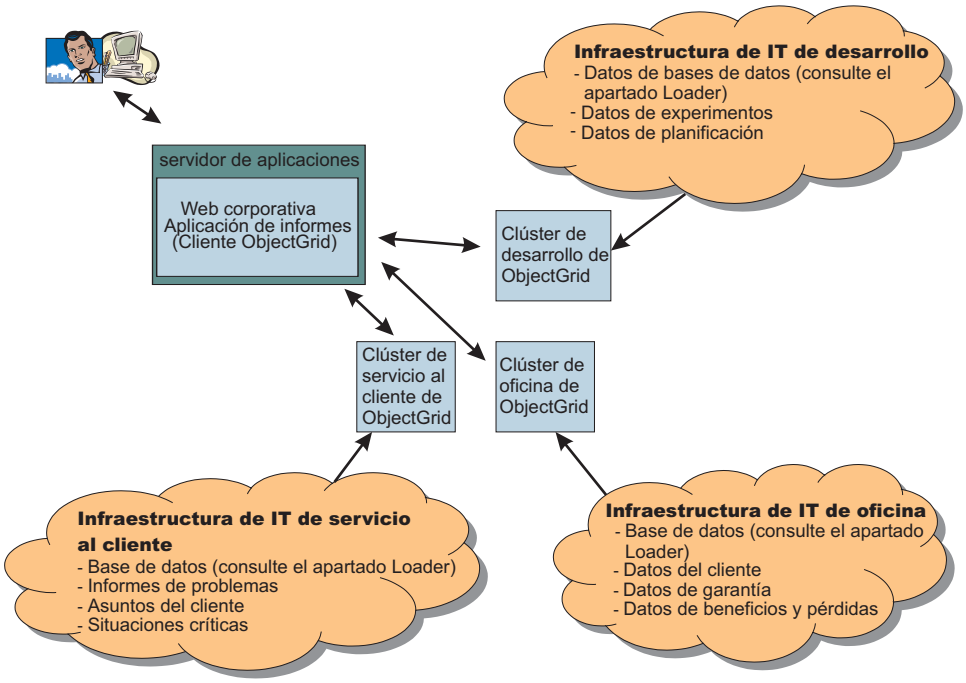


Figura 4. Un usuario de cliente basado en Web genera un informe a partir de tres divisiones corporativas diferentes.

Conceptos de los clústeres ObjectGrid

El término *ObjectGrid distribuida* incluye el concepto de que los clientes pueden interactuar con uno o varios clústeres ObjectGrid. Un clúster ObjectGrid está formado por uno o varios servidores ObjectGrid.

Cliente ObjectGrid

Un cliente ObjectGrid se puede considerar de dos formas. Un cliente se puede considerar una máquina virtual Java (JVM) que utiliza la API de ObjectGrid para conectarse a un clúster ObjectGrid y realizar operaciones de correlación Java en el clúster. La otra forma más formal de considerar el cliente es partir del concepto de varios clientes dentro de la misma JVM. Si utiliza plenamente la función ObjectGrid proporcionada, puede utilizar varios clientes dentro de la misma JVM.

Cada vez que un programador ejecuta la operación de conexión de cliente ObjectGrid en una JVM, se devuelve un contexto de clúster. Este contexto es realmente una instancia de cliente. De modo no visible, las hebras asíncronas manejan muchos aspectos del almacenamiento en antememoria por contexto. Para cada contexto, se puede utilizar ObjectGridManager para adquirir ObjectGrids que se alojan en las instancias específicas de clúster ObjectGrid remoto. Por lo tanto, en general, si se conecta a tres clústeres remotos en la misma JVM, se implementa una solución de tres clientes dentro de la misma JVM.

A continuación, se describen las consideraciones más importantes de este caso de ejemplo. Una sesión de una sola transacción no puede abarcar un conjunto de correlaciones dentro del mismo clúster. Los usuarios no pueden tener una única transacción entre distintos clientes conectados al mismo o a distintos clústeres ObjectGrid. No obstante, los usuarios que intentan integrar grandes cantidades de

información pueden utilizar una transacción para extraer información de cada uno de los clústeres ObjectGrid remotos, e imprimir informes de consolidación o unir la información y enviarla a través de datos de transacción de ObjectGrid a otro clúster ObjectGrid, o simplemente actualizar los clústeres ObjectGrid individuales según el cliente. Esto se debe principalmente a que ObjectGrid ofrece un soporte de transacciones de una fase, por oposición al soporte de transacciones de dos fases que ofrecen normalmente los gestores de transacciones independientes. Para obtener más información sobre este tema, consulte “Demarcación de transacciones ObjectGrid” en la página 39.

Duplicación

Puede crear duplicaciones entre servidores ObjectGrid que estén dentro del mismo clúster ObjectGrid. Con la duplicación, puede recuperarse de una anomalía más rápidamente cuando el servidor ObjectGrid primario que tiene la información concreta que necesita el usuario falla o se cierra para el mantenimiento. En el siguiente diagrama, la ObjectGrid Red y la ObjectGrid Purple están en dos miembros de grupo de duplicación ObjectGrid diferentes. En ObjectGrid, cada MapSet, un subconjunto de una ObjectGrid, se puede duplicar como una unidad. Los PartitionSets son una excepción a esta norma, tal como se describe en el siguiente apartado. La configuración de un solo servidor descrita se modifica en el siguiente diagrama para describir la duplicación.

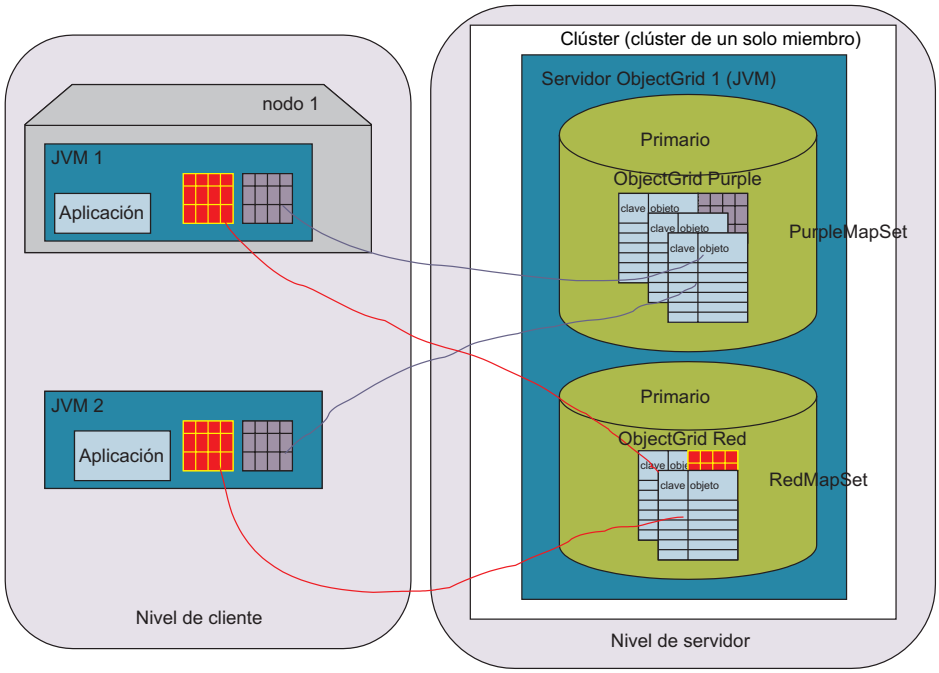


Figura 5. Topología de un solo servidor ObjectGrid distribuido con dos MapSets

El diagrama describe un servidor de aplicaciones como una aplicación cliente y una aplicación Java autónoma. Ambos clientes requieren acceso a dos instancias de ObjectGrid, las instancias Red y Purple en un clúster ObjectGrid de un único servidor. Cada una de estas instancias está contenida en un miembro de grupo de duplicación. Un miembro de grupo de duplicación es un concepto clave, y es el límite de la demarcación de transacciones de ObjectGrid. Una transacción sólo puede comprometer cambios en un único miembro de grupo de duplicación.

En un clúster ObjectGrid, el cliente Java puede iniciar una transacción de ObjectGrid (una sesión), y actualizar los datos dentro de un único miembro de grupo de duplicación. Cada miembro de grupo de duplicación se puede duplicar como una unidad, de forma síncrona o asíncrona, o no duplicarse, dependiendo de los requisitos. Cada petición de cliente ObjectGrid se direcciona a un miembro de grupo de duplicación específico dentro de los servidores del clúster ObjectGrid. La instancia de ObjectGrid dentro del miembro de grupo de duplicación que recibe las peticiones procesa la petición y devuelve un resultado al cliente. Para la duplicación *síncrona*, cada petición, antes de volver al cliente, se envía a la duplicación, o al servidor ObjectGrid 2 del siguiente diagrama, para confirmar que el miembro de grupo de duplicación de la duplicación ha aplicado correctamente la actualización y, a continuación, devuelve los resultados al cliente. En modalidad *asíncrona*, el cliente ObjectGrid puede aplicar un cambio, y el miembro de grupo de duplicación primario del servidor ObjectGrid devuelve el resultado al cliente y no espera a que la duplicación confirme que los cambios se han recibido y aplicado correctamente. En la modalidad asíncrona, la actualización se enviará al miembro de grupo de duplicación de la duplicación del servidor remoto después de que la transacción se haya comprometido satisfactoriamente en el miembro de grupo de duplicación primario.

El siguiente diagrama es una versión diferente del ejemplo de rutina de carga. En este caso, hay tres servidores, cada uno con un rol exclusivo en la duplicación de las dos instancias de ObjectGrid con las que esperan interactuar los usuarios. El clúster ObjectGrid está formado por tres servidores, cada uno de los cuales aloja dos miembros de grupo de duplicación. El servidor server1 aloja dos unidades primarias, el servidor server2 aloja dos duplicaciones y el servidor server3 aloja dos unidades en reposo.

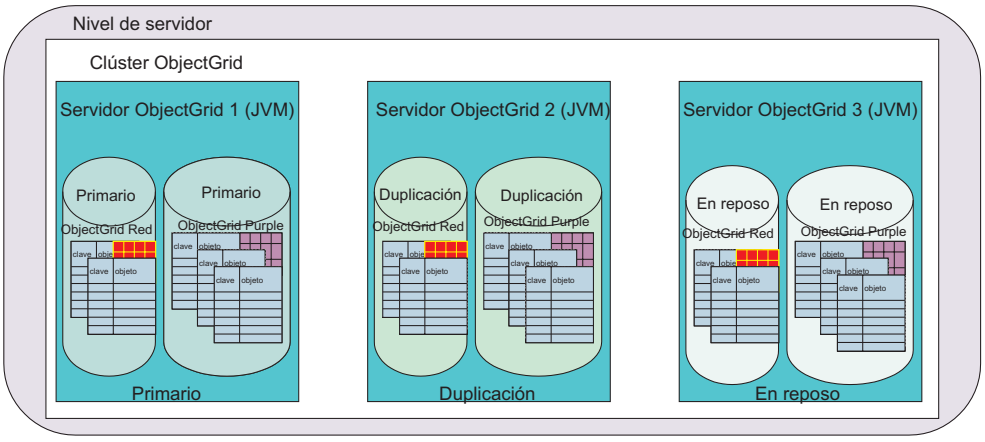


Figura 6. Duplicación de la configuración de ejemplo básica

En el apartado “Visión general de la alta disponibilidad” en la página 30 se describen estos conceptos; no obstante, un concepto clave que debe entender son las diferencias de configuración necesarias para cambiar de la solución de un solo servidor a la solución duplicada de tres servidores que se describe en el diagrama anterior.

Visión general de la configuración de duplicación de varios servidores

La siguiente es una configuración de ObjectGrid básica para las instancias de ObjectGrid Red y Purple.

```
<?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="Red">
<backingMap name="FirstRedMap" readOnly="false" />
<backingMap name="SecondRedMap" readOnly="false" />
</objectGrid>
<objectGrid name="Purple">
<backingMap name="FirstPurpleMap" readOnly="false" />
<backingMap name="SecondPurpleMap" readOnly="false" />
<backingMap name="ThirdPurpleMap" readOnly="false" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

La conversión de esta configuración en un clúster ObjectGrid distribuido requiere un archivo de configuración adicional, el archivo XML de clúster. Para convertir la configuración original de las instancias de objeto Red y Purple en un único servidor se necesitan sólo las adiciones mostradas en el siguiente ejemplo. En concreto, sólo se han añadido dos referencias de servidor. El grupo de duplicación ya existía en el archivo de configuración inicial, que tenía una referencia cruzada con el grupo de duplicación ColorMapsReplicationGroup, tal como se muestra en el siguiente ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster ../objectGridCluster.xsd
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!- single server –>
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12504" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12506" />
<serverDefinition name="server3" host="localhost" clientAccessPort="12507"
peerAccessPort="12508" />
</cluster>
<objectgridBinding ref="Red">
<mapSet name="RedMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstRedMap" />
<map ref="SecondRedMap" />
</mapSet>
</objectgridBinding>
<objectgridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectgridBinding>
<partitionSet name="ColorMapsPartitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup" maxReplicas="1"
minReplicas="1">
<replicationGroupMember serverRef="server1" priority="1" />
<replicationGroupMember serverRef="server2" priority="2" /><!–New–>
<replicationGroupMember serverRef="server3" priority="3" /><!–New->
</replicationGroup>
</clusterConfig>
```

En el ejemplo anterior, los dos MapSets (que se describen a continuación) hacen referencia al ReplicationGroup ColorMapsReplicationGroup, que define los

servidores que se van a incluir en el grupo de duplicación. El archivo de configuración se podría haber ampliado para incluir otro ReplicationGroup, donde cada uno de los MapSets tenga los mismos servidores en órdenes diferentes o distintos servidores que cumplan los requisitos del cliente. La configuración del clúster ObjectGrid da soporte a la reutilización de stanzas. Por omisión, como no se han establecido los atributos de duplicación de MapSet, y el grupo de duplicación tiene más de un servidor, la duplicación está habilitada y la modalidad es asíncrona.

Visión general de la alta disponibilidad

La duplicación habilita la alta disponibilidad en un clúster ObjectGrid.

Para entender la duplicación y la alta disponibilidad, debe entender los tipos de miembros de grupo de duplicación de ObjectGrid. Los tipos de miembros de grupo de duplicación a los que da soporte ObjectGrid son *primario*, *duplicación* y *reposo*. Cada uno de estos tipos tiene un rol determinado en las configuraciones de alta disponibilidad.

Tipos de miembros de grupo de duplicación de ObjectGrid

Miembro de grupo de duplicación primario
El miembro de grupo de duplicación primario mantiene la última vista del cliente de los datos que se están utilizando. Como los datos se actualizan, los datos se propagan a las duplicaciones. El miembro primario es la instancia que se comunica con una base de datos de conexión mediante la interfaz del cargador de ObjectGrid, propaga los compromisos de forma síncrona, asíncrona o no los propaga, dependiendo de la configuración de duplicación.

Miembro de grupo de duplicación de duplicación
Un miembro de grupo de duplicación de duplicación mantiene una versión de los datos que se han propagado desde el miembro primario. El miembro primario se puede configurar para enviar los cambios de varias formas. El grupo de duplicación debe tener al menos dos servidores en la lista para tener un miembro primario y un miembro de duplicación; de lo contrario, la duplicación no estará habilitada.

Miembro de grupo de duplicación en reposo
Un miembro de grupo de duplicación en reposo no recibe actualizaciones cuando se realizan cambios en el miembro primario, como ocurre en la duplicación. Simplemente está configurado y preparado para recibir actualizaciones si el miembro primario o la duplicación fallan. Si el miembro primario falla, la duplicación se convierte en el nuevo miembro primario y el miembro en espera se debe convertir en una duplicación.

Caso de ejemplo de alta disponibilidad

En general, la duplicación habilita la alta disponibilidad en un clúster ObjectGrid. Las dos ilustraciones siguientes describen los casos de ejemplo de anomalía de miembro primario y de recuperación. Si se duplica un miembro de grupo de duplicación primario y se produce una anomalía, se elige una de las duplicaciones para que se convierta en el nuevo miembro primario. En este caso de ejemplo, existe una duplicación.

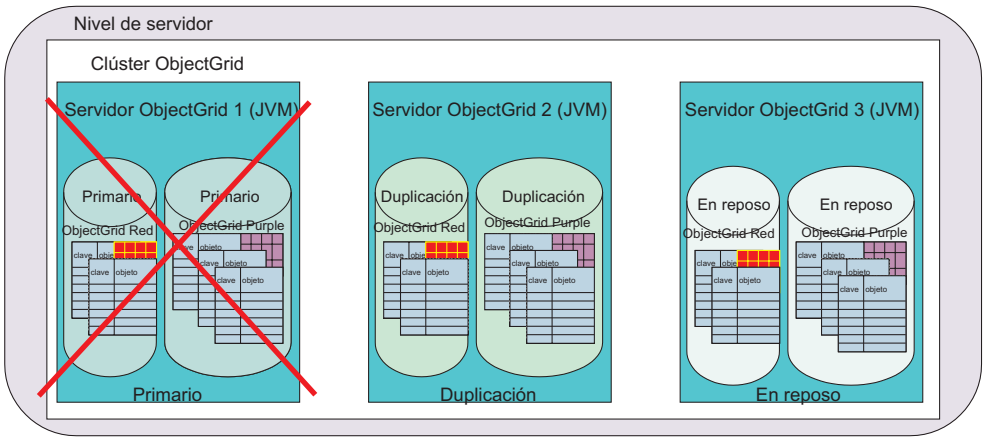


Figura 7. Caso de ejemplo de alta disponibilidad de ObjectGrid

Cuando se detecta una anomalía, el miembro primario deja de estar disponible. La duplicación se convierte en el miembro primario. Si existe un miembro en reposo, se convierte en la duplicación, de la misma forma que la recuperación de ejemplo del siguiente diagrama:

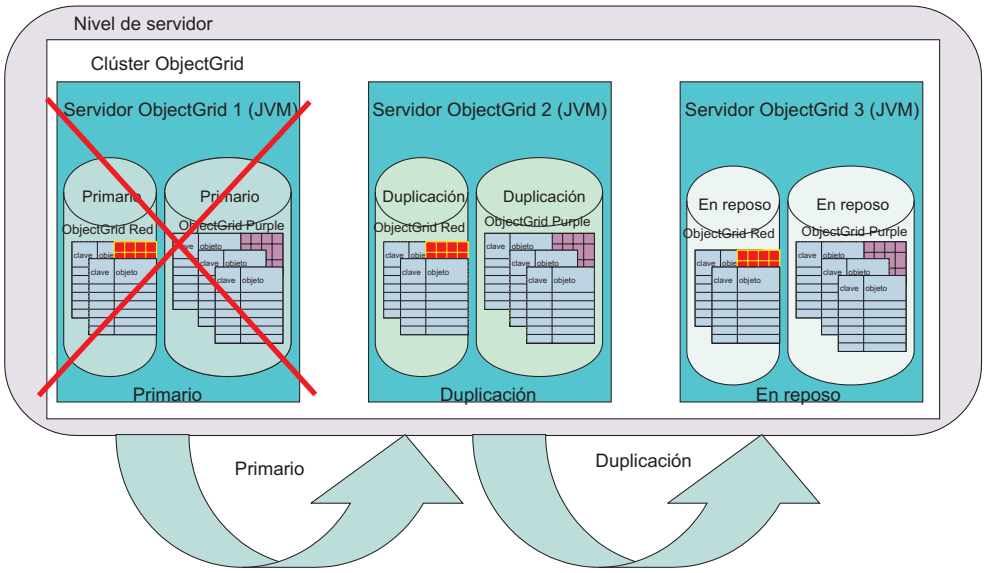


Figura 8. Sustitución por anomalía de ObjectGrid

Los clientes ObjectGrid notarán este ajuste durante su próxima conexión a cualquiera de los servidores afectados. Los clientes que se ponen en contacto con el servidor anómalo utilizan su configuración de tiempo de ejecución y pueden intentar los otros servidores del clúster de forma dinámica. El cliente se pone en contacto con el siguiente servidor de la configuración. Si el servidor está activado pero no operativo, el cliente espera un periodo de tiempo de espera. Los clientes suponen que los miembros de grupo de duplicación duplicados se están recuperando de una anomalía. Después de una determinada cantidad de tiempo, los clientes reintentan la operación y cuando el grupo de duplicación que ahora tiene dos miembros esté operativo, se proporciona una nueva tabla de direccionamiento al cliente. La tabla de direccionamiento describe dónde se encuentra el miembro primario actual, las ubicaciones de las duplicaciones y qué

miembros de grupo de duplicación de este grupo son miembros en reposo actualmente.

Conjuntos de configuración en clústeres ObjectGrid

Cuando se configuran clústeres ObjectGrid, puede separar las ObjectGrids en MapSets. Esta separación es importante porque una ObjectGrid puede contener muchas correlaciones. Los MapSets se pueden particionar con un PartitionSet y se pueden duplicar con un ReplicationGroup. Cada una de estas opciones de configuración afectan a cuántos miembros del grupo de duplicación se crean durante el arranque del servidor ObjectGrid. Una visión general rápida de cada tipo de conjunto permite explicar el rol de cada tipo.

MapSet de ObjectGrid

Cada correlación de ObjectGrid puede tener requisitos de uso y disponibilidad diferentes, aunque se correlacione mediante el uso típico de la aplicación. Por ejemplo, una correlación puede ser de sólo lectura, sin que haya cambios cuando finalice la precarga, y otra puede ser de lectura-grabación y se puede particionar a efectos de escalabilidad. En este caso, cada correlación se incluye en un MapSet exclusivo. En el ejemplo anterior, PurpleMapSet y RedMapSet mantienen todas las correlaciones de cada ObjectGrid proporcionada, que es la opción más sencilla.

Un MapSet es una unidad que puede duplicarse entre servidores ObjectGrid y se correlaciona con un miembro de grupo de duplicación que no está particionado. Cada servidor de grupo de duplicación asociado con un MapSet a través de un PartitionSet aloja un miembro de grupo de duplicación, según corresponda, para dar soporte a la configuración solicitada. Un miembro de grupo de duplicación es un punto final exclusivo en el clúster ObjectGrid y aloja todas las correlaciones que un determinado MapSet dicta en la configuración.

Por ejemplo, en el diagrama anterior, el servidor server1 tiene la unidad primaria, el servidor server2 tiene la duplicación y el servidor server3 tiene una unidad en reposo que se puede convertir en una duplicación o una unidad primaria, dependiendo de los casos de ejemplo de recuperación de la duplicación. Los Mapsets se correlacionan con el PartitionSet que describe tres servidores. Por lo tanto, ambos MapSets, aunque tengan un número diferente de correlaciones en cada uno, se correlacionan con el mismo servidor, ya que las stanzas de PartitionSet y ReplicationGroup son las mismas.

PartitionSet de ObjectGrid

Normalmente, el Mapset y los servidores de grupo de duplicación enumerados determinan el número de servidores que dan soporte a un determinado MapSet en un clúster ObjectGrid. Se crea un miembro de grupo de duplicación en cada servidor de grupo de duplicación de alojamiento. No obstante, las particiones pueden afectar al número de miembros de grupo de duplicación de un MapSet. Las particiones se gestionan en el archivo de configuración a través de la relación de PartitionSet entre el MapSet y el ReplicationGroup.

Un PartitionSet divide un MapSet en varias partes, para que una máquina virtual Java (JVM) no tenga que mantener el MapSet completo en un mismo miembro de grupo de duplicación primario. Por ejemplo, imagine una base de datos de 1.000.000 claves. Si cada objeto al que hace referencia cada clave tiene un gran tamaño, es probable que una única JVM de 32 bits no pueda mantener la correlación en memoria en un solo miembro de grupo de duplicación primario, de

duplicación o en reposo. No obstante, a menudo se necesitan conjuntos de datos de gran tamaño. Para no tener que particionar artificialmente los datos, por ejemplo, para no tener que particionar manualmente la primera instancia de correlación de la ObjectGrid Purple en las correlaciones PurpleFirstMapMap1, PurpleFirstMap2, PurpleFirstMapN, y colocarlas en un Mapset diferente, ObjectGrid puede realizar este trabajo en gran parte.

Posteriormente en este documento, se define el concepto de PartitionKey. Esto equivale en efecto a una API que ObjectGrid puede invocar para determinar cuál es el código hash de clave para una determinada entrada durante la inserción. Si un MapSet tiene dos particiones, se crean dos miembros de grupo de duplicación para el MapSet. A menudo, como los datos tienen un gran tamaño, estos miembros de grupo de duplicación se ubican en servidores diferentes. Para los desarrolladores, los miembros pueden estar en el mismo servidor durante la creación temprana de prototipos. Cada miembro de grupo de duplicación mantiene claves que aplican hash en el mismo valor en todos los miembros de grupo de duplicación particionados que hay disponibles. Como ejemplo sencillo, supongamos que el MapSet se ha particionado de tres formas, 0, 1 y 2. Se establecen tres miembros de grupo de duplicación primarios, y uno mantiene todas las claves que aplican hash en un determinado valor que modula el número de los miembros de grupo de duplicación primarios. Por ejemplo, si el valor hash de una clave es 7, el módulo 7 de 3 es 1, por lo que el miembro de grupo de duplicación primario con el índice de partición 1 contendrá la instancia.

Ejemplo de PartitionSet

El siguiente diagrama ilustra la separación de la correlación Purple en dos particiones. Cada clave en la correlación tiene un entero como hash, y se asigna a un conjunto de particiones específico al insertarse en el miembro de grupo de duplicación correspondiente. Cada partición está en un miembro de grupo de duplicación diferente porque puede existir en una JVM diferente, y ObjectGrid permite que el programador trate en general la PurpleFirstMap como una instancia de correlación lógica, no particionada. El soporte de cliente y servidor ObjectGrid gestiona el direccionamiento correcto de las peticiones entre los miembros de grupo de duplicación.

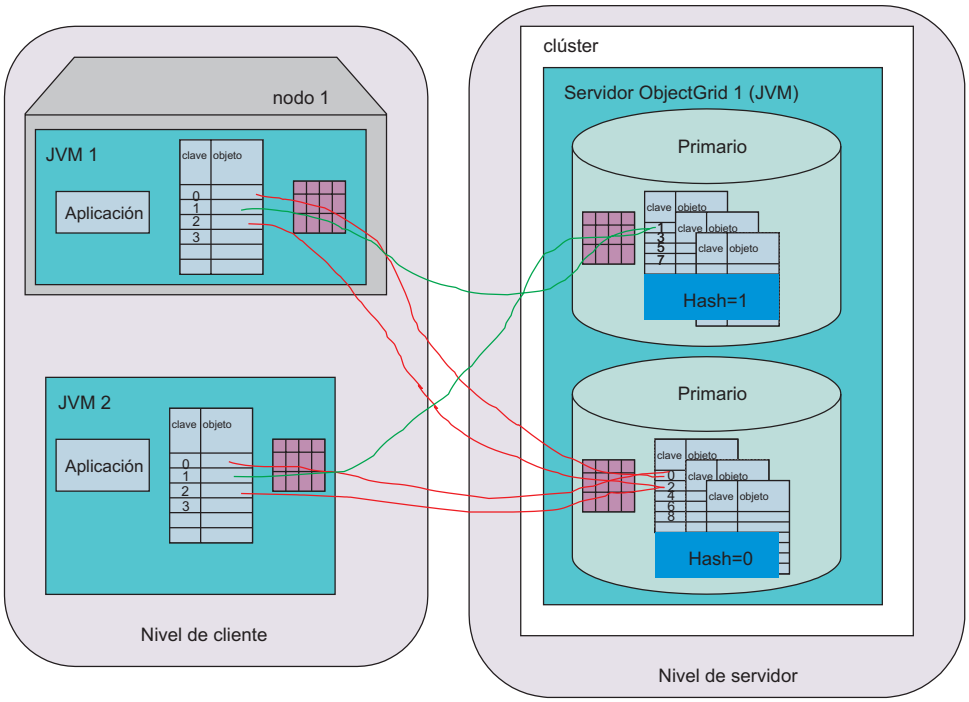


Figura 9. Topología de ObjectGrid distribuido: un solo servidor con particiones

A continuación, se muestra el cambio de configuración del conjunto de particiones para habilitar esta configuración:

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!- single server –>
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectGridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectGridBinding>
<partitionSet name="ColorMapsPartitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
<partition name="partition2" replicationGroupRef="ColorMapsReplicationGroup" />
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup" maxReplicas="1"
minReplicas="1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

La configuración de ejemplo refleja cómo establecer una ObjectGrid Purple duplicada y particionada. En el siguiente caso, existen tres servidores y cada uno de los miembros de grupo de duplicación primarios se correlaciona de la misma forma con el conjunto de tres servidores. Si se utilizara otra stanza de ReplicationGroup, se podrían correlacionar fácilmente de otra forma.

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!- single server –>
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12504" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12506" />
<serverDefinition name="server3" host="localhost" clientAccessPort="12507"
peerAccessPort="12508" />
</cluster>
<objectGridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectGridBinding>
<partitionSet name="ColorMapsPartitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
<partition name="partition2" replicationGroupRef="ColorMapsReplicationGroup" />
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup" maxReplicas="1"
minReplicas="1">
<replicationGroupMember serverRef="server1" priority="1" />
<replicationGroupMember serverRef="server2" priority="2" />
<replicationGroupMember serverRef="server3" priority="3" />
</replicationGroup>
</clusterConfig>
```

Observe que, en el ejemplo anterior, sólo un pequeño cambio en el archivo de configuración obtiene un nuevo nivel de capacidad y requiere la modificación de la aplicación para alcanzar el resultado. El siguiente diagrama muestra la vista de clúster ObjectGrid y cómo se disponen los miembros de grupo de duplicación para dar soporte a la configuración de particiones anterior.

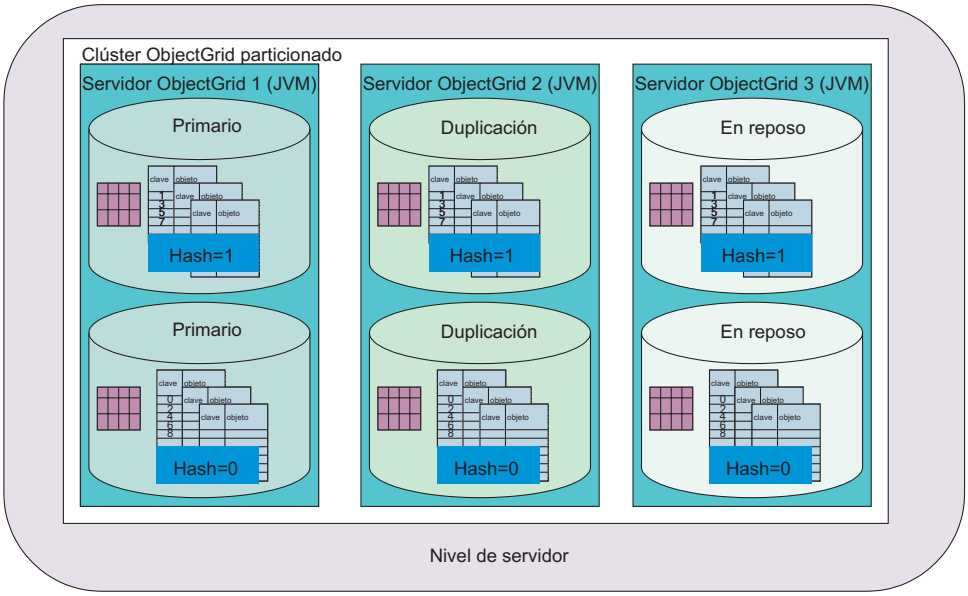


Figura 10. Topología de ObjectGrid distribuida: varios servidores con particiones

Para completar la descripción de PartitionSet, a continuación se proporciona otra variación del ejemplo anterior. En este caso, se ha creado un segundo ReplicationGroup. Cada PartitionSet se incluye ahora en su propio grupo de duplicación, en servidores diferentes.

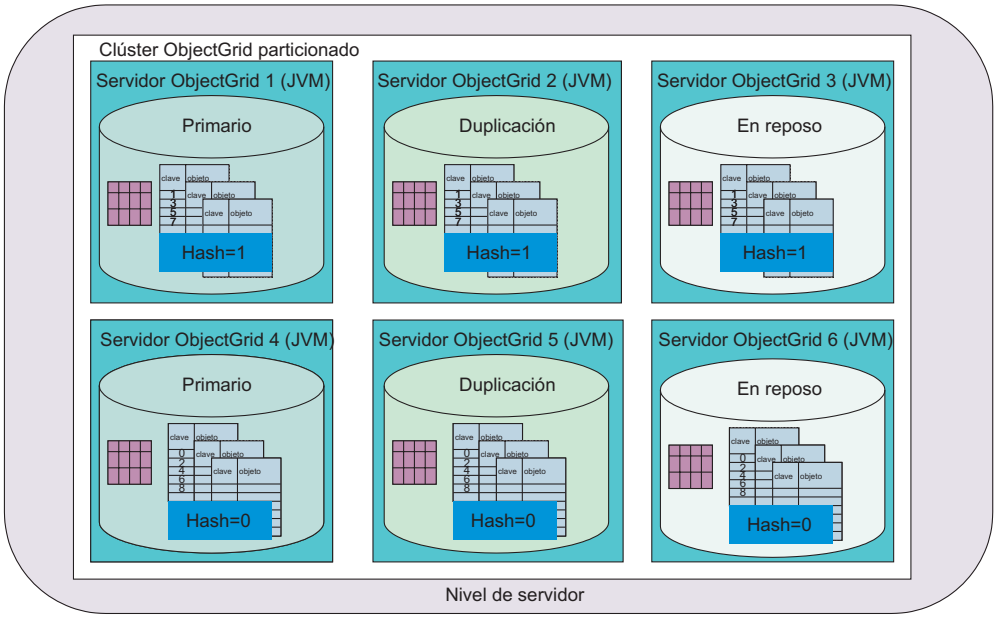


Figura 11. Topología de ObjectGrid distribuida: varios servidores con particiones

La configuración de esta topología es muy parecida a la de los ejemplos anteriores. Los cambios incluyen tres instancias más de servidor y un nuevo ReplicationGroup al que se hace referencia en el segundo PartitionSet.

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!- single server –>
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12504" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12506" />
<serverDefinition name="server3" host="localhost" clientAccessPort="12507"
peerAccessPort="12508" />
<serverDefinition name="server4" host="localhost" clientAccessPort="12513"
peerAccessPort="12514" /><!-*New*->
<serverDefinition name="server5" host="localhost" clientAccessPort="12514"
peerAccessPort="12516" /><!-*New*->
<serverDefinition name="server6" host="localhost" clientAccessPort="12517"
peerAccessPort="12518" /><!-*New*->
</cluster>
<objectGridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectGridBinding>
<partitionSet name="ColorMapsPartitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
<partition name="partition2" replicationGroupRef="ColorMapsReplicationGroup" />
<!-NEW->
```

```
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup" maxReplicas="1"
minReplicas="1">
<replicationGroupMember serverRef="server1" priority="1" />
<replicationGroupMember serverRef="server2" priority="2" />
<replicationGroupMember serverRef="server3" priority="3" />
</replicationGroup>
<replicationGroup name="ColorMapsReplicationGroupNew" maxReplicas="1"
minReplicas="1"> <!-*NEW*->
<replicationGroupMember serverRef="server4" priority="1" />
<replicationGroupMember serverRef="server5" priority="2" />
<replicationGroupMember serverRef="server6" priority="3" />
</replicationGroup>
</clusterConfig>
```

Esta configuración da como resultado el mismo número de miembros de grupo de duplicación, pero obliga al segundo grupo de duplicación a configurarse específicamente mediante su propia stanza XML y, al mismo tiempo, atribuye cada una de las instancias a instancias de servidor diferentes, en lugar de co-ubicarlas como se hizo en el ejemplo anterior.

La ventaja de esta configuración es que cada partición en este caso puede mantener más de 1,5 gigabytes de datos, lo que hace un total de 3 gigabytes o más, ya que los dos miembros de grupo de duplicación están en sus propias instancias de JVM. Esto garantiza el uso óptimo de los 2 gigabytes de JVM de 32 bits de espacio de memoria direccionable.

Clientes ObjectGrid en contacto con varios clústeres ObjectGrid

ObjectGrid se ha diseñado específicamente no sólo para escalar en términos de soporte particionado entre máquinas virtuales Java (JVM), sino también para ampliar el alcance con el que se puede utilizar una interfaz de correlación Java normal para adquirir información. Puede ponerse en contacto con muchos clústeres ObjectGrid con un único cliente.

En el siguiente caso de ejemplo, un nivel de servidor contiene dos clústeres ObjectGrid. Uno de los clientes de aplicaciones Java y uno de los servidores de aplicaciones deben ponerse en contacto con varios clústeres. Esta es una característica muy potente que permite una gran escalabilidad.

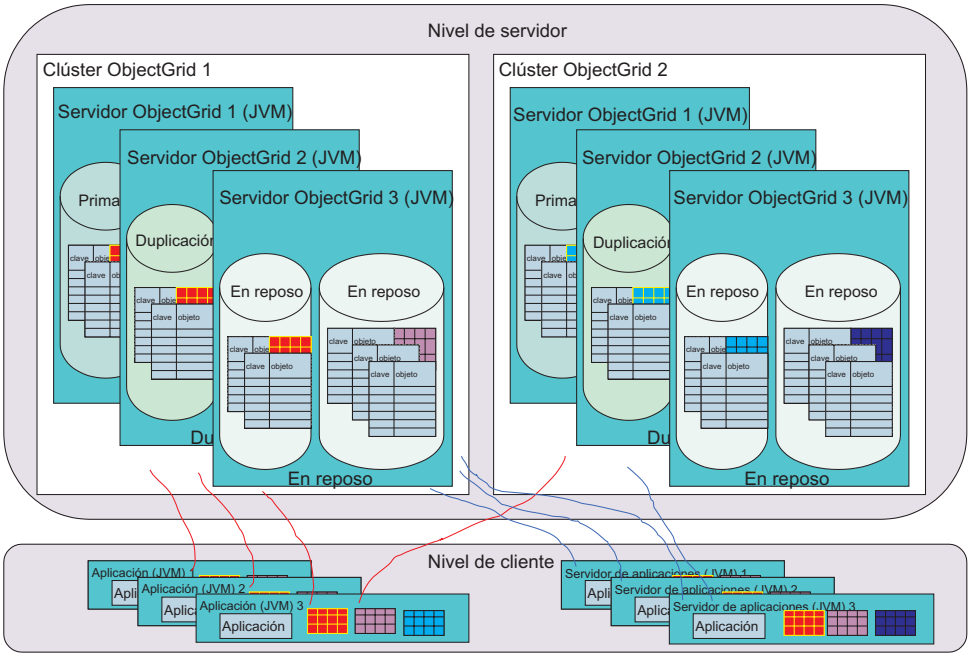


Figura 12. Clientes ObjectGrid en interacción con varios clústeres ObjectGrid

Un clúster ObjectGrid puede dar soporte a varias ObjectGrids, cada una con distintas configuraciones de MapSet y PartitionSet. Cada clúster ObjectGrid puede estar formado por una o varias JVM; probablemente muchas más. En una gran empresa, la posibilidad que tiene el cliente ObjectGrid de ponerse en contacto con no sólo uno, sino varios clústeres ObjectGrid al mismo tiempo, constituye una característica muy ventajosa.

No obstante, debe tener cuidado, ya que el cliente ObjectGrid no puede hacer referencia a datos de varios clústeres dentro de una misma transacción. La aplicación de cliente ObjectGrid debe ejecutar una o varias transacciones para almacenar en antememoria los datos en la instancia de la máquina virtual Java y correlacionar la información recuperada como un objeto Java. Las actualizaciones basadas en la información también deben estar contenidas en una misma transacción para cada clúster. Consulte “Demarcación de transacciones ObjectGrid” en la página 39 para obtener más información al respecto.

Soporte de antememoria cercana de cliente ObjectGrid

El cliente ObjectGrid es realmente un nivel de antememoria. Puede diseñar la aplicación para que aproveche las posibilidades de antememoria local si sabe si los datos que se adquirido previamente del servidor remoto no están obsoletos. Por ejemplo, si los datos se han actualizado en el clúster ObjectGrid pero no en este cliente (para ello necesitará una nueva petición get(...)), el cliente debe actualizar la antememoria local para que sea coherente (no todas las aplicaciones lo requieren) con el clúster ObjectGrid.

Una vez ejecutada una operación get, la siguiente petición de la operación get para el mismo par de clave y objeto hará que el cliente ObjectGrid detecte que los datos ya se han recuperado y utilizará la versión en antememoria de la ″máquina virtual Java (JVM)″ en lugar de ir a través de la red al clúster ObjectGrid para acceder a los datos. Después de que los datos se recuperen una vez a través de la red, los

datos continúan proporcionándose desde la antememoria local hasta que se desaloje la entrada local, de forma manual o mediante un desalojador configurado normal.

Por ejemplo, si sabe que los datos en el servidor se renuevan una vez cada seis horas, puede controlar cuándo se produce la actualización de cliente de antememoria cercana o antememoria local. El usuario invalida la entrada de antememoria cercana y, a continuación, emite la petición get. La petición get se pone en contacto con el servidor y adquiere la información si todo va bien. Supongamos que el objeto es un archivo de imagen. La primera vez que se descarga la imagen después de la ventana de actualización, toda petición posterior no generará una llamada a procedimiento remoto en el servidor para obtener la imagen.

El soporte de antememoria cercana no se aplica en la modalidad pesimista, ya que el cliente puede necesitar un bloqueo en los datos del clúster ObjectGrid para forzar la estrategia de bloqueo solicitada. Revise el método beginNoWriteThrough() para obtener más información sobre cómo borrar o invalidar entradas de la antememoria cercana que se deben eliminar sin modificar la vista de la información del clúster ObjectGrid.

Demarcación de transacciones ObjectGrid

Un concepto clave a tener en cuenta como programador es el de demarcación de transacciones. ObjectGrid no da soporte al protocolo de compromiso de dos fases para el proceso de compromiso de transacciones entre miembros de grupo de duplicación de un clúster ObjectGrid. En una sesión basada en un único clúster ObjectGrid, las actualizaciones de lectura-grabación se deben aplicar a un único miembro de grupo de duplicación primario. Si se han utilizado varios miembros de grupo de duplicación, no se podrán realizar la actualizaciones atómicas durante el proceso de compromiso de transacciones. Esto es diferente al modelo de programación JAR de ObjectGrid local, que permite el compromiso en todas las correlaciones de una única ObjectGrid.

Un caso especial a tener en cuenta es un PartitionSet en el que se ha definido más de una partición. En estos casos de ejemplo, la clave 1 puede estar en el servidor 1, la clave 2 en el servidor 2, etc. Si en una transacción hay actualizaciones de la clave 1 y la clave 2, las actualizaciones fallarán porque una sesión ObjectGrid no puede comprometerse en dos miembros de grupo de duplicación. Como se ha descrito anteriormente, un PartitionSet que da soporte a dos o más particiones se debe utilizar con cuidado en la aplicación. Asegúrese de que la secuencia de transacciones no actualice datos de más de una sola transacción.

Relación de ObjectGrid con las bases de datos

Esta visión general no ha incluido específicamente los servidores ObjectGrid que adquieren datos inicializados de lugares que no sean otros clientes Java. Cuando se inicia una ObjectGrid, se inicializa un cargador para cada correlación de un MapSet. Este cargador permite peticiones de usuario en forma de una petición get o put de correlación Java que se recuperará o se grabará en la base de datos, según corresponda. Las operaciones de la base de datos son invisibles para el usuario de la correlación Java, y no se necesita ninguna codificación especial en su aplicación. No obstante, un programador debe desarrollar la funcionalidad del cargador, que se debe configurar para su uso en la ObjectGrid antes de que el usuario final programador pueda utilizar esta función. Para obtener más información, consulte “Cargadores” en la página 202.

Asimismo, se proporciona soporte a la precarga desde una base de datos después de que se inicialice un clúster ObjectGrid, así como a una precarga particionada para garantizar que se lean los datos correctos para la partición de miembro de grupo de duplicación específica de un determinado MapSet. Los usuarios pueden acceder a la información de lectura-grabación y actualizarla con varias estrategias de bloqueo, por ejemplo, ninguno, optimista y pesimista. El acceso a datos de sólo lectura está soportado y es el modelo más rápido, ya que se pueden proporcionar varias optimizaciones.

Capítulo 4. Guía de Aprendizaje de ObjectGrid: modelo de programación de aplicaciones

Utilice esta tarea para obtener más información sobre el modelo de programación de aplicaciones ObjectGrid.

Prepare el entorno para ejecutar las aplicaciones ObjectGrid. Consulte el Capítulo 1, “Iniciación a ObjectGrid ejecutando la aplicación de ejemplo”, en la página 1 para obtener más información sobre las ubicaciones del archivo JAR (java Archive) los requisitos Java, y cómo ejecutar un archivo sencillo para verificar que el entorno esté configurado correctamente.

Decida que entorno de programación desea utilizar para esta tarea. Puede utilizar un IDE (Integrated Development Environment) como, por ejemplo, Eclipse, pero el entorno Java de la línea de mandatos también funciona. Incorpore ObjectGrid en los enterprise beans y servlets después de que se haya familiarizado con ObjectGrid. Los ejemplos de la Guía de Aprendizaje no presuponen ningún entorno Java determinado, de modo que puede utilizar cualquier entorno familiar.

En su definición más básica, ObjectGrid es una antememoria y un depósito en memoria para objetos. La utilización de una correlación java.util.Map para almacenar objetos y acceder a ellos es parecida a la utilización de ObjectGrid. Asimismo, ObjectGrid es más que una simple antememoria. Al explorar las distintas características y plug-ins de esta tarea, descubrirá que ObjectGrid es muy ampliable y flexible. Puede utilizar ObjectGrid como una antememoria *compartida* o una antememoria más elaborada respaldada por un gestor de recursos.

Los ejemplos de esta guía de aprendizaje no son programas completos. Las importaciones, el proceso de excepciones e incluso algunas de las variables no se declaran completamente en todos los ejemplos. Puede utilizar los ejemplos para escribir sus propios programas.

Utilice esta tarea para utilizar ObjectGrid desde un programa Java.

1. Localice las API y excepciones de ObjectGrid. Todas las API y excepciones públicas de ObjectGrid se encuentran en el paquete `com.ibm.websphere.objectgrid`. Para obtener información avanzada sobre temas de configuración y del sistema, consulte las API y excepciones adicionales del paquete `com.ibm.websphere.objectgrid.plugins`. Allí donde se proporcionen implementaciones de plug-in, ubique esas clases en el paquete `com.ibm.websphere.objectgrid.plugins.builtins`. Para las características de seguridad de ObjectGrid, busque paquetes que incluyan **security** en el nombre como, por ejemplo, `com.ibm.websphere.objectgrid.security`, `com.ibm.websphere.objectgrid.security.plugins` y `com.ibm.websphere.objectgrid.security.plugins.builtins`.

   Esta tarea centra la atención en las API contenidas en el paquete `com.ibm.websphere.objectgrid`. El JavaDoc completo de ObjectGrid puede encontrarse en la siguiente ubicación: `<raíz_instalación>`.

   ```
   com.ibm.websphere.objectgrid
   com.ibm.websphere.objectgrid.plugins
   com.ibm.websphere.objectgrid.plugins.builtins
   com.ibm.websphere.objectgrid.security
   com.ibm.websphere.objectgrid.security.plugins
   com.ibm.websphere.objectgrid.security.plugins.builtins
   ```

2. Obtenga o cree una instancia de ObjectGrid. Utilice ObjectGridManagerFactory para obtener la instancia singleton de ObjectGridManager. A continuación, cree una instancia de ObjectGrid con las siguientes sentencias:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid objectGrid =
objectGridManager.createObjectGrid("someGrid");
```

   La interfaz ObjectGridManager tiene varios métodos para crear, recuperar y eliminar instancias de ObjectGrid. Consulte el tema “Interfaz ObjectGridManager” en la página 93 para escoger una variación para su situación. Asimismo, puede establecer valores de rastreo con la interfaz ObjectGridManager. Si se está ejecutando en WebSphere Extended Deployment o WebSphere Application Server, estos métodos no son necesarios porque el rastreo se gestiona mediante los recursos incluidos. Si se está ejecutando fuera de WebSphere Application Server, estos métodos pueden ser útiles. Consulte el tema “Rastreo de ObjectGrid” en la página 100 para obtener una información más completa sobre estos métodos.

3. Inicialice ObjectGrid.
   a. Establezca un nombre para ObjectGrid, si no ha establecido el nombre con los métodos `create`.
   b. Defina las BackingMaps utilizando la configuración por omisión para una BackingMap para las aplicaciones iniciales.
   c. Después de definir las BackingMaps, inicialice ObjectGrid. La inicialización de ObjectGrid indica que se ha completado toda la configuración y que se desea empezar a utilizar ObjectGrid.
   d. Después de la inicialización de ObjectGrid, obtenga un objeto Session. Consulte “Interfaz ObjectGrid” en la página 107 y el JavaDoc para obtener más información.

   Utilice el siguiente ejemplo como guía en este paso:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid objectGrid =
objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
```

4. Utilice sesiones para gestionar las operaciones transaccionales. Todos los accesos a una antememoria de ObjectGrid son transaccionales: los accesos múltiples, inserciones, actualizaciones y supresiones de objetos de la antememoria están contenidos en una única unidad de trabajo, a la que se hace referencia como sesión. Al final de una sesión, puede comprometer todos los cambios de esta unidad de trabajo o bien retrotraer y olvidar todos estos cambios.

   Asimismo, puede utilizar el compromiso automático para las operaciones atómicas individuales realizadas en la antememoria. En ausencia de un contexto de sesiones activo, los accesos individuales al contenido de antememoria se incluyen en sus propias sesiones comprometidas automáticamente.

   Otro aspecto importante de la interfaz Session consiste en obtener acceso transaccional a BackingMap, o un descriptor de contexto de BackingMap, con la interfaz ObjectMap. Puede utilizar el método getMap para crea un descriptor de contexto de ObjectMap de una BackingMap definida previamente. Todas las operaciones realizadas en la antememoria como, por ejemplo, las inserciones, actualizaciones o supresiones se completan con la instancia de ObjectMap.

Consulte el tema “Interfaz Session” en la página 116 para obtener más información. Utilice el siguiente ejemplo para obtener y gestionar una sesión:

```
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
```

5. Utilice la interfaz ObjectMap para acceder a la antememoria y actualizarla. Al consultar la interfaz ObjectMap, verá que existen varios métodos para acceder y actualizar la antememoria. La interfaz ObjectMap está diseñada como una interfaz de correlación. No obstante, las excepciones comprobadas se introducen como una ayuda para desarrollar las aplicaciones ObjectGrid con un IDE como, por ejemplo, Eclipse. Si desea utilizar una interfaz java.util.Map sin excepciones comprobadas, puede utilizar el método getJavaMap. Para obtener más información, consulte “Interfaces ObjectMap y JavaMap” en la página 120.

   Los métodos explícitos insert y update solucionan el problema de la falta de precisión de la operación put. Puede seguir utilizando el método put, pero la utilización de los métodos explícitos insert y update hacen que su objetivo quede mucho más claro. El uso del método put se clarifica si se define un método put sin ninguna operación get previa como método insert. Si se ha intentado una operación get previa a la operación put, la operación put se tratará como una operación insert o update dependiendo de si la entrada existe en la antememoria.

   Puede realizar las siguientes operaciones básicas de ObjectMap: get, put, insert, update, remove, touch, invalidate y containsKey. Se pueden obtener varios detalles y variaciones en el tema “Visión general de modelo de programación del sistema” en la página 45 o en la documentación de la API de ObjectMap. El siguiente ejemplo muestra el uso de ObjectMap para modificar la antememoria:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid objectGrid =
objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
ObjectMap objectMap = session.getMap("someMap");
// Iniciar una transacción/sesión...
session.begin();
objectMap.insert("key1", "value1");
objectMap.put("key2", "value2");
session.commit();
// Verificar que los cambios se han comprometido
String value1 = (String)objectMap.get("key1");
String value2 = (String)objectMap.get("key2");
System.out.println("key1 = " + value1 + ", key2 = " + value2);
// Iniciar una nueva transacción/sesión...
session.begin();
objectMap.update("key2", "newValue2");
objectMap.remove("key1");
session.rollback();
// Verificar que los cambios no se han comprometido
String newValue1 = (String)objectMap.get("key1");
String newValue2 = (String)objectMap.get("key2");
System.out.println("key1 = " + newValue1 + ", key2 = " + newValue2);
```

6. Utilice el índice para buscar objetos en antememoria. El uso del índice permite a las aplicaciones buscar objetos a partir de un valor específico o de un rango de valores. La correlacionar BackingMap debe tener el plug-in de índice

configurado antes de que las aplicaciones puedan utilizar la función de índice. Las aplicaciones deben obtener el objeto de índice del método getIndex() en la interfaz ObjectMap y convertirlo en la interfaz correcta de índice como MapIndex o MapRangeIndex o una interfaz de índice personalizada.

Actualmente, la característica de creación de índices sólo está soportada en la antememoria local. La característica de índice no está soportada en la antememoria distribuida. Si se intenta llevar a cabo una operación de índice contra una antememoria distribuida, se producirá la excepción `UnsupportedOperationException`.

El siguiente ejemplo demuestra cómo utilizar el índice:

```
MapRangeIndex myIndex = (MapRangeIndex ) objectMap.getIndex("indexName");
Object searchCriteria = "targetAttributeValue";
Iterator iter = myIndex.findAll(searchCriteria);
    while (iter.hasNext()) {
Object key = iter.next();
System.out.println(objectMap.get(key));
}
```

Al terminar de leer este apartado y empezar a experimentar con el código de ejemplo, se irá sintiendo más cómodo con el modelo de programación esencial de ObjectGrid.

Para obtener información específica, consulte el Capítulo 9, “Visión general de la interfaz de programación de la aplicación ObjectGrid”, en la página 93.

Guía de iniciación de ObjectGrid remoto

Escriba una breve descripción aquí, que se utilizará como primer párrafo y como resumen.

El Capítulo 4, “Guía de Aprendizaje de ObjectGrid: modelo de programación de aplicaciones”, en la página 41 suele tratar de un uso de ObjectGrid ″local″ o bien dentro de la aplicación. Una aplicación ha creado una instancia de ObjectGrid y ha utilizado esa instancia. Cuando ha terminado la máquina virtual Java (JVM) ha terminado, también se termina la antememoria de ObjectGrid. La ObjectGrid remota, como el propio nombre indica, permite el acceso a una ObjectGrid que reside en otra JVM. Varios clientes pueden conectarse a la ObjectGrid remota y acceder a la ObjectGrid utilizando la misma API de forma transparente.

1. Revise los apartados siguientes para comenzar:
   - Capítulo 3, “Visión general de ObjectGrid”, en la página 19
   - “Configuración de ObjectGrid” en la página 261
   - “API de conexión de cliente ObjectGrid” en la página 100
   - Capítulo 8, “Soporte de línea de mandatos”, en la página 83
2. Para iniciar los servidores, debe definir el archivo XML de ObjectGrid y el archivo XML de clúster. Consulte “Configuración de ObjectGrid distribuida” en la página 274. Este tema hace referencia a los archivos `university.xml` y `universityCluster.xml`. Puede utilizar estos archivos como ejemplo, modificando el sistema principal y el puerto para iniciar o arrancar el servidor. Consulte Capítulo 8, “Soporte de línea de mandatos”, en la página 83 si desea más detalles acerca del inicio de servidores ObjectGrid.
3. Cuando se ejecuta un servidor, un cliente puede conectarse a este servidor. Consulte “API de conexión de cliente ObjectGrid” en la página 100 si desea más detalles acerca de cómo se conecta un cliente y cómo realiza operaciones de ObjectGrid.

Visión general de modelo de programación del sistema

El modelo de programación del sistema proporciona varias características y puntos de extensión adicionales para ObjectGrid.

El siguiente diagrama ilustra cómo el modelo de programación del sistema proporciona varias características y puntos de extensión adicionales.

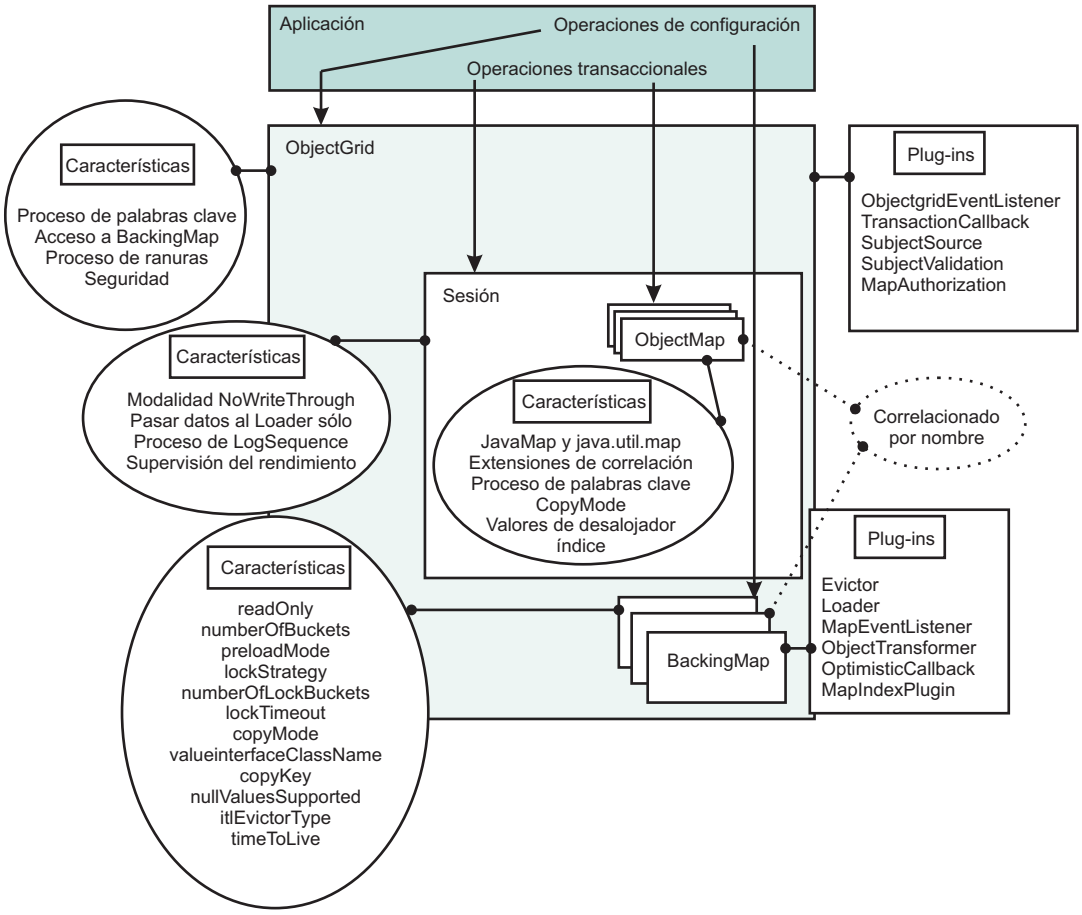


Figura 13. Visión general de ObjectGrid

Un plug-in de ObjectGrid es un componente que proporciona un determinado tipo de función a los componentes conectables de ObjectGrid que incluyen ObjectGrid y BackingMap. Una característica representa una función o rasgo específicos de un componente ObjectGrid, incluidos ObjectGrid, Session, BackingMap, ObjectMap, etc. Si una característica representa una función, puede utilizarse para obtener un objetivo específico de proceso. Si una característica es un rasgo, puede utilizarse para ajustar el comportamiento de los componentes ObjectGrid.

Cada uno de los apartados siguientes describe alguna de las características o extensiones que se ilustran en el diagrama anterior:

- "Visión general del modelo de programación del sistema: puntos de conexión y características de la interfaz ObjectGrid" en la página 46

  La interfaz ObjectGrid tiene varios puntos de conexión y características que permiten interacciones con ObjectGrid más amplias.

- “Visión general del modelo de programación del sistema: puntos de conexión y características de la interfaz BackingMap” en la página 48

  La interfaz BackingMap tiene varios puntos de conexión y características opcionales que permiten interacciones con ObjectGrid más amplias.
- “Visión general del modelo de programación del sistema: características de la interfaz Session” en la página 56

  La interfaz Session tiene varias características que permiten interacciones con ObjectGrid más amplias. Cada uno de los apartados de este tema describe una característica y proporciona un fragmento de código para el caso de ejemplo de uso.
- “Visión general del modelo de programación del sistema: características de la interfaz ObjectMap” en la página 58

  La interfaz ObjectMap tiene varias características que permiten interacciones con ObjectGrid más amplias. Cada uno de los apartados de este tema describe una característica y proporciona un fragmento de código para el caso de ejemplo de uso.

Para obtener más información sobre las características y plug-ins individuales, consulte el Capítulo 9, “Visión general de la interfaz de programación de la aplicación ObjectGrid”, en la página 93.

Visión general del modelo de programación del sistema: puntos de conexión y características de la interfaz ObjectGrid

La interfaz ObjectGrid tiene varios puntos de conexión y características que permiten interacciones con ObjectGrid más amplias.

Cada uno de los apartados siguientes describe una característica y proporciona un fragmento de código para el caso de ejemplo de uso. Siempre que sea adecuado, se facilitará un fragmento de código para mostrar la configuración XML alternativa. Si desea información más extensa, consulte los temas “Interfaz ObjectGrid” en la página 107 y “Configuración de ObjectGrid” en la página 261.

Proceso de palabras clave

La interfaz ObjectGrid proporciona un mecanismo de invalidación flexible basado en las palabras clave. Una palabra clave es una instancia no nula de cualquier objeto serializable. Puede asociar palabras clave con entradas de BackingMap del modo que desee. La mayor parte del proceso de palabras clave se realiza en el nivel de ObjectMap, pero la asociación de una palabra clave con otra palabra clave para formar un árbol de palabras clave con una estructura jerárquica se realiza en el nivel de ObjectGrid.

El método associateKeyword(java.io.Serializable parent, java.io.Serializable child) enlaza las dos palabras clave en una relación direccional. Si se invalida un padre (parent), también se invalidará el hijo (child). La invalidación de un hijo no tiene ningún impacto sobre en el padre. Por ejemplo, este método se utiliza para añadir una entrada de correlación Nueva York como hijo de la entrada de correlación EEUU, de modo que, si se invalida EEUU, también se invalidan todas las entradas Nueva York. Consulte el siguiente ejemplo de código:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
// asociar varias ciudades con la palabra clave "EEUU"
objectGrid.associateKeyword("EEUU", "Nueva York");
objectGrid.associateKeyword("EEUU", "Rochester");
objectGrid.associateKeyword("EEUU", "Raleigh");
:
```

```
// insertar varias entradas con distintas palabras clave
objectMap.insert("key1", "value1", "Nueva York");
objectMap.insert("key2", "value2", "México");
objectMap.insert("key3", "value3", "Raleigh");
objectMap.insert("key4", "value4", "EEUU");
objectMap.insert("key5", "value5", "Rochester");
objectMap.insert("key6", "value6", "Francia");
:
// invalidar todas las entradas asociadas con la palabra clave "EEUU",
// dejando las entradas "key2" y "key6"
objectMap.invalidateUsingKeyword("EEUU", true);
:
```

Para obtener más información, consulte el “Palabras clave” en la página 124.

Acceso a BackingMap

ObjectGrid proporciona acceso a los objetos BackingMap. Puede obtener acceso a un objeto BackingMap con los métodos defineMap o getMap. Para obtener más información, consulte “Interfaz BackingMap” en la página 112. El siguiente ejemplo crea dos referencias a BackingMap:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
BackingMap newBackingMap = objectGrid.defineMap("newMap");
:
```

Proceso de ranuras

Puede reservar una ranura para almacenar objetos que se utilicen en el transcurso de la transacción como, por ejemplo, el objeto de ID de transacción (TxID) o un objeto de conexión de base de datos (Conexión). Posteriormente se hace referencia a estos objetos almacenados mediante un índice específico facilitado por el método reserveSlot. Encontrará información adicional sobre la utilización de ranuras en los temas “Cargadores” en la página 202 y “Plug-in TransactionCallback” en la página 218. El siguiente fragmento de código muestra el proceso de ranuras:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
int index = objectGrid.reserveSlot
   (com.ibm.websphere.objectgrid.TxID.SLOT_NAME);
:
// Utilizar el índice posteriormente al almacenar objetos del objeto TxID
// o recuperarlos de él ...
TxID tx = session.getTxID();
tx.putSlot(index, someObject);
:
Object theTxObject = tx.getSlot(index);
:
```

Proceso de seguridad

Las correlaciones pueden protegerse mediante mecanismos de seguridad. Los siguientes métodos están disponibles en ObjectGrid para configurar y utilizar las características de seguridad.

- getSession(Subject)
- SubjectSource
- SubjectValidation
- AuthorizationMechanism
- MapAuthorization
- PermissionCheckPeriod

Consulte “Seguridad de ObjectGrid” en la página 140 para obtener más información sobre los mecanismos de seguridad disponibles.

ObjectGridEventListener

El receptor ObjectGridEventListener proporciona un modo para que las aplicaciones reciban una notificación en el caso de que una transacción se inicie o se comprometa. Se puede establecer una instancia de ObjectGridEventListener en ObjectGrid. Consulte el tema “Receptores” en la página 187 para obtener más información. A continuación aparece un ejemplo sobre cómo implementar la interfaz ObjectGridEventListener mediante programación:

```
class MyObjectGridEventListener implements
com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
objectGrid.addEventListener(new MyObjectGridEventListener());
:
```

También puede realizar la misma configuración con XML:

```
:
<objectGrids>
<objectGrid name="someGrid">
<bean id="ObjectGridEventListner" className=
"com.somecompany.MyObjectGridEventListener" />
:
</objectGrid>
</objectGrids>
:
```

Plug-in TransactionCallback

La llamada de métodos en la sesión envía los sucesos correspondientes al plug-in TransactionCallback. ObjectGrid puede tener cero o un plug-in TransactionCallback. Las BackingMaps definidas en una ObjectGrid con un plug-in TransactionCallback deben tener el correspondiente Loader. Para obtener más información, consulte “Plug-in TransactionCallback” en la página 218. El siguiente fragmento de código demuestra cómo implementar el plug-in TransactionCallback mediante programación:

```
class MyTransactionCallback implements
com.ibm.websphere.objectgrid.plugins.TransactionCallback { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
objectGrid.setTransactionCallback(new MyTransactionCallback());
:
```

Puede realizar la misma configuración con XML:

```
:
<objectGrids>
<objectGrid name="someGrid">
<bean id="TransactionCallback" className=
"com.somecompany.MyTransactionCallback" />
</objectGrid>
</objectGrids>
:
```

Visión general del modelo de programación del sistema: puntos de conexión y características de la interfaz BackingMap

La interfaz BackingMap tiene varios puntos de conexión opcionales que permiten interacciones con ObjectGrid más amplias.

Cada uno de los apartados siguientes describe una característica y proporciona un fragmento de código para el caso de ejemplo de uso. Siempre que sea adecuado, se facilitará un fragmento de código para mostrar la configuración XML alternativa. Para obtener una información más amplia, consulte los temas “Interfaz BackingMap” en la página 112 y “Configuración de ObjectGrid” en la página 261 o la documentación de la API.

Atributos de configuración

Existen varios elementos de configuración asociados con las BackingMaps:

- **ReadOnly** (toma `false` como valor por omisión): si este atributo se establece en `true`, la correlación de respaldo pasa a ser de sólo lectura. Si se establece en `false`, la correlación de respaldo será de lectura y escritura. Si no se especifica ningún valor, se impondrá el valor por omisión de lectura y escritura.
- **NullValuesSupported** (toma `true` como valor por omisión): el soporte de valores nulos indica que un valor nulo puede colocarse en una correlación. Si este atributo se establece en true, ObjectMap dará soporte a los valores nulos; de lo contrario, éstos no recibirán soporte. Si se da soporte a los valores nulos, una operación `get` que devuelva un valor nulo puede indicar que el valor es nulo o que la correlación no contiene la clave que se ha pasado.
- **NumberOfBuckets** (toma `503` como valor por omisión): especifica el número de cubetas que utiliza esta BackingMap. La implementación de BackingMap utiliza una correlación hash para su implementación. Si existen numerosas entradas en BackingMap, cuantas más cubetas haya, mejor será el rendimiento, ya que el riesgo de colisiones disminuye a medida que crece el número de cubetas. Cuanto más elevado sea el número de cubetas, mayor será la concurrencia.
- **NumberOfLockBuckets** (toma `383` como valor por omisión): especifica el número de cubetas de bloqueo que utiliza el gestor de bloqueos para esta BackingMap. Cuando el atributo 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 numerosas entradas en la correlación hash, cuantas más cubetas de bloqueo haya, mejor será el rendimiento, ya que el riesgo de colisiones disminuye a medida que crece el número de cubetas. Cuanto más elevado sea el número de cubetas de bloqueo, mayor será la concurrencia. Cuando lockStrategy es NONE, esta BackingMap no utiliza ningún gestor de bloqueos. En este caso, el establecimiento del atributo numberOfLockBuckets no tiene ningún efecto.

Ejemplo de configuración mediante programación

El siguiente ejemplo configura propiedades en una correlación de respaldo:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// sobrescribir el valor por omisión de lectura/escritura
backingMap.setReadOnly(true);
// sobrescribir el valor por omisión que permite valores nulos
backingMap.setNullValuesSupported(false);
// sobrescribir el valor por omisión (los números primos funcionan mejor)
backingMap.setNumberOfBuckets(251);
// sobrescribir el valor por omisión (los números primos funcionan mejor)
backingMap.setNumberOfLockBuckets(251);
:
```

Ejemplo de configuración XML

El siguiente ejemplo de configuración XML configura las mismas propiedades que se muestran en el ejemplo anterior realizado mediante programación.

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" readOnly="true" nullValuesSupported="false"
numberOfBuckets="251" numberOfLockBuckets="251" />
</objectGrid>
</objectGrids>
:
```

Estrategia de bloqueo

Cuando la estrategia de bloqueo se establece en OPTIMISTIC o PESSIMISTIC, se crea un gestor de bloqueos para la BackingMap. Para impedir que se produzcan puntos muertos, el gestor de bloqueos tiene un valor de tiempo de espera por omisión que espera a que se conceda un bloqueo. Si se excede el límite de este tiempo de espera, se genera una excepción LockTimeoutException. El valor por omisión de 15 segundos es suficiente para la mayoría de las aplicaciones, pero en un sistema con una carga excesiva, podría darse un tiempo de espera sin que exista ningún punto muerto real. En este caso, el método setLockTimeout puede utilizarse para aumentar el valor de tiempo de espera de bloqueo del valor por omisión al valor que sea necesario para evitar que se generen excepciones de tiempos de espera falsos. Cuando la estrategia de bloqueo es NONE, esta BackingMap no utiliza ningún gestor de bloqueos. En este caso, el establecimiento del atributo lockTimeout no tiene ningún efecto. Para obtener más información, consulte el tema “Bloqueo” en la página 131.

Ejemplo de configuración mediante programación

El siguiente ejemplo establece la estrategia de bloqueo:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// sobrescribir el valor por omisión de OPTIMISTIC
backingMap.setLockStrategy(LockStrategy.PESSIMISTIC);
backingMap.setLockTimeout(30);
// establece el tiempo de espera de bloqueo en 30 segundos
:
```

Ejemplo de configuración XML

El siguiente ejemplo establece la misma estrategia de bloqueo que se define en el ejemplo anterior realizado mediante programación:

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" lockStrategy="PESSIMISTIC" lockTimeout="30" />
</objectGrid>
</objectGrids>
:
```

Copia de claves y valores

La realización de copias de claves y valores puede resultar costosa desde el punto de vista de los recursos y del rendimiento. Sin la posibilidad de realizar estas copias, pueden ocurrir problemas extraños y difíciles de depurar. ObjectGrid proporciona la posibilidad de configurar si se van a hacer copias de claves o valores y cuándo deben realizarse. Generalmente, las claves se consideran

inmutables, de modo que no es necesario hacer copias de los objetos de clave. La modalidad por omisión para los objetos de clave es no realizar copias. Es más probable que la aplicación modifique los objetos de valor. Se puede configurar si se proporciona una copia del objeto Value o una referencia real del objeto Value. Consulte el Capítulo 11, “Procedimientos recomendados para el rendimiento de ObjectGrid”, en la página 329 y el JavaDoc para obtener información detallada adicional sobre los valores CopyKey y CopyMode.

Ejemplo de configuración mediante programación

A continuación, aparece un ejemplo sobre cómo establecer la modalidad de copia y los valores de copia de claves:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.setCopyKey(true); // realizar una copia de todas las claves nuevas
backingMap.setCopyMode(NO_COPY); // Más eficaz: confiar en la aplicación
:
```

Ejemplo de configuración XML

El siguiente ejemplo da como resultado la misma configuración que el ejemplo de configuración anterior realizado mediante programación:

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" copyKey="true" copyMode="NO_COPY" />
</objectGrid>
</objectGrids>
:
```

Desalojadores

Los desalojadores (Evictors) se utilizan para limpiar periódicamente las entradas de la correlación que no sean necesarias. El Evictor define las entradas que se eliminan. Los Evictors incorporados están basados en el tiempo, de modo que la estrategia de desalojo se basa en el periodo de tiempo que una entrada ha estado activa en la correlación. Otras estrategias de desalojo se basan en el uso, el tamaño o una combinación de factores.

- **Desalojador TTL (tiempo de duración) incorporado**: el desalojador de tiempo de duración incorporado proporciona un par de elementos de configuración que se establecen en BackingMap con los métodos setTtlEvictorType y setTimeToLive. Por omisión, este desalojador TimeToLive incorporado no está activo. Puede activarlo llamando al método setTtlEvictorType con uno de estos tres valores: CREATION_TIME, LAST_ACCESS_TIME o NONE (por omisión). Según el tipo de desalojador TimeToLive seleccionado, se utiliza el valor para el método setTimeToLive para establecer la duración de todas las entradas de correlación.
- **Plug-ins Evictor:** además del Evictor de tiempo de duración incorporado, una aplicación puede proporcionar su propio plug-in Evictor de implementación. Puede utilizar cualquier algoritmo periódicamente para invalidar las entradas de correlación.

Configuración mediante programación

La siguiente clase crea un desalojador:

```
class MyEvictor implements com.ibm.websphere.objectgrid.plugins.Evictor { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// el temporizador se inicia cuando se crea la entrada por primera vez
backingMap.setTtlEvictorType(CREATION_TIME);
// Permitir que cada entrada de correlación dure 30 segundos antes
// de la invalidación
backingMap.setTimeToLive(30);
// Los Evictors incorporados y personalizados estarán activos
backingMap.setEvictor(new MyEvictor()); :
```

Configuración XML

El siguiente código XML crea una configuración idéntica a la configuración anterior realizada mediante programación:

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" pluginCollection="default"
ttlEvictorType="CREATION_TIME" timeToLive="30" />
</objectGrid>
</objectGrids>
:
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="Evictor" className="com.somecompany.MyEvictor" />
</backingMapPluginCollection>
</backingMapPluginCollections>
:
```

Para obtener más información, consulte “Desalojadores” en la página 192.

Cargadores

Un cargador (Loader) de ObjectGrid es un componente conectable que permite que una correlación de ObjectGrid se comporte como una antememoria en memoria para los datos que normalmente se mantienen en un almacenamiento persistente bien en el mismo sistema bien en otro sistema. Generalmente, se utiliza una base de datos o un sistema de archivos como almacenamiento persistente. Un cargador posee la lógica para leer y escribir datos en un almacenamiento persistente.

Un Loader es un plug-in opcional de una correlación de respaldo de ObjectGrid. Sólo un único Loader puede asociarse con una determinada correlación de respaldo, y todas las correlaciones de respaldo tienen su propia instancia de Loader. La correlación de respaldo solicita cualquier dato que no provenga de su Loader. Cualquier cambio en la correlación se pasa al Loader. El plug-in Loader proporciona un modo para que la correlación de respaldo pueda mover datos entre la correlación y su almacenamiento persistente.

Configuración mediante programación

A continuación aparece un ejemplo de la implementación de un cargador:

```
class MyLoader implements com.ibm.websphere.objectgrid.plugins.Loader { .. }
:
Loader myLoader = new MyLoader();
myLoader.setDataBaseName("testdb");
myLoader.setIsolationLevel("ReadCommitted");
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
```

```
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.setLoader(myLoader);
backingMap.setPreloadMode(true);
:
```

Configuración XML

El siguiente ejemplo XML da como resultado la misma configuración que el ejemplo de configuración anterior realizado mediante programación:

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" pluginCollectionRef="default" preloadMode="true" />
</objectGrid>
</objectGrids>
:
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="Loader" classname="com.somecompany.MyLoader">
<property name="dataBaseName" type="java.lang.String" value="testdb" />
<property name="isolationLevel" type="java.lang.String" value="ReadCommitted" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
:
```

Para obtener más información, consulte el tema “Cargadores” en la página 202.

Interfaz MapEventListener

La aplicación implementa la interfaz MapEventListener de retorno de llamada cuando desea recibir sucesos sobre una correlación como, por ejemplo, el desalojo de una entrada de correlación o la finalización de una precarga de datos. El siguiente ejemplo de código muestra cómo se puede establecer una instancia de MapEventListener en una instancia de BackingMap:

Configuración mediante programación

```
class MyMapEventListener implements
 com.ibm.websphere.objectgrid.plugins.MapEventListener { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.addMapEventListener(new MyMapEventListener() );
```

Configuración XML

El siguiente ejemplo da como resultado la misma configuración que el ejemplo de configuración anterior realizado mediante programación:

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
:
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="MapEventListener" classname="com.somecompany.MyMapEventListener" />
</backingMapPluginCollection>
</backingMapPluginCollections>
:
```

Consulte el tema “Receptores” en la página 187 para obtener más información.

Interfaz ObjectTransformer

ObjectTransformer puede utilizarse para serializar los valores y claves de entrada de antememoria que no estén definidos como serializables de modo que el usuario pueda definir su propio esquema de serialización sin ampliar o implementar la interfaz Serializable directamente. Esta interfaz además proporciona los métodos para realizar la función de copia en las claves y valores. A continuación, aparece una clase que implementa la interfaz ObjectTransformer:

Configuración mediante programación

```
class MyObjectTransformer implements
 com.ibm.websphere.objectgrid.plugins.ObjectTransformer { ... }
:
ObjectTransformer myObjectTransformer = new MyObjectTransformer();
myObjectTransformer.setTransformType("full");
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.setObjectTransformer(myObjectTransformer);
:
```

Configuración XML

El siguiente ejemplo XML da como resultado la misma configuración que el ejemplo anterior realizado mediante programación:

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
:
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="ObjectTransformer" className="com.somecompany.MyObjectTransformer">
<property name="transformType" type="java.lang.String" value="full"
description="..." />
</bean>
</backingMapCollection>
</backingMapCollections>
:
```

Para obtener más información, consulte el tema “Plug-in ObjectTransformer” en la página 214.

Interfaz OptimisticCallback

La interfaz OptimisticCallback puede utilizarse para crear y procesar un campo de versión que esté asociado con un determinado objeto Value. En muchos casos, la utilización de un objeto Value para determinar directamente si otro cliente de antememoria ha modificado su valor desde su recuperación resulta ineficaz y puede dar lugar a errores. Una alternativa consiste en proporcionar otro campo que represente el estado del objeto Value. El objetivo de la interfaz OptimisticCallback es proporcionar un objeto Versioned Value alternativo que represente al objeto Value. A continuación, aparece una configuración de ejemplo de la interfaz OptimisticCallback:

Configuración mediante programación

```
class MyOptimisticCallback implements
 com.ibm.websphere.objectgrid.plugins.OptimisticCallback { ... }
:
OptimisticCallback myOptimisticCallback = new MyOptimisticCallback();
myOptimisticCallback.setVersionType("Integer");
backingMap.setOptimisticCallback(myOptimisticCallback);
:
```

Configuración XML

El siguiente ejemplo da como resultado la misma configuración que el ejemplo de configuración anterior realizado mediante programación:

```
:
<objectGrids>
<objectGrid name="someGrid">
<backingMap name="someMap" pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
:
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="OptimisticCallBack" classname="com.somecompany.MyOptimisticCallback">
<property name="versionType" type="java.lang.string" value="Integer"
description="..." />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
:
```

Índices

Un MapIndexPlugin, o dicho brevemente un índice, es una opción que utiliza BackingMap para crear un índice basado en el atributo especificado del objeto almacenado. El índice permite a las aplicaciones buscar objetos a partir de un valor específico o de un rango de valores. Para utilizar el índice, las aplicaciones deben obtener el objeto de índice del método getIndex() en la interfaz ObjectMap y convertirlo en la interfaz correcta de índice como MapIndex o MapRangeIndex o una interfaz de índice personalizada.

Actualmente, la característica de creación de índices sólo está soportada en la antememoria local, no en la antememoria distribuida. Si se intenta llevar a cabo una operación de índice contra una antememoria distribuida, se producirá UnsupportedOperationException.

Existen dos tipos de índices: estáticos y dinámicos. Los índices estáticos pueden crearse mediante configuración programada y configuración XML. Los índices dinámicos sólo se pueden crear de forma programada.

Configuración mediante programación

El siguiente ejemplo de código muestra cómo se puede añadir un índice estático a una instancia de BackingMap:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("indexSampleGrid");
BackingMap personBackingMap= objectGrid.getMap("person");
//utilice la clase com.ibm.websphere.objectgrid.plugins.index.HashIndex
//incorporada como clase de plugin de índice.
HashIndex mapIndexPlugin = new HashIndex();
mapIndexPlugin.setName("CODE");
mapIndexPlugin.setAttributeName("EmployeeCode");
mapIndexPlugin.setRangeIndex(true);
```

```
personBackingMap.addMapIndexPlugin(mapIndexPlugin);
//Nota: la configuración de Index anterior da por supuesto que el objeto almacenado
//tiene un atributo denominado EmployeeCode y un método denominado getEmployeeCode()
//que devuelven el valor del atributo EmployeeCode
:
```

El siguiente ejemplo de código muestra cómo se puede crear un índice dinámico en una instancia de BackingMap:

```
class DynamicIndexCallbackImpl implements
com.ibm.websphere.objectgrid.plugins.index.DynamicIndexCallback { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("indexSampleGrid");
BackingMap personBackingMap= objectGrid.getMap("person");
objectGrid.initiallize();
:
//insertar, actualizar o eliminar datos
//El índice dinámico puede crearse después de que la instancia de ObjectGrid
//que lo contiene se haya inicializado
//Si se necesita crear un índice dinámico, créelo sin
//DynamicIndexCallback
personBackingMap.createDynamicIndex("CODE2", true, "employeeCode", null);
:
//Otra opción es crear un índice dinámico con DynamicIndexCallback
//Si se presupone que hay una clase DynamicIndexCallbackImpl se implementa
//la interfaz DynamicIndexCallback
personBackingMap.createDynamicIndex("CODE3", true, "employeeCode",
new DynamicIndexCallbackImpl());
:
```

Configuración XML

El siguiente ejemplo da como resultado la misma configuración que el ejemplo de índice estático anterior realizado mediante programación:

```
:
<objectGrids>
<objectGrid name="indexSampleGrid">
<backingMap name="person" pluginCollectionRef="person" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="person">
<bean id="MapIndexPlugin
className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="CODE"
description="nombre índice" />
<property name="RangeIndex" type="boolean" value="true"
description="true para MapRangeIndex />
<property name="AttributeName" type="java.lang.String"
value="employeeCode" description="nombre atributo" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
:
```

Consulte el apartado sobre índices si desea más información.

Visión general del modelo de programación del sistema: características de la interfaz Session

La interfaz Session tiene varias características que permiten interacciones con ObjectGrid más amplias. Cada uno de los apartados siguientes describe una característica y proporciona un fragmento de código para el caso de ejemplo de uso.

Para obtener más información sobre la interfaz Session, consulte “Interfaz Session” en la página 116.

Modalidad NoWriteThrough

En ocasiones, las aplicaciones sólo desean aplicar cambios a la correlación base pero no al Loader. El método beginNoWriteThrough de la interfaz Session está diseñado para lograr este objetivo. El método isWriteThroughEnabled de la interfaz Session puede utilizarse para verificar si la sesión actual está escribiendo en el Loader de programa de fondo. Esto puede ser útil para que otros usuarios del objeto Session sepan qué tipo de sesión se está procesando actualmente. El ejemplo siguiente habilita la modalidad NoWriteThrough:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
boolean isWriteThroughEnabled = session.isWriteThroughEnabled();
// realizar actualizaciones de la correlación ...
session.commit();
:
```

Pasar datos al Loader sólo

Las aplicaciones sólo pueden aplicar los cambios locales de la sesión al Loader sin comprometer estos cambios permanentemente mediante la invocación del método flush, como se muestra en el ejemplo siguiente:

```
:
Session session = objectGrid.getSession();
session.begin();
// realizar algunos cambios ...
session.flush(); // pasar estos cambios al Loader, sin comprometerlos todavía
// realizar más cambios ...
session.commit();
:
```

Método processLogSequence

El método processLogSequence se utiliza para procesar una LogSequence. Todos los LogElement incluidos en la LogSequence se examinan y se realiza la operación adecuada, por ejemplo, insert, update o invalidate en la BackingMap identificada por el MapName de LogSequence. Una Session de ObjectGrid debe estar activa antes de que se invoque este método. El emisor es responsable de emitir las llamadas de compromiso o retrotracción adecuadas para completar la Session. El proceso de compromiso automático no está disponible para esta invocación de método.

La principal utilidad de este método es procesar una LogSequence recibida por una JVM remota. Por ejemplo, utilizando el soporte de compromiso distribuido, las LogSequences que se asocien con una determinada sesión comprometida se distribuyen a otras ObjectGrids receptoras de otras máquinas virtuales Java (JVM). Después de recibir las LogSequences en la JVM remota, el receptor puede iniciar una Session mediante el método beginNoWriteThrough, invocar el método processLogSequence y realizar el método commit en la Session. A continuación se muestra un ejemplo:

```
:
session.beginNoWriteThrough();
try {
 session.processLogSequence(inputSequence);
}
catch (Exception e) {
session.rollback();
 throw e;
}
session.commit();
:
```

Supervisión del rendimiento

Las correlaciones pueden instrumentarse opcionalmente para la supervisión del rendimiento mientras se ejecutan en WebSphere Application Server. El método `setTransactionType` está disponible en una Session para configurar y utilizar las características de supervisión del rendimiento. Para obtener más información, consulte “Supervisión del rendimiento de ObjectGrid con PMI (Performance Monitoring Infrastructure) de WebSphere Application Server” en la página 295.

Visión general del modelo de programación del sistema: características de la interfaz ObjectMap

La interfaz ObjectMap tiene varias características que permiten interacciones con ObjectGrid más amplias.

Para obtener más información sobre la interfaz ObjectMap, consulte “Interfaces ObjectMap y JavaMap” en la página 120.

La interfaz JavaMap y la interfaz java.util.Map

Para las aplicaciones que desean utilizar la interfaz java.util.Map, ObjectMap dispone del método getJavaMap para que las aplicaciones puedan obtener la implementación de la interfaz java.util.Map respaldada por ObjectMap. La instancia de Map devuelta se puede convertir a la interfaz JavaMap, que amplía la interfaz java.util.Map. La interfaz JavaMap tiene las mismas signaturas de método que ObjectMap, pero se diferencian en el manejo de las excepciones. La interfaz JavaMap amplía la interfaz java.util.Map, de modo que todas las excepciones son instancias de la clase java.lang.RuntimeException. Como la interfaz JavaMap amplía la interfaz java.util.Map, es fácil utilizar ObjectGrid con rapidez con una aplicación existente que utilice una interfaz java.util.Map para la colocación en antememoria de objetos. A continuación, aparece un fragmento de código:

```
:
JavaMap javaMap = (JavaMap)objectMap.getJavaMap();
:
```

Extensiones de correlación

La interfaz ObjectMap también proporciona funciones adiciones además de las funciones de las excepciones comprobadas. Por ejemplo, un usuario puede especificar que una determinada entrada de correlación se actualice con el método getForUpdate, lo que indica al tiempo de ejecución de ObjectGrid y al plug-in Loader que la entrada puede bloquearse durante el proceso, si es necesario. El proceso por lotes es otra posibilidad adicional con los métodos getAll, putAll y removeAll. Para obtener más información sobre estos métodos, consulte la documentación de la API.

Proceso de palabras clave

La mayoría de las operaciones de correlación tienen la versión del parámetro de palabra clave como, por ejemplo, insert, get, getForUpdate, put, remove e invalidate. Para facilitar las cosas, también se proporciona el método setDefaultKeyword. Este método asocia entradas con una palabra clave sin utilizar la versión de la palabra clave de la operación de correlación. A continuación, aparece un ejemplo de palabra clave:

```
:
// setDefaultKeyword
session.begin();
objectMap.setDefaultKeyword("Nueva York");
Person p = (Person) objectMap.get("Billy"); // La entrada "Billy" tiene la
                                                    // palabra clave "Nueva York"
p = (Person) objectMap.get("Bob", "Los Angeles"); // La entrada "Bob"
//tiene la palabra clave "Los Angeles"
objectMap.setDefaultKeyword(null);
p = (Person) objectMap.get("Jimmy"); // La entrada "Jimmy" no tiene palabra clave
session.commit();
:
// versión del parámetro de palabras clave de la operación insert
session.begin();
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person, "Rochester"); // "BillyBob" tiene
//la palabra clave "Rochester"
session.commit();
:
```

Para obtener más información, consulte “Palabras clave” en la página 124.

Método de modalidad de copia

El método setCopyMode permite que la modalidad de copia de la Map se altere temporalmente únicamente en esta correlación de esta sesión o transacción. Este método permite que una aplicación utilice una modalidad de copia óptima por sesión, como sea necesario. La modalidad de copia no puede modificarse durante una sesión activa. Existe el correspondiente método clearCopyMode que restablece la modalidad de copia a la modalidad definida en BackingMap. Sólo puede llamar a este método cuando no existan sesiones activas. A continuación, aparece un ejemplo sobre cómo establecer la modalidad de copia:

```
:
objectMap.setCopyMode(CopyMode.COPY_ON_READ, null);
session.begin();
// modificar objectMap ...
session.commit();
objectMap.clearCopyMode(); // restablecer CopyMode al valor de BackingMap
session.begin();
// modificar objectMap ...
session.commit();
:
```

Para obtener más información, consulte los temas “Plug-in ObjectTransformer” en la página 214 y Capítulo 11, “Procedimientos recomendados para el rendimiento de ObjectGrid”, en la página 329.

Valores de desalojador

Puede alterar temporalmente el valor de tiempo de espera de TimeToLive del desalojador TimeToLive incorporado en el nivel de ObjectMap. El método setTimeToLive establece el número de segundos que puede durar cualquier

entrada de antememoria. Cuando se modifica, se devuelve el valor TimeToLive anterior. Este valor TimeToLive es el tiempo mínimo que una entrada permanece en la antememoria antes de que se la considere apta para el desalojo e indica al desalojador TimeToLive incorporado cuánto tiempo debe permanecer una entrada después de la última hora de acceso. El nuevo valor de TimeToLive sólo se aplica a las entradas de ObjectMap a las que accede la transacción iniciada por el objeto Session utilizado para obtener la instancia de ObjectMap. El nuevo valor de TimeToLive se aplica a cualquier transacción que esté en proceso para la Session y a transacciones futuras que ejecute la Session. El nuevo valor de TimeToLive no afecta a las entradas de una instancia de ObjectMap a las que accede una transacción iniciada por cualquier otra Session. Al llamar a este método en ObjectMap, cualquier valor anterior establecido por el método setTimeToLive en BackingMap se altera temporalmente para esta instancia de ObjectMap. A continuación se muestra un ejemplo:

```
:
session.begin();
int oldTTL = objectMap.setTimeToLive(60); // establecer TTL en 60 segundos
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person); // La entrada "BillyBob" tendrá un TTL
//de 60 segundos
session.commit();
:
objectMap.setTimeToLive(oldTTL); // establecer TTL en el valor original
Person person2 = new Person("Angelina", "Jolie", "somewhere");
objectMap.insert("Brad", person2); // La entrada "Brad" utilizará el valor
                                                 // original de TTL
:
```

Para obtener más información, consulte el tema “Desalojadores” en la página 192.

Capítulo 5. Ejemplos de ObjectGrid

En este tema se describen los ejemplos de ObjectGrid que se proporcionan al instalar el producto WebSphere Extended Deployment.

Visión general

Varios ejemplos de ObjectGrid ilustran la integración con las aplicaciones J2EE (Java 2 Platform, Enterprise Edition) y Partitioning Facility (WPF). En este tema se describe cada uno de los ejemplos, junto con sus características, su ubicación y los entornos donde se ejecutan.

En este tema se describen los ejemplos que se proporcionan al instalar WebSphere Extended Deployment. Se puede acceder a otros ejemplos relacionados con el uso de la integración JMS (Java Message Service) y la integración de ObjectGrid con otras infraestructuras de código fuente abierto en la dirección Web: http://www-1.ibm.com/support/docview.wss?uid=swg27006432.

Ejemplos

- **ObjectGridSamplesSA:** este ejemplo es un conjunto de ejemplos J2SE (Java 2 Platform, Standard Edition) empaquetados en el archivo `objectgridSamples.jar` para demostrar las funciones de ObjectGrid. Estos ejemplos J2SE pueden ejecutarse en un entorno J2SE. El archivo `objectgridSamples.jar` contiene el archivo `SamplesGuide.htm`, que tiene instrucciones para ejecutar estos ejemplos.
- **ObjectGridSample:** este ejemplo es un ejemplo J2EE que muestra cómo los servlets y los enterprise beans de Session utilizan las funciones de ObjectGrid. Este ejemplo se entrega con el archivo EAR (Enterprise Archive) `ObjectGridSample.ear`. El archivo `ObjectGridSample.ear` contiene el archivo `readme.txt`, que tiene instrucciones para configurar y ejecutar este ejemplo.
- **ObjectGridPartitionCluster:** este ejemplo es un ejemplo J2EE para mostrar cómo WPF y ObjectGrid funcionan conjuntamente, cómo utilizar ObjectGridEventListener para propagar los cambios y cómo habilitar el direccionamiento basado en el contexto para mantener la integridad y coherencia de ObjectGrid. Este ejemplo se entrega con el archivo EAR (Enterprise Archive) `D_ObjectGridPartitionClusterSample.ear`. El archivo `D_ObjectGridPartitionClusterSample.ear` contiene el archivo `readme.txt`, que tiene instrucciones para configurar y ejecutar este ejemplo.
- **ObjectGridJMSSamples:** es un conjunto de ejemplos J2EE empaquetados en el archivo ObjectGridJMSSamples.zip, que muestra cómo se utiliza la función JMS para transmitir cambios de una instancia de ObjectGrid a otra instancia de ObjectGrid en una única JVM o un entorno de clúster. Estos ejemplos J2EE sólo están disponibles en Internet en la siguiente dirección Web: http://www-1.ibm.com/support/docview.wss?uid=swg27006432.

Funcionalidad de los ejemplos

Tabla 2. Funciones de los ejemplos

Área funcional	Ejemplo ObjectGrid SamplesSA	Ejemplo ObjectGrid Sample	Ejemplo ObjectGrid PartitionCluster	Ejemplo ObjectGrid JMSSamples
ObjectGrid EventListener			x	x

Tabla 2. Funciones de los ejemplos (continuación)

Área funcional	Ejemplo ObjectGrid SamplesSA	Ejemplo ObjectGrid Sample	Ejemplo ObjectGrid PartitionCluster	Ejemplo ObjectGrid JMSSamples
Transaction callback	x	x	x	
Loader	x	x	x	
Escucha de MapEvent	x			
Object Transformer	x	x	x	x
Optimistic callback	x	x	x	
Modalidad de copia de BackingMap	x	x		
Invalidación distribuida			x	x
Actualización distribuida			x	x
Proceso de LogSequence				x
Partitioning Facility (WPF)			x	
Java Message Service (JMS)				x
Índice de correlación	x			
Clúster ObjectGrid	x	x		
Contexto ClusterClient de ObjectGrid	x	x		
ObjectGrid distribuida	x	x		
Gestión de ObjectGrid	x			
Seguridad de ObjectGrid	x		x	

Ubicación

Después de la instalación de WebSphere Extended Deployment, los siguientes archivos .jar están ubicados en los siguientes directorios:

Tabla 3. Ubicaciones de los ejemplos

Ejemplo	Ubicación
ObjectGridSamplesSA	*raíz_instalación*\optionalLibraries\ObjectGrid\objectgridSamples.jar
ObjectGridSample	*raíz_instalación*\installableApps\ObjectGridSample.ear

Tabla 3. Ubicaciones de los ejemplos (continuación)

Ejemplo	Ubicación
ObjectGridPartitionCluster	*raíz_instalación*\installableApps\ D_ObjectGridPartitionClusterSample.ear

Las versiones actualizadas de los ejemplos facilitados que se listan y de ejemplos adicionales como ObjectGridJMSSamples pueden encontrarse en la Web en la siguiente dirección: http://www-1.ibm.com/support/docview.wss?uid=swg27006432. Asimismo, encontrará artículos en IBM DeveloperWorks, donde se describen temas de interés en la siguiente dirección Web: http://www.ibm.com/developerworks. Busque **ObjectGrid**.

Entornos de los ejemplos

Algunos ejemplos pueden ejecutarse en un entorno J2SE, aunque otros deben ejecutarse en un entorno J2EE. Mientras que algunos pueden ejecutarse en una única instancia de servidor, los demás deben ejecutarse en un clúster. La siguiente tabla muestra el entorno de ejecución de los ejemplos.

Restricción: Si está utilizando ObjectGrid en un entorno de WebSphere Extended Deployment Versión 6.0, puede utilizar ObjectGrid en J2SE (Java 2 Platform, Standard Edition) Versión 1.4.2 o entornos superiores o WebSphere Application Server Versión 6.0.2 o entornos superiores con modificaciones adicionales de los acuerdos de licencia. Póngase en contacto con el representante de ventas para obtener información detallada.

Tabla 4. Entornos de ejecución de los ejemplos

		ObjectGrid SamplesSA	ObjectGrid Sample	ObjectGrid Partition Cluster	ObjectGrid JMSSamples
J2SE	Eclipse	x			
	línea de mandatos	x			
WebSphere Application Server Versión 6.0.*x*	único servidor		x		x
	clúster		x		x
	Entorno de prueba de unidades (UTE) de Rational Application Developer		x		
WebSphere Application Server Versión 5.0.2.*x* y Versión 5.1.*x*	único servidor		x		
	clúster		x		
WebSphere Extended Deployment Versión 6.0.*x*	único servidor		x		x
	clúster		x	x	x

Capítulo 6. Paquete de ObjectGrid

Puede acceder a los paquetes de ObjectGrid de dos formas: instalando WebSphere Extended Deployment, o instalando el entorno de distintos servidores.

Paquete ObjectGrid de WebSphere Extended Deployment Versión 6.0.1

Cuando instala WebSphere Extended Deployment Versión 6.0.1 o posterior, se instalan los siguientes archivos de tiempo de ejecución:

Tabla 5. Archivos de tiempo de ejecución de ObjectGrid de WebSphere Extended Deployment

Nombre de archivo	Entorno de tiempo de ejecución	Descripción
/lib/asm.jar /lib/cglib.jar	Local, cliente y servidor	Estos archivos jar son para la función del programa de utilidad cglib cuando se utilice la copia en modalidad de copia al escribir.
/lib/wsobjectgrid.jar	Local, cliente y servidor	Este archivo Java (JAR) contiene el tiempo de ejecución local, cliente y servidor de ObjectGrid para utilizarlo en el entorno WebSphere Extended Deployment Versión 6.0.1 y posteriores.

Paquete ObjectGrid de WebSphere Extended Deployment para un entorno de distintos servidores Versión 6.0.1

Cuando instala WebSphere Extended Deployment para un entorno de distintos servidores, se instalan los siguientes archivos de tiempo de ejecución:

Tabla 6. Archivos de tiempo de ejecución de ObjectGrid de WebSphere Extended Deployment para un entorno de distintos servidores

Nombre de archivo	Entorno de tiempo de ejecución	Descripción
/ObjectGrid/lib/asm.jar /ObjectGrid/lib/cglib.jar	Local, cliente y servidor	Estos archivos JAR son para la función del programa de utilidad cglib cuando se utilice la copia en modalidad de copia al escribir. Incluya estos archivos JAR en la CLASSPATH si está utilizando la copia en modalidad de copia al escribir y desea utilizar la función de proxy de cglib. Estos archivos JAR se incluyen automáticamente en el tiempo de ejecución del servidor. Añada estos archivos al tiempo de ejecución local o cliente de ObjectGrid.
/ObjectGrid/lib/mx4j.jar /ObjectGrid/lib/mx4j-remote.jar /ObjectGrid/lib/mx4j-tools.jar	Cliente y servidor de pasarela de gestión	Estos archivos JAR son para la función del programa de utilidad mx4j que usan el servidor de pasarela de gestión de ObjectGrid y los programas cliente de pasarela de gestión. Añada estos archivos JAR a la CLASSPATH del cliente de pasarela de gestión cuando se esté conectando al servidor de pasarela de gestión.
/ObjectGrid/lib/objectgrid.jar	Local, cliente y servidor	Este archivo JAR lo utiliza el tiempo de ejecución del servidor autónomo para J2SE (Java 2 Platform, Standard Edition) Versión 1.4.2 y posteriores. Asimismo puede utilizar este archivo JAR para el tiempo de ejecución cliente y local para la versión 1.3 de J2SE y posteriores.

Tabla 6. Archivos de tiempo de ejecución de ObjectGrid de WebSphere Extended Deployment para un entorno de distintos servidores (continuación)

Nombre de archivo	Entorno de tiempo de ejecución	Descripción
/ObjectGrid/lib/ogclient.jar	Local y cliente	Este archivo JAR contiene sólo los tiempos de ejecución local y cliente de ObjectGrid cuando se ejecuta fuera de un proceso WebSphere. Es recomendable utilizar este archivo JAR en lugar de objectgrid.jar, por ocupar menos espacio. Puede utilizar este archivo JAR con la versión 1.3 de J2SE y posteriores.
/ObjectGrid/lib/wsobjectgrid.jar	Local, cliente y servidor	Utilice este archivo JAR en WebSphere Application Server Versión 6.0.2 o posterior. Este archivo JAR es el mismo archivo JAR que se instala con WebSphere Extended Deployment.
/ObjectGrid/lib/wsogclient.jar	Local y cliente	Utilice este archivo JAR para WebSphere Application Server Versión 5.0.2 o posterior. Este archivo JAR contiene sólo los tiempos de ejecución local y cliente de ObjectGrid.

Consideraciones para la utilización de ObjectGrid con J2SE Versión 1.3

Cuando se utiliza el archivo ogclient.jar o objectgrid.jar en un entorno J2SE Versión 1.3.x, debe añadir los siguientes requisitos al entorno J2SE 1.3.x para que sea operativo con ObjectGrid:

- **Implementación JAAS (Java Authentication and Authorization Service)**. J2SE Versión 1.3 no incluía el objeto javax.security.Subject, que forma parte de la especificación JAAS. Las interfaces ObjectGrid y Session necesitan este objeto. Coloque la implementación JAAS en el directorio de extensiones Java jre/lib/ext.
- **Implementación JAXP (Java API for XML Processing)**. Si está pasando archivos XML al tiempo de ejecución de ObjectGrid, es necesaria una implementación JAXP para que el tiempo de ejecución de ObjectGrid analice el archivo XML. ObjectGrid utiliza la validación de sintaxis de definición de esquema XML de forma que se necesita una implementación que de soporte a una validación de esquema. El producto Apache Xerces es un ejemplo de una implementación que da soporte a la validación de esquemas.
- **Implementación JSSE (Java Secure Socket Extension)**. Cuando utiliza el tiempo de ejecución de cliente, se necesita una implementación JSSE. Verifique

que la implementación JSSE utilizada es compatible con la implementación JDK (Java Development Kit) del servidor ObjectGrid si está ejecutándola con la seguridad habilitada.

Si se incluye el tiempo de ejecución local o cliente de ObjectGrid en un entorno compatible con J2EE Versión 1.3, todos estos requisitos ya se cumplen porque todas las implementaciones de las especificaciones necesarias ya se necesitan para J2EE Versión 1.3.

Capítulo 7. Visión general de la gestión de sistemas

Con el release de WebSphere Extended Deployment Versión 6.0.1, ObjectGrid proporciona una infraestructura de gestión de sistemas para que los usuarios puedan supervisar y administrar entornos de ObjectGrid. La arquitectura de gestión de sistemas tiene tres niveles: un cliente usuario se conecta al servidor de pasarela de gestión, que realiza una conexión de cliente ObjectGrid con un clúster ObjectGrid.

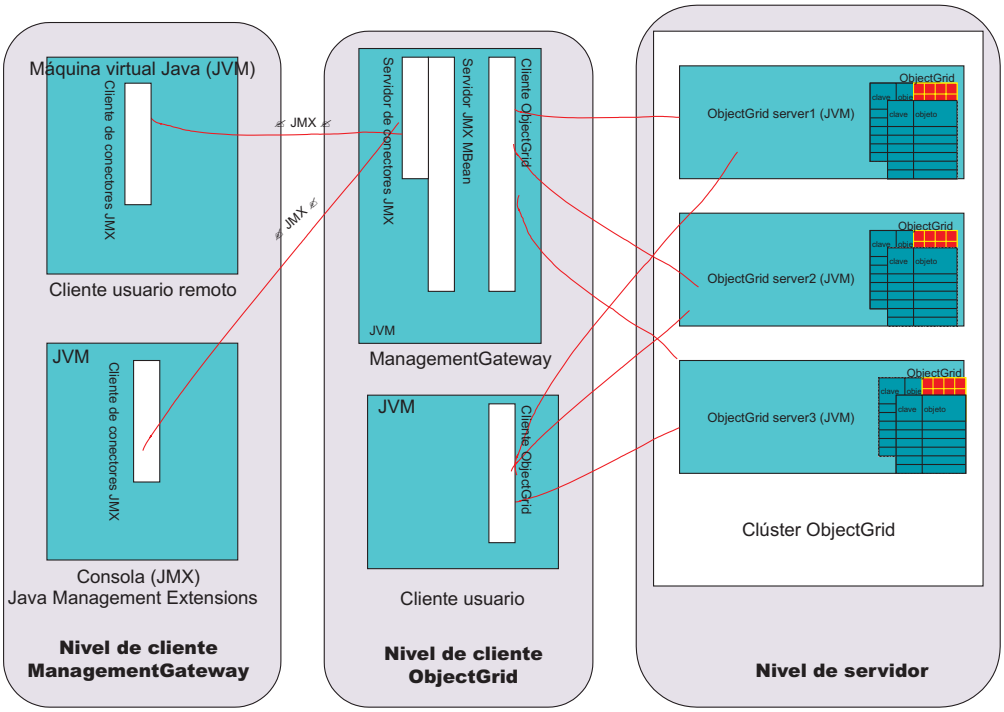


Figura 14. Diagrama de la gestión de sistemas

El *nivel de cliente de pasarela de gestión* contiene los programas que utilizan JMS (Java Management Extensions) para conectarse al servidor de pasarela de gestión. Se incluyen las consolas JMX de terceros así como un programa de cliente que utiliza API MX4J. El *nivel de cliente ObjectGrid* consta del servidor de pasarela de gestión. La pasarela de gestión actúa como servidor para el nivel de cliente de pasarela de gestión y como cliente en un clúster ObjectGrid operativo en el *nivel de servidor*. Asimismo, un programa de cliente ObjectGrid puede invocar las mismas API que invoca el servidor de pasarela de gestión si el usuario no desea implicar JMX. Por último, el nivel de servidor está formado por un clúster ObjectGrid.

La pasarela de gestión aloja un conjunto de beans gestionados (MBeans) y utiliza JMX para administrar y supervisar el entorno de ObjectGrid; está implementada por el proyecto de código fuente abierto MX4J. MX4J se suministra con ObjectGrid.

El modelo de administración de MBean y JMX de ObjectGrid se creó para aprovechar las distintas consolas JMX que hay disponibles para administrar entornos JMX. Puede combinar varios paneles de instrumentos utilizando la consola JMX que prefiera. Las consolas se pueden conectar a los MBeans que se ejecutan en la máquina virtual Java (JVM) de ManagementGateway y los paneles

de instrumentos se pueden ensamblar utilizando estos MBeans. Las consolas ofrecen historiales gráficos o diagramas de valores numéricos y de serie.

Hay dos opciones para ejecutar mandatos de gestión del sistema.

- Invocar los mandatos mediante la infraestructura de cliente-servidor que se aplica actualmente utilizando la interfaz ObjectGridAdministrator.
- Utilizar JMX para invocar los mismos mandatos, con los MBeans de ObjectGrid actuando como una envoltura para ObjectGridAdministrator.

Iniciar el proceso ManagementGateway

Después de iniciar un clúster (o único servidor), puede iniciarse el proceso ManagementGateway. ManagementGateway funciona como servidor para las solicitudes de cliente usuario y como cliente ObjectGrid para el clúster al que está conectado.

Opciones

A continuación figura una lista de opciones que puede pasarse al proceso ManagementGateway:

- **connectorPort** (obligatorio) - Especifica el número de puerto del conector JMX.
- **clusterHost** (obligatorio) - Especifica el nombre de sistema principal de uno de los servidores del clúster ObjectGrid.
- **clusterPort** (obligatorio) - Especifica el puerto de acceso de clientes de uno de los servidores del clúster ObjectGrid.
- **clusterName** (obligatorio) - Especifica el nombre del clúster ObjectGrid.
- **traceEnabled** - Especifica si se ha habilitado el rastreo para el proceso ManagementGateway.
- **traceSpec** - Indica la especificación de rastreo de ManagementGateway.
- **traceFile** - Especifica el archivo en el que se escribe la salida del rastreo.
- **sslEnabled** - Especifica si SSL está habilitado en ManagementGateway.
- **csConfig** - Especifica el objeto ClientSecurityConfiguration para ManagementGateway segura.
- **refreshInterval** - Especifica el intervalo de tiempo tras el que la pasarela de gestión renueva los atributos de MBean.

Interfaz de ManagementGateway

El proceso de ManagementGateway debe iniciarse para hacer disponibles los MBeans. La interfaz de ManagementGateway muestra las opciones que pueden pasarse cuando se inicia ManagementGateway.

```
public interface ManagementGateway {
/**
* Iniciar el servidor de conectores JMX MBean
*/
void startConnector();
/**
* Detener el servidor de conectores JMX MBean
*/
void stopConnector();
/**
* @param puerto del conector JMX
*/
void setConnectorPort(int port);
/**
```

```
* @return puerto del conector JMX
*/
int getConnectorPort();
/**
* @param un objeto {@link com.ibm.websphere.objectgrid.security.config.
ClientSecurityConfiguration}.
*/
void setCsConfig(ClientSecurityConfiguration csConfig);
/**
* @return un objeto {@link com.ibm.websphere.objectgrid.security.config.
ClientSecurityConfiguration}.
*/
ClientSecurityConfiguration getCsConfig();
/**
* @param puerto del servidor al que se conecta el cliente de pasarela
*/
void setPort(String port);
/**
* @return puerto del servidor al que se conecta el cliente de pasarela
*/
String getPort();
/**
* @param sistema principal del servidor al que se conecta el cliente de pasarela
*/
void setHost(String host);
/**
* @return sistema principal del servidor al que se conecta el cliente de pasarela
*/
String getHost();
/**
* @param valor booleano true si SSL está habilitado en la pasarela
*/
void setSSLEnabled(boolean sslEnabled);
/**
* @return valor booleano true si SSL está habilitado en la pasarela
*/
boolean getSSLEnabled();
/**
* @param clúster al que se conecta el cliente de pasarela
*/
void setClusterName(String clusterName);
/**
* @return clúster al que se conecta el cliente de pasarela
*/
String getClusterName();
/**
* @param valor booleano true si el rastreo está habilitado en la pasarela
*/
void setTraceEnabled(boolean traceEnabled);
/**
* @return valor booleano true si el rastreo está habilitado en la pasarela
*/
boolean getTraceEnabled();
/**
* @param especificación de rastreo en la pasarela
*/
void setTraceSpec(String traceSpec);
/**
* @return especificación de rastreo en la pasarela
*/
String getTraceSpec();
/**
* @param archivo de salida de rastreo para el rastreo de pasarela
*/
void setTraceFile(String traceFile);
/**
* @return archivo de salida de rastreo para el rastreo de pasarela
```

```
*/
String getTraceFile();
/**
* @param intervalo (en segundos) para renovar los atributos de MBean del clúster
*/
void setRefreshInterval(int refreshInterval);
/**
* @return intervalo (en segundos) para renovar los atributos de MBean del clúster
*/
int getRefreshInterval();
}
```

Opciones para iniciar el proceso ManagementGateway

Utilización mediante programación de ManagementGatewayFactory

A continuación se ofrece un ejemplo de código para el uso de esta opción:

```
ManagementGateway gw = ManagementGatewayFactory.getManagementGateway();
gw.setConnectorPort(1099);
gw.setClusterName("cluster1");
gw.setHost("localhost");
gw.setPort("12503");
gw.startConnector();
```

Este código debe estar en un programa de usuario que se ejecuta después de que el clúster ObjectGrid que está intentando conectar se haya iniciado.

En la línea de mandatos con el archivo de proceso por lotes startManagementGateway

A continuación se muestra un ejemplo:

```
startManagementGateway.bat -connectorPort 1099 -clusterName cluster1
-clusterHost localhost -clusterPort 12503
```

Para obtener más información sobre los scripts startManagementGateway, consulte “Iniciar el servidor de pasarela de gestión” en la página 88.

ManagementGateway funciona como servidor para un proceso de cliente que desea realizar llamadas JMX, pero también como un cliente ObjectGrid para el clúster al que el usuario desea conectarse. Una vez que se inicia ManagementGateway, se establece una conexión con el clúster y el servicio conector JMX pasa a estar disponible. Puede acceder al servicio conector JMX mediante las API de MX4J o J2SE (Java 2 Platform, Standard Edition) Versión 5.

Ejemplo

A continuación se ofrece un código de ejemplo sobre cómo obtener un MapStatsModule de un servidor denominado Server1 mediante una ManagementGateway con el puerto conector 1.

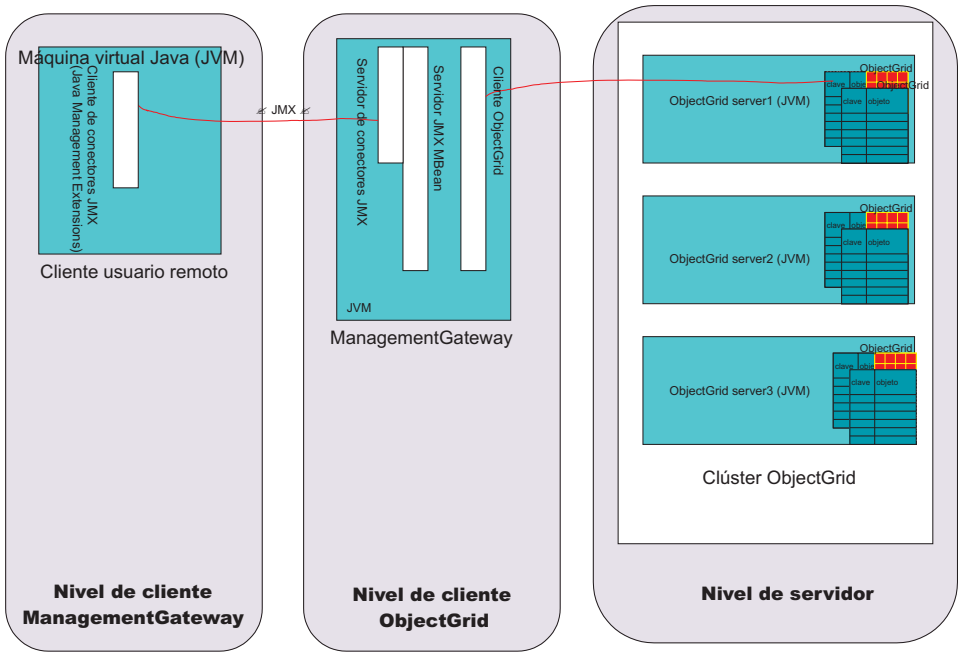


Figura 15. Obtener estadísticas de correlación del servidor server1

Ejecute el siguiente código en un programa de usuario que se ejecute en la sección cliente del usuario remoto del diagrama anterior:

```
JMXServiceURL url = new
JMXServiceURL("service:jmx:rmi://host/jndi/rmi://localhost:1099/jmxconnector");
JMXConnector c = JMXConnectorFactory.connect(url);
MBeanServerConnection mbsc = c.getMBeanServerConnection();
Iterator it = mbsc
.queryMBeans(new ObjectName
("ManagementMap:type=ObjectGrid,OG=OG1,Map=map1,S=server1"), null)
.iterator();
ObjectInstance oi = (ObjectInstance) it.next();
ObjectName mapMBean = oi.getObjectName();
MapStatsModule stats = (MapStatsModule) mbsc.invoke(
mapMBean,
"retrieveStatsModule",
new Object[] { },
new String[] { });
```

Para detener el servidor server1 mediante ManagementGateway:

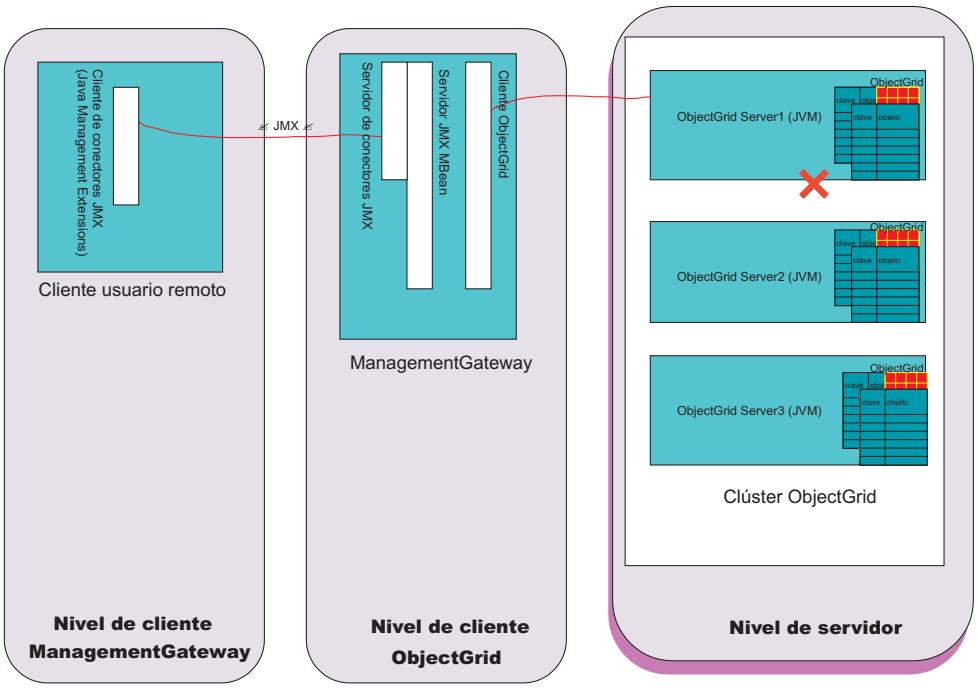


Figura 16. Detener el servidor server1

Ejecute el siguiente código en un programa de usuario que se ejecute en el cliente del usuario remoto del diagrama anterior:

```
JMXServiceURL url = new JMXServiceURL(
"service:jmx:rmi://host/jndi/rmi://localhost:1099/jmxconnector");
JMXConnector c = JMXConnectorFactory.connect(url);
MBeanServerConnection mbsc = c.getMBeanServerConnection();
Iterator it = mbsc
.queryMBeans(new ObjectName("ManagementServer:type=ObjectGrid,S=Server1"), null)
.iterator();
ObjectInstance oi = (ObjectInstance) it.next();
ObjectName server1MBean = oi.getObjectName();
boolean stop = ((Boolean) mbsc.invoke(
server1MBean,
"stopServer",
new Object[] { },
new String[] { })).booleanValue();
```

Después de ejecutar el ejemplo de código anterior, el servidor server1 se detiene. Una vez que el servidor server1 se detiene, no puede reiniciarse con ManagementGateway. El servidor puede reiniciarse mediante la línea de mandatos. Para obtener más información, consulte “Detener los servidores ObjectGrid” en la página 87.

Beans gestionados por ObjectGrid (MBeans)

Existen cinco tipos de MBeans en el entorno de ObjectGrid. Cada MBean hace referencia a una entidad específica, como la correlación, la infraestructura de objetos, el servidor, el grupo de duplicación, o el miembro del grupo de duplicación, y tiene atributos y operaciones.

Todos los MBean de ObjectGrid tienen métodos getxxx que representan valores de atributos. Estos métodos getxxx no pueden llamarse directamente desde un programa de usuario. Esto es debido a que la especificación JMX (Java

Management Extensions) trata los atributos de forma distinta que las operaciones. Los atributos pueden verse a través de cualquier consola JMX de terceros, y las operaciones pueden realizarse a través de un programa de usuario o de una consola JMX de terceros.

Mbean MapMbean

MapMBean permite al usuario supervisar las estadísticas de cada correlación definida para el clúster. Cada correlación tiene asociadas las estadísticas siguientes:

- Tiempo de actualización de proceso por lotes (mín/máx/medio/total)
- Número
- Proporción de coincidencias

Asimismo, como las correlaciones pueden particionarse entre los servidores, puede establecer el ámbito de las estadísticas de correlaciones en un servidor determinado o un miembro de un grupo de duplicación. Asimismo puede correlacionar las estadísticas para todo el clúster. El ObjectName de un MapMBean puede especificarse de diversas formas:

- `"ManagementMap:type=ObjectGrid,OG=ObjectGridName,Map=MapName"`
- `"ManagementMap:type=ObjectGrid,OG=ObjectGridName,Map=MapName, S=ServerName"`
- `"ManagementMap:type=ObjectGrid,OG=ObjectGridName,Map=MapName, RG=ReplicationGroup,IDX=Index"`

Tome una configuración de ejemplo con el componente ObjectGrid OG1, una correlación Map1, con dos servidores en el grupo de duplicación RG1, server1 y server2. Asimismo suponga que el servidor server1 es el principal y el servidor server2 es una duplicación. Para obtener las estadísticas de la correlación Map1 en el principal, utilice cualquiera de estos ObjectNames:

- `"ManagementMap:type=ObjectGrid,OG=OG1,Map=Map1,S=server1"`
- `"ManagementMap:type=ObjectGrid,OG=OG1,Map=Map1,RG=RG1,IDX=0"`

En cualquier ObjectName para MBeans de ObjectGrid, cuando IDX=0, hace referencia al principal del grupo de duplicación. IDX=1-10 hace referencia a las duplicaciones del grupo de duplicación.

A continuación se ofrece una lista de la interfaz MapMBean:

```
public interface MapMBean {
/**
* Operación para hacer que MapStatsModule se asocie con el MBean.
*
* @return MapStatsModule
*/
MapStatsModule retrieveStatsModule();
/**
* La operación sólo irá al servidor a obtener StatsModule si
* StatsModule no está almacenado en antememoria en el ObjectGridAdministrator.
*
*/
void refreshStatsModule();
/**
* Correlación.
*
* @return nombre de correlación
*/
String getMapName();
/**
* ObjectGrid que contiene la correlación.
```

```
*
* @return nombre de la infraestructura de objetos
*/
String getObjectGridName();
/**
* Nombre de servidor del miembro de grupo de duplicación para la correlación.
*
* @return nombre del servidor del miembro de grupo de duplicación
*/
String getServerName();
/**
* Nombre del grupo de duplicación para la correlación.
*
* @return nombre del grupo de duplicación
*/
String getReplicationGroup();
/**
* Índice del miembro de grupo de duplicación para la correlación.
*
* @return índice de miembros de grupo de duplicación
*/
int getIndex();
/**
* Atributo MapStatsModule cargado por
* la llamada retrieveStatsModule.
*
* @return MapStatsModule con formato de serie
*/
String getMapStatsModule();
/**
* Atributo de recuento de correlaciones cargado por
* la llamada retrieveStatsModule.
*
* @return número de entradas en la correlación
*/
long getMapCountStatistic();
/**
* Atributo de proporción de coincidencias cargado por
* la llamada retrieveStatsModule.
*
* @return proporción de coincidencias de la correlación
*/
double getMapHitRateStatistic();
/**
* Atributo de tiempo medio de actualización de proceso por lotes cargado por
* la llamada retrieveStatsModule.
*
* @return tiempo medio de actualización de proceso por lotes cargado para la actualización
*/
double getMapBatchUpdateMeanTime();
/**
* Atributo de tiempo máximo de actualización de proceso por lotes cargado por
* la llamada retrieveStatsModule.
*
* @return tiempo máximo de actualización de proceso por lotes cargado para la actualización
*/
double getMapBatchUpdateMaxTime();
/**
* Atributo de tiempo mínimo de actualización de proceso por lotes cargado por
* la llamada retrieveStatsModule.
*
* @return tiempo mínimo de actualización de proceso por lotes cargado para la actualización
*/
double getMapBatchUpdateMinTime();
/**
* Atributo de tiempo total de actualización de proceso por lotes cargado por
* la llamada retrieveStatsModule.
```

```
*
* @return tiempo total de actualización de proceso por lotes cargado para la actualización
*/
double getMapBatchUpdateTotalTime();
}
```

MBean de ObjectGridMBean

El MBean de ObjectGridMBean permite al usuario supervisar las estadísticas de todas las actualizaciones de cada ObjectGrid que se define para el clúster. Cada ObjectGrid tiene asociadas las estadísticas siguientes:

- Tiempo de transacción (mín/máx/medio/total)
- Número

Asimismo, como ObjectGrids puede particionarse entre los servidores, puede establecer el ámbito de las estadísticas de ObjectGrid en un servidor determinado o un miembro de un grupo de duplicación. Asimismo puede obtener las estadísticas de ObjectGrid para todo el clúster. El ObjectName de un ObjectGridMBean puede especificarse de diversas formas:

- `"ManagementObjectGrid:type=ObjectGrid,OG=ObjectGridName"`
- `"ManagementObjectGrid:type=ObjectGrid,OG=ObjectGridName, S=ServerName"`
- `"ManagementObjectGrid:type=ObjectGrid,OG=ObjectGridName, RG=ReplicationGroup,IDX=Index"`

A continuación se ofrece una lista de la interfaz ObjectGridMbean:

```
public interface ObjectGridMBean {
/**
* Operación para hacer que OGStatsModule se asocie con el MBean.
*
* @return OGStatsModule
*/
OGStatsModule retrieveStatsModule();
/**
* La operación sólo irá al servidor a obtener StatsModule si
* StatsModule no está almacenado en antememoria en el ObjectGridAdministrator.
*
*/
void refreshStatsModule();
/**
* ObjectGrid.
*
* @return nombre de la infraestructura de objetos
*/
String getObjectGridName();
/**
* Nombre de servidor del miembro de grupo de duplicación para la ObjectGrid.
*
* @return nombre del servidor del miembro de grupo de duplicación
*/
String getServerName();
/**
* Nombre del grupo de duplicación para la ObjectGrid.
*
* @return nombre del grupo de duplicación
*/
String getReplicationGroup();
/**
* Índice del miembro de grupo de duplicación para la ObjectGrid.
*
* @return índice de miembros de grupo de duplicación
*/
```

```
int getIndex();
/**
* Atributo OGStatsModule cargado por la llamada retrieveStatsModule.
*
* @return OGStatsModule con formato de serie
*/
String getOGStatsModule();
/**
* Atributo de recuento de ObjectGrid cargado por la llamada retrieveStatsModule.
*
* @return número de transacciones
*/
long getOGCount();
/**
* Atributo de tiempo máximo de transacción cargado por la llamada retrieveStatsModule.
*
* @return tiempo máximo de transacción para la ObjectGrid
*/
long getOGMaxTranTime();
/**
* Atributo de tiempo mínimo de transacción cargado por la llamada retrieveStatsModule.
*
* @return tiempo mínimo de transacción para la ObjectGrid
*/
long getOGMinTranTime();
/**
* Atributo de tiempo medio de transacción cargado por la llamada retrieveStatsModule.
*
* @return tiempo medio de transacción para la ObjectGrid
*/
double getOGMeanTranTime();
/**
* Atributo de tiempo total de transacción cargado por la llamada retrieveStatsModule.
*
* @return tiempo total de transacción para la ObjectGrid
*/
long getOGTotalTranTime();
}
```

MBean de ServerMBean

El Mbean de ServerMBean permite al usuario realizar operaciones en servidores del clúster. El ObjectName de un ServerMBean puede especificarse de la forma siguiente:

- ″ManagementServer:type=ObjectGrid,S=ServerName″

A continuación se ofrece una lista de la interfaz ServerMBean:

```
public interface ServerMBean {
/**
* Operación para cargar el estado de la duplicación para el servidor.
*
*/
void retrieveReplicationStatus();
/**
* Devolver el nombre del servidor.
*
* @return nombre del servidor
*/
String getServerName();
/**
* Operación para obtener el estado del servidor.
*
* @return estado del servidor (true si está ejecutándose, false si no lo está)
*/
boolean retrieveServerStatus();
```

```
/**
* Operación para detener el servidor.
*
* @return true si se ha detenido el servidor, false si no
*/
boolean stopServer();
/**
* Operación para forzar la detención del servidor.
*
* @return true si se ha detenido el servidor, false si no
*/
boolean forceStopServer();
/**
* Operación para detener el clúster del que forma parte el servidor.
*
* @param determina si los servidores se detienen a la fuerza
* @return true si se ha detenido el clúster, false si no
*/
boolean stopCluster(Boolean force);
/**
* Operación para modificar la especificación de rastreo para todos los servidores
* del clúster del que forma parte el servidor.
*
* @param especificación de rastreo
*/
void modifyClusterTraceSpec(String spec);
/**
* Operación para modificar la especificación de rastreo para el servidor.
*
* @param especificación de rastreo
*/
void modifyServerTraceSpec(String spec);
}
```

MBean de ReplicationGroupMBean

El ReplicationGroupMBean le permite supervisar el estado de todos los miembros de grupo de duplicación asociados con un grupo de duplicación específico incluido cuál es el servidor principal y un máximo de diez duplicaciones. El ObjectName para un ReplicationGroupMBean puede especificarse:

- ″ManagementReplicationGroup:type=ObjectGrid,RG=ReplicationGridName″

A continuación se ofrece una lista de la interfaz ReplicationGroupMBean:

```
public interface ReplicationGroupMBean {
/**
* Operación para cargar el estado de los atributos de grupo de duplicación.
*
*/
String[] retrieveReplicationGroupStatus();
/**
* Atributo ReplicationGroupName.
*
* @return nombre de ReplicationGroup
*/
String getReplicationGroupName();
/**
* Atributo Primary.
*
* @return nombre de Primary
*/
String getPrimary();
/**
* Atributo Replica1.
*
```

```
* @return nombre de servidor de Replica1
*/
String getReplica1();
/**
* Atributo Replica2.
*
* @return nombre de servidor de Replica2
*/
String getReplica2();
/**
* Atributo Replica3.
*
* @return nombre de servidor de Replica3
*/
String getReplica3();
/**
* Atributo Replica4.
*
* @return nombre de servidor de Replica4
*/
String getReplica4();
/**
* Atributo Replica5.
*
* @return nombre de servidor de Replica5
*/
String getReplica5();
/**
* Atributo Replica6.
*
* @return nombre de servidor de Replica6
*/
String getReplica6();
/**
* Atributo Replica7.
*
* @return nombre de servidor de Replica7
*/
String getReplica7();
/**
* Atributo Replica8.
*
* @return nombre de servidor de Replica8
*/
String getReplica8();
/**
* Atributo Replica9.
*
* @return nombre de servidor de Replica9
*/
String getReplica9();
/**
* Atributo Replica10.
*
* @return nombre de servidor de Replica10
*/
String getReplica10();
/**
* Todas las duplicaciones de este grupo de duplicación están delimitadas por comas
*
* @return nombres de servidor de todas las duplicaciones
*/
String getReplicas();
}
```

MBean de ReplicationGroupMemberMBean

ReplicationGroupMemberMBean le permite supervisar las siguientes estadísticas para un miembro de grupo de duplicación:

- Estado de un miembro de grupo de duplicación. Puede supervisar los miembros principales o las duplicaciones.
- Proporción del peso de las duplicaciones. Esta estadística sólo se aplica a los miembros de grupo de duplicación que son duplicaciones. Esta proporción es una cuantificación de lo cerca que están las correlaciones de una duplicación de sincronizarse con las correlaciones principales. Cuanto mayor es la proporción, más cerca se encuentra la duplicación de tener la información actualizada sobre la principal.

El ObjectName de un ReplicationGroupMemberMBean puede especificarse de las formas siguientes:

- `"ManagementReplicationGroupMember:type=ObjectGrid, RG=ReplicationGridName,S=ServerName"`
- `"ManagementReplicationGroupMember:type=ObjectGrid, RG=ReplicationGridName,IDX=Index"`

Al especificar IDX=0 se devuelve el principal del grupo de duplicación y de IDX=1 a 10 son las duplicaciones. A continuación se ofrece una lista de la interfaz ReplicationGroupMBean:

```
public interface ReplicationGroupMemberMBean {
/**
* Operación para cargar el estado de los atributos de miembros de grupo de duplicación.
*
*/
void retrieveReplicationGroupMemberStatus();
/**
* Operación para cargar el estado de los atributos de miembros de grupo de
* duplicación.
* Utilizará la antememoria en lugar del método retrieveReplicationGroupMemberStatus
* que irá al servidor a obtener el estado.
*
*/
void refreshReplicationGroupMemberStatus();
/**
* Atributo ReplicationGroupName.
*
* @return nombre de ReplicationGroup al que pertenece este miembro
*/
String getReplicationGroupName();
/**
* Estado de ReplicationGroupMember: principal/duplicación/reposo.
*
* @return estado de ReplicationGroupMember
*/
String getStatus();
/**
* Estadística que representa el porcentaje que indica la proximidad de una duplicación
* a estar actualizada con las correlaciones principales.
*
* @return estadística de duplicación de ReplicationGroupMember
*/
double getReplicaWeightRatio();
/**
* Nombre de servidor en el que reside este ReplicationGroupMember.
*
* @return nombre del servidor
*/
String getServerName();
```

```
/**
* Índice de ReplicationGroupMember.
*
* @return índice de duplicación
*/
int getIndex();
}
```

Capítulo 8. Soporte de línea de mandatos

Utilice scripts de línea de mandatos para gestionar los servidores ObjectGrid.

En el directorio `/ObjectGrid/bin` se proporciona un conjunto de archivos script de una instalación para un entorno de distintos servidores. Estos scripts pueden utilizarse para iniciar o detener un servidor ObjectGrid, para iniciar un servidor de pasarela de gestión y codificar contraseñas en un archivo de propiedades. Antes de intentar utilizar los scripts, verifique si está establecida la variable de entorno JAVA_HOME y que su valor sea una versión de Java soportada por ObjectGrid. Puede actualizar JAVA_HOME en el archivo `setupCmdLine.bat|sh` para apuntar a una versión adecuada de Java si no desea cambiar la variable de entorno de forma global.

Consulte los temas siguientes si desea más información sobre los scripts de línea de mandatos:

- "Iniciar los servidores ObjectGrid"
- "Detener los servidores ObjectGrid" en la página 87
- "Iniciar el servidor de pasarela de gestión" en la página 88
- "Codificación de la contraseña" en la página 90

Iniciar los servidores ObjectGrid

El script startOgServer se proporciona para iniciar un servidor ObjectGrid.

Uso

Utilice el archivo `startOgServer.bat` para iniciar un servidor en un equipo Windows. Utilice el archivo `startOgServer.sh` para iniciar un servidor ObjectGrid en las plataformas Linux y Unix.

Utilizar archivos XML

Un archivo ObjectGrid válido debe emparejarse con un archivo XML de clúster válido para iniciar satisfactoriamente un servidor ObjectGrid. Los archivos XML pueden pasarse al script startOgServer utilizando un nombre de archivo normal o un localizador universal de recursos (URL). La opción del URL permite distintos protocolos además del protocolo file, por ejemplo http, ftp o jarfile.

A continuación figuran los argumentos del script startOgServer para iniciar un servidor mediante los archivos XML:

```
startOgServer.bat
<servidor> -objectgridFile <archivo XML> | -objectgridUrl
<URL archivo XML> -clusterFile <archivo XML> | -clusterUrl <URL archivo XML>
[opciones]
```

Ejemplo

A continuación se ofrecen algunos ejemplos de inicio del servidor ObjectGrid server1. Estos ejemplos utilizan el archivo `startOgServer.bat`.

```
startOgServer.bat server1 -objectgridFile c:\objectgrid\xml\university.xml
-clusterFile c:\objectgrid\xml\universityCluster3Servers.xml
startOgServer.bat server1 -objectgridFile ..\xml\university.xml
-clusterUrl file:///c:/objectgrid/xml/universityCluster3Servers.xml
```

```
startOgServer.bat server1 -objectgridUrl file:///c:/objectgrid/xml/university.xml
-clusterFile ..\xml\universityCluster3Servers.xml
startOgServer.bat server1 -objectgridUrl file:///c:/objectgrid/xml/university.xml
-clusterUrl file:///c:/objectgrid/xml/universityCluster3Servers.xml
```

Los ejemplos utilizan el archivo `universityCluster3Servers.xml`. Como se ha especificado el servidor server1 como servidor de arranque, el archivo `universityCluster3Servers.xml` debe tener un valor serverDefinition con el nombre `server1`.

A continuación figura el archivo `universityCluster3Servers.xml`. Tenga en cuenta que el valor de serverDefinition es server1 y que su sistema principal es lion.ibm.com. El servidor server1 debe iniciarse en el sistema principal lion.ibm.com. Este archivo también define los servidores server2 y server3. Estos servidores deben iniciarse en los sistemas principales tiger.ibm.com y bear.ibm.com, respectivamente.

Archivo *universityCluster3Servers.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
<serverDefinition name="server3" host="bear.ibm.com" clientAccessPort="12505"
peerAccessPort="12506" />
</cluster>
<objectgridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
<replicationGroupMember serverRef="server2" priority="2" />
<replicationGroupMember serverRef="server3" priority="3" />
</replicationGroup>
</clusterConfig>
```

Rutina de carga

Una vez que un servidor ObjectGrid del clúster esté disponible, otros servidores del clúster pueden iniciar la rutina de carga en el servidor disponible. Debe proporcionarse el script startOgServer con el sistema principal y el puerto de acceso de clientes de un servidor que ya esté disponible para iniciar la rutina de carga en él.

Como el primer servidor de un clúster se inicia con XML, ya tiene la información de configuración para todos los servidores del clúster. La rutina de carga permite al servidor que está arrancando conectarse al servidor disponible y descargar la configuración.

A continuación se muestran los argumentos del script startOgServer para iniciar un servidor iniciando la rutina de carga en un servidor disponible.

```
startOgServer.bat
<servidor> -bootstrap <sistema principal:puerto,sistema principal:puerto,>
[opciones]
```

A continuación se ofrecen algunos ejemplos de inicio de un servidor mediante el inicio de rutina de carga en otro servidor. En el primer ejemplo, considere que el servidor server1 del archivo `universityCluster3Servers.xml` se ha iniciado utilizando archivos XML y está disponible. Este ejemplo muestra cómo iniciar una rutina de carga en el servidor server1 para iniciar el servidor server2.

```
startOgServer.bat server2 -bootstrap lion.ibm.com:12501
```

En el siguiente ejemplo, considere que el servidor server2 se ha iniciado satisfactoriamente, pero que el servidor server1 no está disponible. Puede utilizarse una lista separada por comas de combinaciones sistema_principal:puerto cuando se inicia la rutina de carga en otro servidor. Se realizan intentos de contacto con cada sistema principal y puerto de la lista hasta que se encuentra un servidor disponible. En el ejemplo siguiente, el servidor server3 intenta ponerse en contacto con el sistema principal y el puerto del servidor server1. Sin embargo, como el servidor server1 no está disponible en este caso de ejemplo, se produce un error en la conexión. Tomando el elemento siguiente de la lista, el servidor server3 intenta iniciar la rutina de carga en el sistema principal y el puerto del servidor server2. Este intento de rutina de carga ha sido satisfactorio porque el servidor server2 está disponible.

```
startOgServer.bat server3 -bootstrap lion.ibm.com:12501,tiger.ibm.com:12503
```

Argumentos opcionales

Existen varios argumentos opcionales que pueden pasarse al script startOgServer. A continuación se muestran los argumentos válidos de startOgServer.

Opciones:

- -traceSpec <especificación de rastreo>
- -traceFile <archivo de rastreo>
- -serverSecurityFile <archivo de propiedades de seguridad del servidor>
- -timeout <segundos>
- -script <nombre archivo script>
- -jvmArgs <argumentos JVM>

-traceSpec

El argumento -traceSpec puede utilizarse para establecer una especificación de rastreo que tiene un efecto casi inmediato durante el inicio del servidor. Durante el inicio normal del servidor, la especificación de rastreo no se establece hasta que puede leerse desde el archivo XML del clúster o desde la configuración de rutina de carga. Si se producen problemas durante el inicio del servidor, es recomendable establecer antes la especificación de rastreo.

A continuación figura un ejemplo de cómo establecer la opción -traceSpec:

```
startOgServer.bat server1 -objectgridFile c:\objectgrid\xml\university.xml
-clusterFile c:\objectgrid\xml\universityCluster3Servers.xml
-traceSpec ObjectGrid=all=enabled
```

-traceFile

El argumento -traceFile puede utilizarse para especificar una ubicación para el rastreo de salida durante el inicio del servidor. Cuando se ha leído la configuración de este servidor, entran en vigor los valores de rastreo especificados en el archivo XML del clúster.

A continuación figura un ejemplo de cómo establecer la opción -traceFile:

```
startOgServer.bat server2 -bootstrap lion.ibm.com:12501 -traceFile
c:\objectgrid\trace.log
```

-serverSecurityFile

El argumento -serverSecurityFile puede utilizarse para pasar a un servidor su archivo de propiedades de seguridad. Esta opción es necesaria cuando se habilita la seguridad en el servidor. A continuación figura un ejemplo de cómo establecer la opción -serverSecurityFile:

```
startOgServer.bat server1 -objectgridUrl file:///c:/objectgrid/xml/
university.xml
-clusterFile ..\xml\universityCluster3Servers.xml
-serverSecurityFile c:\objectgrid\props\serverSecurity.props
```

-timeout

El argumento -timeout puede utilizarse para especificar el tiempo, en segundos, que puede pasar antes de que el inicio del servidor termine anormalmente. Por omisión, el servidor dispone de 90 segundos para hacerse disponible a partir del momento de iniciarse. Si este tiempo es demasiado breve para un caso de ejemplo determinado, utilice el argumento -timeout para establecer un valor más adecuado. A continuación se ofrece un ejemplo del uso del argumento timeout:

```
startOgServer.bat server1 -objectgridFile ..\xml\university.xml
-clusterUrl file:///c:/objectgrid/xml/universityCluster3Servers.xml
-timeout 120
```

-script

El argumento -script puede utilizarse para crear un script que inicia un proceso del servidor ObjectGrid y mantiene su salida en el indicador de mandatos actual. En circunstancias normales, cuando se inicia un servidor ObjectGrid, el script startOgServer muestra la salida del proceso del servidor en el indicador de mandatos hasta que el servidor pasa a estar disponible. Una vez que el servidor está disponible, startOgServer deja de mostrar la salida del proceso del servidor y sale. En algunos casos, debe iniciar un proceso del servidor que muestra su salida en el indicador de mandatos actual.

Cuando especifique un nombre de archivo para el script, no asigne una vía de acceso para el archivo. El archivo se coloca en el directorio `bin` de la vía de acceso OBJECTGRID_HOME. Especifique el nombre del archivo. El archivo script que se crea incluye los argumentos que se han pasado al script startOgServer de forma que no es necesario facilitar estos mismos argumentos cuando se ejecuta el script creado.

A continuación figura un ejemplo de cómo utilizar la opción -script:

```
startOgServer.bat server1 -objectgridUrl
file:///c:/objectgrid/xml/university.xml
-clusterUrl file:///c:/objectgrid/xml/universityCluster3Servers.xml
-script universityClusterServer1.bat
```

En este ejemplo se crea un script `universityClusterServer1.bat` en el directorio `OBJECTGRID_HOME/bin`. Para ejecutar el script recién creado, vaya al directorio pertinente en el indicador de mandatos, escriba el nombre del script y pulse **Intro**.

-jvmArgs

El argumento -jvmArgs puede utilizarse para enviar argumentos a la máquina virtual Java (JVM) del servidor ObjectGrid que se está iniciando. Los argumentos que pueden pasarse a la JVM pueden pasarse normalmente al servidor utilizando el argumento -jvmArgs.

El argumento -jvmArgs debe ser el último argumento opcional de ObjectGrid especificado como argumento en el script startOgServer. Todo lo que va después del argumento -jvmArgs se pasa a la JVM del servidor como argumento de JVM. A continuación se ofrece un ejemplo de cómo establecer el argumento -jvmArgs:

```
startOgServer.bat server2 -bootstrap lion.ibm.com:12501
-jvmArgs -Xms768M -DmyProp=value1
```

Si el argumento -jvmArgs incluye un argumento de JVM -classpath o -cp, la classpath especificada se añade a la classpath de ObjectGrid. A continuación figura un ejemplo del uso del argumento -jvmArgs para incluir en la classpath los archivos JAR de Xerces que se utilizarán para un servidor ObjectGrid.

```
startOgServer.bat server2 -bootstrap lion.ibm.com:12501 -jvmArgs -cp
C:\xerces2_7_1\xml-apis.jar;c:\xerces2_7_1\xercesImpl.jar
```

Detener los servidores ObjectGrid

Utilice el script stopOgServer para detener los servidores ObjectGrid.

Uso

Utilice el archivo `stopOgServer.bat` para detener un servidor en un equipo Windows. Utilice el archivo `stopOgServer.sh` para detener un servidor ObjectGrid en las plataformas Linux y Unix. El script stopOgServer crea un cliente que puede detener un servidor conectándose a cualquier servidor disponible del clúster. El comportamiento de este script es similar al inicio de rutina de carga a un servidor disponible para iniciar otro servidor. A continuación figuran los argumentos del script stopOgServer para detener un servidor.

```
stopOgServer.bat <servidor> -bootstrap <sistema principal:puerto,
sistema principal:puerto,> [opciones]
```

Ejemplos

A continuación figuran algunos ejemplos de detención de distintos servidores ObjectGrid. Estos ejemplos utilizan el archivo `stopOgServer.bat`. Para estos ejemplos, se supone que tres servidores están activos y funcionando: los servidores server1, server2 y server3 según se definen en el archivo `universityCluster3Servers.xml` en “Iniciar los servidores ObjectGrid” en la página 83.

Este primer ejemplo detiene el servidor server1 iniciando la rutina de carga en su sistema principal y el puerto de acceso de clientes.

```
stopOgServer.bat server1 -bootstrap lion.ibm.com:12501
```

Presuponga que el servidor server1 se ha detenido satisfactoriamente. El ejemplo siguiente detiene el servidor server2 intentando primero el inicio de rutina de carga en el servidor server1. Como el servidor server1 ya se ha detenido, se produce un error en el inicio de rutina de carga. Los siguientes sistema principal y puerto de la lista pertenecen al servidor server3. Como el servidor server3 está disponible, el inicio de rutina de carga en el servidor server3 es satisfactorio y se detiene server2.

```
stopOgServer.bat
server2 -bootstrap lion.ibm.com:12501,bear.ibm.com:12505
```

Argumentos opcionales

Existen algunos argumentos opcionales que pueden pasarse al script stopOgServer. Este apartado muestra cómo utilizar cada uno de estos argumentos opcionales. A continuación se muestran los argumentos stopOgServer válidos, seguidos de argumentos opcionales.

```
stopOgServer.bat
<servidor> -bootstrap <sistema principal:puerto,sistema principal:puerto,>
[opciones]
```

Opciones:

- -traceSpec <especificación de rastreo>
- -traceFile <archivo de rastreo>
- -clientSecurityFile <archivo de propiedades de seguridad del cliente>

-traceSpec

El argumento -traceSpec puede utilizarse para establecer una especificación de rastreo en el cliente que intenta detener un servidor ObjectGrid. A continuación figura un ejemplo de cómo establecer el argumento -traceSpec:

```
stopOgServer.bat server1 -bootstrap lion.ibm.com:12501 -traceSpec
ObjectGrid=all=enabled
```

-traceFile

El argumento -traceFile puede utilizarse para especificar una ubicación para el rastreo de cliente de salida durante la conclusión del servidor. A continuación figura un ejemplo de cómo establecer el argumento traceFile:

```
stopOgServer.bat server2
-bootstrap lion.ibm.com:12501,bear.ibm.com:12505
-traceFile c:\objectgrid\trace.log
```

-clientSecurityFile

El argumento -clientSecurityFile puede utilizarse para pasar al cliente su archivo de propiedades de seguridad. Este argumento es obligatorio cuando intente conectarse a un servidor con la seguridad habilitada.

A continuación figura un ejemplo de cómo establecer el argumento -clientSecurityFile:

```
stopOgServer.bat server1
-bootstrap lion.ibm.com:12501 -c lientSecurityFile
 c:\objectgrid\props\clientSecurity.props
```

Iniciar el servidor de pasarela de gestión

Para supervisar y administrar un clúster ObjectGrid mediante JMX (Java Management Extensions), la pasarela de gestión debe haberse iniciado mediante el script de línea de mandatos o mediante un programa.

Finalidad

Para iniciar la pasarela de gestión mediante la línea de mandatos, utilice el script startManagementGateway. Utilice el archivo `startManagementGateway.bat` para iniciar un servidor ManagementGateway en un equipo Windows. Utilice el archivo `startManagementGateway.sh` para iniciar un servidor ManagementGateway en las plataformas Linux y Unix. Si desea más información sobre el funcionamiento de la pasarela de gestión, MBeans de ObjectGrid y JMX, consulte Capítulo 7, “Visión general de la gestión de sistemas”, en la página 69.

El script startManagementGateway crea un servidor de conectores JMX y un cliente ObjectGrid que se conecta con un clúster ObjectGrid para detener los servidores, reunir el estado y las estadísticas, y realizar varias funciones más.

A continuación se listan los argumentos del script startManagementGateway para iniciar un servidor de pasarela de gestión.

```
startManagementGateway.bat -connectorPort <puerto> -clusterHost <sistema principal>
-clusterPort <puerto> -clusterName <clúster> [opciones]
```

Argumentos opcionales

Algunos argumentos opcionales pueden pasarse al script startManagementGateway. Los argumentos startManagementGateway válidos van seguidos de argumentos opcionales.

```
startManagementGateway.bat -connectorPort <puerto> -clusterHost <sistema principal>
-clusterPort <puerto> -clusterName <clúster> [opciones]
```

Opciones

- -traceEnabled <rastreo true/false habilitado>
- -traceSpec <especificación de rastreo>
- -traceFile <archivo de rastreo>
- -refreshInterval <intervalo de renovación del atributo MBean>
- -sslEnabled <SSL true/false habilitado para pasarela de gestión>
- -clientSecurityFile <vía de acceso al archivo de seguridad del cliente>

-traceEnabled

El argumento -traceEnabled puede utilizarse para establecer si se activa el rastreo para el servidor de pasarela de gestión. El valor por omisión es false, por lo que la única forma de ver el rastreo de ObjectGrid es habilitarlo estableciendo el valor de -traceEnabled en ″true″ y proporcionando valores -traceSpec y -traceFile válidos.

-traceSpec

El argumento -traceSpec puede utilizarse para establecer una especificación de rastreo para el servidor de pasarela de gestión.

-traceFile

El argumento -traceFile puede utilizarse para especificar una ubicación para la salida de rastreo de la pasarela de gestión. A continuación figura un ejemplo de cómo establecer los argumentos traceEnabled, traceSpec y traceFile.

```
startManagementGateway.bat -connectorPort 1099 -clusterHost lion.ibm.com
-clusterPort 12501 -clusterName universityCluster -traceEnabled true
-traceSpec ObjectGrid=all=enabled -traceFile \\objectgrid\\trace.log
```

-refreshInterval

El argumento -refreshInterval puede utilizarse para pasar la cantidad de

tiempo (en segundos) que espera la pasarela de gestión entre las renovaciones de los valores del atributo MBean. El valor por omisión es 120 segundos. A continuación figura un ejemplo de cómo establecer el argumento refreshInterval:

```
startManagementGateway.bat
-connectorPort 1099 -clusterHost lion.ibm.com
-clusterPort 12501 -clusterName universityCluster -refreshInterval 60
```

-sslEnabled

El argumento -sslEnabled puede utilizarse para establecer si se habilita SSL para la pasarela de gestión. Si el valor de este argumento es `true`, los clientes usuarios que se conecten al servidor de pasarela de gestión deben proporcionar las propiedades SSL:

- -Djavax.net.ssl.trustStore
- -Djavax.net.ssl.trustStorePassword

El valor por omisión si no se proporciona el argumento -sslEnabled es ″false″.

-clientSecurityFile

El argumento -clientSecurityFile puede utilizarse para pasar el nombre de archivo que contiene las propiedades de seguridad del cliente para un acceso de cliente seguro entre el servidor de pasarela de gestión y el clúster ObjectGrid. Este argumento es obligatorio cuando intente conectarse a un clúster con la seguridad habilitada. ObjectGrid se entrega con la siguiente plantilla de archivo de propiedades de seguridad del cliente: `security.ogclient.props`.

A continuación figura un ejemplo de cómo establecer las propiedades sslEnabled y clientSecurityFile:

```
startManagementGateway.bat
-connectorPort 1099 -clusterHost lion.ibm.com
-clusterPort 12501 -clusterName universityCluster -sslEnabled true
-clientSecurityFile ..\\properties\\security.ogclient.props
```

Codificación de la contraseña

La codificación de la contraseña impide la observación ocasional de las contraseñas de los archivos de propiedades de seguridad de ObjectGrid.

Uso

ObjectGrid contiene varias contraseñas codificadas que no están cifradas. ObjectGrid proporciona el programa de utilidad FilePasswordEncoder, que puede utilizar para codificar estas contraseñas. Utilice el archivo `FilePasswordEncoder.bat` para codificar las contraseñas en un sistema Windows. Utilice el archivo `FilePasswordEncoder.sh` para codificar las contraseñas en las plataformas Linux y Unix.

La sintaxis del mandato es la siguiente:

```
FilePasswordEncoder.bat lista_propiedades_contraseña_nombre_archivo [tipo_archivo]
```

Opciones

Las opciones siguientes están disponibles para el mandato FilePasswordEncoder:

nombre_archivo

El nombre_archivo se utiliza para especificar el nombre de archivo que tiene contraseñas que codificar. Por ejemplo, `security.ogserver.props`.

lista_prop_contraseñas

La lista_prop_contraseña es una lista de nombres de propiedades de contraseña separados por comas, por ejemplo, ″trustStorePassword,keyStorePassword″.

tipo_archivo

Este argumento es opcional. El tipo_archivo puede ser un valor de propiedad o XML, que indica si el archivo proporcionado es un archivo de propiedades o un archivo XML. El valor por omisión es `propiedad`. Actualmente, ObjectGrid no almacena contraseñas en un archivo XML, de forma que esta opción no es obligatorio. Los ejemplos siguientes demuestran la sintaxis correcta:

- `FilePasswordEncoder.bat security.ogclient.props ″trustStorePassword,keyStorePassword″`
- `FilePasswordEncoder.bat security.ogserver.props ″trustStorePassword,keyStorePassword,secureTokenKeyStorePassword, secureTokenKeyPairPassword,secureTokenSecretKeyPassword″`

Este programa de utilidad FilePasswordEncoder no se entrega con WebSphere Extended Deployment. Puede utilizar el programa de utilidad PropFilePasswordEncoder proporcionado por WebSphere Application Server para codificar estas contraseñas. En la Consulta de mandatos de PropFilePasswordEncoder puede obtener más detalles.

Capítulo 9. Visión general de la interfaz de programación de la aplicación ObjectGrid

En este apartado se describe cómo se configura ObjectGrid con XML o con interfaces mediante programación. Asimismo, se incluye información para implementar las interfaces externas que proporciona ObjectGrid. En todos los casos, se ofrece una visión general y descripciones de las interfaces de la API y de los ejemplos.

Documentación de la API

El JavaDoc para ObjectGrid es la fuente de información definitiva sobre las API. Encontrará el JavaDoc en el siguiente directorio de la instalación de WebSphere Extended Deployment: *raíz_instalación*\web\xd\apidocs

Interfaz ObjectGridManager

La clase ObjectGridManagerFactory y la interfaz ObjectGridManager proporcionan un mecanismo para crear las instancias de ObjectGrid, acceder a ellas y colocarlas en antememoria. La clase ObjectGridManagerFactory es una clase de ayuda estática para acceder a la interfaz ObjectGridManager, un singleton. La interfaz ObjectGridManager incluye varios métodos cómodos para crear instancias de un objeto ObjectGrid. La interfaz ObjectGridManager también facilita la creación y colocación en antememoria de las instancias de ObjectGrid a las que pueden acceder varios usuarios.

Métodos createObjectGrid

Utilice este tema para obtener información sobre los siete métodos createObjectGrid que aparecen en la interfaz ObjectGridManager.

Métodos createObjectGrid

La interfaz ObjectGridManager tiene siete métodos createObjectGrid. A continuación aparece un caso de ejemplo sencillo:

Caso sencillo con la configuración por omisión

A continuación aparece un caso sencillo sobre cómo crear una ObjectGrid para que la compartan 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 continua..*/
```

El fragmento anterior de código Java crea o coloca en antememoria la ObjectGrid Employees. La ObjectGrid Employees se inicializa con la configuración por omisión y está lista para su uso. El segundo parámetro del método createObjectGrid se establece en `true`, que indica a ObjectGridManager que coloque en antememoria la instancia de ObjectGrid que cree. Si este parámetro se establece en `false`, la

instancia no se coloca en antememoria. Todas las instancias de ObjectGrid tienen un `nombre` y éstas pueden compartirse entre numerosos clientes y usuarios en función de ese nombre.

Si se utiliza la instancia de objectGrid en la compartición de igual a igual, la antememoria se debe establecer en `true`. Para obtener más información sobre la compartición de igual a igual, consulte Capítulo 12, “Distribución de cambios entre máquinas virtuales Java de igual”, en la página 341.

Configuración XML

ObjectGrid tiene una gran capacidad de configuración. El ejemplo anterior muestra cómo crear una ObjectGrid sencillo sin ninguna configuración. Con este ejemplo, puede crear una instancia de ObjectGrid configurada previamente que esté basada en un archivo de configuración XML. Puede configurar una instancia de ObjectGrid mediante programación o mediante un archivo de configuración basado en XML. Asimismo, puede configurar ObjectGrid utilizando una combinación de ambos procedimientos.

La interfaz ObjectGridManager permite la creación de una instancia de ObjectGrid basándose en la configuración XML. La interfaz ObjectGridManager tiene varios métodos que toman un URL como argumento. Todos los archivos XML que se pasan a ObjectGridManager deben validarse en ese esquema. La validación XML sólo puede inhabilitarse cuando se ha validado el archivo previamente y no se ha realizado ningún cambio en él desde su última validación. La inhabilitación de la validación supone un ahorro de la sobrecarga asociada pero introduce la posibilidad de que se utilice un archivo XML no válido. IBM Java Developer Kit (JDK) 1.4.2 da soporte a la validación XML. Al utilizar un JDK que no disponga de este soporte, es posible que sea necesario Apache Xerces para validar el XML.

El siguiente fragmento de código Java demuestra cómo transferir un archivo de configuración XML para crear una 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 la validación XML
boolean cacheInstance = true; // Colocar en antememoria la instancia
String objectGridName="Employees"; // Nombre del URL de ObjectGrid
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees = oGridManager.createObjectGrid(objectGridName,
allObjectGrids,
validateXML,
                        cacheInstance);
```

El archivo XML puede contener información de configuración para varias ObjectGrids. El fragmento de código anterior devuelve específicamente la ObjectGrid ″Employees″, presuponiendo que la configuración de ″Employees″ esté definida en el archivo. Para ver la sintaxis XML, consulte “Configuración de ObjectGrid” en la página 261.

Existen siete métodos createObjectGrid. Los métodos se documentan en el siguiente bloque de código.

```
/**
* Un método de fábrica sencillo para devolver una instancia de una
* ObjectGrid. Se asigna un nombre exclusivo.
* La instancia de ObjectGrid no se coloca en antememoria.
* Los usuarios pueden utilizan {@link ObjectGrid#setName(String)} para cambiar el
* nombre de ObjectGrid.
*
* @return ObjectGrid instancia de ObjectGrid con un nombre exclusivo asignado
* @throws ObjectGridException cualquier error durante la creación de ObjectGrid
* @ibm-api
*/
public ObjectGrid createObjectGrid() throws ObjectGridException;
/**
* Un método de fábrica sencillo para devolver una instancia de ObjectGrid con
* el nombre especificado. Las instancias de ObjectGrid pueden colocarse en
* antememoria. Si una ObjectGrid con este nombre ya se ha colocado en
* antememoria, se generará una excepción ObjectGridException
*
* @param objectGridName nombre de la ObjectGrid que se va a crear.
* @param cacheInstance true, si la instancia de ObjectGrid debe colocarse en
*  antememoria
* @return instancia de ObjectGrid
* @this nombre ya se colocado en antememoria o
* cualquier error durante la creación de ObjectGrid.
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, boolean cacheInstance)
throws ObjectGridException;
/**
* Crear una instancia de ObjectGrid con el nombre especificado de ObjectGrid. La
* instancia de ObjectGrid creada se colocará en antememoria.
* @param objectGridName Nombre de la instancia de ObjectGrid que se va a crear.
* @return instancia de ObjectGrid
* @throws ObjectGridException si una ObjectGrid con este nombre ya se ha
* colocado en antememoria o se ha producido un error
* durante la creación de ObjectGrid
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName)
throws ObjectGridException;
/**
* Crear una instancia de ObjectGrid basada en el nombre especificado de ObjectGrid
* y el archivo XML. Se creará y se devolverá la instancia de ObjectGrid definida
* en el archivo XML con el nombre de ObjectGrid especificado. Si esta ObjectGrid
* no se puede encontrar en el archivo xml, se generará una excepción.
*
* Esta instancia de ObjectGrid puede colocarse en antememoria.
*
* Si el URL es nulo, simplemente se ignorará. En este caso, este método se comporta
* como {@link #createObjectGrid(String, boolean)}.
*
* @param objectGridName Nombre de la instancia de ObjectGrid que se va a devolver.
* No debe ser nulo.
* @param xmlFile URL de un archivo XML bien formado basado en el esquema
*  de ObjectGrid.
* @param enableXmlValidation si se valida el XML
* @param cacheInstance un valor booleano que indica si las instancias de ObjectGrid
* definidas en el XML se colocarán en antememoria. Si es true, las instancias
* se colocarán en antememoria. *
* @throws ObjectGridException si una ObjectGrid con el mismo nombre
* se ha colocado en antememoria anteriormente, no se puede encontrar
* ningún nombre de ObjectGrid en el archivo xml,
* o cualquier otro error durante la creación de ObjectGrid.
* @return instancia de ObjectGrid
* @see ObjectGrid
* @ibm-api
*/
```

```
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 basándose
* en el archivo.
* Estas instancias de ObjectGrid se pueden colocar en antememoria.
* Se generará una excepción ObjectGridException cuando se intente colocar
* en antememoria una ObjectGrid recién creado que tenga el mismo nombre
* que una ObjectGrid que ya se haya colocado en antememoria.
*
* @param xmlFile archivo que define una ObjectGrid o varias
* ObjectGrids
* @param enableXmlValidation el establecimiento en true validará el
* archivo XML en el esquema
* @param cacheInstances establecer en true para colocar en antememoria todas
* las instancias de ObjectGrid creadas basándose en el archivo
* @return instancia de ObjectGrid
* @throws ObjectGridException al intentar crear y colocar en antememoria una
* ObjectGrid con el mismo nombre que
* una ObjectGrid que ya se ha colocado en antememoria o si cualquier otro error
* se ha generado durante la
* creación de ObjectGrid
* @ibm-api
*/
public List createObjectGrids(final URL xmlFile,
final boolean enableXmlValidation,
boolean cacheInstances)
throws ObjectGridException;
/*** Crear todas las ObjectGrids que se encuentren en el archivo XML. El archivo
* XML se validará en el esquema. Todas las instancias de ObjectGrid creadas
* se colocarán en antememoria. Se generará una excepción ObjectGridException
* cuando se intente colocar en antememoria una ObjectGrid recién creada que
* tenga el mismo nombre que una ObjectGrid que ya se ha colocado en antememoria.
* @param xmlFile El archivo XML que se va a procesar. Las ObjectGrids se crearán
* basándose en el contenido del archivo.
* @return Lista de instancias de ObjectGrid que se hayan creado.
* @throws ObjectGridException si una ObjectGrid con el mismo nombre que cualquiera
* de aquellos encontrados en el XML ya se ha colocado en antememoria o
* se ha encontrado cualquier otro error durante la creación de ObjectGrid.
* @ibm-api
*/
public List createObjectGrids(final URL xmlFile) throws ObjectGridException;
/**
* Procesar el archivo XML y crear una única instancia de ObjectGrid con el
* objectGridName especificado sólo si una ObjectGrid con ese nombre se encuentra
* en el archivo. Si no existe una ObjectGrid con este nombre definido en el
* archivo XML, se generará una excepción ObjectGridException. La instancia de
* ObjectGrid creada se colocará en antememoria.
* @param objectGridName nombre de la ObjectGrid que se va a crear. Esta ObjectGrid
* debería definirse en el archivo XML.
* @param xmlFile archivo XML que se va a procesar
* @return ObjectGrid recién creada
* @throws ObjectGridException si una ObjectGrid con el mismo nombre se ha
* colocado en antememoria anteriormente, no se puede encontrar ningún
* nombre de ObjectGrid en el archivo xml,
* o cualquier otro error durante la creación de ObjectGrid.
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
throws ObjectGridException;
```

Métodos getObjectGrid

Utilice los métodos getObjectGrid para recuperar las instancias en antememoria.

Recuperación de una instancia en antememoria

Desde que la interfaz ObjectGridManager coloca en antememoria la instancia de la ObjectGrid Employees, cualquier otro usuario puede acceder a ella con el siguiente fragmento de código:

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

A continuación, aparecen los dos métodos getObjectGrid que devuelven instancias de ObjectGrid en antememoria.

```
/**
* Obtener una lista de las instancias de ObjectGrid colocadas en antememoria
* anteriormente.
* Devuelve un valor nulo si no se han colocado en antememoria instancias de
* ObjectGrid.
* @return Lista de instancias de ObjectGrid que se han colocado en antememoria
*  anteriormente.
* @ibm-api
*/
public List getObjectGrids();
/**
* Utilizar este método si ya existe una ObjectGrid. Devuelve una instancia de
* ObjectGrid en antememoria por el nombre. Este método devuelve un valor nulo
* si ninguna ObjectGrid con este objectGridName se ha colocado en antememoria.
*
* @param objectGridName nombre de Objectgrid en antememoria.
* @return ObjectGrid en antememoria que existe actualmente.
*
* @since WAS XD 6.0
* @ibm-api
*
*/
public ObjectGrid getObjectGrid(String objectGridName);
```

Métodos removeObjectGrid

En este tema se describe cómo utilizar los dos métodos removeObjectGrid.

Eliminar una instancia de ObjectGrid

Para eliminar las instancias de ObjectGrid de la antememoria, utilice cualquiera de los métodos removeObjectGrid. ObjectGridManager no mantiene una referencia de las instancias eliminadas. Existen dos métodos remove. Un método toma un parámetro booleano. Si el parámetro booleano se establece en **true**, se llama al método destroy en ObjectGrid. La llamada al método destroy en ObjectGrid cierra la ObjectGrid y libera cualquier recurso que esté utilizando. A continuación, se describe cómo utilizar los dos métodos removeObjectGrid:

```
/**
* Eliminar una ObjectGrid de la antememoria de instancias de ObjectGrid
* @param objectGridName nombre de la instancia de ObjectGrid que se van a eliminar
* de la antememoria
* @throws ObjectGridException si una ObjectGrid con la objectGridName
* no se ha encontrado en la antememoria
* @ibm-api
*/
public void removeObjectGrid(String objectGridName) throws ObjectGridException;
/**
* Eliminar una ObjectGrid de la antememoria de instancias de ObjectGrid y
* destruir los recursos asociados
* @param objectGridName nombre de la instancia de ObjectGrid que se van a eliminar
* de la antememoria
* @param destroy destruir la instancia de ObjectGrid y los recursos asociados
* @throws ObjectGridException si una ObjectGrid con el objectGridName
```

```
* no se ha encontrado en la antememoria
* @ibm-api
*/
public void removeObjectGrid(String objectGridName, boolean destroy)
throws ObjectGridException;
```

Método getObjectGridAdministrator

Devolver una instancia de ObjectGridAdministrator para el clúster

```
public ObjectGridAdministrator getObjectGridAdministrator(ClientClusterContext ctx)
/**
* Devuelva una instancia de ObjectGridAdministrator para este clúster. Cada
* clúster necesitará utilizar un ObjectGridAdministrator diferente.
*
* @param clientClusterContext. Un contexto de clúster exclusivo, con el que el
* cliente debe interactuar.
* @param objectGridName nombre de Objectgrid en antememoria.
* @return una ObjectGrid
*
* @since WAS XD 6.0.1
* @ibm-api
*
*/
public ObjectGridAdministrator getObjectGridAdministrator(ClientClusterContext
ontext);
```

Consulte Capítulo 7, “Visión general de la gestión de sistemas”, en la página 69 para obtener más información sobre este método.

Utilización de la interfaz ObjectGridManager para controlar el ciclo de vida de una instancia de ObjectGrid

Este tema demuestra cómo la interfaz ObjectGridManager puede utilizarse para controlar el ciclo de vida de una instancia de ObjectGrid utilizando beans de arranque y un servlet.

Gestión del ciclo de vida de una instancia de ObjectGrid en un bean de arranque

Un bean de arranque puede utilizarse 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, se puede ejecutar código siempre que se inicie o se detenga una aplicación del modo previsto. Para crear un bean de arranque, utilice la interfaz de factoría `com.ibm.websphere.startupservice.AppStartUpHome` y la interfaz remota `com.ibm.websphere.startupservice.AppStartUp`. Implemente los métodos `start` y `stop` en el bean. El método `start` se invoca siempre que se inicie la aplicación. El método `stop` se invoca cuando se cierra la aplicación. El método `start` puede utilizarse para crear instancias de ObjectGrid. El método `stop` puede utilizarse para destruir instancias de ObjectGrid. A continuación aparece un fragmento de código que muestra la 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 antememoria 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 las ObjectGrids en antememoria y destruirlos
objectGridManager.removeObjectGrid("bookstore", true);
objectGridManager.removeObjectGrid("videostore", true);
  } catch (ObjectGridException e) {
   e.printStackTrace();
}
}
}
```

Después de que se llame al método `start`, las interfaces ObjectGrid recién creadas pueden recuperarse del ObjectGridManager. Por ejemplo, si un servlet se incluye en la aplicación, el servlet puede acceder a estas ObjectGrids mediante el siguiente fragmento de código:

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

Gestión del ciclo de vida de una ObjectGrid en un servlet

Un método para gestionar el ciclo de vida de una ObjectGrid en un servlet consiste en crear la instancia de ObjectGrid en el método `init` y destruir la ObjectGrid en el método `destroy`. Si la instancia de ObjectGrid se coloca en antememoria, puede recuperarse y manipularse en el código del servlet. A continuación, aparece un ejemplo de código que demuestra la creación, manipulación y destrucción de ObjectGrid dentro de un servlet.

```
public class MyObjectGridServlet extends HttpServlet implements Servlet {
private ObjectGridManager objectGridManager;
 public MyObjectGridServlet() {
 super();
}
public void init(ServletConfig arg0) throws ServletException {
 super.init();
 objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
try {
  // crear y colocar en antememoria una ObjectGrid denominada 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");
BackingMap bookMap = bookstoreGrid.getMap("book");
 // realizar operaciones en la ObjectGrid en antememoria
// ...
}
public void destroy() {
 super.destroy();
try {
  // eliminar y destruir la ObjectGrid bookstore en antememoria
objectGridManager.removeObjectGrid("bookstore", true);
  } catch (ObjectGridException e) {
   e.printStackTrace();
}
}
}
```

Rastreo de ObjectGrid

Este tema explica cómo configurar el rastreo para ObjectGrid.

Entorno J2SE (Java 2 Platform, Standard Edition)

Cuando sea necesario enviar la información de depuración a IBM, utilice el mecanismo de rastreo para obtener el rastreo de depuración. A continuación se muestra un ejemplo sobre cómo obtener el rastreo de depuración en un entorno J2SE:

```
oGridManager.setTraceFileName("debug.log");
oGridManager.setTraceSpecification("ObjectGrid=all=enabled");
```

El ejemplo anterior no incluye el rastreo de plug-ins de desalojador incorporados para ObjectGrid. Si esta utilizando uno o más plug-ins de desalojador proporcionados por ObjectGrid y tiene problemas que puedan estar relacionados con el desalojo, habilite el rastreo para ObjectGrid y los desalojadores de ObjectGrid, como se ilustra en el siguiente ejemplo:

```
oGridManager.setTraceFileName("debug.log");
oGridManager.setTraceSpecification
("ObjectGridEvictors=all=enabled:ObjectGrid=all=enabled");
```

Entorno de WebSphere Application Server

No es necesario utilizar ObjectGridManager para establecer el rastreo en un entorno de WebSphere Application Server. Puede utilizar la consola administrativa para establecer la especificación de rastreo.

API de conexión de cliente ObjectGrid

Conceptos básicos

Un cliente se conecta a un proceso de servidor activo o en ejecución en un clúster. El cliente necesita como mínimo el nombre del sistema principal y el número de puerto del servidor con el que se conecta. El nombre de sistema principal y la información de puerto están disponibles en el archivo XML de definición de clúster que se ha utilizado inicialmente para iniciar el servidor. Para obtener más información sobre la configuración XML, consulte “Configuración de ObjectGrid” en la página 261. A continuación, se proporciona un fragmento de código de la definición XML de clúster que se utiliza como ejemplo en este apartado. Utilizando las API que se describen en este apartado, el cliente se conecta a una ObjectGrid

remoto que está configurado para recibir y procesar peticiones de cliente. Tenga en cuenta que el cliente ″descarga″ la ObjectGrid y las definiciones XML de clúster para realizar su propia rutina de carga. La configuración de cliente se basa en la configuración de servidor a la que se conecta.

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster ../
objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1" securityEnabled="false" clientMaxRetries="15"
tcpConnectionTimeout="180"
singleSignOnEnabled="true" loginSessionExpirationTime="300"
statisticsEnabled="true" statisticsSpec="all=enabled">
<serverDefinition name="server1" host="s1.myco.com" clientAccessPort="12503"
peerAccessPort="12500" workingDirectory="/tmp/s1/"
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="s2.myco.com" clientAccessPort="12504"
peerAccessPort="12501" workingDirectory="/tmp/s2/"
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server3" host="10.5.1.22" clientAccessPort="12505"
peerAccessPort="12502" workingDirectory="/tmp/s3/"
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true"/>
</cluster>
<objectgrid-binding
.
.
.
```

Esta definición de clúster está incompleta, pero es suficiente para este ejemplo. Dentro del clúster cluster1 hay tres servidores: server1, server2 y server3. El atributo clientAccessPort especifica el puerto receptor donde escucha el servidor y el puerto con el que el cliente establece una conexión por primera vez. A partir del fragmento XML anterior, los puertos de server1, server2 y server3 son `12503`, `12504` y `12504`, respectivamente.

API de conexión

La interfaz ObjectGridManager tiene métodos de conexión que se describen en el siguiente ejemplo. Las siguientes API de conexión están disponibles en la interfaz ObjectGridManager. Consulte la documentación de la API para obtener una descripción de estos métodos.

```
/**
* Esto permite a un cliente conectarse a una ObjectGrid remota
* La ObjectGrid remota se aloja tal como se especifica en los parámetros:
* @param clusterName: El nombre del clúster al que se conectará este
* cliente
* @param host: El sistema principal al que conectarse
* @param port: El puerto clientAceess que está escuchando
* @param ClientSecurityConfiguration: Configuración de seguridad, puede ser
* nula si no se configura ninguna seguridad.
* @param overRideObjectGrid xml. El parámetro puede ser nulo. Si no es nulo, la
* configuración del lado del cliente del plug-in de ObjectGrid se altera temporalmente.
* No todos los plug-ins se pueden alterar temporalmente. Para obtener más información,
* consulte los documentos de ObjectGrid
* @throws ConnectException
*
*/
public ClientClusterContext connect(String clusterName,
String host,
String port,
```

```
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
/**
*
* @param clusterName
* @param attributes Atributos del par de Sistema principal y Puerto que se prueban
* secuencialmente para conectarse. Si un intento de conectarse falla en un servidor,
* se elige el siguiente par de atributos de sistema principal y puerto para reintentarlo.
* @param ClientSecurityConfiguration: Configuración de seguridad. Puede ser
* nula si no se configura ninguna seguridad.
* @param overRideObjectGrid xml. El parámetro puede ser nulo. Si no es nulo, la
* configuración del lado del cliente del plug-in de ObjectGrid se altera
temporalmente.
* No todos los plug-ins se pueden alterar temporalmente. Para obtener más
información, consulte los documentos de ObjectGrid.
* @return ClientClusterContext
* @throws ConnectException
*
*
*/
public ClientClusterContext connect(String clusterName,
HostPortConnectionAttributes[] attributes,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
/**
* Este método sólo se puede utilizar si se coloca un cliente con un
* un servidor ObjectGrid, especialmente en un entorno J2EE (Java 2 Platform,
* Enterprise Edition) con IBM WebSphere
* Application Server, que da soporte al servidor
* ObjectGrid incorporado.
* Este método conecta el cliente al servidor que se ejecuta en la misma
* Java Virtual Machine (JVM).
* @param securityProps. Puede ser nulo si no se ejecuta en modalidad segura.
* @param overRideObjectGrid xml. El parámetro puede ser nulo. Si no es nulo, la
* configuración del lado del cliente del plug-in de ObjectGrid se altera temporalmente.
* No todos los plug-ins se pueden alterar temporalmente. Para obtener más
información,
* consulte los documentos de ObjectGrid
* @return ClientClusterContext
* @throws ConnectException
*
*/
public ClientClusterContext connect(ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException;
/**
* Esto permite a un cliente conectarse a una ObjectGrid remota
* @param clusterConfigFile Un URL al archivo clusterConfig. Este es el mismo
* archivo que se utiliza para iniciar los servidores. Se utiliza para
* recuperar información de sistema principal y puerto. No puede ser nulo. Si
* es nulo, se genera una excepción IllegalArgumentException.
* @param serverName Una serie; nombre del servidor específico con el que se conecta.
* Si el nombre del servidor no está en la configuración, se genera una
* IllegalArgumentException.
* Este parámetro puede ser nulo, en cuyo caso, se intenta conectar a uno
* de los servidores especificados en el archivo XML de
* clúster. Si el intento de conectarse a uno falla, se elige otro servidor.
* Ha finalizado; hasta entonces, la lista está agotada.
* @param securityProps
* @param overRideObjectGrid xml. El parámetro puede ser nulo. Si no es nulo, la
* configuración del lado del cliente del plug-in de ObjectGrid se altera temporalmente.
* No todos los plug-ins se pueden alterar temporalmente. Para obtener más información,
* consulte los documentos de ObjectGrid
* @return ClientClusterContext
* @throws ConnectException
*
* @ibm-api
*/
```

```
public ClientClusterContext connect(URL clusterConfigFile,
String serverName,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
```

Ejemplo de uso de parámetros de sistema principal y puerto

El siguiente código utiliza el XML de clúster que se describe en “Conceptos básicos” en la página 100. Este cliente se conecta con el sistema principal `s1.myco.com` en el puerto `12503`.

```
import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ConnectException;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
public class C1 {
/**
* @param args
*/
public static void main(String[] args) {
final ObjectGridManager oGridManager=ObjectGridManagerFactory.
getObjectGridManager(); //step 1
ClientClusterContext ctx = null;
try {
ctx=oGridManager.connect("cluster1","s1.myco.com","12503",null,null);
//paso 2
ObjectGrid employees = oGridManager.getObjectGrid(ctx,"employees");
// paso 3
// Ejecutar operaciones de objectGrid
// get
// update
// commit...etc..
} catch (ConnectException e) {
//conexión anómala
   e.printStackTrace();
//terminar
}finally {
if(ctx !=null) {
oGridManager.disconnect(ctx); // paso 4
}
}
}
}
```

1. Obtenga el objeto singleton de ObjectGridManager a partir de ObjectGridManagerFactory.
2. Invoque la API de conexión.
3. Suponiendo que existen los empleados de objectGrid en la ObjectGrid remota, invoque el método getObjectGrid pasando el parámetro ClientClusterContext.
4. Invoque el método de desconexión. Como último paso, todos los clientes deben invocar la desconexión, si el trabajo ha finalizado. Este es un paso muy importante.

Proporcionar varios sistemas principales para reintentar automáticamente la conexión, en el caso de que se produzca una excepción ConnectException

En este ejemplo, los atributos HostPortConnectionAttributes se utilizan para proporcionar una matriz de atributos de sistema principal y puerto a los que se puede conectar un cliente. La API utiliza este par de atributos de sistema principal y puerto en orden secuencial para conectarse. Si un intento de conectarse falla en un servidor, se elige el siguiente par de atributos de sistema principal y puerto para reintentar la conexión.

```
import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ConnectException;
import com.ibm.websphere.objectgrid.HostPortConnectionAttributes;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
public class C2 {
/**
* @param args
*/
public static void main(String[] args) {
final ObjectGridManager oGridManager=ObjectGridManagerFactory.
getObjectGridManager();
ClientClusterContext ctx = null;
HostPortConnectionAttributes[] hca = new HostPortConnectionAttributes[3];
hca[0]=new HostPortConnectionAttributes("s1.myco.com","12503");
hca[1]=new HostPortConnectionAttributes("s2.myco.com","12504");
hca[2]=new HostPortConnectionAttributes("10.5.1.22","12505");
try {
ctx=oGridManager.connect("cluster1",hca,null,null);
ObjectGrid employees = oGridManager.getObjectGrid(ctx,"employees");
// Ejecutar operaciones de objectGrid como, por ejemplo,
// get
// update
// commit.... etc...
} catch (ConnectException e) {
   e.printStackTrace();
}finally {
if(ctx !=null) {
oGridManager.disconnect(ctx);
}
}
}
}
```

Cliente y servidor en el mismo proceso

Si un cliente está en la misma JVM que el servidor, se puede utilizar el siguiente método de conexión.

```
ctx=oGridManager.connect(null,null);
```

Especificar XML de clúster

Si el cliente tiene acceso al archivo XML de clúster, no es necesario especificar el nombre de sistema principal o el número de puerto. Esta API recupera el nombre del servidor y el número de puerto, y los utiliza para conectarse. El nombre de servidor es opcional y puede ser nulo, en cuyo caso, la API intenta conectarse a uno de los servidores definidos en el archivo XML de clúster.

```
ctx=oGridManager.connect(urlToClusterxml,"server1",null,null);
// conectarse a server1
// o
ctx=oGridManager.connect(urlToClusterxml,null,null,null);
//conectarse a cualquier servidor del clúster
```

Seguridad del cliente con la API de conexión

En todos los ejemplo, el parámetro as ClientSecurityConfiguration era nulo. Si pasa el valor nulo, implica que la seguridad está inhabilitada. Si la seguridad está habilitada, pase el objeto ClientSecurityConfiguration como argumento. Para obtener más información, consulte “Seguridad de ObjectGrid” en la página 140.

Alterar temporalmente la configuración XML de ObjectGrid

El cliente ″descarga″ las definiciones de ObjectGrid del servidor para configurarse. Todos los plug-ins definidos en la ObjectGrid estarán disponibles para el cliente. Básicamente, existe una ObjectGrid local en el lado del cliente que se comunica con la ObjectGrid del lado del servidor. Al proporcionar un archivo XML de alteración temporal en la API de conexión, puede ″alterar temporalmente″ la configuración de los plug-ins, que son específicos sólo para el uso del cliente. Estos plug-ins son:

Plug-ins de **ObjectGrid**:

- Plug-in TransactionCallback
- Plug-in ObjectGridEventListener

Plug-ins de **BackingMap**:

- Plug-in Evictor
- Plug-in MapEventListener

Los demás plug-ins definidos en el XML de alteración temporal se ignoran.

Ejemplo

Supongamos que un cliente debe alterar temporalmente la configuración de Evictor para una determinada BackingMap. Es decir, en el lado del cliente, el Evictor debe ser diferente al configurado en el lado del cliente.

Supongamos que el Evictor del lado del servidor se utiliza de la siguiente manera. Utiliza el desalojador LFUEvictor incorporado:

```
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
<property name="maxSize" type="int" value="100" description="..." />
</bean>
```

Los requisitos del lado del cliente son diferentes. En este caso, se debe utilizar un desalojador myco.og.MyEvictor definido por el usuario. El XML de alteración temporal puede incluir el fragmento que se muestra a continuación. Todas las backingMaps configuradas para utilizar LFUEvictor utilizan el siguiente desalojador definido por el usuario:

```
<bean id="Evictor" className="myco.og.MyEvictor">
<property name="name" type="java.lang.String" value="MyEvictor"
description="..." />
</bean>
```

XML completo

El siguiente código XML muestra dos archivos XML: uno que se utiliza para el lado del servidor y otro para el cliente. Esta configuración permite al cliente alterar temporalmente la configuración de Evictor para la BackingMap dow.

Archivo XML de ObjectGrid del lado del servidor

```
<?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="market">
```

```
<backingMap name="dow" ttlEvictorType="NONE" readOnly="false"
pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.
builtins.LRUEvictor">
<property name="maxSize" type="int" value="2"
description="establecer el tamaño máximo del desalojador LRU" />
<property name="numberOfLRUQueues" type="int" value="1"
description="establecer el número de colas de LRU" />
<property name="sleepTime" type="int" value="2" description="tiempo de
inactividad de la hebra de desalojador" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Archivo XML de ObjectGrid del lado del cliente que se altera temporalmente durante la conexión

```
<?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="market">
<backingMap name="dow" pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="Evictor" className="myco.og.MyEvictor">
<property name="name" type="java.lang.String" value="MyEvictor"
description="" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Consideraciones sobre el diseño de las aplicaciones

La API connect() de cliente es una operación costosa. Dependiendo del trabajo, un cliente establece una o varias conexiones físicas en un solo servidor. El número de conexiones de cliente varía entre los valores especificados por los atributos tcpMinConnections y tcpMaxConnections definidos en el elemento cluster de la definición XML de configuración de clúster. Esta es la misma definición XML de clúster que se utilizó para iniciar el servidor. El gestor de conexiones agrupa estas conexiones físicas y ObjectGrid las utiliza según sea necesario. Los atributos tcpMinConnections y tcpMaxConnections especifican el número de conexiones de cliente con un único servidor. Si un cliente se conecta a más de un servidor, el número máximo de conexiones de cliente es menor o igual que el atributo tcpMaxConnections por el número de servidores con los que se conecta el cliente. Por ejemplo, si el cliente se conecta a tres servidores y tcpMaxConnections está especificado como cinco, el cliente tiene un máximo de `(5*3)=15` conexiones y un mínimo de tres conexiones, suponiendo que el valor de tcpMinConnection es `1`. Las conexiones se comparten entre todos los clientes.

El atributo **threadsPerClientConnect** especifica el número de hebras de trabajo. Estas hebras de trabajo entregan el trabajo a través de las conexiones físicas. Procesan datos de configuración, peticiones de cliente, respuestas de servidor y peticiones de administración del sistema. El valor por omisión es `5`. Este atributo

está disponible en el primer iFix. Si el iFix no está disponible, utilice la propiedad de sistema de la máquina virtual Java (JVM) -Dthreads para especificar el número de hebras de trabajo. Dependiendo de la aplicación, si aumenta este número puede aumentar el rendimiento. Como norma general, este número no debe ser menor que el número de conexiones físicas que utiliza el cliente.

Si el diseño de la aplicación lo permite, un cliente puede crear varias hebras para completar el trabajo en una llamada de conexión reutilizando el método *ClientClusterContext*.

Interfaz ObjectGrid

Utilice este tema para consultar los métodos que se necesitan para modificar una ObjectGrid.

Introducción

ObjectGrid es una infraestructura ampliable de colocación en antememoria de objetos transaccionales basada en la interfaz Map de Java. Las operaciones de la API de ObjectGrid se agrupan en una unidad transaccional de trabajo y permiten la ampliación mediante el soporte para plug-ins de diseño personalizado. ObjectGrid es un contenedor lógico con nombre que contiene un número de BackingMaps. Para obtener más información sobre las correlaciones de respaldo, consulte “Interfaz BackingMap” en la página 112.

Creación e inicialización

Consulte el tema sobre la interfaz ObjectGridManager para obtener los pasos que son necesarios para crear una instancia de ObjectGrid. Existen dos métodos diferenciados para crear una ObjectGrid: mediante programación y con archivos de configuración XML. Para obtener más información, consulte “Interfaz ObjectGridManager” en la página 93.

Métodos de fábrica y get o set

Atención: Los métodos *set* se deben invocar antes de inicializar la instancia de ObjectGrid. Si invoca un método set después de invocar el método initialize, se genera una `java.lang.IllegalStateException`. Cada uno de los métodos getSession de la interfaz ObjectGrid también invocan implícitamente el método initialize. Por lo tanto, debe invocar los métodos set antes de invocar los métodos getSession. La única excepción a esta norma es con el establecimiento, la adición y la eliminación de objetos EventListener. Estos objetos se pueden procesar después de que el proceso de ″initialize″ haya finalizado.

La interfaz ObjectGrid contiene los siguientes métodos:

Tabla 7. Métodos de la interfaz ObjectGrid

Método	Descripción
BackingMap defineMap(String name);	*defineMap*: es un método de fábrica para definir una BackingMap denominada exclusivamente. Para obtener más información sobre las correlaciones de respaldo, consulte “Interfaz BackingMap” en la página 112.

Tabla 7. Métodos de la interfaz ObjectGrid (continuación)

Método	Descripción
BackingMap getMap(String name);	*getMap*: devuelve una BackingMap definida anteriormente llamando a *defineMap*. Utilizando este método, puede configurar la BackingMap, si no se ha configurado todavía con la configuración XML.
BackingMap createMap(String name);	*createMap*: crea una BackingMap, pero no la almacena en antememoria para su uso con esta ObjectGrid. Utilice este método conjuntamente con el método setMaps(List) de la interfaz ObjectGrid, que almacena en antememoria BackingMaps para su uso con esta ObjectGrid. Utilice estos métodos cuando configure una ObjectGrid con la infraestructura de Spring.
void setMaps(List mapList);	*setMaps*: borra las BackingMaps que se han definido previamente en esta ObjectGrid y las sustituye por la lista de BackingMaps que se proporciona.
public Session getSession() throws ObjectGridException, TransactionCallbackException;	*getSession*: devuelve una Session, que proporciona las funciones de inicio, compromiso y retrotracción para una unidad de trabajo. Para obtener más información sobre los objetos Session, consulte “Interfaz Session” en la página 116.
Session getSession(CredentialGenerator cg);	*getSession(CredentialGenerator cg)*: obtiene una sesión con un objeto CredentialGenerator. Este método sólo lo puede invocar el cliente ObjectGrid en un entorno de cliente-servidor.
Session getSession(Subject subject);	*getSession(Subject subject)*: permite utilizar un objeto Subject específico en lugar del configurado en la ObjectGrid para obtener una Session.
void initialize() throws ObjectGridException;	*initialize*: se inicializa ObjectGrid y pasa a estar disponible para uso general. Se llama a este método implícitamente cuando se llama al método `getSession`, si ObjectGrid no está en un estado inicializado.
void destroy();	*destroy*: se desensambla la infraestructura y no puede utilizarse después de que se llame a este método.

Tabla 7. Métodos de la interfaz ObjectGrid (continuación)

Método	**Descripción**
void setTxTimeout(int timeout);	*setTxTimeout*: utilice este método para establecer la cantidad de tiempo, en segundos, que tiene una transacción iniciada por una Session que ha creado esta instancia de ObjectGrid antes de finalizar. Si una transacción no finaliza en la cantidad de tiempo especificada, la Session que ha iniciado transacción se marca como ″se ha excedido el tiempo de espera″. Marcar una Session como que se ha excedido el tiempo de espera hace que el siguiente método ObjectMap que invoque la Session caducada genere una excepción `com.ibm.websphere.objectgrid.TransactionTimeoutException` . La Session se marca como *sólo de retrotracción*, lo que hace que la transacción se retrotraiga aunque la aplicación invoque el método commit en lugar del método rollback una vez detectada la excepción `TransactionTimeoutException` en la aplicación. Un valor de tiempo de espera `0` indica que la transacción tiene una cantidad ilimitada de tiempo para finalizarse. La transacción no excede el tiempo de espera si se utiliza un valor de tiempo de espera `0`. Si no se invoca este método, cualquier Session devuelta por el método getSession de esta interfaz tiene un valor de tiempo de espera de transacción establecido en `0` por omisión. Una aplicación puede alterar temporalmente el valor de tiempo de espera de transacción para cada Session utilizando el método setTransactionTimeout de la interfaz com.ibm.websphere.objectgrid.Session.
int getTxTimeout();	*getTxTimeout*: devuelve el valor de tiempo de espera de la transacción en segundos. Este método devuelve el mismo valor que se ha pasado como parámetro de tiempo de espera en el método setTxTimeout. Si no se ha invocado el método setTxTimeout, el método devuelve `0` para indicar que la transacción tiene una cantidad ilimitada de tiempo para finalizarse.
//Palabras clave.	

Tabla 7. Métodos de la interfaz ObjectGrid (continuación)

Método	Descripción
void associateKeyword(Serializable parent, Serializable child);	*associateKeyword*: la palabra clave de ObjectGrid proporciona un mecanismo de invalidación flexible basado en las palabras clave. Para obtener más información sobre las palabras clave, consulte “Palabras clave” en la página 124. Este método enlaza las dos palabras clave en una relación direccional. Si se invalida un padre, también se invalidará el hijo. La invalidación de un hijo no tiene ningún impacto sobre en el padre.
//Seguridad	
void setSecurityEnabled()	*setSecurityEnabled*: habilita la seguridad. La seguridad está inhabilitada por omisión.
void setPermissionCheckPeriod(long period);	*setPermissionCheckPeriod*: este método toma un único parámetro que indica con qué frecuencia se va a comprobar el permiso utilizado para permitir un acceso de cliente. Si el parámetro es `0`, todos los métodos solicitan el mecanismo de autorización, ya sea la autorización JAAS o la autorización personalizada, para comprobar si el sujeto actual tiene permiso. Es posible que esta estrategia ocasione problemas de rendimiento en función de la implementación de autorización. No obstante, este tipo de autorización está disponible, aunque no sea necesaria. Alternativamente, si el parámetro es inferior a `0`, éste indica el número de milisegundos para colocar en antememoria un conjunto de permisos antes de volver al mecanismo de autorización para renovarlos. Este parámetro proporciona un rendimiento mucho mejor, pero, si los permisos de programa de fondo se modifican durante este tiempo, es posible que ObjectGrid permita o niegue el acceso incluso si el proveedor de seguridad de programa de fondo se ha modificado.
void setAuthorizationMechanism(int authMechanism);	*setAuthorizationMechanism*: establece el mecanismo de autorización. El valor por omisión es `SecurityConstants.JAAS_AUTHORIZATION`.
setMapAuthorization(MapAuthorization ma);	*setMapAuthorization*: establece el plug-in MapAuthorization para esta instancia de ObjectGrid. Este plug-in puede utilizarse para autorizar los accesos de ObjectMap o JavaMap a los principales contenidos en el objeto Subject. Una implementación típica de este plug-in consiste en recuperar los principales del objeto Subject y, a continuación, comprobar si los permisos especificados se otorgan a los principales.

Tabla 7. Métodos de la interfaz ObjectGrid (continuación)

Método	Descripción
setSubjectSource(SubjectSource ss);	*setSubjectSource*: establece el plug-in SubjectSource. Este plug-in puede utilizarse para obtener un objeto Subject que represente al cliente ObjectGrid. Este sujeto se utilizará para la autorización de ObjectGrid. El tiempo de ejecución de ObjectGrid invoca el método SubjectSource.getSubject cuando se utiliza el método ObjectGrid.getSession para obtener una sesión y la seguridad está inhabilitada. Este plug-in es útil para un cliente que ya está autenticado: puede recuperar el objeto Subject autenticado y pasarlo a la instancia de ObjectGrid. No es necesaria otra autenticación.
setSubjectValidation(SubjectValidation sv);	*setSubjectValidation*: establece el plug-in SubjectValidation para esta instancia de ObjectGrid. Este plug-in puede utilizarse para validar que un sujeto javax.security.auth.Subject que se pasa a la ObjectGrid es un sujeto válido que no se ha manipulado. El creador del objeto Subject debe dar soporte a la implementación de este plug-in, ya que sólo el creador sabe si se ha manipulado el objeto Subject. No obstante, es posible que un creador del sujeto no sepa si se ha manipulado el objeto Subject. En este caso, no debe utilizarse este plug-in.

Interfaz ObjectGrid: plug-ins

La interfaz ObjectGrid tiene varios puntos de conexión opcionales que permiten interacciones más amplias.

```
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*: una interfaz ObjectGridEventListener se utiliza para recibir notificaciones cuando se producen sucesos significativos en ObjectGrid. Estos sucesos incluyen la inicialización de ObjectGrid, el comienzo de una transacción, la finalización de una transacción y la destrucción de una ObjectGrid. Para escuchar estos sucesos, cree una clase que implemente la interfaz ObjectGridEventListener y añádala a la ObjectGrid. Estos receptores se asocian con cada Session. Consulte “Receptores” en la página 187 y “Interfaz Session” en la página 116 para obtener más información.
- *TransactionCallback*: una interfaz de receptor TransactionCallback permite que se envíen a esta interfaz sucesos de transacción como, por ejemplo, las señales de inicio, compromiso y retrotracción. Generalmente, una interfaz de receptor TransactionCallback se utiliza con un Loader. Para obtener más información,

consulte “Plug-in TransactionCallback” en la página 218 y “Cargadores” en la página 202. Estos sucesos pueden utilizarse seguidamente para coordinar transacciones con un recurso externo o dentro de varios cargadores.

- *reserveSlot*: permite que los plug-ins de esta ObjectGrid reserven ranuras para utilizarlas en las instancias de objeto que tengan ranuras como TxID.
- *SubjectValidation*. Si la seguridad está habilitada, este plug-in puede utilizarse para validar una clase `javax.security.auth.Subject` que se pase a ObjectGrid.
- *MapAuthorization*. Si la seguridad está habilitada, este plug-in puede utilizarse para autorizar que ObjectMap acceda a los principales representados por el objeto Subject.
- *SubjectSource* si la seguridad está habilitada, este plug-in puede utilizarse para obtener un objeto Subject que represente al cliente ObjectGrid. A continuación, este sujeto se utilizará para la autorización de ObjectGrid.

Interfaz BackingMap

Todas las instancias de ObjectGrid contienen una colección de objetos BackingMap.

Todos los objetos BackingMap reciben un nombre y se añaden a la instancia de ObjectGrid utilizando el método defineMap o el método createMap de la interfaz ObjectGrid. Estos métodos devuelven una instancia BackingMap que se utiliza seguidamente para definir el comportamiento de una correlación individual. Para obtener más información, consulte “Interfaz ObjectGrid” en la página 107.

La interfaz Session se utiliza para iniciar una transacción y obtener la ObjectMap o JavaMap que se necesita para realizar una interacción transaccional entre una aplicación y un objeto BackingMap. No obstante, los cambios de transacción no se aplican al objeto BackingMap hasta que se ha comprometido la transacción. Una BackingMap puede considerarse como una antememoria en memoria de datos comprometidos de una correlación individual. Para obtener más información sobre la interfaz Session, consulte Interfaz Session.

La interfaz com.ibm.websphere.objectgrid.BackingMap proporciona los métodos para establecer los atributos de BackingMap. Algunos de los métodos set permiten la ampliación de BackingMap mediante varios plug-ins de diseño personalizado. A continuación, aparece una lista de los métodos set para establecer atributos y proporcionar soporte para plug-ins de diseño personalizado:

```
// Para establecer los atributos 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);
// Para establecer un plug-in personalizado proporcionado por 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 método set enumerado existe el correspondiente método get.

Atributos de BackingMap

Todas las BackingMap tienen los siguientes atributos que pueden establecerse para modificar o controlar el comportamiento de BackingMap:

- Atributo *ReadOnly*. Este atributo indica si la correlación es de sólo lectura o de lectura y grabación. Si nunca se establece este atributo para la correlación, ésta pasa a ser de lectura y grabación por omisión. Cuando una BackingMap se establece para que sea de sólo lectura, ObjectGrid optimiza el rendimiento para sólo lectura siempre que sea posible.
- Atributo *NullValuesSupported*. Este atributo indica si un valor nulo puede colocarse en la Map. Si nunca se establece este atributo, la correlación no da soporte a los valores nulos. Si la correlación da soporte a los valores nulos, una operación get que devuelva un valor nulo puede indicar que el valor es nulo o que la correlación no contiene la clave especificada por la operación get.
- Atributo *LockStrategy*. Este atributo determina si esta BackingMap utiliza un gestor de bloqueos. Si se utiliza un gestor de bloqueos, se usa el atributo LockStrategy para indicar si se adopta un enfoque de bloqueo optimista o pesimista para bloquear las entradas de correlación. Si este atributo no está establecido, se utiliza la LockStrategy optimista. Consulte el tema “Bloqueo” en la página 131 para obtener detalles sobre las estrategias de bloqueo soportadas.
- Atributo *CopyMode*. Este atributo determina si BackingMap hace una copia de un objeto de valor cuando se lee un valor de la correlación o se coloca en la BackingMap durante el ciclo de compromiso de una transacción. Se da soporte a varias modalidades de copia para permitir que la aplicación consiga el equilibrio entre el rendimiento y la integridad de los datos. Si este atributo no está establecido, se utilizará la modalidad de copia `COPY_ON_READ_AND_COMMIT`. Esta modalidad de copia no tiene el mejor rendimiento, pero ofrece la mejor protección frente los problemas relacionados con la integridad de los datos. Para obtener más información sobre las modalidades de copia, consulte Procedimientos recomendados para el método copyMode.
- Atributo *CopyKey*. Este atributo determina si la BackingMap hace una copia de un objeto de clave cuando se crea una entrada por primera vez en la correlación. La acción por omisión es no hacer una copia de los objetos de clave porque las claves normalmente son objetos no modificables.
- Atributo *NumberOfBuckets*. Este atributo indica el número de cubetas hash que va a utilizar BackingMap. La implementación de BackingMap utiliza una correlación hash para su implementación. Si existen numerosas entradas en BackingMap, cuanto más cubetas haya, mejor es el rendimiento. Cuanto mayor es el número de cubetas, menor es el número de claves que tienen el mismo número de cubetas. Cuando más elevado sea el número de cubetas, mayor será la concurrencia. Este atributo es útil para el ajuste del rendimiento con precisión. Se utiliza un valor por omisión de `503` si la aplicación no establece el atributo NumberOfBuckets.
- Atributo *NumberOfLockBuckets*. Este atributo indica el número de cubetas de bloqueo que va a utilizar el gestor de bloqueos para 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 numerosas entradas en la correlación hash, cuantas más cubetas de bloqueo haya, mejor será el rendimiento, ya que el número de claves que colisionan en la misma cubeta disminuye a medida que crece el número de cubetas. Cuanto más elevado sea el número de cubetas de bloqueo, mayor será la concurrencia. Cuando el atributo LockStrategy se establece en NONE, esta BackingMap no utiliza ningún gestor de bloqueos. En este caso, el establecimiento de numberOfLockBuckets no tiene ningún efecto. Si no se establece este valor, se utiliza un valor por omisión de 383.

- Atributo *LockTimeout*. Este atributo se utiliza cuando BackingMap utilice un gestor de bloqueos. BackingMap utiliza un gestor de bloqueos cuando el atributo LockStrategy está establecido en OPTIMISTIC o PESSIMISTIC. El valor del atributo está en segundos y determina durante cuánto tiempo el gestor de bloqueos debe esperar para que se conceda un bloqueo. Si este atributo no está establecido, se utiliza un valor de 15 segundos como valor de LockTimeout. Consulte Bloqueo pesimista para obtener detalles sobre las excepciones de tiempo de espera de bloqueo que ocurren.
- Atributo *TtlEvictorType*. Todas las BackingMap tiene su propio desalojador de tiempo de duración incorporado que utiliza un algoritmo basado en el tiempo para determinar qué entradas de correlación se van a desalojar. Por omisión, el desalojado de tiempo de vida incorporado no está activo. Puede activarlo llamando al método setTtlEvictorType con uno de los tres valores: CREATION_TIME, LAST_ACCESS_TIME o NONE. Un valor de CREATION_TIME indica que el desalojador añade el atributo TimeToLive a la hora a la que se ha creado la entrada de correlación en BackingMap para determinar cuando el desalojador debe desalojar esta entrada de BackingMap. Un valor de LAST_ACCESS_TIME indica que el desalojador añade el atributo TimeToLive a la hora a la que alguna transacción que la aplicación esté ejecutando ha accedido a la entrada por última vez para determinar cuándo el desalojador debe desalojar la entrada de correlación. La entrada de correlación sólo se desaloja si ninguna transacción no accede nunca a una entrada de correlación durante un periodo de tiempo especificado por el atributo TimeToLive. Un valor de NONE indica que el desalojador debe permanecer inactivo y no desalojar nunca ninguna entrada de correlación. Si no se establece nunca este atributo, se utiliza NONE como el valor por omisión y el desalojador de tiempo de duración no está activo. Consulte Desalojadores para obtener detalles sobre el desalojador de tiempo de duración incorporado.
- Atributo *TimeToLive*. Este atributo se utiliza para especificar el número de segundos que el desalojador de tiempo de duración incorporado necesita añadir a la hora de creación o de último acceso de cada entrada, tal como se describe en el caso del atributo TtlEvictorType. Si nunca se establece este atributo, el valor especial de cero se utiliza para indicar que el tiempo de duración es infinito. Si el atributo se establece en infinito, el desalojador nunca desaloja las entradas de correlación.

El ejemplo siguiente ilustra cómo se define la BackingMap someMap en la instancia de la ObjectGrid someGrid y se establecen varios atributos de la 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 ); // sobrescribir el valor por omisión de lectura/grabación
bm.setNullValuesSupported(false); // sobrescribir el valor por omisión que
// permite valores nulos
bm.setLockStrategy( LockStrategy.PESSIMISTIC ); // sobrescribir el valor por
// omisión de OPTIMISTIC
bm.setLockTimeout( 60 ); // sobrescribir el valor por omisión de 15 segundos.
bm.setNumberOfBuckets(251); // sobrescribir el valor por omisión (los
// números primos funcionan mejor)
bm.setNumberOfLockBuckets(251); // sobrescribir el valor por omisión (los
// números primos funcionan mejor)
...
```

Plug-ins BackingMap

La interfaz BackingMap tiene varios puntos de conexión opcionales que permiten interacciones con ObjectGrid más amplias:

- **Plug-in ObjectTransformer**: para algunas operaciones de correlación, es posible que una BackingMap necesite serializar, deserializar o copiar una clave o valor de una entrada en la BackingMap. La BackingMap puede realizar estas acciones proporcionando una implementación por omisión de la interfaz ObjectTransformer. Una aplicación puede mejorar el rendimiento proporcionando un plug-in ObjectTransformer de diseño personalizado que la BackingMap pueda utilizar para serializar, deserializar o copiar una clave o valor de una entrada en la BackingMap. Para obtener más información, consulte “Plug-in ObjectTransformer” en la página 214.
- **Plug-in Evictor**: el desalojador de tiempo de espera incorporado utiliza un algoritmo basado en el tiempo para decidir cuando se debe desalojar una entrada de BackingMap. Es posible que algunas aplicaciones necesiten utilizar un algoritmo distinto para decidir cuando una entrada de BackingMap debe desalojarse. El plug-in Evictor pone a disposición de la BackingMap un Evictor de diseño personalizado para que lo utilice. El plug-in Evictor existe además del desalojador de tiempo de espera incorporado. No sustituye al desalojador de tiempo de espera incorporado. ObjectGrid proporciona un plug-in Evictor personalizado que implementa los algoritmos conocidos como, por ejemplo ″menos utilizado recientemente″ o ″utilizado menos frecuentemente″. Las aplicaciones pueden conectar uno de los plug-ins Evictor proporcionados o pueden proporcionar su propio plug-in Evictor. Para obtener más información, consulte “Desalojadores” en la página 192.
- **Plug-in MapEventListener**: es posible que una aplicación desee tener conocimiento de los sucesos de BackingMap como, por ejemplo, el desalojo de una entrada de correlación o la precarga de la finalización de una BackingMap. Una BackingMap llama a los métodos en el plug-in MapEventListener para notificar a una aplicación los sucesos de BackingMap. Una aplicación puede recibir la notificación de varios sucesos de BackingMap utilizando el método setMapEventListener para proporcionar uno o más plug-ins MapEventListener de diseño personalizado a BackingMap. La aplicación puede modificar los objetos MapEventListener enumerados mediante los métodos addMapEventListener o removeMapEventListener. Para obtener más información, consulte “Interfaz MapEventListener” en la página 190.
- **Plug-in Loader**: una BackingMap es una antememoria en memoria de una correlación. Un plug-in Loader es una opción que BackingMap utiliza para mover datos en la memoria y como almacenamiento persistente de BackingMap. Por ejemplo, un Loader JDBC (Java Database Connectivity) puede utilizarse para mover datos dentro y fuera de una BackingMap y una o más tablas relacionales de una base de datos relacional. No es necesario utilizar una base de datos relacional como almacenamiento persistente de una BackingMap. El Loader

también puede utilizarse para mover datos entre una BackingMap y un archivo, entre una BackingMap y una correlación Hibernate, entre una BackingMap y un bean de entidad J2EE (Java 2 Platform, Enterprise Edition) y entre una BackingMap y otro servidor de aplicaciones, etc. La aplicación debe proporcionar un plug-in Loader de diseño personalizado para mover datos entre la BackingMap y el almacenamiento persistente para todas las tecnologías que se utilicen. Si no se facilita un Loader, BackingMap simplemente se convierte en una antememoria en memoria. Consulte “Cargadores” en la página 202 para obtener más información sobre este plug-in.

- **Plug-in OptimisticCallback**: cuando el atributo LockStrategy para una BackingMap se establece en OPTIMISTIC, la BackingMap o el plug-in Loader deben realizar operaciones de comparación para los valores de la correlación. La BackingMap y el Loader utilizan el plug-in OptimisticCallback para realizar las operaciones de comparación optimista de versiones. Para obtener más información, consulte “Interfaz OptimisticCallback” en la página 225.
- **Plug-in MapIndexPlugin**: un plug-in MapIndexPlugin o Index, de forma abreviada, es una opción que utiliza la BackingMap para crear un índice basado en el atributo especificado del objeto almacenado. El índice permite a la aplicación encontrar objetos por un valor específico o un rango de valores. Existen dos tipos de índice: estático y dinámico. Consulte “Indexación” en la página 239 para obtener más información.

Interfaz Session

En este apartado se describe cómo las aplicaciones empiezan y terminan las transacciones utilizando la interfaz Session. La interfaz Session también proporciona acceso a las interfaces ObjectMap y JavaMap basadas en aplicaciones.

Introducción

Todas las instancias de ObjectMap o JavaMap están unidas a un objeto Session determinado. Todas las hebras que requieren acceso a ObjectGrid primero deben obtener una Session del objeto ObjectGrid. Una instancia de Session no puede compartirse de modo concurrente entre las hebras. ObjectGrid no utiliza ningún almacenamiento local de hebras, pero las restricciones de plataforma pueden limitar la oportunidad de pasar una Session 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; ObjectGrid getObjectGrid();
 TxID getTxID()
 throws NoActiveTransactionException;
```

```
 boolean isWriteThroughEnabled();
 void setTransactionType(String tranType);
 public void processLogSequence(LogSequence logSequence)
 throws NoActiveTransactionException, UndefinedMapException, ObjectGridException;

public 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 objeto ObjectGrid mediante el método ObjectGrid.getSession. El siguiente fragmento de código demuestra cómo obtener 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.

Métodos de transacciones y sesiones

Una Session puede utilizarse para iniciar, comprometer 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 antememoria de transacciones especial hasta que se comprometa la transacción. Cuando se compromete una transacción, los cambios pendientes se aplican a las BackingMaps y los Loaders y se hacen visibles a los demás clientes ObjectGrid.

ObjectGrid también da soporte a la posibilidad de comprometer automáticamente las transacciones, también conocida como compromiso automático. 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 compromete 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 Loader a una BackingMap. El método flush invoca al Loader con el conjunto de cambios actuales de la antememoria de transacciones. El Loader aplica los cambios al programa de fondo. Estos cambios no se comprometen cuando se invoca el vaciado. Si una transacción de Session se compromete después de una invocación de vaciado, sólo las actualizaciones que ocurran después de esa invocación se aplican al

Loader. Si una transacción de Session se retrotrae después de una invocación de vaciado, los cambios vaciados 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 Loader. 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 Loader, sin comprometerlos todavía
// realizar más cambios
...
session.commit();
```

Método NoWriteThrough

Algunas correlaciones de ObjectGrid están respaldadas por un Loader, que proporciona almacenamiento persistente para los datos de la correlación. A veces, es útil comprometer los datos sólo en la correlación de ObjectGrid y no pasar los datos al Loader. La interfaz Session proporciona el método beginNoWriteThough con este fin. El método beginNoWriteThrough inicia una transacción como el método begin. Con el método beginNoWriteThrough, cuando se compromete la transacción, los datos sólo se comprometen en la correlación en memoria de ObjectGrid, y no se comprometen en el almacenamiento persistente proporcionado por el Loader. 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 antememoria cercana, sin modificar la antememoria lejana en el servidor. Si se sabe que los datos están obsoletos en la antememoria cercana, el uso del método beginNoWriteThrough permite invalidar entradas en la antememoria 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 Loader
```

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 “Plug-in TransactionCallback” en la página 218 y “Cargadores” en la página 202 para obtener información adicional sobre la característica de ranuras de los objetos.

Establecimiento del tipo de transacción para el método de supervisión del rendimiento

Si está utilizando ObjectGrid dentro de un servidor de aplicaciones de WebSphere Application Server, es posible que sea 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” en la página 295 para obtener más información sobre el método setTransactionType.

Proceso de un método LogSequence completo

ObjectGrid puede propagar conjuntos de cambios de correlación a otros receptores de ObjectGrid como un mecanismo para distribuir las correlaciones de una máquina virtual Java (JVM) 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. Una Session de ObjectGrid debe está activa antes de que se invoque el método processLogSequence. La aplicación también es responsable de emitir las llamadas de compromiso o retrotracción adecuadas para completar la Session. El proceso de compromiso automático 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, comprometer 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;
}
// comprometer 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 comprometer 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 de retrotracción″. Las siguientes llamadas a este método para una transacción marcada previamente como ″sólo de retrotracción″ se ignorarán. 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 de retrotracción″ para que se pueda comprometer la siguiente transacción iniciada por la sesión.

Método isMarkedRollbackOnly

Devuelve si la sesión está marcada actualmente como ″sólo de 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 boolean true si y sólo si todos los cambios de transacción se desechan en el plug-in Loader como resultado del método flush de la interfaz Session que se está invocando. Este método es muy útil para los plug-ins Loader cuando necesitan 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 comprometen como resultado del método commit de la interfaz Session que se está invocando. Este método es muy útil para los plug-ins Loader cuando necesitan saber por qué se ha invocado el método batchUpdate.

Interfaces ObjectMap y JavaMap

Este tema describe cómo las aplicaciones interactúan con ObjectGrid utilizando las interfaces ObjectMap y JavaMap. Estas dos interfaces se utilizan para la interacción transaccional entre las aplicaciones y las BackingMaps.

Interfaz ObjectMap

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

Obtención de 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.

Transacciones de compromiso automático

Como se ha mencionado previamente, las operaciones efectuadas en BackingMaps que utilizan ObjectMaps y JavaMaps se realizan con más eficacia dentro de una transacción de Session. ObjectGrid proporciona el soporte de compromiso automático 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 comprometen la transacción implícita.

Semántica de los métodos

A continuación, aparece 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” en la página 124. El método setTimeToLive se describe en el tema “Desalojadores” en la página 192. Consulte la documentación de la API para obtener más información sobre estos métodos.

Método containsKey
: Determina si una clave tiene un valor en la BackingMap o el 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 de este método es similar a la del método flush de la interfaz Session. La diferencia importante es que el vaciado 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 ObjectMap se vacían en el Loader.

Método get
: Recoge la entrada de la BackingMap. Si no se encuentra la entrada en la BackingMap pero existe un Loader asociado a la BackingMap, intenta recoger la entrada del Loader. El método getAll se proporciona para permitir el proceso de recogida por lotes.

Método getForUpdate

El método getforUpdate es idéntico al método get, pero la utilización del 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 recogida por lotes.

Método insert

Inserta una entrada en la BackingMap y el Loader. La utilización de este método indica a la BackingMap y al Loader que se desea insertar una entrada no existente previamente. Al invocar este método en una entrada existente, se genera una excepción cuando se invoca al método o se compromete 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 invalidación por lotes.

La invalidación local se especifica cuando el valor *false* se pasa como el parámetro **isGlobal** del método invalidate. La invalidación local descarta cualquier cambio de la entrada de la antememoria de transacciones. Si la aplicación emite un método `get`, se recoge la entrada del último valor comprometido de la BackingMap. Si no existe ninguna entrada en la BackingMap, se recoge la entrada del último valor comprometido o vaciado del Loader. Cuando una transacción está comprometida, cualquier entrada marcada como invalidada localmente no tiene impacto alguno en la BackingMap. Todos los cambios que se hayan vaciado en el Loader están todavía comprometidos incluso si se había invalidado la entrada.

La invalidación global se especifica cuando el valor *true* se pasa como el parámetro **isGlobal** del método invalidate. La invalidación global descarta cualquier cambio pendiente de la entrada de la antememoria de transacciones y omite el valor de BackingMap en operaciones posteriores que se realicen en la entrada. Cuando una transacción está comprometida, cualquier entrada marcada como invalidada globalmente se desaloja de la BackingMap.

Considere el siguiente caso de uso de invalidación como un ejemplo: la BackingMap está respaldada mediante 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 invalidación 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("La columna de versión es: " + 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 vaciado, 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 actualiza el objeto, la aplicación realiza una invalidación global. El siguiente método get que se emite obtiene la entrada del Loader ignorando el valor de la transacción. La entrada se recoge de la base de datos con el valor de la versión actualizada.

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 de la transacción. Una invocación del método put sin una invocación del método get anterior o una invocación del método get anterior que no haya encontrado una entrada se interpreta como una operación insert. La semántica de los métodos insert y update se aplica cuando se compromete la operación put. El método putAll se proporciona para habilitar el proceso de actualización y de inserción por lotes.

Método remove

Elimina la entrada de la BackingMap y el Loader, si existe alguno conectado. Este método devuelve el valor del objeto que se ha eliminado. Si el objeto no existe, este método devuelve un valor nulo. El método removeAll se proporciona para habilitar el proceso de supresión por lotes sin los valores de retorno.

Método setCopyMode

Especifica una CopyMode para esta ObjectMap. Con este método, una aplicación puede alterar temporalmente la CopyMode que se especifica en la BackingMap. La CopyMode especificada está vigente hasta que se invoque al método clearCopyMode. Ambos métodos se invocan fuera de los límites transaccionales. Una CopyMode no se puede modificar durante una transacción.

Método touch

Actualiza la última hora de acceso de 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 una invalidación o eliminación, se genera una excepción durante el proceso de compromiso.

Método update

Actualiza explícitamente una entrada de la BackingMap y el Loader. La utilización de este método indica a la BackingMap y al Loader que se desea actualizar una entrada existente. Se genera una excepción si invoca a este método en una entrada que no existe durante la invocación del método o durante el proceso de compromiso.

Método getIndex

Intenta obtener un índice con nombre que se basa en la BackingMap. El índice no se puede compartir entre las hebras y trabaja con las mismas normas 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.

Interfaz JavaMap

Una instancia de JavaMap se obtiene a partir de un objeto ObjectMap. La interfaz JavaMap tiene las mismas signaturas de método que ObjectMap, pero se diferencian en el manejo de las excepciones. JavaMap amplía la interfaz java.util.Map, de modo que todas las excepciones son instancias de la clase java.lang.RuntimeException. Como JavaMap amplía la interfaz java.util.Map, es fácil utilizar ObjectGrid con rapidez con una aplicación existente que utilice una interfaz java.util.Map para la colocación en antememoria de objetos.

Obtención de 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 muestra 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. Al llamar a getJavaMap en varias ocasiones utilizando una determinada ObjectMap, siempre se devuelve la misma instancia de JavaMap.

Métodos soportados

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

- `containsKey(java.lang.Object)`
- `get(java.lang.Object)`
- `put(java.lang.Object, java.lang.Object)`
- `putAll(java.util.Map)`
- `remove(java.lang.Object)`

Todos los demás métodos heredados de la interfaz java.util.Map generan la excepción `java.lang.UnsupportedOperationException`.

Palabras clave

ObjectGrid proporciona un mecanismo de invalidación flexible basado en las palabras clave. Una *palabra clave* es una instancia no nula de cualquier objeto serializable. Puede asociar palabras clave con entradas de BackingMap del modo que desee.

Asociación de palabras clave con entradas

Un conjunto de entradas se puede asociar con cero o más palabras clave. Los métodos de ObjectMap y JavaMap que manipulan las entradas, incluidos todos los métodos `get`, `update`, `put`, `insert` y `touch`, tienen versiones que permiten que una única palabra clave se asocie con todas las entradas que modifique el método. Las nuevas asociaciones de palabras clave sólo son visibles en la transacción actual hasta que la transacción se comprometa. Después de un compromiso, la nueva asociación se aplica a la BackingMap y es visible para las demás transacciones. Si

se produce un error durante el proceso de compromiso que genere una retrotracción, o si el usuario retrotrae una transacción activa, las nuevas asociaciones de palabras clave se retrotraerán. El siguiente código muestra cómo una nueva entrada se asocia con una palabra clave:

```
Session sess = objectGrid.getSession();
ObjectMap map = sess.getMap("MapA");
sess.begin();
map.insert("Billy", new Person("Joe", "Bloggs", "Manhattan"), "Nueva York");
sess.commit();
```

El código de ejemplo anterior inserta una nueva entrada en la BackingMap y la asocia con la palabra clave ″Nueva York″. Una aplicación que inserta entradas también debe asociar palabras clave cuando se recuperan las entradas. La aplicación debe asociar palabras clave con las entradas cada vez que se obtengan. Consideremos el siguiente código de ejemplo:

```
sess.begin();
Person p = (Person)map.get("Billy", "Nueva York");
sess.commit();
```

El código de ejemplo anterior garantiza que la entrada recuperada se asocie con la palabra clave ″Nueva York″. Una aplicación puede asociar varias palabras clave con una entrada, aunque sólo una palabra clave por invocación de método. Para asociar más palabras clave debe emitir otra invocación de método, como se muestra en el siguiente ejemplo:

```
sess.begin();
Person p = (Person)map.get("Billy", "Nueva York");
map.touch("Billy", "Otra palabra clave");
map.get("Billy", "Otra palabra clave más");
sess.commit();
```

Palabras clave por omisión

El método `setDefaultKeyword` de las interfaces ObjectMap y JavaMap proporcionan un modo para asociar entradas con una palabra clave determinada sin utilizar la versión de la palabra clave de los métodos `get`, `insert`, `put`, `update` o `touch`. Si se utiliza la versión de la palabra clave de un método, se ignora la palabra clave por omisión y se utiliza el objeto de palabra clave proporcionado.

```
sess.begin();
map.setDefaultKeyword("Nueva York");
Person p = (Person)map.get("Billy");
p = (Person)map.get("Bob", "Los Angeles");
map.setDefaultKeyword(null);
p = (Person)map.get("Jimmy");
sess.commit();
```

En el ejemplo anterior `Billy` se asocia con la palabra clave por omisión, ″Nueva York″. `Bob` no se asocia con la palabra clave por omisión porque se ha pasado una palabra explícita a la invocación del método get para recuperar la entrada Bob. Ninguna palabra clave se asocia con ″Jimmy″ porque se ha vuelto a establecer la palabra clave por omisión y no se ha pasado ningún argumento de clave explícita a la invocación del método `get`.

Invalidación de entradas con palabras clave

La utilización del método `invalidateUsingKeyword` de las interfaces ObjectMap y JavaMap invalida todas las entradas asociadas con una palabra clave en la BackingMap correspondiente. Con este enfoque, puede invalidar con eficacia entradas relacionadas en una única operación.

```
sess.begin();
map.insert("Billy", new Person("Joe", "Bloggs", "Manhattan"), "Nueva York");
map.invalidateUsingKeyword("Nueva York", false);
map.insert("Bob", new Person("Paul", "Henry", "Albany"), "Nueva York");
sess.commit();
```

En el ejemplo anterior, la entrada para ″Billy″ se invalida y no se inserta en la BackingMap. La entrada para ″Bob″ no se invalida porque se ha insertado después de la invocación del método `invalidateUsingKeyword`. El método `invalidateUsingKeyword` invalida las entradas en función de las asociaciones de palabras clave al invocarse el método.

Agrupación de palabras clave

Las palabras clave también se pueden agrupar en una relación de padre e hijo. Una palabra clave padre puede tener varios hijos y una palabra clave hijo puede tener varios padres. Por ejemplo, si una aplicación utiliza las palabras clave ″Dublín″, ″París″, ″Nueva York″ y ″Los Angeles″, puede añadir los siguientes grupos de palabras clave:

- ″EEUU″ agrupa ″Nueva York″ y ″Los Angeles″
- ″Europa″ agrupa ″Dublín″ y ″París″
- ″Mundo″ agrupa ″EEUU″ y ″Europa″

Si se invalida la palabra clave ″EEUU″, se invalidan todas las entradas asociadas con las palabras clave ″Nueva York″ y ″Los Angeles″. Si se invalida la palabra clave ″Mundo″, se invalidan todas las entradas asociadas con los grupos ″EEUU″ y ″Europa″. Las asociaciones de palabras clave se definen utilizando el método `associateKeyword` en la interfaz ObjectGrid. La adición de palabras clave hijo a un padre después de una invocación del método `invalidateUsingKeyword` no provoca que se invaliden las entradas asociadas con la palabra clave hijo. El siguiente código de ejemplo define el conjunto de asociaciones de palabras clave que se describen:

```
ObjectGrid objectGrid = ...;
objectGrid.associateKeyword("EEUU", "Nueva York");
objectGrid.associateKeyword("EEUU", "Los Angeles");
objectGrid.associateKeyword("Europa", "Dublín");
objectGrid.associateKeyword("Europa", "París");
objectGrid.associateKeyword("Mundo", "EEUU");
objectGrid.associateKeyword("Mundo", "Europa");
```

Objetos LogElement y LogSequence

Cuando una aplicación está realizando cambios en una Map durante una transacción, un objeto LogSequence hace un seguimiento de estos cambios. Si la aplicación cambia una entrada de la correlación, existe el correspondiente LogElement para proporcionar información detallada del cambio. Los Loaders reciben un objeto LogSequence determinado para una correlación específica siempre y cuando una aplicación solicite vaciar o comprometer la transacción. El Loader se itera en los LogElements contenidos en la LogSequence y aplica todos los LogElements al programa de fondo.

Los ObjectGridEventListeners registrados con una ObjectGrid anterior también utilizan objetos LogSequence. Estos receptores reciben un objeto LogSequence para todas las correlaciones de una transacción comprometida. Las aplicaciones pueden utilizar estos receptores para detectar el cambio de ciertas entradas como el desencadenante de una base de datos convencional.

En este tema se describen cuatro interfaces o clases relacionadas con las anotaciones cronológicas y proporcionadas por la infraestructura de ObjectGrid:

- `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 en una entrada durante una transacción. Un objeto LogElement tiene los siguientes atributos. Los atributos más utilizados son los atributos *tipo* y *valor actual*:

Atributo *tipo*
: Un elemento de anotaciones cronológicas *tipo* indica el tipo de operación que este elemento representa. El *tipo* puede ser cualquiera de las siguientes constantes definidas en la interfaz `LogElement`: `INSERT`, `UPDATE`, `DELETE`, `EVICT`, `FETCH`, o `TOUCH`.

Atributo *deshacer tipo*
: Devuelve qué operación debe realizarse para ″deshacer″ un cambio anterior que la transacción ha realizado en la entrada de correlación.

Atributo *valor actual*
: El *valor actual* representa el nuevo valor para la operación `INSERT`, `UPDATE` o`FETCH`. Si la operación es `TOUCH`, `DELETE` o `EVICT`, el valor actual es nulo. Este valor puede convertirse a ValueProxyInfo cuando se utilice una ValueInterface.

Atributo *CacheEntry*
: Puede obtener una referencia al objeto CacheEntry del LogElement y utilizar los métodos definidos en este objeto para recuperar la información necesaria.

Atributo *estado pendiente*
: Si el *estado pendiente* es `true`, el cambio representado por este elemento de anotación cronológica no se ha aplicado todavía al cargador. Si es `false`, el cambio se ha aplicado al cargador probablemente por la operación de vaciado.

Atributo *valor con versión*
: El valor con versión es un valor que puede utilizarse para crear versiones.

Atributo *nuevas palabras clave*
: La nueva colección de palabras clave contiene cualquier palabra clave nueva que se haya asociado con esta entrada.

Atributo *última hora de acceso*
: Representa la última hora de acceso para la entrada.

Atributos *imagen previa / imagen posterior*
: Hay disponibles métodos getter para obtener la imagen del objeto de valor antes o después de aplicar los cambios a la correlación.

Interfaz LogSequence

En la mayoría de las transacciones, ocurren operaciones en más de una entrada de una correlación, de modo que se crean varios objetos LogElement. Es recomendable tener un objeto que actúe como una combinación de varios objetos

LogElement. La interfaz LogSequence cumple este objetivo mediante una lista de los objetos LogElement. La interfaz LogSequence tiene los siguientes métodos:

Método size
Devuelve el número de objetos LogElement en la secuencia específica.

Método getAllChanges
Devuelve un iterador de todos los cambios en la secuencia de anotaciones cronológicas especificada.

Método getPendingChanges
Devuelve un iterador de todos los cambios pendientes. Es muy posible que lo utilice un cargador para aplicar sólo los cambios pendientes al almacén persistente.

Método getChangesByKeys
Devuelve un iterador de los objetos LogElement que tienen la clave de destino en función del parámetro de entrada.

Método getChangesByTypes
Devuelve un iterador de los objetos LogElement que son del tipo de LogElement especificado.

Método getMapName
Devuelve el nombre de la correlación de respaldo a las que se aplican los cambios. El emisor puede utilizar este nombre como entrada al método `Session.getMap(string)`.

Método isDirty
Devuelve si este LogSequence tiene algún LogElement que *ensucie* una Map. Es decir, si la LogSequence contiene cualquier objeto LogElement que sea de un tipo distinto de Fetch o Get, la LogSequence se considera ″sucia″.

Método isRollback
Devuelve si esta LogSequence se ha generado para retrotraer una transacción.

Método getObjectGridName
Devuelve el nombre de la ObjectGrid que aloja la correlación para la que se aplican estos cambios.

LogElement y LogSequence se utilizan ampliamente en ObjectGrid, y también los utilizan los plug-in ObjectGrid que escriben los usuarios cuando se propagan las operaciones de un componente o servidor a otro servidor o componente. Por ejemplo, la función de propagación distribuida de transacciones de ObjectGrid puede utilizar un objeto LogSequence para propagar los cambios a los demás servidores o puede aplicarse al almacén persistente mediante el cargador. Las siguientes interfaces son las que utilizan LogSequence principalmente.

- `com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener`
- `com.ibm.websphere.objectgrid.plugins.Loader`
- `com.ibm.websphere.objectgrid.plugins.Evictor`
- `com.ibm.websphere.objectgrid.Session`

Para obtener más información detallada sobre estas interfaces, consulte la documentación de la API.

Ejemplo de Loader

Este apartado muestra cómo se utilizan los objetos LogSequence y LogElement en un Loader. Un *Loader* se utiliza para cargar y conservar datos en un almacenamiento persistente. El método `batchUpdate` de la interfaz Loader utiliza LogSequence:

```
void batchUpdate(TxID txid, LogSequence sequence)
throws LoaderException, OptimisticCollisionException;
```

Se llama al método `batchUpdate` siempre que una ObjectGrid necesite aplicar todos los cambios actuales al Loader. El Loader recibe una lista de objetos LogElement para la correlación encapsulada en un objeto LogSequence. La implementación del método `batchUpdate` debe iterarse en los cambios y aplicarlos al programa de fondo. El siguiente fragmento de código muestra cómo un Loader utiliza un objeto LogSequence. El fragmento se itera en el conjunto de cambios y construye tres sentencias JDBC (Java Database Connectivity) por lotes: una con inserciones, otra con actualizaciones y la última con supresiones:

```
public void batchUpdate(TxID tx, LogSequence sequence)
throws LoaderException
{
 // Obtener una conexión SQL para utilizarla.
 Connection conn = getConnection(tx);
try
{
  // Procesar la lista de cambios y crear un conjunto de sentencias
  // preparadas para ejecutar operaciones SQL de actualización, inserción
// o supresión por lotes. Estas sentencias se almacenan en antememoria 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;
}
}
 // Ejecutar las sentencias 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, aunque no se ilustran los detalles de cómo se construye una sentencia `insert`, `update` o `delete` de SQL. Este ejemplo ilustra cómo el método `getPendingChanges` se llama en el argumento LogSequence para obtener un iterador de los objetos LogElement que un Loader necesita procesar y cómo el método `LogElement.getType().getCode()` se utiliza para determinar si un LogElement es para una operación `insert`, `update` o `delete` de SQL.

Ejemplo de desalojador

Este ejemplo describe cómo LogSequence y LogElement se utilizan en un desalojador. Un Evictor se utiliza para desalojar las entradas de correlación de respaldo en función de varios criterios. El método apply de la interfaz Evictor utiliza LogSequence:

```
/**
* Se llama durante el compromiso de antememoria para permitir
* que el desalojador haga un seguimiento del uso de objetos
* de una correlación de respaldo. También dará soporte a las
* entradas que se hayan desalojado satisfactoriamente.
*
* @param sequence LogSequence de cambios de la correlación
*/
void apply(LogSequence sequence);
```

Para obtener información sobre cómo el método apply utiliza LogSequence, consulte el ejemplo de código que aparece en el tema “Desalojadores” en la página 192.

Interfaces LogSequenceFilter y LogSequenceTransformer

En ocasiones, es necesario filtrar los objetos LogElement de modo que sólo se acepten los objetos LogElement con ciertos criterios y no con otros. Por ejemplo, es posible que desee serializar un cierto LogElement en función de un determinado criterio. LogSequenceFilter soluciona este problema con el siguiente método:

```
public boolean accept (LogElement logElement);
```

Este método devuelve el valor true si el LogElement específico debe utilizarse en la operación y devuelve el valor false si no debe utilizarse el LogElement específico.

LogSequenceTransformer es una clase que utiliza la función LogSequenceFilter descrita anteriormente. Ésta utiliza el LogSequenceFilter para filtrar algunos objetos LogElement y, a continuación, serializa los objetos LogElement aceptados. Esta clase tiene dos métodos. El primer método aparece a continuación:

```
public static void serialize(Collection logSequences, ObjectOutputStream stream,
LogSequenceFilter filter, DistributionMode mode)
throws IOException
```

Este método permite que el emisor proporcione un filtro para determinar qué LogElements deben incluirse en el proceso de serialización. El parámetro **DistributionMode** permite al emisor controlar el proceso de serialización. Por ejemplo, si la modalidad de distribución es sólo la invalidación, no es necesario serializar el valor. El segundo método de esta clase aparece a continuación:

```
public static Collection inflate(ObjectInputStream stream, ObjectGrid objectGrid)
throws IOException, ClassNotFoundException.
```

Este método lee el formato serializado de la secuencia de anotaciones cronológicas, creado por el método `serialize`, en la corriente de entrada de objetos proporcionada.

Bloqueo

Este tema describe la estrategia de bloqueo a la que da soporte una BackingMap de ObjectGrid.

Todas las BackingMaps pueden configurarse para utilizar una de las siguientes estrategias de bloqueo:

- Bloqueo pesimista
- Bloqueo optimista
- Ninguno

A continuación, aparece un ejemplo sobre cómo la estrategia de bloqueo puede establecerse en las BackingMaps `map1`, `map2`, y `map3`, donde cada correlación utiliza una estrategia de correlación distinta:

```
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 = og.defineMap("map2");
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

Para evitar una excepción `java.lang.IllegalStateException`, se debe llamar al método `setLockStrategy` antes de llamar a los métodos `initialize` o `getSession` de la instancia de ObjectGrid.

Cuando se utiliza la estrategia de bloqueo `PESSIMISTIC` o `OPTIMISTIC`, se crea un gestor de bloqueos para la BackingMap. El gestor de bloqueos utiliza una correlación hash para realizar un seguimiento de las entradas bloqueadas por una o más transacciones. Si existen numerosas entradas de correlación en la correlación hash, cuantas más cubetas de bloqueos haya, mejor es el rendimiento. Cuanto mayor es el número de cubetas, menor es el riesgo de colisiones de sincronización Java. Cuanto más elevado sea el número de cubetas de bloqueo, mayor será la concurrencia. El siguiente ejemplo muestra cómo una aplicación puede establecer el número de cubetas de bloqueo que se vayan a utilizar para una determinada BackingMap:

```
bm.setNumberOfLockBuckets( 503 );
```

De nuevo, para evitar una excepción `java.lang.IllegalStateException`, se debe llamar al método `setNumberOfLockBuckets` antes de llamar a los métodos `initialize` o `getSession` de la instancia de ObjectGrid. El parámetro del método `setNumberOfLockBuckets` es un número entero primitivo Java que especifica el número de cubetas de bloqueo que se van a utilizar. La utilización de un número primo garantiza una distribución uniforme de las entradas de correlación entre las cubetas de bloqueos. Un buen punto de inicio para un mayor rendimiento es establecer el número de cubetas de bloqueo en aproximadamente el diez por ciento del número previsto de entradas de BackingMap.

Bloqueo pesimista

Utilice la estrategia de bloqueo pesimista para las correlaciones de lectura y grabación cuando las demás estrategias de bloqueo no sean posibles.

Al configurar una Map de ObjectGrid para utilizar la estrategia de bloqueo PESSIMISTIC, se obtiene un bloqueo de transacciones pesimista para una entrada de correlación cuando una transacción obtenga por primera vez la entrada de la BackingMap. El bloqueo pesimista se mantiene hasta que la aplicación completa la transacción. Generalmente, la estrategia de bloqueo pesimista se utiliza en las siguientes situaciones:

- La BackingMap está configurada con o un cargador o sin él y la información de mantenimiento de versiones no está disponible.
- La BackingMap está siendo utilizada directamente por una aplicación que necesita ayuda de la ObjectGrid para el control de concurrencia.
- La información de mantenimiento de versiones está disponible, pero las transacciones de actualización colisionan con frecuencia en las entradas de respaldo y dan como resultado errores de actualización optimistas.

Como la estrategia de bloqueo pesimista es la que tiene el mayor impacto sobre el rendimiento y la escalabilidad, sólo debería utilizarse para las correlaciones de lectura y grabación cuando las demás estrategias de bloqueo no sean viables. Por ejemplo, las anomalías de actualización optimistas ocurren frecuentemente, o una aplicación maneja con dificultad la recuperación de una anomalía optimista.

Métodos ObjectMap y modalidades de bloqueo

Cuando una aplicación utiliza los métodos de la interfaz ObjectMap, ObjectGrid automáticamente intenta un bloqueo pesimista para la entrada de correlación a la que se acceda. ObjectGrid utiliza las siguientes modalidades de bloqueo en función de los métodos a los que llame la aplicación en la interfaz ObjectMap:

- Los métodos `get` y `getAll` adquieren un *bloqueo S* o una modalidad de bloqueo compartida, para la clave de una entrada de correlación. El bloqueo S se mantiene hasta que se completa la transacción. La modalidad de bloqueo S permite la concurrencia entre transacciones que intenten adquirir una modalidad de bloqueo S o de bloqueo actualizable (bloqueo U) para la misma clave, pero bloquea las demás transacciones que intenten obtener un 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 complete la transacción. La modalidad de bloqueo U permite la concurrencia entre transacciones que adquieran una modalidad de bloqueo S para la misma clave, pero bloquea las demás transacciones que intenten adquirir una modalidad de bloqueo U o bloqueo 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 determinado valor de clave. Un bloqueo X bloquea todas las demás transacciones que intenten adquirir una modalidad de bloqueo S, U o X para la misma clave.
- Los métodos `global invalidate` y `global invalidateAll` adquieren un bloqueo X para cada entrada de correlación que se invalide. El bloqueo X se mantiene hasta que se completa la transacción. No se adquiere ningún bloqueo para los

métodos `local invalidate` y `local invalidateAll` porque las llamadas del método invalidate local no invalidan ninguna de las entradas de BackingMap.

A raíz de las definiciones anteriores, es obvio que una modalidad de bloqueo S es menos potente que una de bloqueo U ya que permite más transacciones que se ejecuten de forma concurrente al acceder a la misma entrada de correlación. La modalidad de bloqueo U es ligeramente más potente que la modalidad de bloqueo S porque bloquea las demás transacciones que soliciten una modalidad de bloqueo U o X . La modalidad de bloqueo S sólo bloquea las demás transacciones que soliciten una modalidad de bloqueo X. Esta pequeña diferencia es importante para evitar que ocurran ciertos puntos muertos. La modalidad de bloqueo X es la modalidad de bloqueo más potente porque bloquea todas las demás transacciones que intenten obtener una modalidad de bloqueo S, U o X para la misma entrada de correlación. El efecto neto de una modalidad de bloqueo X es garantizar que sólo una transacción pueda insertar, actualizar o eliminar una entrada de correlación y evitar que se pierdan actualizaciones cuando más de una transacción intente actualizar la misma entrada de correlación.

La siguiente tabla es una matriz de compatibilidad de modalidades de bloqueo que resume las modalidades de bloqueo descritas y se utiliza para determinar qué modalidades de bloqueo son compatibles entre sí. Para leer esta matriz, las filas de la matriz indican una modalidad de bloqueo que ya se ha otorgado. Las columnas indican la modalidad de bloqueo solicitada por otra transacción. Si aparece **Sí** en la columna, se ha otorgado la modalidad de bloqueo solicitada por las demás transacciones porque es compatible con la modalidad de bloqueo ya otorgada. **No** indica que la modalidad de bloqueo no es compatible y que las demás transacciones deben esperar a que la primera transacción libere el bloqueo del que es propietaria.

Tabla 8. Compatibilidad y potencia de las modalidades de bloqueo.

bloqueo	**bloqueos compatibles**			**potencia**
	S (compartido)	U (actualizable)	X (exclusivo)	
S (Compartido)	Sí	Sí	No	menos potente
U (Actualizable)	Sí	No	No	normal
X (Exclusivo)	No	No	No	más potente

Tiempo de espera de bloqueo

Todas las BackingMap de ObjectGrid tienen un valor de tiempo de espera de bloqueo por omisión. El valor de tiempo de espera se utiliza para garantizar que una aplicación no espere indefinidamente a que se le otorgue una modalidad de bloqueo debido a una condición de punto muerto ocurrida a causa de un error de la aplicación. La aplicación puede utilizar la interfaz BackingMap para alterar temporalmente el valor de tiempo de espera de bloqueo por omisión. El siguiente ejemplo 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 a los métodos setLockStrategy y setLockTimeout antes de llamar a los métodos initialize o getSession en la instancia de ObjectGrid. El parámetro del método setLockTimeout es un número entero primitivo Java que especifica el número de segundos que ObjectGrid espera para que se otorgue una modalidad de bloqueo. Si una transacción espera más tiempo que el valor de tiempo de espera de bloqueo configurado para la BackingMap, se genera una excepción `com.ibm.websphere.objectgrid.LockTimeoutException`.

Cuando se genera una excepción LockTimeoutException, la aplicación debe determinar si el tiempo de espera ocurre porque la aplicación se está ejecutando más lentamente de lo habitual o porque se ha producido una condición de punto muerto. Si se ha producido una condición de punto muerto, el incremento del valor de tiempo de espera de bloqueo no elimina la excepción. El incremento del tiempo de espera hace que la excepción tarde más en generarse. No obstante, si, al aumentarse el valor de tiempo de espera de bloqueo, se elimina la excepción, el problema se ha producido porque la aplicación se estaba ejecutando más lentamente de lo habitual. La aplicación en este caso debe determinar la causa por la que el rendimiento sea tan lento. Consulte Capítulo 14, “Resolución de problemas”, en la página 349 y Capítulo 11, “Procedimientos recomendados para el rendimiento de ObjectGrid”, en la página 329 para obtener más información.

Puntos muertos

Considere la siguiente secuencia de peticiones de la modalidad de bloqueo:

```
El bloqueo X se otorga a la transacción 1 para la clave1.
El bloqueo X se otorga a la transacción 2 para la clave2.
Bloqueo X solicitado por la transacción 1 para la clave2.
(La transacción 1 bloquea a la espera del bloqueo propiedad de la transacción 2).
Bloqueo X solicitado por la transacción 2 para la clave1.
(La transacción 2 bloquea a la espera del bloqueo propiedad de la transacción 1).
```

La secuencia anterior es el clásico ejemplo de punto muerto de dos transacciones que intentan adquirir más de un único bloqueo y cada transacción adquiere los bloqueos en un orden distinto. Para evitar este punto muerto, cada transacción debe obtener los distintos 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 compromiso. Durante el ciclo de compromiso, la ObjectGrid determina las claves para las entradas de correlación que necesiten bloquearse y solicita las modalidades de bloqueo en la secuencia de las claves. Con este método, ObjectGrid evita la mayor parte de los puntos muertos clásicos. Sin embargo, ObjectGrid no previene ni puede prevenir todos los casos de ejemplo de puntos muertos posibles. Existe un par de casos de ejemplo que la aplicación debe considerar. A continuación, aparecen los casos de ejemplo de los que la aplicación debe tener conocimiento y tomar medidas preventivas frente a ellos.

Existe un caso de ejemplo donde ObjectGrid puede detectar un punto muerto sin necesidad de esperar a que ocurra un tiempo de espera de bloqueo. Si ocurre este caso de ejemplo, se genera una excepción `com.ibm.websphere.objectgrid.LockDeadlockException`. Considere el siguiente fragmento de código.

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Billy");
// Billy ha cumplido años, así que tiene un año más.
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
sess.commit();
```

En esta situación, la mujer de Billy desea que sea mayor de los que es, y ambos ejecutan la transacción de forma concurrente. En esta situación, ambas transacciones tienen una modalidad de bloqueo S en la entrada **Billy** de la correlación PERSON como resultado de la invocación del método person.get("Billy"). Como resultado de la llamada de método person.put ("Billy", p), ambas transacciones intentan actualizar la modalidad de bloqueo S a la de bloqueo X. Ambas transacciones bloquean la espera a que la otra transacción libere la modalidad de bloqueo S que poseen. Como consecuencia, se produce un punto muerto ya que existe una condición de espera circular entre las dos transacciones. Se genera una condición de espera circular cuando más de una transacción intenta aumentar un bloqueo desde un bloqueo a otro más potente para la misma entrada de correlación. En este caso de ejemplo, ObjectGrid genera una excepción `LockDeadlockException` en lugar de una excepción `LockTimeoutException`. Para obtener más información, consulte "LockDeadlockException" en la página 353.

La aplicación puede evitar la excepción `LockDeadlockException` en el ejemplo anterior utilizando la estrategia de bloqueo OPTIMISTIC en lugar de la estrategia de bloqueo PESSIMISTIC. La utilización de la estrategia de bloqueo OPTIMISTIC es la solución preferida cuando la correlación se lee principalmente y las actualizaciones de la correlación no son frecuentes. Consulte "Bloqueo optimista" en la página 137 para obtener más detalles sobre la estrategia optimista. Si se debe utilizar la estrategia de bloqueo PESSIMISTIC, se puede utilizar el método getForUpdate en lugar del método get en el ejemplo anterior. En este caso, la primera transacción que llame al método getForUpdate adquiere una modalidad de bloqueo U en lugar de una modalidad de bloqueo S. Esta modalidad de bloqueo provoca que la segunda transacción se bloquee cuando llame al método getForUpdate ya que sólo se otorga una modalidad de bloqueo U a una transacción. Como la segunda transacción está bloqueada, no tiene ninguna modalidad de bloqueo en la entrada de correlación Billy. La primera transacción no se bloquea cuando intenta actualizar una modalidad de bloqueo U a una de bloqueo X como resultado de la llamada de método put de la primera transacción. Esta característica muestra por qué la modalidad de bloqueo U se llama modalidad de bloqueo ″actualizable″. Cuando se completa la primera transacción, se desbloquea la segunda transacción y se concede la modalidad de bloqueo U. Una aplicación puede evitar el caso de ejemplo de punto muerto de aumento de bloqueo utilizando el método getForUpdate en lugar del método get cuando se esté utilizando la estrategia de bloqueo PESSIMISTIC.

Importante: Esta solución no evita 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 o remove. La concurrencia es tan alta como cuando se utiliza el método get habitual. La única reducción de la concurrencia ocurre cuando más de una transacción para la misma entrada de correlación llama al método getForUpdate.

Debe tener cuidado cuando una transacción llame al método getForUpdate en más de una entrada de correlación para garantizar que todas las transacciones

adquieran los bloqueos U en el mismo orden. Por ejemplo, suponga que la primera transacción llama al método getForUpdate para la clave 1 y el método getForUpdate para la clave 2. Otra transacción concurrente llama al método getForUpdate para las mismas claves, pero en sentido contrario. Esta secuencia provoca el punto muerto clásico ya que distintas transacciones obtienen varios bloqueos en distinto orden. La aplicación todavía debe garantizar que todas las transacciones accedan a varias entradas de correlación en la secuencia de las claves para evitar que se produzca el punto muerto. Como el bloqueo U se obtiene cuando se llama al método getForUpdate y no en el momento de compromiso, ObjectGrid no puede ordenar las solicitudes de bloqueo del mismo modo que lo hace durante el ciclo de compromiso. La aplicación debe controlar el orden de bloqueos en este caso.

La utilización del método flush en la interfaz ObjectMap antes de un compromiso puede dar lugar a consideraciones adicionales sobre el orden de los bloqueos. El método flush generalmente se utiliza para forzar que los cambios realizados en la correlación se pasen al programa de fondo mediante el plug-in Loader. Es esta situación, el programa de fondo utiliza su propio gestor de bloqueos para controlar la concurrencia, de modo que la condición de espera de bloqueo y el punto muerto puedan ocurrir en el programa de fondo en lugar de en el gestor de bloqueos de ObjectGrid. Consideremos la siguiente transacción:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
sess.begin();
 activeTran = true;
Person p = (IPerson)person.get("Billy");
p.setAge( p.getAge() + 1 );
person.put( "Billy", 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 haya actualizado a la persona Tom, llamado al método flush y, a continuación, actualizado a la persona Billy. Si ocurre esta situación, el siguiente intercalado de las dos transacciones genera una condición de punto muerto de la base de datos:

```
El
bloqueo X se otorga a la transacción 1 para "Billy" cuando se ejecuta el vaciado.
El bloqueo X se otorga a la transacción 2 para "Tom" cuando se ejecuta el vaciado.
Bloqueo X solicitado por la transacción 1 para "Tom" durante el proceso de
compromiso.
(La transacción 1 bloquea a la espera del bloqueo propiedad de la transacción 2).
Bloqueo X solicitado por la transacción 2 para "Billy" durante el proceso de
compromiso.
(La transacción 2 bloquea a la espera del bloqueo propiedad de la transacción 1).
```

Este ejemplo demuestra que la utilización del método flush puede provocar que ocurra un punto muerto en la base de datos en lugar de en ObjectGrid. Este ejemplo de punto muerto puede ocurrir independientemente de la estrategia de

bloqueo que se utilice. La aplicación debe evitar que ocurra este tipo de bloqueo al utilizar el método flush y cuando un Loader esté conectado a la BackingMap. El ejemplo anterior también ilustra otra razón por la que ObjectGrid tiene el mecanismo de tiempo de espera de bloqueo. Es posible que una transacción que espere un bloqueo de la base de datos esté esperando mientras tenga un bloqueo de entradas de correlación de ObjectGrid. Por consiguiente, los problemas que ocurran en la base datos pueden ocasionar demasiados tiempos de espera para una modalidad de bloqueo de ObjectGrid y dar como resultado que se genere una excepción `LockTimeoutException`.

Gestión de las excepciones

No existe ninguna gestión de excepciones para los ejemplos de este tema. Para evitar que se mantengan los bloqueos durante periodos de tiempo excesivos cuando se genera una excepción `LockTimeoutException` o `LockDeadlockException`, una aplicación debe asegurarse de que coloca en antememoria las excepciones imprevistas y llama al método rollback cuando algo imprevisto suceda. Modifique el siguiente fragmento de código tal como se demuestra en el siguiente ejemplo:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
sess.begin();
 activeTran = true;
Person p = (IPerson)person.get("Billy");
// Billy ha cumplido años, así que tiene un año más.
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
sess.commit();
 activeTran = false;
}
finally
{
 if ( activeTran ) sess.rollback();
}
```

El bloque `finally` del fragmento de código garantiza que una transacción se retrotraiga cuando ocurra una excepción inesperada. No sólo maneja una excepción `LockDeadlockException`, sino también cualquier excepción imprevista que pueda ocurrir. El bloque `finally` maneja el caso en el que ocurra una excepción durante una invocación del método commit. Este ejemplo no es el único modo de gestionar las excepciones imprevistas, y es posible que haya casos en los que una aplicación desee capturar alguna de las excepciones imprevistas que puedan ocurrir y muestre una de sus excepciones de aplicación. Puede añadir bloques de captura si es necesario, pero la aplicación debe asegurarse de que el código de bloqueo no existe sin que se complete la transacción.

Bloqueo optimista

La estrategia de bloqueo optimista asume que dos transacciones nunca intentarán actualizar la misma entrada de correlación mientras se estén ejecutando de forma concurrente. Debido a esta presuposición, no es necesario mantener una modalidad de bloqueo durante la duración de la transacción porque es poco probable que más de una transacción actualice la entrada de correlación de forma concurrente.

La estrategia de bloqueo optimista se utiliza generalmente cuando:

- Una BackingMap está configurada con un cargador o sin él y la información de mantenimiento de versiones está disponible.
- Una BackingMap se lee principalmente. Es decir, las transacciones con frecuencia leen las entradas de transacción y sólo ocasionalmente insertan, actualizan o eliminan una entrada de correlación.
- Una BackingMap se inserta, actualiza o elimina con mayor frecuencia que se lee, pero las transacciones rara vez colisionan en la misma entrada de correlación.

Como con la estrategia de bloqueo pesimista, los métodos de la interfaz ObjectMap determinan cómo ObjectGrid intenta adquirir automáticamente una modalidad bloqueo para la entrada de correlación a la que se acceda. No obstante, a continuación, se ofrecen algunas diferencias importantes entre las estrategias pesimista y optimista:

- Como en la estrategia de bloqueo pesimista, los métodos `get` y `getAll` adquieren una *modalidad de bloqueo S* cuando se invoca el método. No obstante, con el bloqueo optimista, la modalidad de bloqueo S no se mantiene hasta que se complete la transacción. Por el contrario, la modalidad de bloqueo S se libera antes de que el método vuelva a la aplicación. El propósito de adquirir la modalidad de bloqueo es para que la ObjectGrid pueda garantizar que sólo los datos comprometidos de otras transacciones sean visibles para la transacción actual. Después de que ObjectGrid haya verificado que los datos se han comprometido, se libera la modalidad de bloqueo S. Durante el tiempo de compromiso, se realiza una comprobación optimista del mantenimiento de versiones para garantizar que ninguna otra transacción haya cambiado la entrada de correlación después de que la transacción actual haya liberado la modalidad de bloqueo S. Si una entrada no se recoge de la correlación antes de que se actualice, invalide o suprima, el tiempo de ejecución de ObjectGrid recoge implícitamente la entrada de la correlación. Esta operación de obtención implícita se realiza para obtener el valor actual en el momento en el que se solicitó la entrada para modificarla.
- A diferencia de la estrategia de bloqueo pesimista, los métodos `getForUpdate` y `getAllForUpdate` se manejan exactamente como 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 éste vuelva a la aplicación.
- Todos los demás métodos ObjectMap se manejan del mismo modo que para la estrategia de bloqueo pesimista. Es decir, cuando se invoca al método commit, se obtiene una modalidad de bloqueo X para cualquier entrada de correlación que se inserte, se actualice, se toque o se invalide, y esta modalidad se mantiene hasta que la transacción complete el proceso de compromiso.

Esta estrategia de compromiso se denomina optimista porque existe un enfoque optimista. En la estrategia de bloqueo optimista se asume que dos transacciones nunca intentarán actualizar la misma entrada de correlación mientras se estén ejecutando de forma concurrente. Debido a esta presuposición, no es necesario mantener una modalidad de bloqueo durante la duración de la transacción porque es poco probable que más de una transacción actualice la entrada de correlación de forma concurrente. No obstante, como no se ha mantenido una modalidad de bloqueo, otra transacción concurrente puede actualizar potencialmente la entrada de correlación después de que la transacción actual haya liberado la modalidad de bloqueo S. Para hacer frente a esta posibilidad, ObjectGrid obtiene un bloqueo X durante el tiempo de compromiso y realiza una comprobación de mantenimiento de versiones optimista para verificar que ninguna otra transacción haya modificado la entrada de correlación desde que la transacción actual haya leído la entrada de

correlación de la BackingMap. Si otra transacción modifica la entrada de correlación, se produce un error en la comprobación de versiones y se genera la excepción `OptimisticCollisionException`. La excepción fuerza a la transacción actual a que se retrotraiga y la aplicación debe volver a intentar la transacción completa. La estrategia de bloqueo optimista es muy útil cuando una correlación se lee principalmente y es poco probable que ocurran actualizaciones de la misma entrada de correlación.

Estrategia de ningún bloqueo de BackingMap

Cuando se configura una BackingMap para que utilice una estrategia de bloqueo de NONE, no se obtiene ningún bloqueo de transacciones para una entrada de correlación. Un caso de ejemplo, donde es útil esta estrategia es cuando una aplicación es un gestor de persistencia como, por ejemplo, un contenedor de EJB (Enterprise JavaBeans) de J2EE (Java 2 Platform, Enterprise Edition) o utiliza Hibernate para obtener los datos persistentes. En este caso de ejemplo, la BackingMap se configura sin un cargador y se utiliza como antememoria de datos por el gestor de persistencia. El gestor de persistencia de este caso de ejemplo proporciona el control de la concurrencia entre las transacciones que accedan a las mismas entradas de la Map de ObjectGrid. La ObjectGrid no necesita obtener ningún bloqueo de transacciones para controlar la concurrencia. Con esto se asume que el gestor de persistencia no libera los bloqueos de transacciones antes de actualizar la correlación de ObjectGrid con los cambios comprometidos. Si ese no es el caso, debe utilizarse una estrategia de bloqueo PESSIMISTIC o OPTIMISTIC. Por ejemplo, suponga que el gestor de persistencia de un contenedor de EJB esté actualizando la correlación de ObjectGrid con datos comprometidos en la transacción gestionada pro el contenedor de EJB. Si la actualización de la correlación de ObjectGrid ocurre antes de que se liberen los bloqueos de transacciones del gestor de persistencia, puede utilizarse la estrategia de bloqueo NONE. Si la actualización de la correlación de ObjectGrid ocurre después de que se liberen los bloqueos de transacciones del gestor de persistencia, será necesaria una de las estrategias de bloqueo OPTIMISTIC o PESSIMISTIC.

Otro caso de ejemplo donde puede utilizarse la estrategia de bloqueo NONE es cuando la aplicación utiliza una BackingMap directamente y se configura un Loader para la correlación. En este caso de ejemplo, el cargador utiliza el soporte de control de concurrencia proporcionado por un sistema de gestión de bases de datos relacionales (RDBMS) utilizando JDBC (Java Database Connectivity) o Hibernate para acceder a los datos de una base de datos relacional. La implementación del cargador puede utilizar indistintamente un enfoque optimista o pesimista.

Un cargador que utiliza un enfoque optimista de bloqueo o mantenimiento de versiones ayuda a obtener el mayor grado de concurrencia o rendimiento. Para obtener más información sobre cómo implementar un enfoque de bloqueo optimista, consulte el apartado “OptimisticCallback” en la página 211 del tema “Consideraciones acerca del cargador” en la página 208.

Es posible que un cargador que utilice el soporte de bloqueo pesimista del programa de fondo subyacente desee utilizar el parámetro **forUpdate** que se pase al método get de la interfaz Loader. Este parámetro se establece en *true* si la aplicación ha utilizado el método `getForUpdate` de la interfaz ObjectMap para obtener los datos. El cargador utiliza este parámetro para determinar si va a solicitar un bloqueo actualizable en la fila que se esté leyendo. Por ejemplo, DB2 obtiene un bloqueo actualizable cuando una cláusula select de SQL contiene una

cláusula `for update`. Este enfoque ofrece la misma prevención frente a puntos muertos que se describe en el tema Bloqueo pesimista.

Seguridad de ObjectGrid

Utilice los mecanismos de seguridad de ObjectGrid para proteger el acceso de datos de correlación y las tareas de gestión mediante la programación o la configuración.

ObjectGrid proporciona mecanismos de seguridad para proteger el acceso a los datos de correlación y las tareas de gestión. La seguridad de ObjectGrid se basa en el mecanismo JAAS (Java Authentication and Authorization Services). JAAS es una parte integral de la seguridad de Java 2.

En este apartado se describen los mecanismos de seguridad de ObjectGrid y cómo utilizar las API de seguridad de ObjectGrid.

- "Visión general de seguridad de ObjectGrid" proporciona una visión general de la seguridad de ObjectGrid.
- "Seguridad de cliente-servidor" en la página 145 describe la seguridad de cliente-servidor para el modelo de programación de ObjectGrid distribuido.
- "Seguridad de ObjectGrid local" en la página 164 describe la seguridad de ObjectGrid local.
- "Autorización" en la página 171 describe el mecanismo de autorización y los plug-ins relacionados que se aplican al modelo de programación de ObjectGrid local y distribuido.
- "Seguridad del clúster ObjectGrid" en la página 180 describe el mecanismo de seguridad del clúster ObjectGrid y los plug-ins relacionados.
- "Seguridad de pasarela" en la página 183 describe la seguridad de pasarela.
- "Integración de la seguridad con WebSphere Application Server" en la página 186 resalta la integración con WebSphere Application Server.

La mayoría del código de ejemplo que se muestra en este apartado proviene de los ejemplos proporcionados de ObjectGrid. Puede encontrar una visión general de seguridad de ejemplo en el Capítulo 5, "Ejemplos de ObjectGrid", en la página 61.

Visión general de seguridad de ObjectGrid

ObjectGrid es un sistema de almacenamiento en antememoria distribuido. El acceso a los datos en antememoria se puede proteger. Generalmente, la seguridad se basa en tres conceptos clave:

- *Autenticación de confianza*: determinar de forma fiable la identidad del solicitante.
- *Autorización*: otorgar derechos de acceso al solicitante con permisos.
- *Transporte seguro*: transmitir de forma segura los datos a través de las redes.

ObjectGrid proporciona seguridad en los siguientes aspectos:

- "Seguridad de cliente-servidor" en la página 141 describe la autenticación y la seguridad de las comunicaciones de cliente-servidor utilizando Secure Sockets Layer (SSL).
- Los mecanismos de "Autorización" en la página 141 garantizan que sólo los clientes autorizados puedan acceder a los datos de correlación de ObjectGrid y a las tareas de gestión.

- “Seguridad del clúster ObjectGrid” verifica que sólo los servidores autorizados puedan unirse al clúster ObjectGrid.
- “Seguridad de pasarela” en la página 142 describe la autenticación de cliente de pasarela.
- “Seguridad de ObjectGrid local” en la página 142 proporciona un mecanismo de seguridad cuando la aplicación crea directamente una instancia de ObjectGrid.

La seguridad ObjectGrid se basa en una arquitectura abierta y proporciona varios puntos de plug-in que se pueden personalizar. El mecanismo de plug-in juega un papel importante. ObjectGrid también proporciona algunas implementaciones incorporadas para estos plug-ins. Algunas implementaciones son para el uso de producción predefinida, mientras que otras se proporcionan como pruebas o como ejemplos. Consulte “Plug-ins de seguridad” en la página 143 para ver un resumen de los plug-ins y las implementaciones incorporadas.

Seguridad de cliente-servidor

ObjectGrid da soporte a una infraestructura de cliente-servidor distribuida. La infraestructura de seguridad de cliente-servidor existe para proteger el acceso a los servidores ObjectGrid.

Un cliente ObjectGrid puede utilizar la credencial que desee para autenticarse en el servidor ObjectGrid. Se debe establecer un contrato entre los clientes y los servidores, de forma que el mecanismo de autenticación del servidor entienda esta credencial. Cuando se utiliza SSL (Secure Sockets Layer), el cliente también puede utilizar certificados SSL para autenticarse en el servidor ObjectGrid.

Para proteger la comunicación de cliente-servidor, ObjectGrid da soporte a SSL. El protocolo SSL proporciona seguridad de capa de transportes con autenticidad, integridad y confidencialidad, para garantizar una conexión segura entre el cliente y el servidor ObjectGrid. Algunas de las características de seguridad que proporciona SSL son: cifrado de datos para impedir la exposición de información confidencial mientras fluyen los datos, firma de datos para impedir la modificación no autorizada de datos mientras fluyen los datos, y autenticación de cliente y servidor para garantizar que está comunicándose con la persona o la máquina correcta. SSL puede proteger perfectamente un entorno empresarial.

Para obtener más información, consulte “Seguridad de cliente-servidor” en la página 145.

Autorización

Las autorizaciones de ObjectGrid se basan en sujetos y permisos. En ObjectGrid, existen dos categorías de permisos: permisos para el acceso a datos y permisos para las tareas de gestión. Puede utilizar JAAS (Java Authentication and Authorization Services) para autorizar el acceso o conectar sus propios mecanismos para manejar las autorizaciones.

Para obtener más información, consulte “Autorización” en la página 171.

Seguridad del clúster ObjectGrid

En un entorno seguro, un servidor debe poder comprobar la autenticidad de otro servidor. Para ello, ObjectGrid utiliza un mecanismo de serie de clave secreta compartida. Este mecanismo de clave secreta es parecido a una contraseña

secreta. Todos los servidores ObjectGrid acuerdan una serie secreta compartida. Cuando un servidor se une al clúster, se le solicita que presente la serie secreta. Si la serie secreta del servidor que se une coincide con la del servidor maestro, el servidor que se une puede unirse al clúster; de lo contrario, se rechaza la petición de unión.

El envío de una serie secreta en texto normal no es seguro. La infraestructura de seguridad de ObjectGrid proporciona un plug-in SecureTokenManager que permite al servidor “proteger” la serie secreta antes de enviarla. La forma de implementar la operación “segura” está abierta. ObjectGrid proporciona una implementación predefinida, en la que la operación “segura” se implementa para cifrar y firmar la serie secreta.

Para obtener más información, consulte “Seguridad del clúster ObjectGrid” en la página 180

Seguridad de pasarela

Una pasarela ObjectGrid sirve como punto para delegar las peticiones de gestión de cliente al servidor ObjectGrid. La pasarela de gestión aloja un conjunto de mbeans. Un cliente de pasarela invoca estos mbeans para administrar o supervisar los servidores ObjectGrid.

La pasarela de gestión y la comunicación de servidor utilizan el mecanismo de comunicación de cliente-servidor ObjectGrid, en el que la pasarela se trata como un cliente ObjectGrid. La comunicación del cliente de pasarela y la pasarela (servidor MBean) se puede proteger mediante SSL. Esta posibilidad se debe a la capa de conector JMX, que implementa el proyecto de código fuente abierto mx4j. ObjectGrid requiere mx4j para que la pasarela funcione.

Para la autenticación, la pasarela propaga la credencial presentada por el cliente de pasarela al servidor ObjectGrid. La autenticación y la autorización se aplican en los servidores ObjectGrid.

Para obtener más información, consulte “Seguridad de pasarela” en la página 183.

Seguridad de ObjectGrid local

En WebSphere Extended Deployment Server release 6.0, se introdujo el modelo de programación de ObjectGrid local. En este modelo, la aplicación crea directamente una instancia de ObjectGrid y la utiliza. La aplicación y las instancias de ObjectGrid están en la misma Java Virtual Machine (JVM). No existen conceptos de cliente o servidor en este modelo.

La autenticación no está soportada en el modelo de programación de ObjectGrid local. Las aplicaciones deben gestionar su propia autenticación y pasar el objeto Subject autenticado a la ObjectGrid.

En el modelo de programación de ObjectGrid local se utiliza el mismo mecanismo de autorización que el que se utilizaba en el modelo de cliente-servidor.

Para obtener más información, consulte “Seguridad de ObjectGrid local” en la página 164.

Plug-ins de seguridad

Los plug-ins de seguridad suplementan a la infraestructura de seguridad de ObjectGrid. Estos plug-ins se pueden implementar para ampliar o personalizar la infraestructura de seguridad.

Un ejemplo es el plug-in CredentialGenerator, representado por la interfaz com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator. Cuando un cliente ObjectGrid se conecta a un servidor ObjectGrid, se invoca el método getCredential() de este plug-in para generar un objeto Credential. Este objeto Credential se envía al servidor. A continuación, el servidor utiliza el objeto Credential para autenticarse utilizando el plug-in Authenticator, que se representa mediante la interfaz com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator.Authenticator.

Los plug-ins juegan un papel importante en la infraestructura de seguridad de ObjectGrid. Puede implementar el CredentialGenerator para generar una credencial específica, por ejemplo, un par de ID de usuario y contraseña, un vale kerberos o un símbolo de seguridad. Puede implementar el plug-in Authenticator para autenticar el cliente. Si lo desea, puede implementar el plug-in Authenticator para dar soporte a la contraseña de usuario o a un símbolo de seguridad.

Todos los plug-ins de seguridad que puede utilizar se muestran en la siguiente tabla:

Tabla 9. Plug-ins de seguridad

Categoría	Nombre de clase de plug-in	Instancia
Autenticación	`com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator`	cliente
	`com.ibm.websphere.objectgrid.security.plugins.Credential`	cliente
	`com.ibm.websphere.objectgrid.security.plugins.Authenticator`	servidor
Autorización	`com.ibm.websphere.objectgrid.security.plugins.MapAuthorization`	ObjectGrid
	`com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization`	clúster
Seguridad del clúster ObjectGrid	`com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager`	servidor
otras	`com.ibm.websphere.objectgrid.security.plugins.SubjectSource`	ObjectGrid local
	`com.ibm.websphere.objectgrid.security.plugins.SubjectValidation`	ObjectGrid local

El siguiente diagrama muestra los plug-ins y las instancias aplicadas. Por ejemplo, el plug-in MapAuthorization se aplica en las instancias de ObjectGrid, mientras que AdminAuthorization se aplica en las instancias de servidor.

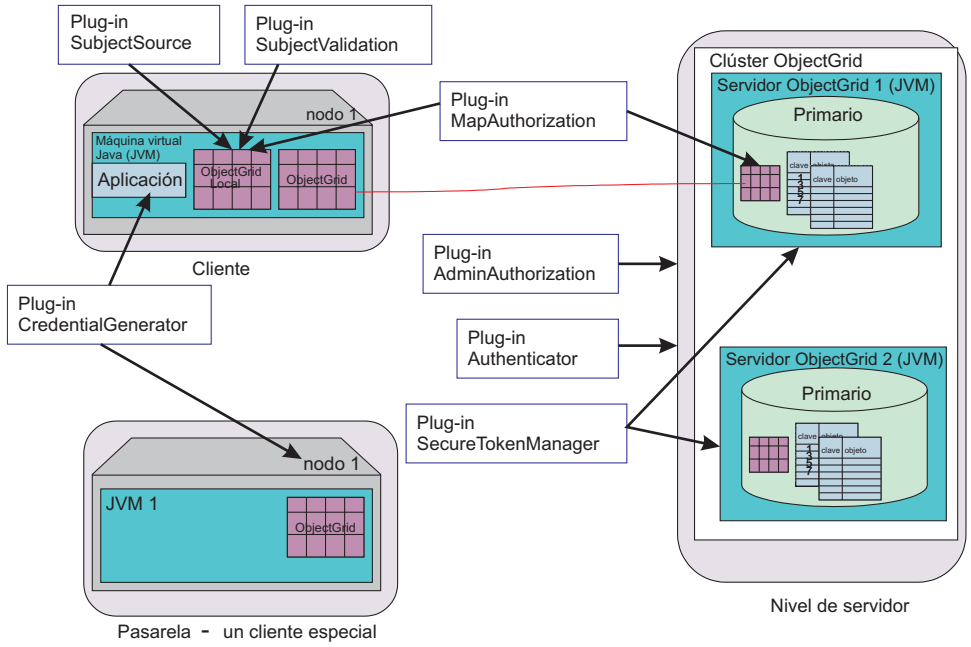


Figura 17. Plug-ins de seguridad

La siguiente tabla muestra la implementación incorporada. La columna de objetivo muestra el objetivo. El objetivo puede ser para producción directamente o para pruebas.

Tabla 10. Implementaciones incorporadas de seguridad

Plug-in	nombre de clase incorporada	objetivo
Credential Generator Credential	com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator	producción
	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator	producción
	com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential	producción
	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential	producción
	com.ibm.websphere.objectgrid.security.plugins.builtins.ClientCertificateCredential	producción
Authenticator	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator	producción
	com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator	pruebas
	com.ibm.websphere.objectgrid.security.plugins.builtins.CertificateMappingAuthenticator	pruebas
	com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator	pruebas
Map Authorization	com.ibm.websphere.objectgrid.security.plugins.builtins.JAASMapAuthorizationImpl	producción
	com.ibm.websphere.objectgrid.security.plugins.builtins.TAMMapAuthorizationImpl	pruebas
SubjectSource	com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl	producción

Tabla 10. Implementaciones incorporadas de seguridad (continuación)

Plug-in	nombre de clase incorporada	objetivo
Subject Validation	`com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl`	producción

Seguridad de cliente-servidor

En este tema se describe el mecanismo de autenticación y cómo proteger la comunicación del servidor del cliente.

La seguridad de cliente-servidor incluye los siguientes aspectos importantes:

- Cómo "Habilitar la seguridad de cliente-servidor"
- Cómo obtener una credencial que represente el cliente con la "Credencial y el generador de credenciales" en la página 146
- Cómo configurar los parámetros utilizados para la configuración SSL utilizando la "Comunicación segura" en la página 162
- Cómo autenticar el cliente en el lado del servidor utilizando un "Autenticador" en la página 152

Habilitar la seguridad de cliente-servidor

Habilitar la seguridad de cliente

Para habilitar la seguridad en la seguridad de cliente, establezca la propiedad securityEnabled en el archivo `security.ogclient.props` en **true**. ObjectGrid proporciona un archivo de plantilla de propiedad de seguridad de cliente, el archivo `security.ogclient.props`, en el directorio `[WAS_HOME]/optionalLibraries/ObjectGrid/properties` para una instalación de WebSphere o el directorio `/ObjectGrid/properties` para una instalación de distintos servidores. Puede modificar este archivo de plantilla con los valores adecuados.

A continuación, se proporciona la descripción de la propiedad securityEnabled:

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 debe ser true en ambos o false en ambos. Por ejemplo, si la seguridad del servidor conectado está habilitada, el cliente debe establecer esta propiedad en true para conectarse al servidor.

La interfaz com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration representa el archivo `security.ogclient.props`. Puede utilizar la API pública com.ibm.websphere.objectgrid.security.config. ClientSecurityConfigurationFactory para crear una instancia de esta interfaz con los valores por omisión, o puede crear una instancia pasando el archivo de propiedades de seguridad del cliente ObjectGrid. El archivo `security.ogclient.props` contiene otras propiedades.

Habilitar la seguridad de servidor

Para habilitar la seguridad en el lado del servidor, puede establecer la propiedad securityEnabled en el XML de clúster en true. A continuación, se muestra un ejemplo:

```
<cluster>
<objectGrid name="cluster" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300">
```

Credencial y el generador de credenciales

Cuando se conecta a un servidor, el cliente necesita presentar su propia credencial. La credencial del cliente se representa mediante una interfaz com.ibm.websphere.objectgrid.security.plugins.Credential. La credencial puede contener un par de contraseñas de usuario, un vale Kerberos, etc.

A continuación, se muestra la interfaz de la credencial:

```
package com.ibm.websphere.objectgrid.security.plugins;
import java.io.Serializable;
/**
* Esta interfaz representa una credencial utilizada por un cliente ObjectGrid. Representa
* una identidad de cliente. Esta credencial se envía al servidor
* ObjectGrid para su autenticación. Debe ser serializable.
*
* Una credencial tiene que implementar los métodos equals(Object) y
* hashCode(). Dos objetos Credential se consideran iguales si y sólo si
* representan la misma identidad y la misma información de seguridad. Supongamos
* que la credencial contiene un ID de usuario y una contraseña. Dos credenciales
* serán iguales si y sólo si los ID de usuario y las contraseñas coinciden.
*
* ObjectGrid proporciona tres implementaciones incorporadas para esta interfaz:
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* ClientCertificateCredential:
* Una credencial que contiene una cadena de certificados SSL.
* com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential:
* Una credencial que contiene un par de ID de usuario y contraseña.
* com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential:
* Una credencial que contiene símbolos de autorización y autenticación
* específicos de WebSphere Application Server.
*
* Consulte la documentación de la API correspondiente para obtener más información.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see CredentialGenerator
*/
public interface Credential extends Serializable {
/**
* Comprueba si dos objetos Credential son iguales.
*
* Dos objetos Credential se consideran iguales si y sólo si
* representan la misma identidad y la misma información de seguridad.
*
* @param o el objeto que se está comprobando que sea igual que este objeto.
*
* @return true si los dos objetos Credential son equivalentes.
*/
boolean equals(Object o);
/**
* Devuelve el código hash del objeto Credential
*
* @return el código hash del objeto Credential
*/
int hashCode();
}
```

Esta interfaz define explícitamente los métodos equals(Object) y hashCode(). Estos métodos son importantes para garantizar el comportamiento. Los objetos Subject autenticados se guardan en antememoria basándose en los objetos Credential en el lado del servidor.

ObjectGrid proporciona tres implementaciones por omisión para las interfaces de credencial.

1. La implementación com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential. Esta credencial contiene un par de ID de usuario y contraseña.
2. La implementación com.ibm.websphere.objectgrid.security.plugins.builtins.ClientCertificateCredential. Esta credencial contiene una cadena de certificados de cliente. Esta credencial se puede utilizar para la autenticación de certificados de cliente ObjectGrid. No puede crear esta credencial en el lado del cliente. Tiene que generarla el servidor como parte del reconocimiento de comunicación SSL.
3. La implementación com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential. Esta credencial contiene símbolos de autenticación y autorización específicos de WebSphere Application Server. Estos símbolos se pueden utilizar para propagar los atributos de seguridad a través de los servidores de aplicaciones en el mismo dominio de seguridad.

Consulte la documentación de la API para obtener más información.

ObjectGrid también proporciona un plug-in para generar una credencial. Este plug-in se representa mediante la interfaz com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator. A continuación, se muestran las interfaces CredentialGenerator:

```
/**
* Este plug-in se utiliza para obtener una credencial que represente a este cliente. Es
* una fábrica para el objeto Credential.
* Una implementación de ejemplo es devolver un objeto Credential que contenga
* un par de ID de usuario y contraseña. La implementación de la credencial
* generada por una implementación de esta clase debe ser comprensible para el
* plug-in Authenticator del servidor.
*
* Una clase de implementación de esta interfaz debe tener un constructor por omisión.
* Cuando ejecute el cliente en un entorno seguro, establezca el nombre de clase
* de implementación (credentialGeneratorClass) en el archivo de propiedades de
* configuración de seguridad de cliente. El tiempo de ejecución del cliente construye
* un objeto de esta clase de implementación e invoca getCredential() para obtener la
* credencial para conectarse a un clúster ObjectGrid.
*
* Los usuarios también pueden especificar las propiedades adicionales de esta fábrica
* utilizando la propiedad credentialGeneratorProps en el archivo de propiedades
* de configuración de seguridad de cliente.
* Estas propiedades se pasan a esta fábrica
* fábrica utilizando el método setProperties(String). De esta forma, puede
* personalizar la fábrica.
*
* También puede establecer CredentialGenerator mediante programación invocando
* el método ClientSecurityCinfiguration.setCredentialGenerator
* (CredentialGenerator).
*
*
* Supongamos que tiene los siguientes valores en el archivo de propiedades de
* configuración de seguridad de cliente:
* credentialGeneratorClass=com.myco.CredGenFactory
*
```

```
* credentialGeneratorProps=user1 password1
*
*
* y se pasa una serie "user1 password1" al método setProperties(String),
* donde "user1" indica el nombre de usuario y "password1"
* indica la contraseña.
*
* ObjectGrid proporciona dos implementaciones incorporadas para esta interfaz:
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* UserPasswordCredentialGenerator:
* Un generador de credenciales que genera una UserPasswordCredential
* que contiene un par de ID de usuario y contraseña.
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* WSTokenCredentialGenerator:
* Un generador de credenciales que genera una WSTokenCredential
* que contiene símbolos de autorización y autenticación
* específicos de WebSphere Application Server.
*
*
* La relación entre CredentialGenerator y Credential puede ser una relación de uno
* con uno o una relación de uno con muchos. Por ejemplo,
* UserPasswordCredentialGenerator
* tiene una relación de uno con uno con UserPasswordCredential, pero
* WSTokenCredentialGenerator
* tiene una relación de uno con muchos con WSTokenCredential, ya que puede generar
* distintos WSTokenCredential según qué sujeto se asocie con la hebra
* actual.
*
* Consulte la documentación de la API correspondiente para obtener más información.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see Autenticador
* @see ClientSecurityConfiguration#setCredentialGenerator(CredentialGenerator)
* @see Credencial
* @see CredentialGeneratorFactory#getCredentialGenerator()
*/
public interface CredentialGenerator {
/**
* Obtiene una credencial que representa al cliente.
*
* @return la credencial que representa al cliente.
*
* @throws CannotGenerateCredentialException si se produce una anomalía al
* generar la credencial para el cliente.
*
* @see Credencial
*/
Credential getCredential() throws CannotGenerateCredentialException;
/**
* Establecer las propiedades definidas por el usuario en la fábrica
*
* Este método se utiliza para añadir propiedades de CredentialGenerator adicionales
* al objeto. Estas propiedades se pueden establecer utilizando la propiedad
* credentialGeneratorProps en el archivo de propiedades de configuración de seguridad
* de cliente. De esta forma puede personalizar la fábrica.
*
* @param properties Propiedades definidas por el usuario
*/
void setProperties(String properties);
}
```

ObjectGrid proporciona dos implementaciones por omisión incorporadas:

1. El constructor com.ibm.websphere.objectgrid.security.plugins.builtins. UserPasswordCredentialGenerator necesita un ID de usuario y una contraseña.

Cuando se invoca el método getCredential(), devuelve un objeto UserPasswordCredential que contiene el ID de usuario y la contraseña.

2. com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator representa un generador de credenciales (símbolo de seguridad) cuando se ejecuta en WebSphere Application Server. Cuando se invoca el método getCredential(), se recupera el sujeto asociado con la hebra actual. A continuación, la información de seguridad en este objeto Subject se convierte en un objeto WSTokenCredential. Puede especificar si desea recuperar un sujeto runAs o un sujeto emisor de la hebra utilizando la constante WSTokenCredentialGenerator.RUN_AS_SUBJECT o WSTokenCredentialGenerator.CALLER_SUBJECT.

Consulte la documentación de la API para obtener más información.

Conectar

Si un cliente ObjectGrid desea conectarse a un servidor de forma segura, puede utilizar un método de conexión cualquiera en la interfaz ObjectGridManager. Supongamos que utiliza el siguiente método de conexión:

```
/**
* Esto permite al cliente conectarse a una ObjectGrid remota
* La ObjectGrid remoto se aloja tal como se especifica en los parámetros
* @param clusterName: El nombre del clúster al que se conectará este
* cliente
* @param host: El sistema principal al que conectarse
* @param port: El puerto clientAceess que está escuchando.
* @param ClientSecurityConfiguration: Configuración de seguridad. Puede ser
* nula si no se configura ninguna seguridad.
* @param overRideObjectGrid xml. El parámetro puede ser nulo. Si no es nulo, la
* configuración del lado del cliente del plug-in de objectgrid se altera temporalmente.
* No todos los plug-ins se pueden alterar temporalmente. Para obtener más
información, consulte los documentos de ObjectGrid
* @throws ConnectException
* @ibm-api
*/
public ClientClusterContext connect(String clusterName, String host, String port,
ClientSecurityConfiguration securityProps, URL overRideObjectGrid) throws
ConnectException ;
```

Este método necesita un parámetro de tipo ClientSecurityConfiguration entre otros. Esta interfaz representa una configuración de seguridad de cliente. Puede utilizar la API pública com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory para crear una instancia de esta interfaz con los valores por omisión, o puede crear una instancia pasando el archivo de propiedades de seguridad del cliente ObjectGrid. El archivo `security.ogclient.props` contiene las siguientes propiedades relacionadas con la autenticación. El valor marcado con + es el valor por omisión.

- `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 debe ser true en ambos o false en ambos. Por ejemplo, si la seguridad del servidor conectado está habilitada, el cliente debe establecer esta propiedad en true para conectarse al servidor.
- `credentialAuthentication (Nunca, Con soporte+, Necesario)`: esta propiedad indica si el cliente da soporte a la autenticación de credenciales.
  - Si el valor de la propiedad es **Nunca**, este cliente no da soporte a ninguna autenticación de credenciales.

  - Si el valor de propiedad es **Con soporte**, se ejecuta la autenticación de cliente cuando se comunica con otro servidor que da soporte o necesita la autenticación de credenciales. La autenticación de credenciales de cliente transmite una credencial o un símbolo de inicio de sesión único (SSO)
  - Si el valor de la propiedad es **Necesario**, el cliente debe enviar una credencial al servidor para su autenticación.
- `authenticationRetryCount (un valor entero, 0+)`. Esta propiedad determina cuántos reintentos de inicio de sesión se realizan cuando caduca una credencial. Si el valor es `0`, no se realiza ningún reintento. El reintento de autenticación sólo se aplica en los casos en los que caduca la credencial. Si la credencial no es válida, no se realiza ningún reintento. La aplicación es responsable de reintentar la operación.
- `clientCertificateAuthentication (Nunca+, Con soporte, Necesario)`: esta propiedad indica si el cliente da soporte a la autenticación de certificados de cliente.
  - Si el valor de la propiedad es **Nunca**, no se proporciona soporte a ninguna autenticación de certificados de cliente en el lado del cliente.
  - Si el valor de la propiedad es **Con soporte**, se puede ejecutar la autenticación de cliente de capa de transporte y el cliente envía un certificado digital al servidor durante la fase de autenticación.
  - Si el valor de la propiedad es **Necesario**, el cliente sólo se autentica con los servidores que den soporte a la autenticación de cliente de capa de transporte.
- `transportType (TCP/IP, Da soporte a SSL+, Precisa SSL)`: indica qué protocolo de transporte desea el cliente para conectarse al servidor. Con qué protocolo se conecta el cliente al servidor también depende del valor transportType en el lado del servidor. Para obtener más información, consulte “Comunicación segura” en la página 162. .
  - Si el valor es **TCP/IP**, el cliente tiene que utilizar TCP/IP para conectarse al servidor.
  - Si el valor es **Da soporte a SSL**, el cliente puede utilizar TCP/IP o SSL para conectarse al servidor. El cliente intenta primero utilizar SSL para conectarse al servidor. Si la conexión SSL falla, el cliente intenta utilizar TCP/IP.
  - Si el valor es **Precisa SSL**, el cliente tiene que utilizar SSL para conectarse al servidor.
- `SSOEnabled`: especifica si el cliente da soporte al paso de símbolos de inicio de sesión único al servidor. Establezca esta propiedad en `false` si el cliente se autentica en cada servidor. Establezca esta propiedad en `true` si el cliente sólo se autentica en un servidor. Si establece SSOEnabled true en el cliente, compruebe que la propiedad habilitada para el inicio de sesión único en la configuración XML del clúster también esté establecida en `true`.

También puede establecer estas propiedades utilizando métodos de establecimiento en la interfaz ClientSecurityConfiguration.

Una vez creado un objeto de tipo ClientSecurityConfiguration, establezca el credentialGenerator en el objeto utilizando el siguiente método:

```
/**
* Establezca el objeto {@link CredentialGenerator} para este cliente.
* @param generator El objeto CredentialGenerator asociado con este cliente
*/
void setCredentialGenerator(CredentialGenerator generator);
```

Puede establecer también el CredentialGenerator en el archivo de propiedades de seguridad del cliente ObjectGrid. Las propiedades son las siguientes:

- **credentialGeneratorClass**: el nombre de implementación de clase de CredentialGenerator. Debe tener un constructor por omisión.
- **credentialGeneratorProps**: las propiedades de la clase CredentialGenerator. Si el valor no es nulo, se establece en el objeto CredentialGenerator construido utilizando el método setProperties(String).

A continuación, se proporciona un ejemplo de cómo crear una instancia de ClientSecurityConfiguration y utilizarla para conectarse al servidor.

```
/**
* Obtener un ClientClusterContext seguro
* @return 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 invoca la conexión, el cliente ObjectGrid llama al método CredentialGenerator.getCredential() para obtener la credencial del cliente. Esta credencial se envía junto con la petición de conexión al servidor para su autenticación.

Utilizar un CredentialGenerator diferente para cada sesión

En algunos casos, un cliente ObjectGrid representa sólo una identidad de cliente; en otros casos, puede representar varias identidades. A continuación, se proporciona un ejemplo del segundo caso: un cliente ObjectGrid se crea y se comparte en un servidor Web. Todos los servlets del servidor Web utilizan este cliente ObjectGrid. Como cada servlet representa un cliente Web diferente, utilice distintas credenciales cuando envíe peticiones a los servidores ObjectGrid.

ObjectGrid permite cambiar la credencial a nivel de sesión. Esto es, cada sesión puede utilizar un CredentialGenerator diferente. Por lo tanto, los casos de ejemplo anteriores se pueden ejecutar dejando que el servidor obtenga una sesión con un CredentialGenerator diferente. A continuación, se muestra el método en la interfaz ObjectGridManager.

```
/**
* Obtener una sesión con un CredentialGenerator. Este método sólo lo puede invocar
* el cliente ObjectGrid en un entorno de cliente-servidor.
*
* Si ObjectGrid se utiliza en un modelo principal, esto es, en la misma JVM sin
* que exista ningún cliente o servidor, se debe utilizar getSession(Subject) para
* proteger la ObjectGrid.
*
* @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");
// Obtener una sesión con CredentialGenerator;
Session session = og.getSession(credGenManager );
// Obtener la correlación de empleados
ObjectMap om = session.getMap("employee");
// iniciar una transacción.
session.begin();
Object rec1 = map.get("xxxxxx");
session.commit();
// Obtener otra sesión con un CredentialGenerator diferente;
session = og.getSession(credGenEmployee );
// Obtener la correlación de empleados
om = session.getMap("employee");
// iniciar una transacción.
session.begin();
Object rec2 = map.get("xxxxx");
session.commit();
```

Si utiliza el método ObjectGird.getSession() para obtener un objeto Session, la sesión utiliza el CredentialGenerator establecido en el objeto ClientConfigurationSecurity. Por lo tanto, puede tratar el CredentialGenerator pasado al método ObjectGird.getSession(CredentialGenerator) que altera temporalmente el CredentialGenerator establecido en el objeto ClientConfigurationSecurity.

Si puede reutilizar el objeto Session, se obtiene un aumento del rendimiento. No obstante, la invocación del método ObjectGrid.getSession(CredentialGenerator) no resulta muy costosa; la carga adicional principal es el aumento del tiempo de recogida de basura del objeto. Asegúrese de liberar las referencias cuando termine con los objetos Session. En resumen, si el objeto Session puede compartir la identidad, intente reutilizarlo; si el objeto Session no puede compartir la identidad, utilice el método ObjectGrid.getSession(CredentialGenerator).

Autenticador

Cuando el cliente ObjectGrid recupera el objeto Credential utilizando el objeto CredentialGenerator, el objeto Credential se envía junto con la petición de cliente al servidor ObjectGrid. El servidor ObjectGrid autentica el objeto Credential antes de procesar la petición. Si el objeto Credential se autentica satisfactoriamente, se devuelve un objeto Subject que representa a este objeto Credential. A continuación, este objeto Subject se utilizará para la autorización de la petición.

Este objeto Subject también se guarda en la antememoria. Caducará cuando su ciclo de vida alcance el valor de tiempo de espera de sesión. El valor de tiempo de espera de sesión de inicio de sesión se puede establecer utilizando la propiedad loginSessionExpirationTime en el archivo XML de clúster. Por ejemplo, si se establece loginSessionExpirationTime=″300″, el objeto Subject caducará en 300 segundos.

El servidor ObjectGrid utiliza el plug-in Authenticator para autenticar el objeto Credential. A continuación, se muestra la interfaz del autenticador:

```
/**
* Este plug-in puede utilizarse para autenticar un cliente ObjectGrid en un servidor
* ObjectGrid basándose en la credencial proporcionada por el cliente. Se devuelve un
* objeto Subject como resultado de la autenticación.
*
* Este plug-in se utiliza en un servidor ObjectGrid. Se puede configurar en el
* archivo XML de clúster ObjectGrid.
*
```

```
* La credencial pasada en el método authenticate(Credential) puede contener
* toda la información de credencial que deseen los usuarios. Por ejemplo, puede ser
* un objeto Credential que contenga un par de contraseñas de usuario.
*
* ObjectGrid proporciona varias implementaciones incorporadas para esta interfaz:
* * com.ibm.websphere.objectgrid.security.plugins.builtins.
* CertificateMappingAuthenticator:
* Un autenticador que simplemente correlaciona un certificado SSL con un sujeto.
* com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator:
* Un autenticador que autentica un ID de usuario y una contraseña en un archivo de claves.
* com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator:
* Un autenticador que autentica un ID de usuario y una contraseña en un servidor LDAP.
* com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator:
* Un autenticador que autentica un símbolo de seguridad de WebSphere Application Server.
*
* Consulte la documentación de la API correspondiente para obtener más información.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see Credencial
*/
public interface Authenticator {
/**
* Autentica un usuario representado por el objeto de credencial.
*
* @param credential La credencial del usuario
*
* @return Un objeto Subject que representa al usuario
*
* @throws InvalidCredentialException si la credencial no es válida
* @throws ExpiredCredentialException si la credencial ha caducado
*
* @see Credencial
*/
Subject authenticate(Credential credential)
throws InvalidCredentialException, ExpiredCredentialException;
}
```

Aquí es donde la implementación obtiene el objeto Credential y lo autentica en un registro de usuarios, por ejemplo, un servidor LDAP (Lightweight Directory Access Protocol), etc. ObjectGrid no proporciona una configuración de registro de usuarios directamente. La conexión a un registro de usuarios y la autenticación en él se deben implementar en este plug-in.

Por ejemplo, una implementación de autenticador extrae el ID de usuario y la contraseña de la credencial, los utiliza para conectarse y validarse en un servidor LDAP, y crea un objeto Subject como resultado de la autenticación. La implementación puede utilizar módulos de inicio de sesión JAAS. Se devuelve un objeto Subject como resultado de la autenticación.

Tenga en cuenta que este método genera 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 excepciones a partir del método de autenticación, las excepciones se devuelven al cliente. No obstante, el tiempo de ejecución del cliente maneja estas dos excepciones de forma diferente:

- Si la excepción es una InvalidCredentialException, el tiempo de ejecución del cliente muestra la excepción. Se espera que la aplicación maneje la excepción. Puede corregir el CredentialGenerator, por ejemplo, y reintentar la operación.

- Si la excepción es una ExpiredCredentialException, y el número de reintentos no es 0, el tiempo de ejecución del cliente vuelve a llamar al método CredentialGenerator.getCredential() y envía el nuevo objeto Credential al servidor. Si la autenticación de la nueva credencial es satisfactoria, el servidor procesa la petición. Si la autenticación de la nueva credencial falla, la excepción se devuelve al cliente. Si el número de reintentos de autenticación alcanza el valor permitido y el cliente continúa obteniendo ExpiredCredentialException, se genera una ExpiredCredentialException. La aplicación debe manejar la excepción.

La interfaz del autenticador proporciona una gran flexibilidad. Puede implementar la interfaz del autenticador de varias formas. Por ejemplo, puede implementar esta interfaz para que ejecute la autenticación de credenciales y la autenticación de certificados de cliente, para dar soporte a ambas autenticaciones. O bien, puede implementar la interfaz para dar soporte a dos registros de usuarios diferentes.

ObjectGrid da soporte a dos tipos de autenticaciones: autenticación de credenciales y autenticación de certificados de cliente. Qué mecanismo se debe utilizar dependerá del valor de la propiedad de seguridad en el lado del cliente y el servidor. A continuación, se muestran estas propiedades:

- credentialAuthentication en el archivo `security.ogclient.props`
- credentialAuthentication en el archivo `security.ogserver.props`
- clientCertificateAuthentication en el archivo `security.ogclient.props`
- clientCertificateAuthentication en el archivo `security.ogserver.props`

Recuerde que también puede establecer estas propiedades utilizando las API de programación.

Las dos tablas siguientes muestran qué mecanismo de autenticación se utiliza con los distintos valores.

Tabla 11. Autenticación de credenciales con los valores de cliente y servidor

credentialAuthentication de cliente	**credentialAuthentication de servidor**	**Resultado**
No	Nunca	inhabilitado
	Con soporte	inhabilitado
	Necesario	Caso de error
Con soporte	Nunca	inhabilitado
	Con soporte	habilitado
	Necesario	habilitado
Necesario	Nunca	Caso de error
	Con soporte	habilitado
	Necesario	habilitado

Cuando no existe ninguna autenticación de credenciales (el resultado está inhabilitado), se puede realizar la autenticación de certificados de cliente.

En la tabla siguiente se muestran las autenticaciones de certificados de cliente que se utilizan con los distintos valores. Tenga en cuenta que la autenticación de certificados de cliente sólo es posible si se utiliza SSL como protocolo de comunicaciones y no se utiliza la autenticación de credenciales.

Tabla 12. Autenticación de certificados de cliente con los valores de cliente y servidor

Autenticación clientCertificate de cliente	Autenticación clientCertificate de servidor	Resultado
No	Nunca	inhabilitado
	Con soporte	inhabilitado
	Necesario	Caso de error
Con soporte	Nunca	inhabilitado
	Con soporte	habilitado*
	Necesario	habilitado*
Necesario	Nunca	Caso de error
	Con soporte	habilitado*
	Necesario	habilitado*

* ClientCertificateAuthentication sólo se realiza cuando se utiliza SSL como protocolo y no se utiliza CredentialAuthentication.

Tenga en cuenta que existen algunos matices: cuando se utilizan la autenticación de credenciales y la autenticación de certificados de cliente, pero la credencial enviada desde el cliente es nula, se utiliza la autenticación de certificados de cliente.

El autenticador se puede configurar en el archivo XML de clúster. A continuación se muestra un ejemplo:

```
<cluster name="cluster1" securityEnabled="true" singleSignOnEnabled="true"
loginSessionExpirationTime="300" statisticsEnabled="true"
statisticsSpec="map.all=enabled">
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12500" workingDirectory="" traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12501" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<authenticator
className ="com.ibm.websphere.objectgrid.security.plugins.builtins.
WSTokenAuthenticator">
</authenticator>
</cluster>
```

ObjectGrid proporciona estas cuatro implementaciones incorporadas de autenticación por omisión: autenticación de ID de usuario y contraseña en un registro de usuarios de archivo de claves, autenticación de ID de usuario y contraseña en un servidor LDAP, autenticación de correlación sencilla de certificados de cliente SSL y mecanismo de seguridad de WebSphere Application Server. Excepto la implementación de autenticador del mecanismo de seguridad de WebSphere Application Server, las implementaciones incorporadas sólo se proporcionan a efectos de prueba. El objetivo principal de estos dos programas incorporados es permitir realizar una prueba sencilla sin escribir ningún código. La implementación de autenticador de WebSphere Application Server es una implementación predeterminada que se puede conectar cuando los clientes y los servidores ObjectGrid están en el mismo dominio de seguridad.

Para los servidores ObjectGrid que deseen utilizar registros de usuarios de WebSphere Application Server, puede utilizar las API de WebSphere Application

Server para obtener el registro de usuarios configurado en el servidor de aplicaciones y, a continuación, utilizarlo en la implementación de autenticador. No obstante, esta implementación está fuera del ámbito de esta guía de programación.

Implementación de autenticador de registro de archivos de claves

Puede almacenar el ID de usuario y la contraseña en un archivo denominado archivo de almacén de claves. Puede utilizar la herramienta keytool para crear un archivo de almacén de claves y entradas. Por ejemplo, el siguiente mandato crea una entrada con el alias user1:

```
keytool -genkey -v -keystore ./keys.jks -storepass password -alias user1
-keypass password -dname CN=user1,O=MyCompany,L=MyCity,ST=MyState
```

A efectos de prueba, ObjectGrid proporciona la implementación por omisión com.ibm.websphere.objectgrid.security.plugins.builtins. KeyStoreLoginAuthenticator para que este plug-in maneje la autenticación de nombre de usuario y contraseña. Esta implementación utiliza el nombre de inicio de sesión KeyStoreLogin para que el usuario inicie una sesión en un archivo de almacén de claves.

A continuación, se proporciona un fragmento de código que muestra la implementación del método authenticate(Credential) en la clase KeyStoreLoginAuthenticator.

```
public Subject authenticate(Credential credential)throws
InvalidCredentialException,
ExpiredCredentialException {
UserPasswordCredential cred = (UserPasswordCredential) credential;
LoginContext lc = null;
lc = new LoginContext("KeyStoreLogin",
new UserPasswordCallbackHandlerImpl(cred.getUserName(),
cred.getPassword().toCharArray()));
lc.login();
Subject subject = lc.getSubject();
```

Este fragmento convierte primero la credencial en una UserPasswordCredential, que es una implementación de la interfaz de credencial, ya que tiene un contrato con el cliente por el que el cliente sólo puede pasar un objeto de tipo UserPasswordCredential. A continuación, llama al módulo de inicio de sesión KeyStoreLogin para iniciar una sesión.

ObjectGrid suministra un módulo de inicio de sesión com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule con este fin. Debe proporcionar un archivo de almacén de claves que contenga el par de nombre de usuario y contraseña de cada usuario. El archivo de almacén de claves se configura como una opción en el módulo de inicio de sesión.

A continuación, se proporciona el fragmento de código que muestra cómo inicia una sesión el módulo de inicio de sesión en el archivo de claves.

```
/**
* Autentica un usuario basándose en el archivo de almacén de claves.
*
* @see javax.security.auth.spi.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("La inicialización del módulo ha fallado");
}
NameCallback nameCallback = new NameCallback("Username:");
PasswordCallback pwdCallback = new PasswordCallback("Password:", false);
try {
handler.handle(new Callback[] { nameCallback, pwdCallback });
}
catch (Exception e) {
throw new LoginException("El retorno de llamada ha fallado: " + 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: "
+ "el usuario ha entrado el nombre de usuario: " +
name);
}
if (ObjectGridManagerImpl.isTraceEnabled && TC.isDebugEnabled())
Tr.debug(TC, "login", "userName="+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(
"Se ha producido una excepción durante el inicio de sesión");
le.initCause(se);
throw le;
}
if (debug) {
System.out.println("[KeyStoreLoginModule] login: exit");
}
  return true;
}
/**
* Validar 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 ha encontrado alguna excepción
*/
protected 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("Nombre no válido: " + userName);
}
// Comprobar los certificados
Certificate certs[] = null;
try {
certs = keyStore.getCertificateChain(userName);
}
catch (KeyStoreException kse) {
SecurityException se = new SecurityException();
se.initCause(kse);
throw se;
}
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 al 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 añadido: " + principal);
}
}
catch (CertificateException ce) {
SecurityException se = new SecurityException();
se.initCause(ce);
throw se;
}
}
}
}
```

Debe crear un nombre de inicio de sesión “KeyStoreLogin” en el archivo de configuración de la autenticación JAAS. Si no está familiarizado con el archivo de configuración de la autenticación JAAS, consulte la Guía de aprendizaje de autenticación JAAS para obtener más información.

```
KeyStoreLogin {
com.ibm.websphere.objectgrid.jaas.KeystoreLoginModule required
keyStoreFile="${user.dir}${/}security${/}.keystore";
};
```

Esta implementación se proporciona sólo a efectos de prueba.

Implementación de autenticador LDAP

ObjectGrid proporciona la implementación por omisión com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator para que este plug-in maneje la autenticación de nombre de usuario y contraseña en un servidor LDAP. Esta implementación utiliza el módulo de inicio de sesión LDAPLogin para que el usuario inicie una sesión en un servidor LDAP.

El siguiente fragmento demuestra cómo se implementa el método de autenticación:

```
/**
* @see com.ibm.ws.objectgrid.security.plugins.Authenticator#
* authenticate(LDAPLogin)
*/
public Subject authenticate(Credential credential) throws
InvalidCredentialException, ExpiredCredentialException {
UserPasswordCredential cred = (UserPasswordCredential) credential;
LoginContext lc = null;
try {
lc = new LoginContext("LDAPLogin",
new UserPasswordCallbackHandlerImpl(cred.getUserName(),
cred.getPassword().toCharArray()));
lc.login();
Subject subject = lc.getSubject();
return subject;
}
catch (LoginException le) {
throw new InvalidCredentialException(le);
}
catch (IllegalArgumentException ile) {
throw new InvalidCredentialException(ile);
}
}
```

ObjectGrid suministra un módulo de inicio de sesión com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule con este fin. Debe proporcionar las dos opciones siguientes en el archivo de configuración de inicio de sesión JAAS:

- **providerURL**: el URL del proveedor del servidor LDAP
- **factoryClass**: la clase de implementación de la fábrica de contexto LDAP

LDAPLoginModule llama al método com.ibm.websphere.objectgrid.security.plugins.builtins. LDAPAuthentcationHelper.authenticate. El siguiente fragmento de código muestra cómo se implementa el método de autenticación de LDAPAuthentcationHelper:

```
/**
* Autenticar el usuario en el directorio LDAP.
* @param user el ID de usuario, p.ej., uid=xxxxxx,c=us,ou=bluepages,o=ibm.com
* @param pwd la contraseña
*
* @throws NamingException
*/
public String[] authenticate(String user, String pwd)
throws NamingException {
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, factoryClass);
env.put(Context.PROVIDER_URL, providerURL);
env.put(Context.SECURITY_PRINCIPAL, user);
env.put(Context.SECURITY_CREDENTIALS, pwd);
env.put(Context.SECURITY_AUTHENTICATION, "simple");
InitialContext initialContext = new InitialContext(env);
// 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 satisfactoria, el ID y la contraseña se consideran válidos. A continuación, el módulo de inicio de sesión obtiene la información de UID y la información del departamento a partir de este método de autenticación. El módulo de inicio de sesión crea dos principales: SimpleUserPrincipal y SimpleDeptPrincipal. Puede utilizar el sujeto autenticado para la autorización de grupo (en este caso, el departamento es un grupo) y la autorización individual.

A continuación, se muestra un ejemplo de configuración de módulo de inicio de sesión que se utiliza para iniciar una 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 servidor `ldap://directory.acme.com:389/`. Cambie este valor por su servidor LDAP. Este módulo de inicio de sesión utiliza el ID de usuario y la contraseña proporcionados para conectarse al servidor LDAP. Esta implementación se proporciona sólo a efectos de prueba.

Implementación de autenticador de WebSphere Application Server

ObjectGrid también 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 si se cumplen las siguientes condiciones:

- La seguridad global de WebSphere Application Server está activada.
- Los clientes ObjectGrid y los servidores ObjectGrid están ejecutados en las máquinas virtuales Java de WebSphere Application Server.
- Estos servidores de aplicaciones están en el mismo dominio de seguridad.
- El cliente ObjectGrid ya está autenticado en WebSphere Application Server.

El cliente ObjectGrid puede utilizar la clase com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator para generar una credencial y el servidor ObjectGrid utiliza esta clase de implementación de autenticador para autenticar la credencial. Si el símbolo se ha autenticado satisfactoriamente, se devuelve un objeto Subject.

Este caso de ejemplo aprovecha el hecho de que el cliente ObjectGrid ya se ha autenticado. Como los servidores de aplicaciones que tienen los servidores ObjectGrid están en el mismo dominio de seguridad que los servidores de aplicaciones que alojan los clientes ObjectGrid, los símbolos de seguridad se pueden propagar desde el cliente ObjectGrid al servidor ObjectGrid de forma que el mismo registro de usuarios no se tenga que volver a autenticar.

Implementación de autenticador de correlación de certificados sencilla
ObjectGrid también proporciona una implementación incorporada com.ibm.websphere.objectgrid.security.plugins.builtins. CertificateMappingAuthenticator para correlacionar el certificado con un objeto Subject. La implementación extrae el nombre distinguido (DN) del primer certificado de la cadena y crea un principal con ese nombre. Esta implementación se proporciona sólo a efectos de prueba.

Implementación de autenticador de Tivoli Access Manager
Tivoli Access Manager se ha utilizado ampliamente como servidor de seguridad. También puede implementar el autenticador utilizando los módulos de inicio de sesión proporcionados por Tivoli Access Manager.

Para autenticar un usuario utilizando Tivoli Access Manager, el LoginModule proporcionado por Tivoli, com.tivoli.mts.PDLoginModule, requiere que la aplicación emisora proporcione lo siguiente:

- Un nombre de principal, especificado como un nombre abreviado o un nombre X.500 (DN)
- Una contraseña

LoginModule autentica el principal y devuelve la credencial de Tivoli Access Manager. LoginModule espera que la aplicación emisora proporcione la siguiente información:

- El nombre de usuario, a través de una javax.security.auth.callback.NameCallback
- La contraseña, a través de una javax.security.auth.callback.PasswordCallback.

Cuando la credencial de Tivoli Access Manager se recupera satisfactoriamente, JAAS LoginModule crea un objeto Subject y un objeto PDPrincipal. No se proporciona ningún programa incorporado para la autenticación de TAM, ya que es obvio con PDLoginModule. Consulte IBM Tivoli Access Manager Authorization Java Classes Developer Reference para obtener más información.

Inicio de sesión único

Cuando un cliente ObjectGrid se autentica satisfactoriamente en un servidor, el servidor ObjectGrid crea un objeto Subject. Si el cliente y el servidor dan soporte al inicio de sesión único (SSO), este objeto Subject se convierte en un símbolo SSO. Este símbolo se pasa al lado del cliente para asociarse con el socket. El símbolo SSO se puede pasar a un nuevo servidor para su autenticación, para que no se tenga que reautenticar en un servidor diferente.

El símbolo SSO se implementa utilizando el mecanismo de gestor de símbolos seguros de ObjectGrid. Para obtener más información sobre el gestor de símbolos seguros, consulte “Seguridad del clúster ObjectGrid” en la página 180. Básicamente, el mecanismo de símbolos seguros utiliza claves criptográficas

(claves secretas) para cifrar y descifrar datos de usuario que se pasan entre los servidores, y claves públicas-privadas para firmar los datos.

El símbolo SSO también contiene una hora de caducidad. Todos los servidores de productos que participan en un dominio de protección deben tener sincronizados la hora, la fecha y el huso horario. De lo contrario, los símbolos SSO aparecen caducados prematuramente y provocan anomalías de autenticación o validación. (Esto no es necesario si se utiliza la hora universal).

Cuando un cliente ObjectGrid se conecta a un servidor diferente, este símbolo SSO se puede pasar al nuevo servidor. Este servidor validará el símbolo SSO para asegurarse de que no se ha manipulado eliminando la firma y descifrándolo. También comprueba la indicación de la hora para comprobar que no ha caducado. Si el símbolo es válido, el cliente no tiene que autenticarse en este servidor.

Si un símbolo SSO ha caducado, el servidor tiene que volver a autenticar el cliente. El servidor solicita al cliente que proporcione de nuevo la credencial.

Habilitar inicio de sesión único del cliente

El inicio de sesión único del cliente se puede habilitar de dos formas:

- **Configuración**. Utilice la propiedad SSOEnabled en el archivo `security.ogclient.props` para habilitar el inicio de sesión único en el lado del cliente.
- **Mediante programación**. Utilice ClientSecurityConfiguration para habilitar el inicio de sesión único con el siguiente método.

```
/**
* Establezca si el inicio de sesión único está habilitado.
* @param enabled Si el inicio de sesión único está habilitado para
* este cliente o no.
*/
void setSingleSignOnEnabled(boolean enabled);
```

Habilitar inicio de sesión único del servidor

Para habilitar el inicio de sesión único en el lado del servidor, establezca el atributo singleSignOnEnabled en true en el archivo XML de clúster. A continuación se muestra un ejemplo:

```
<cluster>
<objectGrid name="cluster" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300">
```

Observe que el inicio de sesión único sólo se habilita si la seguridad está habilitada.

Comunicación segura

ObjectGrid da soporte a TCP/IP y SSL para la comunicación segura. SSL proporciona una comunicación segura entre el cliente y el servidor. Qué mecanismo de comunicación se debe utilizar dependerá de los valores de las siguientes propiedades:

- La propiedad transportType en el archivo security.ogclient.props
- La propiedad transportType en el archivo security.ogserver.props

Tabla 13. El protocolo de transporte que se utiliza con los valores de transporte de cliente y de transporte de servidor

transportType de cliente	transportType de servidor	Protocolo resultante
TCP/IP	TCP/IP	TCP/IP
	Da soporte a SSL	TCP/IP
	Precisa SSL	Error
Da soporte a SSL	TCP/IP	TCP/IP
	Da soporte a SSL	SSL (si SSL falla, TCP/IP)
	Precisa SSL	SSL
Precisa SSL	TCP/IP	Error
	Da soporte a SSL	SSL
	Precisa SSL	SSL

Cuando se utiliza SSL, se debe proporcionar la configuración SSL en el lado del cliente y del servidor.

Configurar parámetros SSL para los clientes ObjectGrid

Los parámetros SSL en el lado del cliente se pueden configurar de las formas siguientes:

- Cree un objeto com.ibm.websphere.objectgrid.security.config.SSLConfiguration utilizando la clase de fábrica com.ibm.websphere.objectgrid.security.config. ClientSecurityConfigurationFactory. Para obtener más información, consulte la documentación de la API.
- Configure los parámetros en el archivo `security.ogclient.props` y, a continuación, utilice el método ClientSecurityConfigurationFactory.getClientSecurityConfiguration(String) para rellenar la instancia del objeto.

Las siguientes propiedades se aplican a las configuraciones SSL en el archivo `security.ogclient.props`.

- **provider**: especifica el proveedor SSL JSSE. Los valores posibles son IBMJSSE+, IBMJSSE2, SunJSSE, etc. Establezca este valor según el JDK (Java Development Kit) que utilice.
- **protocol**: especifica el protocolo SSL. Los valores posibles son SSL+, SSLV2, SSLV3, TLS, TLSv1, etc. Establezca este valor de protocolo según el proveedor JSSE (Java Secure Socket Extension) que utilice.
- **alias**: la serie representa el alias en el almacén de claves. No existe ningún valor por omisión. Esta propiedad se utiliza si el almacén de claves tiene varios certificados de par de claves y desea seleccionar uno de los certificados.
- **keyStoreType**: especifica el tipo de almacén de claves SSL. Los valores posibles son JKS+, JCEK, PKCS12, etc. Establezca este valor según el proveedor JSSE (Java Secure Socket Extension) que utilice.
- **keyStore**: especifica el nombre de archivo y la vía de acceso del almacén de claves que tienen las claves privadas y los certificados públicos de cliente. Por ejemplo, `[OBJECTGRID_HOME]/properties/DummyClientKeyFile.jks`. En este release, el soporte de hardware no está soportado.
- **keyStorePassword**: especifica la contraseña para proteger la vía de acceso del almacén de claves. La contraseña se codifica simplemente

utilizando el algoritmo “xor” en ObjectGrid. Utilice la herramienta PropFilePasswordEncoder para codificar este archivo de propiedades. A continuación, se muestra un ejemplo de una contraseña codificada: `{x0r}CDo9Hgw\\`.

- **trustStoreType**: especifica el tipo de almacén de confianza. Los valores posibles son JKS+, JCEK, PKCS12, etc. Puede establecer este valor según el proveedor de JSSE que utilice.
- **trustStore**: especifica el nombre de archivo y la vía de acceso del almacén de confianza que tienen los certificados públicos de servidor. Por ejemplo, `[OBJECTGRID_HOME]/properties/DummyClientTrustFile.jks`
- **trustStorePassword**: especifica la contraseña para proteger la vía de acceso del almacén de confianza. La contraseña se codifica simplemente utilizando el algoritmo xor en ObjectGrid. Utilice la herramienta PropFilePasswordEncoder para codificar este archivo de propiedades. A continuación, se muestra un ejemplo de una contraseña codificada: `{x0r}CDo9Hgw\`
- **certReqSubjectDN**: esta es la serie que se necesita en el nombre distinguido (DN) del sujeto de certificado del servidor. Un cliente puede conectarse al servidor sólo si el DN del certificado del servidor contiene esta serie. Si el valor es nulo, el cliente no necesita un DN de sujeto particular en el certificado del cliente. Por ejemplo, si el DN del sujeto de certificado es ″CN=Server1, OU=Your Organizational Unit, O=Your Organization, S=Your State,C=Your Country″, entonces ″CN=server1″, ″O=Your Organization″, ″OU=Your Organizational Unit, O=Your Organization, S=Your State,C=Your Country″ da como resultado una coincidencia, pero ″CN=server2″ y ″OU=Your Organizational Unit, L=smething, O=Your Organization, S=Your State,C=Your Country″ no coincide. La coincidencia de comodines no está soportada.

Configurar parámetros SSL para los servidores ObjectGrid

Los parámetros SSL en el lado del cliente se pueden configurar en el archivo `security.ogserver.props`. Este archivo de propiedades se puede pasar como un parámetro cuando ejecuta un servidor ObjectGrid.

Excepto las propiedades SSL anteriores, la configuración SSL del lado del servidor tiene una propiedad adicional:

- **clientAuthentication** (true+, false). Si esta propiedad se establece en true, el cliente SSL se debe autenticar. Esto es distinto a la autenticación de certificados de cliente. La autenticación de certificados de cliente significa autenticar un cliente en un registro de usuarios basándose en la cadena de certificados, mientras que esta propiedad garantiza que el servidor se conecte al cliente adecuado.

Seguridad de ObjectGrid local

En este tema se describe la seguridad del modelo de programación de ObjectGrid local. En el modelo de programación de ObjectGrid local, la función de seguridad principal es la autorización. El modelo de programación de ObjectGrid local no da soporte a ninguna autenticación. Debe autenticarse fuera de ObjectGrid. No obstante, ObjectGrid proporciona plug-ins para obtener y validar objetos Subject.

La habilitación de la seguridad de ObjectGrid se puede hacer de dos formas:

- **Configuración**. Puede utilizar el archivo XML de ObjectGrid para definir una ObjectGrid y habilitar la seguridad para esa ObjectGrid. A continuación, aparece 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 estableciendo el atributo securityEnabled en `true`.

```
<objectGrids>
<objectGrid name="secureClusterObjectGrid" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_JASS">
<bean id="TransactionCallback"
classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
...
</objectGrids>
```

- **Mediante programación**. Si desea crear una ObjectGrid utilizando las API, llame al siguiente método de la interfaz ObjectGrid para habilitar la seguridad.

  ```
  /**
  * Habilitar la seguridad de ObjectGrid
  */
  void setSecurityEnabled();
  ```

  En el modelo de programación de ObjectGrid local, no existe ninguna autenticación. Cuando se establece la seguridad con este método, se configura la autorización. Esta condición es coherente con el modelo de cliente-servidor. La habilitación de la seguridad de una ObjectGrid en el modelo de cliente-servidor sólo habilita la autorización en esa instancia de ObjectGrid.

Autenticación

En el modelo de programación de ObjectGrid local, ObjectGrid no proporciona ningún mecanismo de autenticación. ObjectGrid se basa en el entorno, ya sean los servidores de aplicaciones o las aplicaciones, para las autenticaciones. Cuando ObjectGrid se utiliza en WebSphere Application Server o WebSphere Extended Deployment, las aplicaciones pueden utilizar el mecanismo de autenticación de seguridad de WebSphere Application Server. Cuando ObjectGrid se ejecuta en un entorno J2SE (Java 2 Platform, Standard Edition), la aplicación debe gestionar las autenticaciones con la autenticación JAAS (Java Authentication and Authorization Service) u otros mecanismos de autenticación. Para obtener más información sobre el uso de la autenticación JAAS, consulte la guía de consulta de JAAS.

El contrato entre una aplicación y una instancia de ObjectGrid es el objeto javax.security.auth.Subject. Después de que el servidor de aplicaciones o la aplicación autentique al cliente, la aplicación puede recuperar el objeto javax.security.auth.Subject autenticado y utilizar el objeto Subject para obtener una sesión de la instancia de ObjectGrid llamando al método ObjectGrid.getSession(Subject). Este objeto Subject se utiliza para autorizar los accesos a los datos de correlación. Este contrato se denomina un mecanismo para pasar el sujeto. A continuación se muestra la API ObjectGrid.getSession(Subject):

```
/**
* Esta API permite que la antememoria utilice un sujeto específico y no el
* configurado en la ObjectGrid para obtener una sesión.
* @param subject
* @return Instancia de Session
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws InvalidSubjectException el sujeto pasado no es válido en función
* del mecanismo SubjectValidation.
*/
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;
```

El método getSession de la interfaz ObjectGrid también puede utilizarse para obtener un objeto Session:

```
/**
* Éste devuelve un objeto Session que una única hebra puede utilizar cada vez.
* No está permitido que varias hebras compartan esta Session sin colocar una
* sección crítica a su alrededor. Mientras que la infraestructura principal
* permite que se mueva el objeto entre hebras, es posible que TransactionCallback
* y el Loader impidan este uso,
* especialmente en entornos J2EE. Cuando la seguridad está habilitada, éste
* utilizará 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. Esto
* garantiza que toda la configuración se complete antes de que se requiera
* usar el tiempo de ejecución.
*
* @see #initialize()
* @return Instancia de Session
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws IllegalStateException si se llama a este método después de
* que se llame al método destroy().
*/
public Session getSession()
throws ObjectGridException, TransactionCallbackException;
```

Como especifica la documentación de la API, cuando la seguridad está activada, este método utiliza el plug-in SubjectSource para obtener un objeto Subject. El plug-in SubjectSource es uno de los plug-in de seguridad que se definen en ObjectGrid para dar soporte a la propagación de objetos Subject. Para obtener más información, consulte “Plug-ins relacionados con la seguridad”.

El método getSession(Subject) sólo se puede invocar en la instancia de ObjectGrid local. Si invoca el método getSession(Subject) en el lado del cliente en una configuración de ObjectGrid distribuida, se genera una excepción.

Plug-ins relacionados con la seguridad

ObjectGrid proporciona dos plug-ins de seguridad relacionados con el mecanismo para pasar el sujeto: los plug-ins SubjectSource y SubjectValidation.

Plug-in SubjectSource

El plug-in SubjectSource, representado mediante la interfaz com.ibm.websphere.objectgrid.security.plugins.SubjectSource, se utiliza para obtener un objeto Subject de un entorno de ejecución de ObjectGrid. Este entorno de ObjectGrid puede ser una aplicación que utilice ObjectGrid o un servidor de aplicaciones que aloje la aplicación. A continuación, aparece la interfaz:

```
/**
* Este plug-in puede utilizarse para obtener un objeto Subject que represente al
* cliente ObjectGrid.
* A continuación, este sujeto se utilizará para la autorización de ObjectGrid. El
* tiempo de ejecución de ObjectGrid llama al método getSubject cuando se utiliza
* el método ObjectGrid.getSession() para obtener una sesión y la
* seguridad está habilitada.
*
* Este plug-in es útil para un cliente que ya está autenticado: puede
* recuperar el objeto Subject autenticado y pasarlo a la instancia de
* ObjectGrid. Por lo tanto, no se necesita ninguna otra
* autenticación.
*
* Por ejemplo, utilice
* Subject.getSubject(AccessControlContext)
* para obtener el sujeto asociado con el AccessControlContext y,
* a continuación, devuélvalo en la implementación de getSubject.
```

```
*
* Este plug-in sólo se puede utilizar en un dominio seguro como, por
* ejemplo, en un servidor ObjectGrid.
*
* @ibm-api
* @since WAS XD 6.0
*/
public interface SubjectSource {
/**
* Obtener un objeto Subject que pueda representar el cliente ObjectGrid.
*
* @return Un objeto Subject
* @throws ObjectGridSecurityException cualquier excepción durante la recuperación
* del sujeto
*/
Subject getSubject() throws ObjectGridSecurityException;
}
```

Considere el plug-in SubjectSource como una alternativa al mecanismo para pasar el sujeto. Al utilizar el mecanismo para pasar el sujeto, la aplicación recupera el objeto Subject y lo utiliza para obtener el objeto de sesión de ObjectGrid. Con el plug-in SubjectSource, el tiempo de ejecución de ObjectGrid recupera el objeto Subject y lo utiliza para obtener el objeto de sesión. El mecanismo para pasar el sujeto otorga el control de los objeto Subject a las aplicaciones, mientras que el mecanismo del plug-in SubjectSource libra a las aplicaciones de recuperar el objeto Subject.

Este plug-in SubjectSource puede utilizarse para obtener un objeto Subject que representa a un cliente ObjectGrid que se utiliza para la autorización de ObjectGrid. Cuando se llama al método ObjectGrid.getSession(), el Subject de getSubject() genera una excepción ObjectGridSecurityException; el tiempo de ejecución de ObjectGrid llama al método, si la seguridad está habilitada.

ObjectGrid proporciona una implementación por omisión de este plug-in: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl. Esta implementación puede utilizarse para recuperar un sujeto emisor o un sujeto RunAs de la hebra cuando una aplicación se ejecute en WebSphere Application Server. Puede configurar esta clase como la clase de implementación SubjectSource cuando utilice ObjectGrid en WebSphere Application Server. A continuación, aparece un fragmento de código que muestra el flujo principal de 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 el sujeto emisor
  s = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
}
}
catch (WSSecurityException wse) {
 throw new ObjectGridSecurityException(wse);
}
return s;
```

Para obtener más detalles, consulte la documentación de la API del plug-in SubjectSource y la implementación de WSSubjectSourceImpl.

Plug-in SubjectValidation

El plug-in SubjectValidation, que se representa mediante 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 javax.security.auth.Subject, que se haya pasado a ObjectGrid o recuperado mediante el plug-in SubjectSource, sea un sujeto válido que no se haya manipulado. Esta es la interfaz.

```
/**
* Este plug-in puede utilizarse para validar que un javax.security.auth.Subject
* pasado a la ObjectGrid es un sujeto válido que no se ha
* manipulado.
*
* Una implementación de este plug—in necesita soporte del creador del
* objeto Subject, ya que sólo el creador sabe si el objeto Subject
* se ha manipulado. No obstante, puede que el creador del sujeto no
* no sepa si se ha manipulado el objeto Subject. En este caso,
* no debe utilizarse este plug-in.
*
* Este plug-in sólo se puede utilizar en un dominio seguro como, por
* ejemplo, en un servidor de aplicaciones. No coloque este plug-in en
* el lado del cliente; se ignora.
*
* @ibm-api
*
* @since WAS XD 6.0
*/
public interface SubjectValidation {
/**
* Validar que no se ha manipulado el objeto Subject.
* @param subject El sujeto a validar
* @return el objeto Subject validado
* @throws InvalidSubjectException
*/
Subject validateSubject(Subject subject) throws
InvalidSubjectException;
}
```

El Subject de validateSubject(Subject) genera una excepción InvalidSubjectException; el método de la interfaz SubjectValidation toma un objeto Subject y devuelve un objeto Subject. La validez de un objeto Subject, o qué objeto Subject se devuelve, depende 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 que se pasa a este método. Este caso es excepcional, teniendo en cuenta que confiamos en los desarrolladores de la aplicación que desarrollan el código para recuperar el objeto Subject.

El creador del objeto Subject debe dar soporte a la implementación de este plug-in ya que sólo el creador sabe si se ha manipulado el objeto Subject. No obstante, es posible que algún creador del sujeto no sepa si se ha manipulado el objeto Subject. En este caso, este plug-in no es de utilidad.

ObjectGrid proporciona una implementación por omisión de SubjectValidation: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl. Esta implementación puede utilizarse para validar al sujeto autenticado de WebSphere. Los usuarios pueden configurar esta clase como la clase de implementación SubjectValidation cuando utilicen ObjectGrid en WebSphere

Application Server. La implementación de WSSubjectValidationImpl considera válido a un objeto Subject si y sólo si no se ha manipulado el símbolo de credencial asociado con este objeto Subject. En otras palabras, puede haber modificado otras partes del objeto Subject. La implementación de WSSubjectValidationImpl solicita a WebSphere Application Server el objeto Subject original correspondiente al símbolo de credencial y devuelve este objeto Subject original como el objeto Subject validado. Por lo tanto, los cambios realizados en el contenido del Subject que no estén relacionados con el símbolo de credencial no tiene ningún efecto. El siguiente fragmento de código muestra el flujo básico de WSSubjectValidationImpl.validateSubject(Subject):

```
// Crear un LoginContext con el esquema WSLogin y
// pasar un manejador de retorno de llamada.
LoginContext lc = new LoginContext("WSLogin",
new WSCredTokenCallbackHandlerImpl(subject));
// Cuando se llama a este método, se llamará a los métodos de manejador de
// retorno de llamada para iniciar la sesión del usuario.
lc.login();
// Obtener el objeto del LoginContext
return lc.getSubject();
```

En el fragmento de código anterior, un objeto del manejador de retorno de llamada del símbolo de credencial, WSCredTOkenCallbackHandlerImpl, se crea con el objeto Subject que se vaya a validar. A continuación, se crea un LoginContext con el esquema de inicio de sesión ″WSLogin″. Cuando se llama al método lc.login(), la seguridad de WebSphere Application Server recupera el símbolo de credencial del objeto Subject y, a continuación, devuelve el Subject correspondiente como el objeto Subject validado.

Para obtener más detalles, consulte la documentación de la API de SubjectValidation y WSSubjectValidationImpl.

Configuración de plug-ins

Los plug-ins SubjectValidation y SubjectSource se pueden configurar de dos modos:

- **Configuración**. Puede utilizar el archivo XML de ObjectGrid para definir una ObjectGrid y establecer estos dos plug-ins. A continuación, aparece un ejemplo en el que la clase WSSubjectSourceImpl se configura como el plug-in SubjectSource y la clase WSSubjectValidation como el 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>
  ```

- **Mediante programación.** Si desea crear una ObjectGrid a través de las API, puede llamar a los siguientes métodos para establecer los plug-ins SubjectSource o SubjectValidation.

```
/**
* Establecer el plug-in SubjectValidation para esta instancia de
* ObjectGrid. Un plug-in SubjectValidation puede utilizarse para
* validar el objeto Subject pasado como un Subject válido.
* Consulte {@link SubjectValidation} para obtener más detalles.
* @param subjectValidation el plug-in SubjectValidation
*/
void setSubjectValidation(SubjectValidation subjectValidation);
/**
* Establecer el plug-in SubjectSource. Puede utilizarse un plug-in
* SubjectSource para obtener un objeto Subject del entorno para que
* represente al cliente ObjectGrid.
*
* @param source plug-in SubjectSource
*/
void setSubjectSource(SubjectSource source);
```

Escriba su propio código de autenticación JAAS

Puede escribir su propio código de autenticación JAAS para manejar la autenticación. Es necesario que escriba sus propios módulos de inicio de sesión y, a continuación, configure los módulos de inicio de sesión para el módulo de autenticación.

El módulo de inicio de sesión recibe información sobre un usuario y lo autentica. Esta información puede ser cualquier cosa que sirva para identificar al usuario. Por ejemplo, puede ser un ID de usuario y contraseña, un certificado de cliente, etc. Después de recibir la información, el módulo de inicio de sesión verifica que representa a un sujeto válido y, a continuación, crea un objeto Subject. Actualmente, están disponibles para el público varias implementaciones de módulos de inicio de sesión.

Después de la escritura de un módulo de inicio de sesión, configúrelo para que el tiempo de ejecución pueda utilizarlo. Debe configurarse un archivo de configuración del módulo de inicio de sesión JAAS. Este módulo de inicio de sesión contiene el módulo de inicio de sesión y el 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. El símbolo necesario indica que el módulo FileLoginModule debe validar este inicio de sesión o, de lo contrario, se produce un error del esquema en conjunto.

El establecimiento del archivo de configuración del módulo de inicio de sesión JAAS puede realizarse mediante uno de los siguientes modos:

- Establezca el archivo de configuración del módulo de inicio de sesión JAAS en el **login.config.url** del archivo `java.security`, por ejemplo, `login.config.url.1=file:${java.home}/lib/security/file.login`
- Establezca el archivo de configuración del módulo de inicio de sesión JAAS desde la línea de mandatos utilizando los argumentos JVM **-Djava.security.auth.login.config**, por ejemplo, `-Djava.security.auth.login.config ==$JAVA_HOME/lib/security/file.login`

Para obtener más información sobre cómo escribir y configurar módulos de inicio de sesión, consulte la guía de aprendizaje de autenticación JAAS.

Si el código se ejecuta en WebSphere Application Server, debe configurar el inicio de sesión JAAS en la consola administrativa y almacenar esta configuración de inicio de sesión en la configuración del servidor de aplicaciones. Consulte Configuración de inicio de sesión de JAAS (Java Authentication and Authorization Service) para obtener más información.

Autorización

Una vez autenticado el cliente, puede utilizar mecanismos de autorización de ObjectGrid para autorizar el acceso a las tareas de gestión y los datos de correlación de ObjectGrid. La autorización de ObjectGrid se basa en el objeto Subject. ObjectGrid da soporte a dos tipos de mecanismos de autorización: la autorización JAAS (Java Authentication and Authorization Service) y la autorización personalizada.

Clase de permiso

Existen dos tipos diferentes de autorización de ObjectGrid: autorización en los datos de la correlación y autorización en las tareas de gestión. Cada autorización utiliza una clase de permiso. El permiso para acceder a la correlación viene representado por la clase MapPermission y el permiso para ejecutar las tareas de gestión viene representado por la clase AdminPermission.

Clase MapPermission

En ObjectGrid, la clase pública com.ibm.websphere.objectgrid.security.MapPermission representa los permisos de los recursos de ObjectGrid, específicamente, los métodos de las interfaces ObjectMap o JavaMap. ObjectGrid define las siguientes series de permiso para acceder a los métodos de ObjectMap y JavaMap:

- **read**: otorga permiso para leer los datos de la correlación. La constante de entero se define como MapPermission.READ.
- **write**: otorga permiso para actualizar los datos de la correlación. La constante de entero se define como MapPermission.WRITE.
- **insert**: otorga permiso para insertar los datos en la correlación. La constante de entero se define como MapPermission.INSERT.
- **remove**: otorga permiso para eliminar los datos de la correlación. La constante de entero se define como MapPermission.REMOVE.
- **invalidate**: otorga permiso para invalidar los datos de la correlación. La constante de entero se define como MapPermission.INVALIDATE.
- **all**: otorga todos los permisos: read, write, insert, remote e invalidate. La constante de entero se define como MapPermission.ALL.

Puede construir un objeto MapPermission pasando el nombre de correlación de ObjectGrid plenamente cualificado (con el formato [nombre_ObjectGrid].[Correlación_ObjectMap]) y la serie o valor entero de permiso. Una serie de permiso puede ser una serie delimitada por comas de las series de permisos anteriores como, por ejemplo, ″read, insert″, o puede ser ″all″, que significa que se otorgan todos los permisos. Un valor entero de permiso puede ser cualquiera de las constantes de entero de permiso anteriores o un valor matemático “o” de varias constantes de entero de permiso como, por ejemplo, DGMapPermission.GET|DGMapPermission.PUT.

La autorización ocurre cuando un cliente llama a un método de ObjectMap o JavaMap. El tiempo de ejecución de ObjectGrid comprueba diferentes permisos para los distintos métodos. Si no se otorgan a los clientes los

permisos necesarios, se generará una AccessControlException.

Tabla 14. Lista de métodos y los permisos necesarios

	com.ibm.websphere.objectgrid.ObjectMap **com.ibm.websphere.objectgrid.JavaMap**
read	boolean containsKey(Object)
	boolean equals(Object)
	Object get(Object)
	Object get(Object, Serializable)
	List getAll(List)
	List getAll(List keyList, Serializable)
	List getAllForUpdate(List, Serializable)
	Object getForUpdate(Object)
	Object getForUpdate(Object, Serializable)
write	Object put(Object key, Object value)
	void put(Object, Object, Serializable)
	void putAll(Map)
	void putAll(Map, Serializable)
	void update(Object, Object)
	void update(Object, Object, Serializable)
insert	public void insert (Object, Object)
	void insert(Object, Object, Serializable)
	remove Object remove (Object)
	void removeAll(Collection)
invalidate	public void invalidate (Object, boolean)
	void invalidateAll(Collection, boolean)
	void invalidateUsingKeyword(Serializable)
	int setTimeToLive(int)

La autorización se basa únicamente en qué método se utiliza, en lugar de en qué hace realmente el método. Por ejemplo, un método put puede insertar o actualizar un registro según si existe o no el registro. No obstante, los casos de inserción o actualización no se distinguen en este momento.

Observe también que un tipo de operación se puede conseguir mediante la combinación de otros tipos. Por ejemplo, una actualización se puede obtener mediante una eliminación y una inserción posterior. Tenga esto en cuenta cuando diseñe las políticas de autorización.

AdminPermission

El permiso de administración está representado por la clase com.ibm.websphere.objectgrid.security.AdminPermission. ObjectGrid define dos acciones de permiso para los permisos de administración:

- **admin**: otorga permisos para realizar tareas de administración.
- **monitor**: otorga permisos a las acciones que son tareas de administración de sólo acceso de lectura.

Las operaciones detalladas que se otorgan a los usuarios con los distintos permisos se enumeran en la siguiente tabla. Estas operaciones se corresponden con los métodos de la interfaz ManagementMBean:

Tabla 15. Relación entre las tareas de gestión y los permisos de administración

operaciones	admin	monitor
startServer	S	N
stopServer	S	N
forceStopServer	S	N
setServerTrace	S	N
retrieveServerStatus	S	S
getMapStats	S	S
getOGStats	S	S
getReplicationStats	S	S

Si el cliente tiene el permiso admin, puede ejecutar la tarea startServer; si el cliente tiene el permiso monitor, no puede ejecutar la tarea startServer.

Mecanismos de autorización

ObjectGrid da soporte a dos tipos de mecanismos de autorización: la autorización JAAS y la autorización personalizada. Esto se aplica a la autorización de acceso a datos de correlación y a la autorización de administración. La autorización JAAS incrementa las políticas de seguridad Java con controles de acceso centrados en el usuario. Se pueden otorgar los permisos en función no sólo del código que se está ejecutando, sino también de quién (principal) lo está ejecutando. Esto es parte de JDK 1.4.

ObjectGrid también da soporte a la autorización personalizada con el plug-in com.ibm.websphere.objectgrid.security.plugins.MapAuthorization y el plug-in com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization. Puede implementar su propio mecanismo de autorización si no desea utilizar la autorización JAAS. Mediante los mecanismos de autorización personalizados, puede utilizar la base de datos de políticas, el servidor de políticas o Tivoli Access Manager para gestionar las autorizaciones de ObjectGrid.

El mecanismo de autorización de ObjectGrid se puede configurar de dos modos:

- **Configuración**. Puede utilizar el archivo XML de ObjectGrid para definir una ObjectGrid y establecer los mecanismos de autorización en AUTHORIZATION_MECHANISM_JAAS o AUTHORIZATION_MECHANISM_CUSTOM. A continuación, se muestra el archivo `secure-objectgrid-definition.xml` que se utiliza en el ejemplo de aplicación de empresa ObjectGridSample.

```
<objectGrids>
<objectGrid name="secureClusterObjectGrid" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
<bean id="TransactionCallback"
classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
...
</objectGrids>
```

- **Mediante programación**. Si desea crear una ObjectGrid mediante las API, puede llamar al siguiente método para establecer el mecanismo de autorización. Esto sólo se aplica al modelo de programación de ObjectGrid local cuando se crea directamente la instancia de ObjectGrid.

```
/**
* Establecer el mecanismo de autorización. El valor por omisión es
* com.ibm.websphere.objectgrid.security.SecurityConstants.
* AUTHORIZATION_MECHANISM_JAAS.
* @param authMechanism mecanismo de autorización de correlación
*/
void setAuthorizationMechanism(int authMechanism);
```

Autorización JAAS

Un objeto javax.security.auth.Subject representa a un usuario autenticado. Un Subject consta de un conjunto de principales, donde cada Principal representa una identidad para ese usuario. Por ejemplo, un Subject podría tener un principal de nombre (″Joe Smith″) y un principal de grupo (″manager″).

Mediante la política de autorización JAAS, los permisos pueden otorgarse a Principales específicos. ObjectGrid asocia el Subject con el contexto de control de acceso actual. Para cada llamada de método a ObjectMap o Javamap, el tiempo de ejecución Java determina automáticamente si la política otorga el permiso necesario sólo a un Principal específico.

Debe estar familiarizado con la sintaxis de política del archivo de políticas. Para ver una descripción detallada de la autorización JAAS, consulte la guía de aprendizaje JAAS Authorization Tutorial.

ObjectGrid tiene una base de código especial que se utiliza para comprobar la autorización JAAS en las llamadas de método a 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 otorgar los permisos de ObjectMap o JavaMap a los principales. Este código especial se ha creado porque se otorgan todos los permisos al archivo JAR (Java Archive) de ObjectGrid.

La plantilla de la política para otorgar MapPermission es:

```
grant codeBase "http://www.ibm.com/com/ibm/ws/objectgrid/security/
PrivilegedAction"
<Campo(s) Principal>{
permission com.ibm.websphere.objectgrid.security.MapPermission
"[nombre_ObjectGrid].[nombre_ObjectMap]", "acción";
....
permission com.ibm.websphere.objectgrid.security.MapPermission
"[nombre_ObjectGrid].[nombre_ObjectMap]", "acción";
};
```

Un campo Principal tiene un aspecto parecido al siguiente:

```
Principal
Principal_class "nombre_principal"
```

Esto es, la palabra ″Principal″ seguida del nombre plenamente cualificado de una clase de Principal y un nombre de principal. map_name es el nombre de correlación plenamente cualificado en el formato [Nombre de ObjectGrid].[Nombre de correlación] como, por ejemplo, ″secureClusterObjectGrid.employees″. La acción es una serie delimitada por comas de las series de permisos definidas en la clase MapPermission como, por ejemplo, “read, insert” o “all”.

Sólo se da soporte a la función de comodín de forma limitada. Puede sustituir el nombre de ObjectGrid o el nombre de correlación por “*” para indicar “cualquiera”. No obstante, ObjectGrid no da soporte a la sustitución

de parte del nombre de ObjectGrid o nombre de correlación por "*". Por lo tanto, "*.employees", "clusterObjectGrid.*", y "*.*" son nombres válidos, pero "cluster*.employees" no lo es

Por ejemplo, en la aplicación de ejemplo `ObjectGridSample.ear`, están definidos dos archivos de políticas de autorización: `fullAccessAuth.policy` y `readInsertAccessAuth.policy`. El contenido de `readInsertAccessAuth.policy` es el siguiente:

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/
PrivilegedAction"
Principal com.ibm.ws.security.common.auth.WSPrincipalImpl
"principal_name" {
permission com.ibm.websphere.objectgrid.security.MapPermission
"secureClusterObjectGrid.employees", "read,insert";
permission com.ibm.websphere.objectgrid.security.MapPermission
"secureClusterObjectGrid.offices", "read,insert";
permission com.ibm.websphere.objectgrid.security.MapPermission
"secureClusterObjectGrid.sites", "read,insert";
permission com.ibm.websphere.objectgrid.security.MapPermission
"secureClusterObjectGrid.counters", "read,insert";
};
```

En esta política, sólo se otorgan los permisos ″insert″ y ″read″ a las cuatro correlaciones con un determinado principal. El otro archivo de políticas `fullAccessAuth.policy` otorga ″todos″ los permisos a estas correlaciones con un principal. Antes de ejecutar la aplicación, cambie el principal_name y la clase de principal por los valores correspondientes. El valor de principal_name depende del registro de usuario. Por ejemplo, si se utiliza el sistema operativo local como registro de usuario, el nombre de máquina es MAQUINA1 y el ID de usuario es usuario1, entonces el principal_name es ″MAQUINA1/usuario1″.

La política de autorización JAAS puede incluirse directamente en el archivo de políticas Java o puede incluirse en un archivo de autorización JAAS separado, que debe establecerse a continuación utilizando el argumento `-Djava.security.auth.policy=file:[JAAS_AUTH_POLICY_FILE]` de JVM o utilizando `auth.poliyc.url.x= file:[JAAS_AUTH_POLICY_FILE]` en el archivo `java.security`.

La descripción de la autorización JAAS también se aplica cuando desea escribir y configurar las políticas para autorizar el acceso a las tareas de gestión. La única diferencia será que en lugar de utilizar el formato ″com.ibm.websphere.objectgrid.security.MapPermission map name, actions;″, se utiliza ″com.ibm.websphere.objectgrid.security.AdminPermission action;″. La acción puede ser ″admin″ o ″monitor″.

Autorización de correlación personalizada

ObjectGrid también da soporte a la autorización de correlación personalizada mediante el plug-in MapAuthorization. A continuación, aparece la interfaz:

```
/**Este
plug-in se puede utilizar para autorizar los accesos de ObjectMap/JavaMap a los
* principales representados por el objeto Subject.
*
* Una implementación típica de este plug-in consiste en recuperar los
* principales del objeto Subject y, a continuación, comprobar si los permisos
* especificados se otorgan a los principales.
*
```

```
*
* @ibm-api
* @since WAS XD 6.0
*/
public interface MapAuthorization {
/**
* Compruebe si los principales representados por el objeto Subject
* en el sujeto tienen el MapPermission especificado. Si se han
* otorgado los permisos, se devuelve true; de lo contrario, se
* devuelve false.
*
* @param subject el sujeto
* @param permission el permiso para acceder a ObjectMap
*
* @return true si se otorga el permiso; de lo contrario, false.
*/
boolean checkPermission(Subject subject, MapPermission permission);
}
```

Este plug-in puede utilizarse para autorizar los accesos de ObjectMap y JavaMap a los principales contenidos en el objeto Subject. El siguiente método:

```
boolean checkPermission(Subject subject, MapPermission permission)
```

El tiempo de ejecución de ObjectGrid llama a la interfaz MapAuthorization para comprobar si el objeto Subject que se ha pasado tiene permiso para ser transferido. En caso afirmativo, la implementación de la interfaz MapAuthorization devuelve true; de lo contrario, devuelve false.

Una implementación típica de este plug-in consiste en recuperar los principales del objeto Subject y comprobar si los permisos especificados se otorgan a los principales mediante la consulta de políticas específicas. Son los usuarios quienes definen estas políticas. Por ejemplo, las políticas pueden definirse en una base de datos, un archivo sencillo o un servidor de políticas de Tivoli Access Manager.

ObjectGrid proporciona dos implementaciones por omisión para este plug-in. La clase com.ibm.websphere.objectgrid.security.plugins.builtins. JAASMapAuthorizationImpl es una implementación de MapAuthorization, que utiliza el mecanismo JAAS para la autorización. Otra clase de implementación es la clase com.ibm.websphere.objectgrid.security.plugins.builtins. TAMMapAuthorizationImpl. Muestra cómo puede utilizarse Tivoli Access Manager para gestionar las autorizaciones de ObjectGrid. A continuación, aparece un fragmento de código que muestra el flujo básico de JAASMapAuthorizationImpl.checkPermission(Subject, MapPermission):

```
// Crea una PrivilegedExceptionAction para comprobar los permisos.
PrivilegedExceptionAction action =
  MapPermissionCheckAction.getInstance(permission);
Subject.doAsPrivileged(subject, action, null);
```

Consulte IBM Tivoli Access Manager Authorization Java Classes Developer Reference para obtener más información.

No utilice este plug-in TAMMapAuthorizationImpl en un caso de ejemplo predefinido. Utilice este plug-in sólo a efectos de prueba. Requiere determinadas precondiciones restrictivas:

- El objeto Subject contiene un principal com.tivoli.mts.PDPrincipal.

- El servidor de políticas TAM ha definido los siguientes permisos para el objeto de nombre de ObjectMap o JavaMap. El objeto definido en el servidor de políticas debe tener el mismo nombre que el nombre de ObjectMap o JavaMap en el formato [nombre_ObjectGrid].[nombre_ObjectMap]. El permiso es el primer carácter de las series de permisos definidas en MapPermission. Por ejemplo, el permiso ″r″ definido en el servidor de políticas representa el permiso ″read″ de ObjectMap.

El siguiente fragmento 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,
MapPermission permission) {
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) {
// Manejar la excepción
}
}
  return false;
}
```

El plug-in MapAuthorization se puede configurar de dos formas:

- **Configuración**. Puede utilizar el archivo XML de ObjectGrid para definir un plug-in MapAuthorization. A continuación se muestra un ejemplo:

  ```
  <objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
  ...
  <bean id="MapAuthorization"
  className="com.ibm.websphere.objectgrid.security.plugins.builtins.
  JAASMapAuthorizationImpl" />
  </objectGrids>
  ```

- **Mediante programación**. Si desea crear una ObjectGrid mediante las API, puede llamar al siguiente método para establecer el plug-in de autorización. Esto sólo se aplica al modelo de programación de ObjectGrid local cuando se crea directamente la instancia de ObjectGrid.

  ```
  /**
  * Establece el plug-in MapAuthorization para esta instancia de ObjectGrid.
  *
  * Un plug-in {@link MapAuthorization} se puede utilizar para autorizar el
  * el acceso a las correlaciones. Consulte {@link MapAuthorization}
  ```

```
   * para obtener más detalles.
   * @param mapAuthorization el plug-in MapAuthorization
   */
   void setMapAuthorization(MapAuthorization mapAuthorization);
```

Autorización de administración personalizada

Como el soporte de autorización de acceso a datos de correlación personalizada, ObjectGrid da soporte a la autorización de administración personalizada. El plug-in es com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization.

```
/**
* Este plug-in puede utilizarse para autorizar operaciones de gestión en los
* principales contenidos en el objeto Subject. Los permisos para las
* operaciones de gestión están representados por objetos
* AdminPermission.
*
* Este plug-in se utiliza en un servidor ObjectGrid. Se puede configurar en el
* archivo XML de clúster ObjectGrid.
*
* Una implementación típica de este plug-in consiste en recuperar el conjunto
* de principales del objeto Subject y, a continuación,
* comprobar si los permisos especificados se otorgan a los principales.
*
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see AdminPermission
*/
public interface AdminAuthorization {
/**
* Comprueba si el usuario representado por el objeto Subject tiene el
* AdminPermission especificado o no.
*
* Si se han otorgado los permisos, se devuelve true; de lo contrario, se
* devuelve false.
*
* @param subject el objeto Subject que representa al usuario
* @param permission el permiso de administración que se debe comprobar
*
* @return true si se otorga el permiso; de lo contrario, false.
*
* @see AdminPermission
*/
boolean checkPermission(Subject subject, AdminPermission permission);
}
```

Este plug-in puede utilizarse para autorizar los accesos de admin a los principales contenidos en el objeto Subject. El tiempo de ejecución de ObjectGrid llama al método

```
boolean checkPermission(Subject subject, AdminPermission permission)
```

de la interfaz AdminAuthorization para comprobar si el objeto Subject que se ha pasado tiene permiso admin para ser transferido. En caso afirmativo, la implementación de la interfaz AdminAuthorization debe devolver true; de lo contrario, devuelve false.

Puede implementar esta interfaz según los requisitos de seguridad. ObjectGrid no proporciona una clase de implementación de esta interfaz.

Puede establecer el plug-in AdminAuthorization a nivel de clúster en el XML de clúster. A continuación se muestra un ejemplo:

```
<cluster name="cluster1" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300"
statisticsEnabled="true"
statisticsSpec="map.all=enabled">
<serverDefinition name="server1" host="localhost"
clientAccessPort="12503" peerAccessPort="12500" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="localhost"
clientAccessPort="12504" peerAccessPort="12501" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<authenticator className ="com.ibm.websphere.objectgrid.security.plugins.
builtins.WSTokenAuthenticator"></authenticator>
<adminAuthorization className= "com.ibm.ws.objectgrid.test.security.util.
TestAdminAuthorization"></adminAuthorization>
</cluster>
```

Periodo de comprobación de permisos

ObjectGrid da soporte a la colocación en antememoria de los resultados de comprobación de permisos de correlación a efectos de rendimiento. Sin este mecanismo, cuando se llama a un método incluido en la Tabla 14 en la página 172, el tiempo de ejecución de ObjectGrid llama al mecanismo de autorización configurado para autorizar el acceso. Con este periodo de comprobación de permisos establecido, se llama al mecanismo de autorización periódicamente en función del periodo de comprobación de permisos.

La información de autorización de permisos se almacena en antememoria basándose en el objeto Subject. Cuando un cliente intenta acceder a los métodos, el tiempo de ejecución de ObjectGrid buscará en la antememoria basándose en el objeto Subject. Si no se puede encontrar en la antememoria, el tiempo de ejecución comprobará los permisos otorgados para este objeto Subject y, a continuación, los almacenará en una antememoria.

El periodo de comprobación de permisos se debe definir antes de inicializar la ObjectGrid. El periodo de comprobación de permisos puede configurarse de dos modos:

- **Configuración**. Puede utilizar el archivo XML de ObjectGrid para definir una ObjectGrid y establecer el periodo de comprobación de permisos. A continuación, se proporciona un ejemplo para establecer el periodo de comprobación de permisos en 45 segundos.

  ```
  <objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS"
  permissionCheckPeriod="45">
  <bean id="bean id="TransactionCallback"
  className="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- **Mediante programación**. Si desea crear una ObjectGrid con las API, llame al siguiente método para establecer el periodo de comprobación de permisos. Este método sólo se puede invocar antes de inicializar la instancia de ObjectGrid. Este método sólo se aplica al modelo de programación de ObjectGrid local cuando se crea directamente la instancia de ObjectGrid.

  ```
  /**
  * Este método toma un único parámetro que indica con qué frecuencia el cliente
  * desea comprobar el permiso utilizado para permitir un acceso de cliente. Si el
  * parámetro es 0, cada una de las llamadas a get/put/update/remove/evict solicitará
  * el mecanismo de autorización, ya sea la autorización JAAS o la autorización
  ```

```
* personalizada, para comprobar si el sujeto actual tiene permiso. Esto puede ser
* extraordinariamente costoso desde el punto de vista del rendimiento, dependiendo
* de la implementación de autorización, pero si es necesario, puede hacerlo.
* Alternativamente, si el parámetro es > 0, indica el número de segundos
* para colocar en antememoria un conjunto de permisos antes de volver al
* mecanismo de autorización para renovarlos. Esto mejora el rendimiento
* considerablemente pero se corre el riesgo de que, si se modifican los
* permisos de programa de fondo durante este tiempo, ObjectGrid permitirá
* o impedirá el acceso aunque el proveedor de seguridad de programa de fondo
* se haya modificado.
*
* @param period periodo de comprobación de permisos en segundos.
*/
void setPermissionCheckPeriod(int period);
```

Seguridad del clúster ObjectGrid

La seguridad del clúster ObjectGrid garantiza que todo servidor que se una tenga la credencial correcta, para que un servidor malicioso no pueda unirse al clúster. Para ello, ObjectGrid utiliza un mecanismo de serie secreta compartida.

Todos los servidores ObjectGrid acuerdan una serie secreta compartida. Cuando un servidor se une al clúster, se le solicita que presente la serie secreta. Si la serie secreta del servidor que se une coincide con la del servidor presidente, el servidor que se une puede unirse al clúster; de lo contrario, se rechaza la petición de unión.

El envío de una serie secreta en texto normal no es seguro. La infraestructura de seguridad de ObjectGrid proporciona un plug-in de gestor de símbolos seguros que permite al servidor “proteger” la serie secreta antes de enviarla. Debe decidir cómo desea implementar la operación “segura”. ObjectGrid proporciona una implementación predefinida, en la que la operación “segura” se implementa para cifrar y firmar la serie secreta.

La serie secreta (authenticationSecret) se establece en el archivo `security.ogserver.props`:

- `authenticationSecret`: la serie secreta para desafiar al servidor. Cuando se inicia un servidor, debe presentar esta serie al servidor presidente. Si la serie secreta coincide con la del servidor presidente, este servidor puede unirse al clúster.

Plug-in SecureTokenManager

Un plug-in de gestor de símbolos seguros se representa mediante la interfaz com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager. A continuación, aparece la interfaz:

```
package com.ibm.websphere.objectgrid.security.plugins;
import com.ibm.websphere.objectgrid.security.ObjectGridSecurityException;
import com.ibm.websphere.objectgrid.security.SecurityConstants;
/**
* Esta interfaz la utilizan los servidores ObjectGrid para transformar un objeto en
* un símbolo seguro y viceversa. Un símbolo seguro es una matriz de bytes.
* Este es un ejemplo de uso posible: cuando un servidor se une al clúster,
* el servidor debe presentar una contraseña al servidor presidente en el
* clúster. Antes de enviar la contraseña fuera, el servidor que se une invoca
* el método generateToken(Object) para generar un símbolo para esta contraseña.
* El símbolo debe ser difícil de dividir para que la contraseña se pueda proteger de
* forma segura. El símbolo se enviará a través de la conexión. Normalmente, el símbolo
* está asociado con una indicación de la hora para impedir ataques de reproducción
* maliciosos. En el lado receptor, el servidor invoca el método verifyToken(byte[])
* para comprobar el símbolo y reconstruir el objeto correspondiente a partir del
* símbolo.
```

```
*
* ObjectGrid utiliza JCE para proporcionar una implementación por omisión de esta
* interfaz. En esta implementación, cuando se genera el símbolo, el objeto se cifra
* con una indicación de la hora y luego se firma. Para verificar un símbolo, la
* firma del símbolo se verifica y se descifra. Esta implementación necesitará un
* almacén de claves configurado en los servidores ObjectGrid para dar soporte al
* cifrado y descifrado de datos, y la firma y la verificación de firma. Utilice
* security.ogserver.props para los valores de clave de símbolo seguro.
*
* Una clase de implementación debe tener un constructor por omisión. Los usuarios
* pueden establecer la propiedad CustomSecureTokenManagerProps en el archivo de
* de propiedades de configuración de seguridad de servidor. Esta propiedad se
* establecerá en el objeto utilizando el método setProperties(String).
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see SecurityConstants#SECURE_TOKEN_MANAGER_CUSTOM_STRING
* @see SecurityConstants#SECURE_TOKEN_MANAGER_DEFAULT_STRING
*/
public interface SecureTokenManager {
/**
*
Establecer las propiedades definidas por el usuario en la fábrica
*
*
Este método se utiliza para establecer propiedades de SecureTokenManager adicionales
* en el objeto. Estas propiedades se pueden establecer utilizando la propiedad
* SecureTokenManagerProps en el archivo de propiedades de configuración de seguridad
* de servidor. De esta forma puede personalizar la fábrica.
*
* @param properties Propiedades definidas por el usuario
*/
void setProperties(String properties);
/**
* Genera el símbolo del objeto especificado.
*
* El símbolo generado será difícil de dividir.
*
* @param o el objeto que se va a proteger
*
* @return un símbolo que representa el objeto que se va a proteger
*
* @throws ObjectGridSecurityException si se produce una excepción durante
* la generación de la matriz de bytes de símbolos
*/
byte[] generateToken(Object o) throws ObjectGridSecurityException;
/**
* Verifica el símbolo y reconstruye el objeto.
*
* @param bytes la matriz de bytes de símbolos que representa el objeto protegido.
*
* @return el objeto protegido
*
* @throws ObjectGridSecurityException si se produce una excepción durante
* la verificación de la matriz de bytes de símbolos
*/
Object verifyToken(byte[] bytes) throws ObjectGridSecurityException;
}
```

El método generateToken(Object) parte de un objeto que se va a proteger y genera un símbolo que los demás no pueden entender. El método verifyTokens(byte[]) realiza el proceso inverso: convierte el símbolo de nuevo en el objeto original.

Una implementación de SecureTokenManager sencilla es utilizar un algoritmo de codificación sencillo (por ejemplo, el algoritmo XOR) para codificar el objeto en

forma serializada y, a continuación, utilizar el algoritmo de descodificación correspondiente para descodificar el símbolo. Esta implementación no es segura y es fácil de romper.

ObjectGrid proporciona una implementación predefinida para esta interfaz. La implementación no es una API pública y es transparente.

La implementación por omisión utiliza un par de claves para firmar y verificar la firma, y utiliza una clave secreta para cifrar el contenido. Para ello, todo servidor tiene un almacén de claves de tipo JCKES para almacenar el par de claves (clave privada y clave pública) y una clave secreta. El almacén de claves tiene que ser de tipo JCKES para almacenar claves secretas.

Estas claves se utilizan para cifrar y firmar o verificar la serie secreta en el extremo emisor. Asimismo, el símbolo se asocia con un tiempo de caducidad, por lo que caduca después de una determinada cantidad de tiempo. En el extremo receptor, los datos se verifican, descifran y comparan con la serie secreta del receptor. No se necesitan protocolos de comunicación tipo SSL entre un par de servidores para la autenticación, ya que las claves privadas y las claves públicas sirven al mismo propósito. No obstante, si la comunicación de los servidores no está cifrada, podrían robarse los datos pinchando la comunicación. Como el símbolo caduca pronto, la amenaza de ataques de reproducción se minimiza. Esta posibilidad se reduce significativamente si todos los servidores se despliegan detrás de un cortafuegos.

La desventaja de este enfoque es que los administradores de ObjectGrid tienen que generar claves y transportarlas a todos los servidores, lo que supondrá un problema de seguridad.

Configuraciones

Para utilizar el gestor de símbolos seguros, se deben configurar las siguientes propiedades en el archivo `security.ogserver.props`:

- Propiedad **secureTokenManagerType**: esta propiedad indica qué gestor de símbolos seguros se debe utilizar.
  - Si el valor es **ninguno**, no se utiliza ningún gestor de símbolos seguros.
  - Si el valor es **por omisión**, se utiliza el gestor de símbolos seguros proporcionado directamente por omisión.
  - Si el valor es **personalizado**, se utiliza el gestor de símbolos seguros proporcionado por el usuario.
- Propiedad **customSecureTokenManagerClass**: esta propiedad especifica la clase de implementación SecureTokenManager. Sólo se utiliza si el valor de secureTokenManagerType es ″personalizado″. La clase de implementación debe tener un constructor por omisión para poder crear una instancia.
- Propiedad **customSecureTokenManagerProps**: esta propiedad especifica las propiedades de SecureTokenManager personalizadas. Sólo se utiliza si el valor de secureTokenManagerType es ″personalizado″. El valor se establece en el objeto SecureTokenManager con el método setProperties(String).
- Si el valor de secureTokenManagerType se establece en el valor por omisión, se necesitarán las siguientes configuraciones para las claves de firma y cifrado:
  - `secureTokenKeyStore`: especifica el nombre de la vía de acceso de archivos del almacén de claves que almacena el par de claves pública-privada y la clave secreta.

- `secureTokenKeyStoreType`: especifica el tipo del almacén de claves, por ejemplo, JCKES. Puede establecer este valor según el proveedor JSSE (Java Secure Socket Extension) que utilice. No obstante, este almacén de claves debe poder dar soporte a las claves secretas.
- `secureTokenKeyStorePassword`: especifica la contraseña para proteger el almacén de claves.
- `secureTokenKeyPairAlias`: especifica el alias del par de claves pública-privada que se utiliza para firmar y verificar.
- `secureTokenKeyPairPassword`: especifica la contraseña para proteger el alias del par de claves que se utiliza para firmar y verificar.
- `secureTokenSecretKeyAlias`: especifica el alias de clave secreta que se utiliza para cifrar.
- `secureTokenSecretKeyPassword`: especifica la contraseña para proteger la clave secreta.
- `secureTokenCipherAlgorithm`: especifica el algoritmo que se utiliza para el cifrado. Puede establecer este valor según el proveedor de JSSE que utilice.
- `secureTokenSignAlgorithm`: especifica el algoritmo que se utiliza para firmar el objeto. Puede establecer este valor según el proveedor de JSSE que utilice.

Seguridad de pasarela

Una pasarela de gestión de ObjectGrid sirve como punto para delegar las peticiones de administración de cliente al servidor ObjectGrid. En este tema se describe cómo proteger el acceso a la pasarela.

El siguiente diagrama es un ejemplo. Si el cliente ObjectGrid desea obtener las estadísticas de un clúster, envía primero una petición a la pasarela. La pasarela envía esta petición a ambos servidores para obtener las estadísticas y, a continuación, combina las estadísticas. Las estadísticas combinadas se devuelven al cliente.

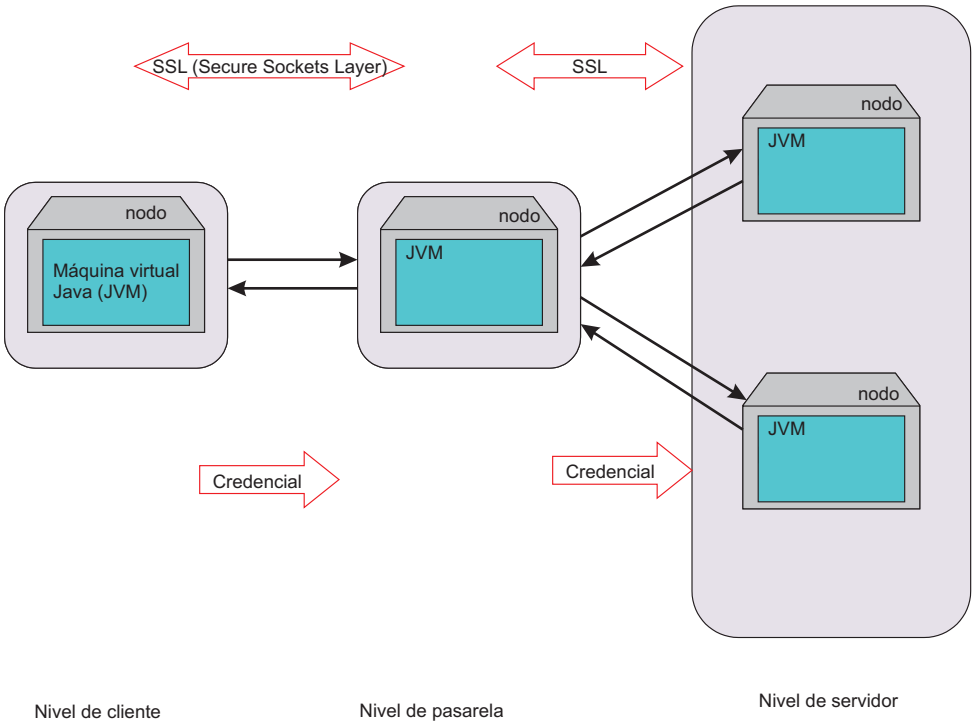


Figura 18. Seguridad de pasarela

La pasarela y la comunicación de servidor utilizan el mecanismo de comunicación de cliente-servidor ObjectGrid. La pasarela se trata como un cliente ObjectGrid. La comunicación del cliente y la pasarela se puede proteger mediante SSL. Esta posibilidad se debe a la capa de conector JMX, que es el proyecto de código fuente abierto mx4j. ObjectGrid requiere mx4j para que la pasarela funcione.

Para la autenticación, la pasarela propaga la credencial (por ejemplo, un ID de usuario y una contraseña) presentada por el cliente al servidor. La autenticación y la autorización se aplican en los servidores ObjectGrid.

La autenticación de certificados de cliente para el cliente de pasarela no está soportada.

Seguridad de servidor de pasarela

Un servidor de pasarela es un cliente ObjectGrid. Todos los aspectos de seguridad son los mismos que los de un cliente ObjectGrid. Consulte “Iniciar el servidor de pasarela de gestión” en la página 88 para obtener más información sobre cómo iniciar un servidor de pasarela desde una línea de mandatos.

El siguiente fragmento de código demuestra cómo iniciar la pasarela segura mediante programación:

```
// Obtener ClientSecurityConfiguration del archivo de propiedades de
// seguridad del cliente
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration("etc/test/security/security.client.props");
CredentialGenerator creGen = new UserPasswordCredentialGenerator("admin",
"xxxxxx");
csConfig.setCredentialGenerator(credGen);
// Inicializar la pasarela
ManagementGateway gateway = ManagementGatewayFactory.getManagementGateway();
```

```
gateway.setConnectorPort(namingPort);
gateway.setClusterName("cluster1");
gateway.setHost("localhost");
gateway.setPort("12503");
gateway.setTraceEnabled(true);
gateway.setTraceSpec("ObjectGrid=all=enabled");
gateway.setTraceFile("logs/GatewayTrace.log");
// Establecer el objeto ClientSecurityConfiguration
gateway.setCsConfig(csConfig);
// Iniciar la pasarela
gateway.startConnector();
```

En el código anterior, se crea un objeto ClientSecurityConfiguration y se establece en la instancia de ManagementGateway.

Seguridad de cliente de pasarela

El cliente de pasarela necesita pasar una credencial a un servidor de pasarela en el momento de realizar la conexión. El siguiente fragmento de código demuestra cómo pasar una credencial:

```
/**
* recuperar el estado del servidor de la pasarela
*/
public boolean retrieveServerStatus()
throws Exception {
String serverProtocol = "rmi";
String serverHost = "host";
String namingHost = "localhost";
String jndiPath = "/jmxconnector";
JMXServiceURL url = new JMXServiceURL("service:jmx:" + serverProtocol + "://"
+ serverHost + "/jndi/rmi://" + namingHost + ":" + namingPort + jndiPath);
// Crear el JMXCconnectorServer
JMXConnector cntor = JMXConnectorFactory.newJMXConnector(url, null);
// La correlación de entorno de conexión
Map environment = new HashMap();
// crear una credencial
UserPasswordCredential gatewayClientCred =
new UserPasswordCredential("admin", "admin1");
environment.put(JMXConnector.CREDENTIALS, gatewayClientCred);
// Conectar e invocar una operación en el MBeanServer remoto
try {
cntor.connect(environment);
}
catch (SecurityException x) {
// Cuidado. Credenciales erróneas.
throw x;
}
// Obtener un apéndice para el MBeanServer remoto
mbsc = cntor.getMBeanServerConnection();
Iterator it = mbsc.queryMBeans(
new ObjectName("ManagementServer:type=ObjectGrid,S=server1"),
null).iterator();
ObjectInstance oi = (ObjectInstance) it.next();
server1MBean = oi.getObjectName();
boolean status = ((Boolean) mbsc.invoke(
server1MBean,
"retrieveServerStatus",
new Object[] {},
new String[] {})).booleanValue();
return status;
}
```

En este fragmento de código, se crea un objeto gatewayClientCred y se coloca en el entorno. A continuación, se utiliza este entorno para conectarse al servidor de pasarela.

Si desea utilizar SSL para conectarse desde el cliente de pasarela al servidor de pasarela, debe utilizar las propiedades del sistema para almacenar el almacén de confianza y la contraseña del almacén de confianza. Por ejemplo, puede pasar las siguientes propiedades cuando inicie un cliente de pasarela.

- `-Djavax.net.ssl.trustStore=etc/test/security/client.public`
- `-Djavax.net.ssl.trustStorePassword=public`

Consulte el sitio Web de MX4J - Open Source Java Management Extensions para obtener más información.

Integración de la seguridad con WebSphere Application Server

ObjectGrid proporciona varias características de seguridad para integrarse con la infraestructura de seguridad de WebSphere Application Server.

Integración de la seguridad de ObjectGrid distribuido con WebSphere Application Server

Para el modelo de ObjectGrid distribuido, la integración de seguridad se puede realizar utilizando las siguientes clases:

- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator.`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential`

  .

Estas clases se describen en “Seguridad de cliente-servidor” en la página 145. A continuación, se muestra un ejemplo sobre cómo utilizar la clase WSTokenCredentialGenerator.

```
/**
* conectarse al servidor ObjectGrid.
*/
protected ClientClusterContext connect() throws ConnectException {
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration(proFile);
CredentialGenerator gen = getWSCredGen();
csConfig.setCredentialGenerator(gen);
return objectGridManager.connect(csConfig, null);
}
/**
* Obtener un WSTokenCredentialGenerator
*
private CredentialGenerator getWSCredGen() {
WSTokenCredentialGenerator gen = new WSTokenCredentialGenerator(
WSTokenCredentialGenerator.RUN_AS_SUBJECT);
return gen;
}
```

Por otro lado, WSTokenAuthentication se puede utilizar como autenticador para autenticar el objeto WSTokenCredential.

Integración de la seguridad de ObjectGrid local con WebSphere Application Server

Para el modelo de ObjectGrid local, la integración de seguridad se puede realizar utilizando las dos clases siguientes:

- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl`

Para obtener más información sobre estas clases, consulte “Seguridad de ObjectGrid local” en la página 164. Puede configurar la clase WSSubjectSourceImpl como el plug-in SubjectSource y la clase WSSubjectValidationImpl como el plug-in SubjectValidation.

Receptores

ObjectGrid proporciona interfaces del tipo receptor que pueden ampliarse. Las ampliaciones pueden guiarle a través de la interfaz de ampliación y describen las operaciones que se ejecutan en una instancia de ObjectGrid o una instancia de correlación.

Interfaz ObjectGridEventListener

Utilice la interfaz ObjectGridEventListener para recibir notificaciones cuando se produzcan sucesos significativos en una ObjectGrid. Estos sucesos incluyen la inicialización de ObjectGrid, el comienzo de una transacción, la finalización de una transacción y la destrucción de una ObjectGrid. Para escuchar estos sucesos, cree una clase que implemente la interfaz ObjectGridEventListener y añádala a la ObjectGrid.

Interfaz ObjectGridEventListener

La interfaz ObjectGridEventListener tiene los siguientes métodos. Se llama a estos métodos cuando determinados sucesos significativos ocurren en la ObjectGrid.

```
/**
* Este método se invoca cuando la ObjectGrid se ha inicializado.
* Una instancia de Session utilizable se pasa a este receptor para
* permitir la reproducción opcional de una LogSequence en una Map.
*
* @param session Instancia de Session con la que está asociada este
*  receptor.
*/
void initialize(Session session);
/**
* Este suceso indica el inicio de una transacción (sesión).
* Se proporciona una versión de serie del TxID para
* correlacionarlo con el final de la transacción (sesión),
* si desea utilizar esta versión. El tipo de transacción
* (sesión) también se proporciona mediante el parámetro
* booleano isWriteThroughEnabled.
*
* @param txid versión de serie del TxID
* @param isWriteThroughEnabled Distintivo booleano que indica si
* la Session se ha iniciado mediante veginNoWriteThrough
*/
void transactionBegin(String txid, boolean isWriteThroughEnabled);
/**
* Este suceso indica el final de una transacción (sesión). Una versión del
* TxID se facilita para correlacionarla con el el inicio de la transacción
* (sesión), si así se desea. Los cambios también están soportados. Los usos
```

```
* típicos de este suceso son para la invalidación personalizada de igual
* o "push" de compromiso de igual. Este receptor de sucesos
* proporciona los cambios. Las llamadas
* a este método se realizan después del compromiso y se secuencian
* para enviarlas de una en una, no en paralelo. El orden de sucesos
* es el orden de compromiso.
*
* @param txid versión de serie del TxID
* @param isWriteThroughEnabled distintivo booleano que indica
* si la Session se ha
* iniciado a través de beginNoWriteThrough
* @param committed distintivo booleano que indica si la Session
* se ha comprometido
* (true) o retrotraído (false)
* @param changes Collection de LogSequences que se han
* procesado para la Session actual. */
void transactionEnd(String txid, boolean isWriteThroughEnabled,
boolean committed, Collection /*/* <LogSequence> */*/ changes);
/**
* Se invocará a este método cuando se haya destruido la ObjectGrid. Es
* el caso contrario de la inicialización. Cuando se llama a este método,
* el ObjectGridEventListener puede liberar cualquier recurso que utilice.
*/
void destroy();
```

Adición y eliminación de los objetos ObjectGridEventListeners

Un objeto puede tener varios ObjectGridEventListeners. Existen dos métodos en la ObjectGrid que permiten que se añadan los ObjectGridEventListeners. Los ObjectGridEventListeners añadidos también pueden eliminarse de una ObjectGrid.

El método addEventListener puede utilizarse para añadir un ObjectGridEventListener a una ObjectGrid.

```
/**
* Añadir un EventListener a la Session. Los sucesos significativos se
* comunicarán a los receptores interesados a través de este retorno de
* llamada.
* Se pueden registrar varios receptores de sucesos, sin un orden
* determinado de las notificaciones de sucesos.
*
* Tenga en cuenta que se puede invocar a este método antes y después del
* método {@link ObjectGrid#initialize()}.
*
* @param cb Instancia de ObjectGridEventListener
*/
void addEventListener(ObjectGridEventListener cb);
```

Para añadir una lista de ObjectGridEventListeners, utilice el método setEventListeners:

```
/**
* Éste sobrescribe la lista actual de retornos de llamada y la sustituye
* por la lista proporcionada de retornos de llamada.
*
* Tenga en cuenta que se puede invocar a este método antes y después del
* método {@link ObjectGrid#initialize()}.
* @param callbacks
*/
void setEventListeners(List callbacks);
```

Para eliminar un ObjectGridEventListener de una ObjectGrid utilice el método removeEventListener:

```
/**
* Elimina un EventListener de la Session. Si el EventListener deseado
* no se encuentra en la Session, no se devolverá ningún error.
```

```
*
* Tenga en cuenta que se puede invocar a este método antes y después del
* método {@link ObjectGrid#initialize()}.
* @param cb Instancia de ObjectGridEventListener
*/
void removeEventListener(ObjectGridEventListener cb);
```

Creación de un receptor de sucesos de ObjectGrid personalizado

Para utilizar un receptor de sucesos de ObjectGrid personalizado, cree primero una clase que implemente la interfaz com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener. Añada el receptor personalizado a una ObjectGrid para recibir la notificación de sucesos significativos. Un ObjectGridEventListener puede configurarse mediante programación o con XML:

- **Mediante programación.** Presuponga que el nombre de clase del receptor de sucesos de ObjectGrid 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 una ObjectGrid:

  ```
  ObjectGridManager objectGridManager =
   ObjectGridManagerFactory.getObjectGridManager();
  ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
  MyObjectGridEventListener myListener = new MyObjectGridEventListener();
  myGrid.addEventListener(myListener);
  ```

- **Mediante XML.** Un ObjectGridEventListner también puede configurarse utilizando XML. El siguiente XML crea una configuración equivalente al receptor de sucesos de ObjectGrid creado mediante programación que se ha descrito. 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" />
  </objectGrid>
  </objectGrids>
  </objectGridConfig>
  ```

  Proporcione este archivo al ObjectGridManager para facilitar la creación de esta configuración. El siguiente fragmento de código demuestra cómo se crea una ObjectGrid utilizando este archivo XML. La ObjectGrid creada tiene un ObjectGridEventListener establecido en la ObjectGrid myGrid.

  ```
  ObjectGridManager objectGridManager =
   ObjectGridManagerFactory.getObjectGridManager();
  ObjectGrid myGrid =
  objectGridManager.createObjectGrid("myGrid", new URL(
  "file:etc/test/myGrid.xml"), true, false);
  ```

Detección de los cambios de una Map

El método transactionEnd en la interfaz ObjectGridEventListener es muy útil para las aplicaciones interesadas en supervisar las entradas de las Maps locales. Una aplicación puede añadir estos receptores y, a continuación, utilizar el método transactionEnd para ver cuándo se modifican las transacciones. Por ejemplo, si la ObjectGrid está funcionando en la modalidad distribuida, una aplicación puede detectar los cambios de

entrada. Suponga que las entradas duplicadas eran de los precios más recientes de las cotizaciones. El receptor puede estar pendiente y detectar estos cambios al llegar y actualizar una segunda Map que mantenga el valor de una posición en una cartera. El receptor debe realizar todos los cambios utilizando la Session proporcionada al receptor en el método initialize en la interfaz ObjectGridEventListener. El receptor puede distinguir entre los cambios locales y los cambios remotos de entrada generalmente comprobando si la transacción es WriteThrough. Los cambios de entrada de ObjectGrids de igual siempre son WriteThrough.

Interfaz MapEventListener

Utilice la interfaz MapEventListener para recibir sucesos significativos sobre una correlación. Los sucesos se envían al MapEventListener cuando una entrada se desaloja de la correlación y cuando se completa la precarga de una correlación.

Interfaz MapEventListener

La interfaz MapEventListener tiene los siguientes métodos. Implemente la interfaz com.ibm.websphere.objectgrid.plugins.MapEventListener para crear un MapEventListener personalizado.

```
/**
* Este método se invoca cuando la entrada especificada se desaloja de
* la correlación. El desalojo puede haber ocurrido mediante el proceso
* de Evictor o mediante uno de los métodos invalidate de
* ObjectMap.
*
* @param key Clave de la entrada de correlación que se ha desalojado.
* @param value Valor aparecido en la entrada de correlación desalojada.
* El objeto de valor no debe modificarse.
*
*/
void entryEvicted(Object key, Object value);
/**
* Se invoca a este método cuando la precarga de esta correlación se ha
* completado.
*
* @param t Objeto Throwable que indica si la precarga se ha completado sin
* que ninguna Throwable haya ocurrido durante la precarga de la correlación.
* Una referencia nula indica que se ha completado la precarga sin que
* ocurran objetos Throwable durante la precarga de la correlación.
*/
void preloadCompleted( Throwable t );
```

Adición y eliminación de MapEventListeners

El siguiente método BackingMap permite que se añadan MapEventListeners a una correlación y se eliminen de ésta.

```
/**
* Añade un MapEventListener a esta BackingMap.
*
* Tenga en cuenta que se puede invocar a este método antes y después del
* método ObjectGrid.initialize().
* @param eventListener Referencia no nula a un MapEventListener que se
*  vaya a añadir a la lista.
*
* @throws IllegalArgumentException si eventListener es nulo.
*
* @see MapEventListener
*/
public void addMapEventListener(MapEventListener
 eventListener );
/**
* Establece la lista de objetos MapEventListener.
*
```

```
* Si esta BackingMap ya tiene una List de
* MapEventListeners, esa lista se sustituirá por la
* List pasada como argumento en la invocación actual
* de este método. Se puede llamar a este método antes y
* después del método ObjectGrid.initialize().
*
* @param eventListenerList Referencia no nula a una List de
* objetos MapEventListener.
*
* @throws IllegalArgumentException se genera si
* eventListenerList es nulo
* o el eventListenerList contiene una referencia
* nula o un objeto que no sea una instancia de
* MapEventListener.
*
* @see MapEventListener
*/
public void setMapEventListeners( List /*MapEventListener*/
 eventListenerList );
/**
* Elimina un MapEventListener de esta BackingMap.
*
* Tenga en cuenta que se puede invocar a este método antes y después del
* método ObjectGrid.initialize().
*
* @param eventListener Referencia no nula a un receptor de sucesos
* que se haya añadido previamente al invocar los métodos
* addMapEventListener(MapEventListener) o
* setMapEventListeners(List) de este interfaz.
*
* @throws IllegalArgumentException si eventListener es nulo.
*
* @see MapEventListener
*/
public void removeMapEventListener(MapEventListener eventListener );
```

Creación de un MapEventListener

Para crear un MapEventListener personalizado, implemente la interfaz com.ibm.websphere.objectgrid.plugins.MapEventListener. Para utilizar el MapEventListener, añádalo a una BackingMap. Se puede crear y configurar un MapEventListener mediante programación o con XML:

- **Mediante programación**. El nombre de clase para el MapEventListener personalizado es la clase com.company.org.MyMapEventListener. Esta clase implementa la interfaz MapEventListener. El siguiente fragmento de código crea el MapEventListener personalizado y lo añade a una BackingMap:

  ```
  ObjectGridManager objectGridManager =
   ObjectGridManagerFactory.getObjectGridManager();
  ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
  BackingMap myMap = myGrid.defineMap("myMap");
  MyMapEventListener myListener = new MyMapEventListener();
  myMap.addMapEventListener(myListener);
  ```

- **Creación de XML**. También se puede configurar un MapEventListner utilizando XML. El siguiente XML consigue una configuración equivalente a la creación anterior realizada mediante programación. 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>
</backingMapPluginCollection>
</objectGridConfig>
```

Al proporcionar este archivo al ObjectGridManager, se facilita la creación de esta configuración. El siguiente fragmento de código muestra cómo se crea una ObjectGrid utilizando este archivo XML. La ObjectGrid recién creada tiene un MapEventListener establecido en la BackingMap myMap.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
objectGridManager.createObjectGrid("myGrid", new URL(
"file:etc/test/myGrid.xml"), true, false);
```

Desalojadores

ObjectGrid proporciona una mecanismo de desalojador por omisión. También puede proporcionar un mecanismo de desalojador conectable.

Un *desalojador* (evictor) controla la pertenencia de las entradas en todas las BackingMap. El desalojador por omisión utiliza una política de desalojo de *tiempo de duración* para todas las BackingMap. Si proporciona un mecanismo de desalojador conectable, generalmente utiliza una política de desalojo basada en el número de entradas en lugar del tiempo. En este tema se describen ambos tipos de desalojadores.

Desalojador de tiempo de duración por omisión

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

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

- **NONE** indica que una entrada de la BackingMap nunca caduca. El desalojador TTL no desaloja estas entradas.
- **CREATION_TIME** indica que la hora en que se ha creado una entrada se utiliza en el cálculo de la hora de caducidad.
- **LAST_ACCESS_TIME** indica que la hora a la que se ha accedido a una entrada por última vez se utiliza en el cálculo de la hora de caducidad.

Si el atributo ttlType no se establece en una BackingMap, se utiliza el tipo por omisión de **NONE** para que el desalojador TTL no desaloje ninguna entrada. Si el

atributo ttlType se establece en **CREATION_TIME** o **LAST_ACCESS_TIME**, el valor del atributo de tiempo de duración de la BackingMap se añade a la hora de creación o a la última hora de acceso para calcular la hora de caducidad. La precisión de tiempo del atributo de correlación de tiempo de duración es en segundos. Un valor de `0` para el atributo de tiempo de duración es un valor especial que se utiliza para indicar que la entrada de correlación puede tener una duración permanente, es decir, la entrada permanece en la correlación hasta que la aplicación elimina o invalida explícitamente la entrada de correlación.

Especificación de atributos para los desalojadores TTL

Los desalojadores TTL están asociados a las instancias BackingMap. El siguiente fragmento de código demuestra cómo puede utilizarse la interfaz BackingMap para establecer los atributos necesarios de modo que cuando se crean todas las entradas, éstas tienen una hora de caducidad establecida en diez minutos después de su creación.

```
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` ya que indica que el tiempo de duración está en segundos. El código anterior debe ejecutarse antes que se invoque al método initialize en la instancia de ObjectGrid. Estos atributos BackingMap no pueden modificarse después de que se inicialice la instancia de ObjectGrid. Después de que se ejecute el código, cualquier entrada que se inserte en la BackingMap myMap tiene una hora de caducidad. Después de que se alcance la hora de caducidad, el desalojador TTL depura la entrada.

Si una aplicación requiere que la hora de caducidad se establezca en la última hora de acceso más diez minutos, se debe modificar una línea del código anterior. El argumento que se pasa al método setTtlEvictorType se cambia de `TTLType.CREATION_TIME` a `TTLType.LAST_ACCESS_TIME`. Con este valor, la hora de duración se calcula como la última hora de acceso más diez minuto. Cuando se crea una entrada por primera vez, la última hora de acceso es la hora de creación.

Cuando se utiliza `TTLType.LAST_ACCESS_TIME`, las interfaces ObjectMap y JavaMap pueden utilizarse para alterar temporalmente el valor de tiempo de duración de BackingMap. Este mecanismo permite que una aplicación utilice un valor de tiempo de duración distinto para cada entrada que se cree. Presuponga que el fragmento de código anterior se ha utilizado para establecer el atributo ttlType en `LAST_ACCESS_TIME` y el valor de tiempo de duración se ha establecido en 10 minutos en BackingMap. Una aplicación entonces puede alterar temporalmente el valor de tiempo de duración de todas las entradas ejecutando el siguiente código anterior a la creación o modificación de 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 ObjectMap. La variable oldTimeToLive1 se establece en 600 ya que el valor de tiempo de duración de BackingMap se utiliza como valor por omisión si no se ha llamado previamente al método setTimeToLive en ObjectMap.

La entrada con la clave `key2` tiene un tiempo de caducidad del tiempo de inserción más veinte minutos como resultado de la llamada de método setTimeToLive(1200) en ObjectMap. La variable oldTimeToLive2 se establece en 1800 ya que el valor de tiempo de duración de la invocación anterior del método ObjectMap.setTimeToLive ha establecido el tiempo de duración en 1800.

El ejemplo anterior muestra la inserción de dos entradas de correlación en la correlación myMap para los valores de clave key1 y key2. Posteriormente, puede que la aplicación de una nueva hebra desee actualizar estas entradas de correlación con nuevos valores de correlación. No obstante, la aplicación desea mantener los valores de tiempo de duración 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 duración utilizando una constante definida en la interfaz ObjectMap a estos efectos:

```
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();
```

Como el valor especial ObjectMap.USE_DEFAULT se utiliza en la llamada al método setTimeToLive, key1 mantiene su valor de tiempo de duración de 1800 segundos y key2 mantiene su valor de tiempo de duración de 1200 segundos, ya que son los valores que se utilizaron cuando la transacción anterior insertó estas entradas de correlación.

El ejemplo anterior también muestra la inserción de una nueva entrada de correlación para key3. En este caso, el valor especial USE_DEFAULT indica que se debe utilizar el valor por omisión del valor de tiempo de duración de esta correlación. El atributo BackingMap del tiempo de duración define el valor por omisión. Consulte “Atributos de BackingMap” en la página 113 para obtener más información sobre cómo se define el atributo de tiempo de duración en la BackingMap.

Consulte la documentación de la API para obtener información sobre el método setTimeToLive de las interfaces ObjectMap y JavaMap. Le avisa de que se genera una excepción `IllegalStateException` si el método BackingMap.getTtlEvictorType() devuelve cualquier cosa que no sea valor `TTLType.LAST_ACCESS_TIME`. ObjectMap y JavaMap sólo pueden utilizarse para alterar temporalmente el valor de tiempo de duración cuando utilice el tipo de desalojador TTL `LAST_ACCESS_TIME`. Este método no puede utilizarse para alterar temporalmente el valor de tiempo de duración cuando utilice el tipo de desalojador TTL `CREATION_TIME` o `NONE`.

Utilización de un archivo XML para especificar los atributos para el desalojador TTL

En lugar de utilizar la interfaz BackingMap para establecer mediante programación los atributos BackingMap que vaya a utilizar el desalojador TTL, se puede utilizar un archivo XML para configurar todas las BackingMap. El siguiente código demuestra cómo establecer estos atributos para tres BackingMaps distintas:

```
<?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" timeToLive="1800" />
<backingMap name="map3" ttlEvictorType="CREATION_TIME" timeToLive="1200" />
</objectgrid>
</objectGrids>
```

El ejemplo anterior muestra que la BackingMap map1 utiliza un tipo de desalojador TTL `NONE`. La BackingMap map2 utiliza un tipo de desalojador TTL `LAST_ACCESS_TIME` y tiene un valor de tiempo de duración de 1800 segundos o 30 minutos. La BackingMap map3 está definida para que utilice un tipo de desalojador TTL `CREATION_TIME` y tiene un valor de tiempo de duración de 1200 segundos o 20 minutos.

Desalojadores conectables opcionales

El desalojador TTL por omisión 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. Un desalojador conectable opcional puede utilizarse para desalojar entradas en función del número de entradas que existen en lugar de la hora. Los 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 utilizadas recientemente* 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 *utilizadas menos frecuentemente* para decidir qué entradas se van a desalojar cuando la BackingMap supere un número máximo de entradas.

La BackingMap informa a un desalojador de la creación, modificación o eliminación de entradas en una transacción. La 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 lograr la estrategia de desalojo que desee.

Especificación de un desalojador conectable

Como los desalojadores están asociados con BackingMaps, la interfaz BackingMap se utiliza para especificar el desalojador conectable que se va a utilizar. El siguiente fragmento de código es un ejemplo para la especificación de un desalojador LRUEvictor para la BackingMap map1 y un desalojador LFUEvictor para la BackingMap map2:

```
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 de `1000`. El desalojador LFUEvictor se utiliza para la BackingMap map2 con un número máximo de entradas de `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 alguna entrada requiere que sea desalojada. El tiempo de inactividad se especifica en segundos. Un valor de 15 segundos constituye un compromiso adecuado entre el impacto en el rendimiento y cómo evitar que BackingMap crezca demasiado. El objetivo es utilizar el mayor tiempo posible de inactividad 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 inicio es utilizar el diez por ciento del número máximo de entradas como número de colas. La utilización de un número primo es generalmente mejor que la utilización de uno que no lo sea.

El método setNumberOfHeaps establece la propiedad de LFUEvictor para establecer cuántos objetos de almacenamiento dinámico binarios utiliza LFUEvictor para gestionar la información 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 inicio, y un número primo es generalmente mejor que uno que no lo sea.

Utilización de XML para especificar 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:

```
<?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="establecer el tamaño máximo del desalojador LRU">
<property name="sleepTime" type="int" value="15"
description="tiempo de inactividad de la hebra de desalojador" />
<property name="numberOfLRUQueues" type="int" value="53"
description="establecer el número de colas de LRU" />
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="LFU">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
<property name="maxSize" type="int" value="2000"
description="establecer el tamaño máximo del desalojador LFU">
<property name="sleepTime" type="int" value="15"
description="tiempo de inactividad de la hebra de desalojador" />
<property name="numberOfHeaps" type="int" value="211"
description="establecer el número de almacenamientos dinámicos de LFU" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Escritura de un desalojador personalizado

ObjectGrid puede ampliarse para que utilice un algoritmo de desalojo. Puede crear un desalojador personalizado que implemente la interfaz com.ibm.websphere.objectgrid.plugins.Evictor. A continuación, aparece la interfaz:

```
public interface Evictor
{
void initialize(BackingMap map, EvictionEventCallback callback);
void destroy();
void apply(LogSequence sequence);
}
```

- El método initialize se invoca durante la inicialización del objeto BackingMap. Este método inicializa un plug-in Evictor con una referencia a BackingMap y otra referencia a un objeto que implemente la interfaz com.ibm.websphere.objectgrid.plugins.EvictionEventCallback.
- El método apply se invoca cuando las transacciones que tienen acceso a una o más entradas de BackingMap estén comprometidas. Se pasa al método apply

una referencia a un objeto que implemente la interfaz com.ibm.websphere.objectgrid.plugins.LogSequence. La interfaz LogSequence permite que un plug-in Evictor determine qué entradas de BackingMap ha creado, modificado o eliminado la transacción. Un Evictor utiliza esta información para decidir qué entradas se van a desalojar y cuándo.

- El método destroy se invoca cuando la BackingMap se esté destruyendo. Este método permite que un Evictor finalice cualquier hebra que pueda haber creado.

La interfaz EvictionEventCallback tiene los métodos siguientes:

```
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);
}
```

Un plug-in Evictor utiliza los métodos EvictionEventCallback para volver a llamar a la infraestructura de ObjectGrid del modo siguiente:

- Un desalojador utiliza el método setEvictorData para solicitar a la infraestructura utilizada que almacene y asocie algún objeto de desalojador que cree con la entrada indicada por el argumento de clave. Los datos son específicos del desalojador y están determinados por la información que el desalojador necesita para implementar el algoritmo que está utilizando. Por ejemplo, en el caso de un algoritmo de utilizado menos frecuentemente, el desalojador mantiene un recuento en el objeto de datos del desalojador para hacer un seguimiento sobre cuántas veces se invoca al método apply con un LogElement que se refiera a una entrada de una determinada clave.
- Un desalojador utiliza el método getEvictorData para recuperar los datos que haya pasado al método setEvictorData durante una invocación del método apply anterior. Si no se encuentran los datos del desalojador para el argumento de clave especificado, se devolverá un objeto `KEY_NOT_FOUND` especial definido en la interfaz EvictorCallback.
- Un desalojador utiliza el método evictMapEntries 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. Asimismo, la misma instancia de EvictorData que se pasa al método setEvictorData debe estar en el parámetro de lista de datos del desalojador de este método. El método getKey de la interfaz EvictorData se utiliza para determinar qué entrada de correlación se debe desalojar. La entrada de correlación se desaloja si la entrada de antememoria contiene actualmente la misma instancia de EvictorData exacta que hay en la lista de datos del desalojador para esta entrada de antememoria.
- Un desalojador utiliza el método evictEntries para solicitar el desalojo de una o más entradas de correlación. Este método se utiliza sólo si el objeto que se pasa al método setEvictorData no implementa la interfaz com.ibm.websphere.objectgrid.plugins.EvictorData.

ObjectGrid invoca el método apply de la interfaz Evictor *después* de que finalice una transacción. Todos los bloqueos de transacciones que ha adquirido la transacción finalizada dejan de mantenerse. En teoría, varias hebras pueden llamar al método apply al mismo tiempo y cada hebra puede completar su propia transacción. Como los bloqueos de transacciones los ha liberado la transacción finalizada, el método apply debe proporcionar su propia sincronización para garantizar que el método apply sea seguro para las hebras.

El motivo de implementar la interfaz EvictorData y utilizar el método evictMapEntries en lugar del método evictEntries es para cerrar una posible ventana de temporización. Considere la siguiente secuencia de sucesos:

1. La transacción 1 finaliza y llama al método apply con una LogSequence que suprime la entrada de correlación de la clave 1.
2. La transacción 2 finaliza y llama al método apply con una LogSequence que inserta una nueva entrada de correlación para la clave 1. En otras palabras, la transacción 2 vuelve a crear la entrada de correlación que se suprimió en la transacción 1.

Como el desalojador se ejecuta de forma asíncrona con respecto a las hebras que ejecutan transacciones, puede que cuando el desalojador decida desalojar la clave 1, desaloje la entrada de correlación que existía antes de que terminara la transacción 1 o que desaloje la entrada de correlación que se volvió a crear en la transacción 2. Para eliminar las ventanas de temporización y la ambigüedad de qué versión de la entrada de correlación de la clave 1 va a desalojar el desalojador, implemente la interfaz EvictorData mediante el objeto que se ha pasado al método setEvictorData. Utilice la misma instancia de EvictorData el tiempo que dure una entrada de correlación. Cuando se suprime esa entrada de correlación y se vuelve a crear en otra transacción, el desalojador debe utilizar una nueva instancia de la implementación de EvictorData. Si utiliza la implementación de EvictorData y el método evictMapEntries, el desalojador puede garantizar que la entrada de correlación se desalojará si y sólo si la entrada de antememoria asociada con la correlación contiene la instancia de EvictorData correcta.

Las interfaces Evictor y EvictonEventCallback permiten que una aplicación conecte un desalojador que implemente un algoritmo definido por el usuario para el desalojo. El siguiente fragmento de código ilustra cómo implementar 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 del desalojador
 evictorThread = new Thread( this );
String threadName = "MyEvictorForMap-" + bm.getName();
 evictorThread.setName( threadName );
 evictorThread.start();
}
```

El código anterior guarda las referencias a la correlación y a los objetos de retorno de llamada en variables de instancia de modo estén disponibles para los métodos apply y destroy. En este ejemplo, se crea una lista enlazada que se utiliza como una cola *primero en entrar, primero en salir* para implementar un algoritmo LRU (menos utilizado recientemente). Se genera una hebra y se mantiene una referencia a ésta como una variable de instancia. Al mantener esta referencia, el método destroy puede interrumpirse y terminar la hebra generada.

Si se ignoran los requisitos de sincronización para hacer que la hebra de código sea segura, el siguiente fragmento de código ilustra cómo puede implementarse el método apply 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.getCacheEntry().getKey();
  LogElement.Type type = elem.getType();
  if ( type == LogElement.INSERT )
{
   // insertar el proceso aquí añadiéndolo a la parte anterior 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 )
{
   // actualizar el proceso aquí moviendo el objeto EvictorData a
   // la parte anterior de la cola.
   EvictorData data = evictorCallback.getEvictorData(key);
   queue.remove(data);
   queue.addFirst(data);
}
  else if ( type == LogElement.DELETE || type == LogElement.EVICT )
{
   // eliminar el proceso eliminando el objeto EvictorData
   // de la cola.
   EvictorData data = evictorCallback.getEvictorData(key);
   if ( data == EvictionEventCallback.KEY_NOT_FOUND )
{
    // Aquí se presupone que la hebra del desalojador asíncrono
    // ha desalojado la entrada de correlación antes de que esta hebra pudiese
    // procesar la petición de LogElement. Por lo tanto, es posible que
    // no necesite hacer nada cuando esto ocurra.
}
else
{
    // Se ha encontrado la clave. Por lo tanto, procese los datos del desalojador.
    if ( data != null )
{
     // Ignorar el valor nulo devuelto por el método remove ya que
     // la hebra del desalojador generada ya puede haberlo eliminado de la cola.
     // Pero, este código es necesario en el caso de que no haya sido la
     // hebra del desalojador la que haya generado este LogElement.
     queue.remove( data );
}
else
{
     // Según como escriba el Evictor, esta posibilidad
     // puede que no exista o puede indicar un defecto del desalojador
     // debido a una lógica inadecuada de sincronización de la hebra.
}
}
}
}
}
```

El proceso de inserción del método apply generalmente maneja la creación de un objeto de datos del desalojador que se pasa al método setEvictorData de la interfaz EvictionEventCallback. Como este desalojador ilustra una implementación LRU, EvictorData también se añade a la parte anterior de la cola creada por el método initialize. El proceso de actualización del método apply generalmente actualiza el objeto de datos del desalojador creado por alguna invocación anterior del método apply (por ejemplo, por el proceso de inserción del método apply). Como este desalojador es una implementación LRU, es necesario que mueva el objeto EvictorData de su posición actual de cola a la parte anterior de la cola. El desalojador generado elimina el último objeto EvictorData de la cola ya que el último elemento de la cola representa la entrada menos utilizada recientemente. Se presupone que el objeto EvictorData contiene un método getKey de modo que la hebra de desalojador conozca las claves de entrada que es necesario desalojar. Tenga en cuenta que en este ejemplo se ignoran los requisitos de sincronización para que el código sea seguro. Un desalojador personalizado real es más complejo porque debe hacer frente a los cuellos de botella relacionados con el rendimiento y la sincronización que ocurren como resultado de los puntos de sincronización.

Los siguientes fragmentos de código ilustran los métodos destroy y run de la hebra ejecutable que ha generado el método initialize.

```
// El método destroy simplemente interrumpe la hebra generada por el
// método initialize.
public void destroy()
{
 evictorThread.interrupt();
}
// Este es el método run de la hebra generada por el método initialize.
public void run()
{
 // Repetir en bucle hasta que el método destroy interrumpa esta hebra.
 boolean continueToRun = true;
 while ( continueToRun )
{
try
{
   // Pasar a inactividad durante un tiempo antes de barrer la cola.
   // La propiedad sleepTime de un desalojador es una buena candidata
   // para que se establezca.
   Thread.sleep( sleepTime );
   int queueSize = queue.size();
   // Desalojar las entradas si el tamaño de cola ha excedido el
   // tamaño máximo. Obviamente, el tamaño máximo
   // sería otra propiedad del desalojador.
   int numToEvict = queueSize - maxSize;
if ( numToEvict > 0 )
{
    // Eliminar de la parte final de la cola ya que es aquí donde está la
    // entrada utilizada menos 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 )
{
      // Esta cola está vacía.
      queueSize = 0;
}
```

```
}
    // Solicitar el desalojo si la lista de claves no está vacía.
    if ( ! evictList.isEmpty() )
{
evictorCallback.evictMapEntries( evictList );
}
}
}
  catch ( InterruptedException e )
{
   continueToRun = false;
}
 } // fin del bucle While
} // fin de ejecución del método.
```

Interfaz RollBackEvictor opcional

La interfaz com.ibm.websphere.objectgrid.plugins.RollbackEvictor la puede implementar opcionalmente un plug-in Evictor. Al implementar esta interfaz, se puede invocar un desalojador no sólo cuando se comprometen las transacciones, sino también cuando se retrotraen.

```
public interface RollbackEvictor
{
void rollingBack( LogSequence ls );
}
```

El método apply sólo se invoca si se compromete una transacción. Si se retrotrae una transacción y el desalojador implementa la interfaz RollbackEvictor, se invoca el método rollingBack. Si no se implementa la interfaz RollbackEvictor y se retrotrae la transacción, no se invocan el método apply y el método rollingBack.

Cargadores

Un cargador de ObjectGrid es un componente conectable que permite que una correlación ObjectGrid se comporte como una antememoria que generalmente se mantiene en un almacenamiento persistente en el mismo sistema o en 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 origen de datos lo que permite que las antememorias basadas en concentradores se creen utilizando ObjectGrid. Un cargador posee la lógica para leer y grabar datos desde y a un almacenamiento persistente.

Un cargador es un plug-in para una correlación de respaldo ObjectGrid. Sólo se puede asociar un cargador a una correlación de respaldo determinada. Cada correlación de respaldo tiene su propia instancia de cargador. La correlación de respaldo solicita los datos que no contiene de su cargador. Los cambios realizados en la correlación se pasan al cargador. El plug-in del cargador permite que la correlación de respaldo traslade los datos entre la correlación y el almacenamiento persistente.

Conexión de un cargador

El siguiente fragmento de código ilustra cómo un cargador suministrado por la aplicación se conecta a la correlación de respaldo para map1 utilizando la API ObjetGrid:

```
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 );
```

Se presupone que `MyLoader` es la clase suministrada por la aplicación que implementa la interfaz com.ibm.websphere.objectgrid.plugins.Loader. Debido a que la asociación de un cargador con una correlación de respaldo no se puede modificar después de inicializar ObjectGrid, el código se debe ejecutar antes de invocar el método `initialize` de la interfaz ObjectGrid que se está invocando. Se produce una excepción `IllegalStateException` en una llamada al método `setLoader` si se invoca después de que se haya producido la inicialización.

El cargador suministrado por la aplicación puede tener establecidas las propiedades. En el ejemplo, se utiliza el cargador MyLoader para leer y grabar datos de una tabla en una base de datos relacional. El cargador debe tener el nombre de la base de datos y el nivel de aislamiento SQL que se ha de utilizar. El cargador MyLoader tiene los métodos `setDataBaseName` y `setIsolationLevel` que permiten que la aplicación establezca estas dos propiedades del cargador.

También se puede conectar un cargador suministrado por la aplicación utilizando un archivo XML. El ejemplo siguiente ilustra cómo se conecta el cargador MyLoader en la correlación de respaldo map1 con el mismo nombre de base de datos y propiedades de nivel de aislamiento del cargador establecidas.

```
<?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="nombre de base de datos" />
<property name="isolationLevel" type="java.lang.String"
value="read committed" description="nivel de aislamiento" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Implementar la interfaz del cargador

Un cargador suministrado por la aplicación debe implementar la interfaz *com.ibm.websphere.objectgrid.plugins.Loader*. La interfaz del cargador tiene la definición 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;
}
```

Cada uno de los apartados siguientes proporciona una descripción y las consideraciones que se han de tener en cuenta cuando se implementa cada uno de los métodos de la interfaz del cargador.

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 de valores java.lang.util.List, uno para cada clave que figura en la lista de claves. El primer valor devuelto en la lista de valores corresponde a la primera clave de la lista de claves, el segundo valor devuelto de la lista de claves corresponde a la segunda clave de la lista de claves y así sucesivamente. Si el cargador no encuentra el valor para una clave en la lista de claves, el cargador debe devolver el objeto de valor `KEY_NOT_FOUND` especial definido en la interfaz del cargador. Debido a que se puede configurar la correlación de respaldo para que permita que `null` sea un valor válido, es muy importante que el cargador devuelva el objeto `KEY_NOT_FOUND` especial cuando el cargador no pueda encontrar la clave. Este valor permite que la correlación de respaldo distinga entre un valor `null` y un valor que no existe debido a que no se ha encontrado la clave. Si una correlación de respaldo no da soporte a valores `null`, cuando un cargador devuelve un valor null en lugar del objeto `KEY_NOT_FOUND` para una clave que no existe, se genera una excepción.

El argumento **forUpdate** indica al cargador si la aplicación ha llamado a un método `get` o a un método `getForUpdate` en la correlación. Para obtener más información, consulte la interfaz com.ibm.websphere.objectgrid.ObjectMap. El cargador es el responsable de implementar una política de control de concurrencia que controla el acceso simultáneo al almacenamiento persistente. Por ejemplo, muchos sistemas de gestión de base de datos dan soporte a la sintaxis `for update` en la sentencia SQL select que se utiliza para leer datos de una tabla relacional. El cargador puede elegir entre utilizar la sintaxis `for update` o la sentencia SQL `select` basándose en si se pasa el valor booleano `true` como el valor del argumento para el parámetro **forUpdate** de este método. Generalmente, el cargador utiliza la sintaxis update solamente cuando se utiliza una política de control de concurrencia pesimista. Para un control de concurrencia optimista, el cargador no utiliza nunca la sintaxis update en la sentencia SQL select. El cargador es responsable de decidir si utiliza el argumento `forUpdate` basándose en la política de control de concurrencia que utiliza el cargador.

Para obtener una explicación del parámetro **txid**, consulte el tema “Plug-in TransactionCallback” en la página 218.

Método `batchUpdate`

El método `batchUpdate` resulta crítico en la interfaz del cargador. Este método se invoca cuando ObjectGrid tiene que aplicar todos los cambios actuales al cargador. Al cargador se le proporciona una lista de cambios para esta correlación. Los cambios son iterativos y se aplican al programa de fondo. El método recibe el valor de TxID actual y los cambios que se han de aplicar. El ejemplo siguiente se itera sobre el conjunto de cambios y

agrupa por lotes tres sentencias JDBC (Java Database Connectivity), una con `insert`, otra con `update` y otra 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 para utilizarla.
 Connection conn = getConnection(tx);
try
{
  // Procesar la lista de cambios y crear un conjunto de sentencias
  // preparadas para ejecutar una operación SQL de actualización, inserción
  // o supresión por lotes.
  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( 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 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 nivel superior del proceso del argumento LogSequence pero los detalles de cómo se crea una sentencia SQL `insert`, `update` o `delete` no se ilustran. Algunos de los puntos clave ilustrados son:

- Se llama al método `getPendingChanges` en el argumento LogSequence para obtener un iterador sobre la lista de LogElements que necesita procesar el cargador.

- Se utiliza el método `LogElement.getType().getCode()` para determinar si LogElement es una operación SQL `insert`, `update` o `delete`.
- Se captura una excepción `SQLException` y se encadena a una excepción `LoaderException` que imprime para informar que se ha producido una excepción durante el proceso de actualización por lotes.
- El soporte de actualización por lotes JDBC se utiliza para minimizar el número de consultas que deben realizarse al programa de fondo.

Método `preloadMap`

Durante la inicialización de ObjectGrid, se inicializa cada una de las correlaciones de respaldo definidas. Si se conecta un cargador a una correlación de respaldo, la correlación de respaldo invoca el método `preloadMap` en la interfaz del cargador para que el cargador pueda realizar una búsqueda previa de los datos desde su programa de fondo y cargar los datos en la correlación. El ejemplo siguiente presupone que se leen de las primeras 100 filas de una tabla de empleados de la base de datos y se cargan en la correlación. La clase EmployeeRecord es una clase suministrada por la aplicación que contiene los datos de los empleados que se han leído de la tabla de empleados.

```
import java.util.Map;
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 compromiso automático para utilizarla que se
  // haya establecido en un nivel de aislamiento de compromiso de lectura.
  conn = getAutoCommitConnection(tx);
  // Cargar previamente la correlación de empleados con
  // objetos EmployeeRecord.
  // Leer todos los empleados de la tabla pero
  // limitar la carga previa 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;
}
  // Comprometer 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 cualquier error de retrotracción
    // y permitir que se genere el Throwable original.
}
}
  // Asegurarse de que se limpian también otros recursos de bases de datos,
  // por ejemplo, sentencias de cierre, conjuntos de resultados, etc.
}
}
```

Este ejemplo ilustra los siguientes puntos claves:

- La correlación de respaldo preloadMap utiliza el objeto Session que se pasa a la misma como el 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 en el objeto javaEmployeeRecord.
- El cargador captura todas las excepciones generables que se producen y genera una excepción`LoaderException` con la excepción generable captada encadenada a la misma.
- El bloque `finally` asegura que cualquier excepción generable que se produce entre el momento en que se llama al método `beginNoWriteThrough` y al método `method` haga que el bloque `finally` retrotraiga la transacción activa. Esta acción resulta crítica para garantizar que cualquier transacción que inicie el método `preloadMap` se complete antes de que se devuelva al que llama. El bloque `finally` es un buen lugar para realizar otras acciones de limpieza que pueden resultar necesarias como, por ejemplo, cerrar la conexión y otros objetos JDBC.

El ejemplo `preloadMap` utiliza una sentencia SQL que selecciona todas las filas de la tabla. En el cargador suministrado por la aplicación, es posible que necesite establecer una o más propiedades del cargador para controlar qué cantidad de la tabla se ha de cargar previamente en la correlación.

Debido a que el método `preloadMap` únicamente se llama una vez durante la inicialización de BackingMap, también es un buen lugar para ejecutar el código de inicialización del cargador de una vez. Incluso si un cargador decide no realizar la búsqueda previa de datos desde el programa de fondo y cargar los datos en la correlación, probablemente tenga que realizar otra inicialización de una vez para que otros métodos del cargador resulten más eficaces. El siguiente es un ejemplo de almacenamiento en antememoria

del objeto TransactionCallback y del objeto OptimisticCallback como variables de instancia del cargador, de modo que los demás métodos del cargador no tengan que realizar llamadas a métodos para obtener acceso a estos objetos. Este almacenamiento en antememoria de los valores del plug-in ObjectGrid se puede llevar a cabo porque una vez inicializado BackingMap, los objetos TransactionCallback y OptimisticCallback no se pueden modificar ni sustituir. Se acepta el almacenamiento en antememoria de 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
{
 // Almacenar en antememoria los objetos TransactionCallback y
 // OptimisticCallback en variables de instancia de este cargador.
 ivTcb = (MyTransactionCallback)
 session.getObjectGrid().getTransactionCallback();
 ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
 // Resto del código preloadMap (como se muestra en el ejemplo anterior).
}
```

Para obtener más información sobre la carga previa y la carga previa recuperable relacionada con la sustitución por anomalía, consulte “Programación de duplicación” en la página 229.

Consideraciones acerca del cargador

Tenga en cuenta las consideraciones siguientes cuando implemente un cargador.

Consideraciones acerca de la carga previa

Cada correlación de respaldo tiene un atributo preloadMode booleano que se puede establecer para indicar si la carga previa de una correlación se completa de forma asíncrona. Por omisión, el atributo preloadMode se establece en `false`, lo que indica que la inicialización de la correlación de respaldo no se completa hasta que se completa la carga previa de la correlación. Por ejemplo, la inicialización de la correlación de respaldo no se completa hasta que se devuelve el método `preloadMap`. Si el método `preloadMap` va a leer una gran cantidad de datos de su programa de fondo y lo carga en la correlación, es posible que tarde un período de tiempo considerable en completarse. En este caso, puede configurar una correlación de respaldo para que utilice una carga previa asíncrona de la correlación estableciendo el atributo preloadMode en true. Este valor hace que el código de inicialización de la correlación de respaldo genere una hebra que invoca él método `preloadMap` y permite que se complete una correlación de respaldo mientras la carga previa de la correlación todavía se está procesando.

El siguiente fragmento de código ilustra cómo se establece el atributo preloadMode de modo que permite la carga previa asíncrona:

```
BackingMap bm = og.defineMap( "map1" );
bm.setPreloadMode( true );
```

El atributo preloadMode también se puede establecer utilizando un archivo XML como el que se ilustra en el ejemplo siguiente:

```
<backingMap name="map1" preloadMode="true"
pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
```

TxID y el uso de la interfaz TransactionCallback

Al método `get` y a los métodos `batchUpdate` de la interfaz del cargador se les pasa un objeto TxID que representa la transacción de la sesión que requiere que se lleve a cabo la operación `get` o `batchUpdate`. Es posible que se llame a los métodos `get` y `batchUpdate` más de una vez por transacción. Por lo tanto, los objetos del ámbito de la transacción que necesita el cargador se mantienen generalmente en una ranura del objeto TxID. Se utiliza un cargador JDBC (Java Database Connectivity) para ilustrar cómo un cargador utiliza las interfaces TxID y TransactionCallback.

También se pueden almacenar varias correlaciones ObjectGrid en la misma base de datos. Cada correlación tiene su propio cargador y cada cargador puede conectarse a la misma base de datos. Cuando se conecta a la misma base de datos, cada cargador desea utilizar la misma conexión JDBC de modo que los cambios realizados en cada tabla se comprometan como parte de la misma transacción de base de datos. Generalmente, la misma persona que escribe la implementación del cargador escribe también la implementación TransactionCallback. El mejor método es si la interfaz TransactionCallback se amplía para añadir métodos que necesita el cargador para obtener una conexión de base de datos y para el almacenamiento intermedio de las sentencias preparadas. El motivo por el que se utiliza esta metodología resulta aparente cuando se observa cómo utiliza el cargador las interfaces TransactionCallback y TxID.

Por ejemplo, es posible que el cargador necesite que se amplíe la interfaz TransactionCallback como se muestra a continuación:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
public interface MyTransactionCallback extends TransactionCallback
{
 Connection getAutoCommitConnection(TxID tx, String databaseName)
 throws SQLException;
 Connection getConnection(TxID tx, String databaseName,
 int isolationLevel ) throws SQLException;
 PreparedStatement getPreparedStatement(TxID tx, Connection conn,
 String tableName, String sql) throws SQLException;
 Collection getPreparedStatementCollection( TxID tx, Connection conn,
 String tableName );
}
```

Utilizando estos métodos nuevos, los métodos `get` y `batchUpdate` pueden obtener una conexión como se indica a continuación:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getConnection(TxID tx, int isolationLevel)
{
 Connection conn = ivTcb.getConnection(tx, databaseName, isolationLevel );
 return conn;
}
```

En el ejemplo anterior y en los ejemplos siguientes, *ivTcb* e *ivOcb* son variables de la instancia del cargador que se han inicializado como se ha descrito en el apartado “Consideraciones acerca de la carga previa” en la página 208. La variable *ivTcb* es una referencia a la instancia MyTransactionCallback y la variable *ivOcb* es una referencia a la instancia MyOptimisticCallback. La variable *databaseName* es una variable de la instancia del cargador que se ha establecido como una propiedad del cargador durante la inicialización de la correlación de respaldo. El argumento `isolationLevel` es una de las constantes de la conexión JDBC definidas para los diferentes niveles de aislamiento a los que da soporte JDBC. Si el cargador utiliza una implementación optimista, el método `get` utiliza generalmente una conexión de compromiso automático JDBC para buscar los datos de la base de datos. En este caso, el cargador puede tener un método `getAutoCommitConnection` que se implemente como se indica a continuación:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getAutoCommitConnection(TxID tx)
{
 Connection conn = ivTcb.getAutoCommitConnection(tx, databaseName);
 return conn;
}
```

Recuerde que el método `batchUpdate` tiene la sentencia switch siguiente:

```
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;
}
```

Cada uno de los métodos `buildBatchSQL` utiliza la interfaz MyTransactionCallback para obtener una sentencia preparada. El siguiente es un fragmento de código que muestra el método `buildBatchSQLUpdate` creando una sentencia SQL update para actualizar una entrada EmployeeRecord y añadirla al proceso de actualización por lotes:

```
private void buildBatchSQLUpdate( TxID tx, Object key, Object value, Connection
conn )
throws SQLException, LoaderException
{
String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?, DEPTNO = ?,
 SEQNO = ?, MGRNO = ? where EMPNO = ?";
PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn, "employee",
sql );
 EmployeeRecord emp = (EmployeeRecord) value;
 sqlUpdate.setString(1, emp.getLastName());
 sqlUpdate.setString(2, emp.getFirstName());
 sqlUpdate.setString(3, emp.getDepartmentName());
 sqlUpdate.setLong(4, emp.getSequenceNumber());
 sqlUpdate.setInt(5, emp.getManagerNumber());
 sqlUpdate.setInt(6, key);
 sqlUpdate.addBatch();
}
```

Cuando el bucle batchUpdate ha creado todas las sentencias preparadas, llama al método getPreparedStatementCollection. Este método se puede implementar del modo siguiente:

```
private Collection getPreparedStatementCollection( TxID tx, Connection conn )
{
return ( ivTcb.getPreparedStatementCollection( tx, conn, "employee" ) );
}
```

Cuando la aplicación invoca el método commit en la sesión, el código de sesión llama al método commit en el método TransactionCallback después de que haya pasado al cargador todos los cambios realizados por la transacción en cada correlación modificada. Debido a que todos los cargadores utilizan el método MyTransactionCallback para obtener una conexión y las sentencias preparadas que necesitan, el método TransactionCallback sabe qué conexión debe utilizar para solicitar que el programa de fondo comprometa los cambios. De este modo, ampliar la interfaz TransactionCallback con métodos que necesita cada uno de los cargadores tiene las ventajas siguientes:

- El objeto TransactionCallback encapsula el uso de las ranuras TxID para los datos del ámbito de la transacción y el cargador no requiere información acerca de las ranuras TxID. El cargador sólo tiene que conocer los métodos que se añaden a TransactionCallback utilizando la interfaz MyTransactionCallback para las funciones de soporte que necesita el cargador.
- El objeto TransactionCallback puede garantizar que se compartan las conexiones entre cada cargador que se conecta al mismo programa de fondo de modo que se pueda evitar un protocolo de compromiso a dos fases.
- El objeto TransactionCallback puede garantizar que la conexión con el programa de fondo se complete mediante una operación de compromiso o retrotracción en la conexión cuando corresponda.
- TransactionCallback puede garantizar la limpieza de los recursos de base de datos cuando se completa una transacción.
- TransactionCallback puede ocultar si está obteniendo una conexión gestionada desde un entorno gestionada como, por ejemplo, WebSphere Application Server u otro servidor de aplicaciones compatible con J2EE (Java 2 Platform Enterprise Edition). Esta ventaja permite utilizar el mismo código del cargador en entornos gestionados y no gestionados. Sólo se debe modificar el plug-in TransactionCallback.

Para obtener información detallada acerca de cómo la implementación de TransactionCallback utiliza las ranuras TxID para los datos del ámbito de las transacciones, consulte el tema “Plug-in TransactionCallback” en la página 218.

OptimisticCallback

Como se ha mencionado anteriormente, es posible que el cargador decida utilizar un método optimista para el control de concurrencia. Si este es el caso, el ejemplo del método buildBatchSQLUpdate se ha de modificar ligeramente para implementar un método optimista. Existen varios modos de utilizar un método optimista. Un modo habitual es tener una columna de indicación de la hora o una columna de contador de número de secuencia para las versiones de cada actualización de la fila. Presuponga que la tabla de empleados tiene una columna de número de secuencia que se incrementa cada vez que se actualiza la fila.

A continuación, modifica la signatura del método buildBatchSQLUpdate de modo que se le pase el objeto LogElement en lugar del par de clave y valor. También tiene que utilizar el objeto OptimisticCallback que se conecta a la correlación de respaldo para obtener tanto el objeto de versión inicial como para actualizar el

objeto de versión. El siguiente es un ejemplo de un método `buildBatchSQLUpdate` modificado que utiliza la variable de instancia *ivOcb* que se ha inicializado, como se describe en el apartado de preloadMap:

```
private void buildBatchSQLUpdate( TxID tx, LogElement le,
 Connection conn )throws SQLException, LoaderException
{
 // Obtener el objeto de versión inicial en la última lectura o actualización
 // de esta entrada de correlación en la base de datos.
 Employee emp = (Employee) le.getCurrentValue();
 long initialVersion = ((Long) le.getVersionedValue()).longValue();
 // Obtener el objeto de versión del objeto Employee actualizado para la operación
 // de actualización SQL.
 Long currentVersion = (Long)ivOcb.getVersionedObjectForValue( emp );
 long nextVersion = currentVersion.longValue();
 // Crear ahora la actualización SQL que incluya el objeto de versión en la
 // cláusula where para la comprobación optimista.
String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?,
 DEPTNO = ?,SEQNO = ?, MGRNO = ? where EMPNO = ? and SEQNO = ?";
 PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn,
"employee", sql );
 sqlUpdate.setString(1, emp.getLastName());
 sqlUpdate.setString(2, emp.getFirstName());
 sqlUpdate.setString(3, emp.getDepartmentName());
 sqlUpdate.setLong(4, nextVersion );
 sqlUpdate.setInt(5, emp.getManagerNumber());
 sqlUpdate.setInt(6, key);
 sqlUpdate.setLong(7, initialVersion);
 sqlUpdate.addBatch();
}
```

El ejemplo muestra que se utiliza LogElement para obtener el valor de versión inicial. Cuando la transacción accede por primera vez a la entrada de correlación, se crea un LogElement con el objeto Employee inicial que se obtiene de la correlación. El objeto Employee inicial también se pasa al método getVersionedObjectForValue en la interfaz OptimisticCallback y el resultado se guarda en LogElement. Este proceso se lleva a cabo antes de que se proporcione a una aplicación una referencia al objeto Employee inicial y de que tenga la oportunidad de llamar a algún método que modifique el estado del objeto Employee inicial.

El ejemplo muestra que el cargador utiliza el método getVersiondObjectForValue para obtener el objeto de versión del objeto Employee actualizado actual. Antes de llamar al método batchUpdate en la interfaz del cargador, ObjectGrid llama al método updateVersionedObjectForValue en la interfaz OptimisticCallback para que se genere un objeto de versión nuevo para el objeto Employee actualizado. Cuando el método batchUpdate vuelve a la ObjectGrid, LogElement se actualiza con el objeto de versión actual para que se convierta en el nuevo objeto de versión inicial. Este paso es necesario debido a que la aplicación puede haber llamado al método flush en la correlación, en lugar de al método commit en la sesión. Una transacción individual puede llamar varias al cargador para la misma clave. Por este motivo, ObjectGrid garantiza que se actualice LogElement con el nuevo objeto de versión cada vez que se actualice la fila en la tabla de empleados.

Ahora que el cargador tiene el objeto de versión inicial y el objeto de versión siguiente, puede ejecutar una sentencia SQL update que establezca la columna SEQNO en el siguiente valor del objeto de versión y utiliza el valor del objeto de versión inicial en la cláusula `where`. A veces, se hace referencia a este objeto como una sentencia `update` sobrecualificada. El uso de la sentencia `update` sobrecualificada permite a la base de datos relacional comprobar que otra transacción no haya modificado la fila mientras la transacción lee los datos de la

base de datos y actualiza esta transacción. Si durante este período de tiempo otra transacción ha modificado la fila, entonces la matriz del contador que devuelve el proceso de actualización por lotes indica que se han actualizado cero filas para esta clave. El cargador es responsable de verificar que la operación SQL `update` ha actualizado de hecho la fila. Si no es así, el cargador visualiza una excepción `com.ibm.websphere.objectgrid.plugins.OptimisticCollisionException` para informar a la sesión que el método batchUpdate ha fallado debido a que más de una transacción simultánea ha intentado actualizar la misma fila en la tabla de base de datos. Esta excepción hace que la sesión se retrotraiga y la aplicación debe volver a intentar toda la transacción. El objetivo es que el reintento sea satisfactorio, por eso este método se denomina optimista. De hecho, el método optimista se ejecuta mejor si los datos se modifican con poca frecuencia o si las transacciones simultáneas rara vez intentan actualizar la misma fila.

Es importante para el cargador utilizar el parámetro de *clave* del constructor OptimisticCollisionException para identificar qué clave o conjunto de claves ha provocado la anomalía del método batchUpdate optimista. El parámetro de clave puede ser el propio objeto de clave o una matriz de objetos de clave si más de una clave ha generado una anomalía de actualización optimista. ObjectGrid utiliza el método getKey del constructor OptimisticCollisionException para determinar qué entradas de correlación contienen datos obsoletos y han provocado la excepción. Parte del proceso de retrotracción es desalojar de la correlación cada una de las entradas de correlación obsoletas. El desalojo de entradas obsoletas es necesario para que las próximas transacciones que accedan a la misma clave o las mismas claves generen la invocación del método get de la interfaz del cargador para renovar las entradas de correlación con los datos actuales de la base de datos.

Otras formas de que un cargador implemente un método optimista son:

- No existen columnas de indicación de la hora o de número de secuencia. En este caso, el método getVersionObjectForValue de la interfaz OptimisticCallback devuelve el objeto de valor propiamente dicho como la versión. Con este método, el cargador ha de crear una cláusula `where` que incluye cada uno de los campos del objeto de versión inicial. Este método no resulta muy eficaz y no todos los tipos de columnas se pueden elegir para utilizarlos en la cláusula `where` de una sentencia SQL update sobrecualificada. Generalmente, este método no se utiliza.
- No existen columnas de indicación de la hora o de número de secuencia. No obstante, a diferencia del método anterior, la cláusula `where` sólo contiene los campos de valor que ha modificado la transacción. Un modo de detectar qué campos se modifican es establecer la modalidad de copia de la correlación de respaldo en la modalidad `CopyMode.COPY_ON_WRITE`. Esta modalidad de copia requiere que se pase una interfaz de valor al método `setCopyMode` en la interfaz BackingMap. BackingMap crea objetos del proxy dinámicos que implementan la interfaz de valor proporcionada. Con esta modalidad de copia, el cargador puede convertir cada valor en un objeto com.ibm.websphere.objectgrid.plugins.ValueProxyInfo. La interfaz ValueProxyInfo tiene un método que permite que el cargador obtenga la lista de nombres de atributos que ha modificado la transacción. Este método permite al cargador llamar a los métodos `get` en la interfaz de valor para que los nombres de atributos obtengan los datos modificados y para crear una sentencia SQL update que solamente establezca los atributos modificados. Ahora se puede crear la cláusula `where` de modo que tenga la columna de la clave primaria más cada una de las columnas de atributos modificadas. Este método resulta más eficaz que el método anterior pero requiere que se escriba más código en el cargador y puede darse el caso de que la antememoria de sentencias preparadas deba ser

mayor para poder manejar las diferentes permutaciones. No obstante, si las transacciones generalmente sólo modifican unos cuantos atributos, es posible que esta limitación no sea un problema.

- Es posible que algunas bases de datos tengan una API que ayude a mantener automáticamente los datos de la columna que resultan útiles para las versiones optimistas. Consulte la documentación de su base de datos para determinar si existe esta posibilidad.

Plug-in ObjectTransformer

Utilice el plug-in ObjectTransformer cuando requiera un alto nivel de rendimiento. Si observa problemas de rendimiento con el uso de CPU, añada un plug-in de ObjectTransformer a cada correlación. Si no proporciona un plug-in de ObjectTransformer, hasta un 60-70% del tiempo total de CPU se utiliza realizando series y copias de las entradas.

Finalidad

La finalidad del plug-in ObjectTransformer es permitir que las aplicaciones proporcionen métodos personalizados para las operaciones siguientes:

- Serializar o deserializar la clave de entrada
- Serializar o deserializar el valor de una entrada
- Copiar una clave o un valor para una entrada

Si no se proporciona un plug-in ObjectTransformer, debe poder serializar las claves y los valores ya que ObjectGrid utiliza la secuencia de serializar y deserializar para copiar los objetos. Este método resulta caro, por lo tanto, utilice un plug-in ObjectTransformer cuando el rendimiento resulte crítico. La copia se lleva a cabo cuando una aplicación busca un objeto en una transacción por primera vez. Puede evitar esta copia estableciendo la modalidad de copia de la correlación en `NO_COPY` o puede disminuir la copia estableciendo la modalidad de copia en `COPY_ON_READ`. Optimice la operación de copia cuando lo necesite la aplicación proporcionando un método de copia personalizado en este plug-in. Este tipo de plug-in puede disminuir la actividad general de copia del 65–70% al 2/3% del tiempo total de CPU.

Las implementaciones de método copyKey y copyValue por omisión intentan primero utilizar el método clone(), si se proporciona. Si no se proporciona ningún método clone(), la implementación toma por omisión el valor de serialización.

También se utiliza la serialización de los objetos directamente cuando ObjectGrid se ejecuta en modalidad distribuida. LogSequence utiliza el plug-in ObjectTransformer para ayudarle a serializar las claves y los valores antes de transmitir los cambios a iguales en ObjectGrid. Debe prestar atención cuando proporcione un método de serialización personalizado en lugar de utilizar la serialización JDK incorporada. Las versiones de los objetos es un tema complejo y pueden surgirle problemas con la compatibilidad de las versiones si no se asegura de que los métodos personalizados estén diseñados para versiones.

La lista siguiente muestra detalladamente cómo ObjectGrid intenta serializar las claves y los valores:

- Si se escribe un plug-in ObjectTransformer personalizado y se conecta, ObjectGrid llama a los métodos de ObjectTransformer para serializar las claves y los valores y obtener copias de las claves y valores de los objetos.

- Si no se utiliza un plug-in ObjectTransformer personalizado, ObjectGrid se serializa y se deserializa según el valor por omisión. Si se utiliza el valor por omisión, cada uno de los objetos se implementa como externalizable o se implementa como serializable.
  - Si el objeto da soporte a la interfaz Externalizable, se llama al método `writeExternal`. Los objetos que se implementan como externalizables presentan un rendimiento mejor.
  - Si el objeto no da soporte a la interfaz Externalizable e implementa Serializable, el objeto se guarda utilizando el método `ObjectOutputStream`.

Interfaz ObjectTransformer

Consulte la documentación de la API para obtener más información acerca de la interfaz ObjectTransformer. La interfaz ObjectTransformer contiene los métodos siguientes que serializan y deserializan las claves o valores y copian las claves o valores:

```
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 copyKey(Object value);
 Object copyValue(Object value);
}
```

Uso de la interfaz ObjectTransformer

Puede utilizar la interfaz ObjectTransformer en las situaciones siguientes:

- objeto no serializable
- objeto serializable pero para mejorar el rendimiento de la serialización
- copia de clave o de valor

En el ejemplo siguiente, se utiliza ObjectGrid para almacenar la clase Stock:

```
/**
* Objeto Stock para la demo de ObjectGrid
*
*
*/
public class Stock implements Cloneable {
 String ticket;
 double price;
 String company;
 String description;
 int serialNumber;
 long lastTransactionTime;
/**
 * @return devuelve la descripción.
*/
 public String getDescription() {
  return description;
}
/**
 * @param description La descripción que se ha de establecer.
*/
 public void setDescription(String description) {
```

```
  this.description = description;
}
/**
 * @return Devuelve lastTransactionTime.
*/
 public long getLastTransactionTime() {
  return lastTransactionTime;
}
/**
 * @param lastTransactionTime La última lastTransactionTime que se ha de establecer.
*/
 public void setLastTransactionTime(long lastTransactionTime) {
  this.lastTransactionTime = lastTransactionTime;
}
/**
 * @return Devuelve el precio.
*/
 public double getPrice() {
  return price;
}
/**
 * @param price El precio que se ha de establecer.
*/
 public void setPrice(double price) {
  this.price = price;
}
/**
 * @return Devuelve serialNumber.
*/
 public int getSerialNumber() {
  return serialNumber;
}
/**
 * @param serialNumber El serialNumber que se ha de establecer.
*/
 public void setSerialNumber(int serialNumber) {
  this.serialNumber = serialNumber;
}
/**
 * @return Devuelve ticket.
*/
 public String getTicket() {
  return ticket;
}
/**
 * @param ticket El ticket que se ha de establecer.
*/
 public void setTicket(String ticket) {
  this.ticket = ticket;
}
/**
 * @return Devuelve la empresa.
*/
 public String getCompany() {
  return company;
}
/**
 * @param company La empresa que se ha de establecer.
*/
 public void setCompany(String company) {
  this.company = company;
}
//clonar
 public Object clone() throws CloneNotSupportedException
```

```
{
  return super.clone();
}
}
```

Puede escribir una clase de transformador de objeto personalizada para la clase Stock:

```
/**
* La implementación personalizada de ObjectGrid ObjectTransformer para el
* objeto Stock
*
*/
public class MyStockObjectTransformer implements ObjectTransformer {
/* (non-Javadoc)
* @see
* com.ibm.websphere.objectgrid.plugins.ObjectTransformer#serializeKey
 * (java.lang.Object,
 * java.io.ObjectOutputStream)
*/
 public void serializeKey(Object key, ObjectOutputStream stream)
 throws IOException {
  String ticket= (String) key;
  stream.writeUTF(ticket);
}
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
 ObjectTransformer#serializeValue(java.lang.Object,
java.io.ObjectOutputStream)
*/
 public void serializeValue(Object value, ObjectOutputStream stream)
 throws IOException {
  Stock stock= (Stock) value;
  stream.writeUTF(stock.getTicket());
  stream.writeUTF(stock.getCompany());
  stream.writeUTF(stock.getDescription());
  stream.writeDouble(stock.getPrice());
  stream.writeLong(stock.getLastTransactionTime());
  stream.writeInt(stock.getSerialNumber());
}
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateKey(java.io.ObjectInputStream)
*/
 public Object inflateKey(ObjectInputStream stream) throws IOException,
ClassNotFoundException {
  String ticket=stream.readUTF();
  return ticket;
}
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateValue(java.io.ObjectInputStream)
*/
 public Object inflateValue(ObjectInputStream stream) throws IOException,
ClassNotFoundException {
  Stock stock=new Stock();
  stock.setTicket(stream.readUTF());
  stock.setCompany(stream.readUTF());
  stock.setDescription(stream.readUTF());
  stock.setPrice(stream.readDouble());
  stock.setLastTransactionTime(stream.readLong());
  stock.setSerialNumber(stream.readInt());
  return stock;
}
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyValue(java.lang.Object)
```

```
*/
 public Object copyValue(Object value) {
  Stock stock = (Stock) value;
try{
   return stock.clone();
}
  catch (CloneNotSupportedException e)
{
//crear la corriente de uno
}
}
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyKey(java.lang.Object)
*/
 public Object copyKey(Object key) {
  String ticket=(String) key;
  String ticketCopy= new String (ticket);
  return ticketCopy;
}
}
```

A continuación, conecte esta clase 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);
```

Plug-in TransactionCallback

Generalmente, una aplicación conecta un plug-in TransactionCallback y un cargador como un par. El cargador es el responsable de capturar los datos del programa de fondo y también de aplicar los cambios al programa de fondo. Esta captura y vaciado se suele llevar a cabo dentro del contexto de una transacción ObjectGrid.

El plug-in TransactionCallback tiene las responsabilidades siguientes:

- Reserva ranuras para un estado específico de la transacción necesario para la transacción y el cargador
- Convierte o correlaciona una transacción ObjectGrid en una transacción de plataforma
- Configura el estado por transacción cuando ObjectGrid inicia una transacción
- Compromete la transacción cuando se compromete la transacción ObjectGrid
- Retrotrae la transacción cuando se retrotrae la transacción ObjectGrid

ObjectGrid no es un coordinador de transacciones XA. ObjectGrid se basa en la plataforma para proporcionar dicha posibilidad. Los métodos `begin`, `commit` y `rollback` de ObjectGrid presentes en una sesión son llamadas del ciclo de vida. El plug-in TransactionCallback debe recibir estos sucesos y hacer que la plataforma proporcione la posibilidad de transacciones para los recursos que utilizan los cargadores. En este tema se analizan diferentes casos de ejemplo y se describe cómo se puede escribir el plug-in TransactionCallback para trabajar en estos casos de ejemplo.

Vista general del plug-in TransactionCallback

El plug-in TransactionCallback es un POJO que implementa la interfaz TransactionCallback. La interfaz TransactionCallback es similar al ejemplo siguiente:

```
public interface TransactionCallback
{
 void initialize(ObjectGrid objectGrid) throws TransactionCallbackException;
 void begin(TxID id) throws TransactionCallbackException;
 void commit(TxID id) throws TransactionCallbackException;
 void rollback(TxID id) throws TransactionCallbackException;
 boolean isExternalTransactionActive(Session session);
}
```

Método `initialize`

Se llama al método `initialize` cuando se inicializa ObjectGrid. Callback reserva ranuras para el objeto TxID que necesita. Generalmente, reserva una ranura para cada pieza del estado o el objeto que desea crear en el método `begin` cuando se inicia una transacción. Por ejemplo, desea utilizar un gestor de persistencia con ObjectGrid como cargador. Presuponiendo que este gestor de persistencia tiene objetos de estado de transacción y sesión, TransactionCallback obtendrá una sesión y una transacción y mantendrá las referencias a estos dos objetos en las ranuras de TxID. En este caso, el método `initialize` es similar al ejemplo siguiente:

```
/**
* Se llama cuando se inicializa la ObjectGrid por primera vez. Simplemente
* reservaremos las ranuras en TxID.
*/
public void initialize(ObjectGrid objectGrid) throws
TransactionCallbackException
{
 // reservar una ranura para la transacción del gestor de persistencia
 TXslot = objectGrid.reserveSlot(TxID.SLOT_NAME);
 // reservar una ranura para la sesión del gestor de persistencia
 SessionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
}
```

Un TxID tiene ranuras. Las ranuras son entradas de una matriz ArrayList. Los plug-ins pueden reservar una entrada en la matriz ArrayList llamando al método `ObjectGrid.reserveSlot` e indicando que desean una ranura en el objeto TxID. A continuación, el método devuelve el índice de entrada siguiente a la aplicación. Luego el método puede almacenar la información en esta ranura. Los métodos siguientes demuestran esta técnica.

Método `begin`

ObjectGrid invoca este método cuando inicia una transacción nueva. El plug-in correlaciona este suceso con una transacción real que los cargadores pueden utilizar posteriormente para las llamadas a los métodos `get` y `update` que lleguen antes de que se invoque el método `commit`. El siguiente es un ejemplo del método `begin` que correlaciona un método begin de ObjectGrid con un método begin de transacción del gestor de persistencia:

```
/**
* Esto se invoca cuando la ObjectGrid comienza una nueva transacción.
* Simplemente creamos una transacción del gestor de persistencia e invocamos
* el método begin para la misma. A continuación, se guarda la transacción
* en la ranura TxID de modo que se pueda volver a obtener sin necesidad
* de ThreadLocal, etc.
*/
public void begin(TxID id) throws TransactionCallbackException
{
```

```
 Session PMsession = getPMcurrentSession();
 Transaction tx = PMsession.beginTransaction();
 id.putSlot(TXslot, tx);
 id.putSlot(SessionSlot, PMsession);
}
```

Este ejemplo se basa en el hecho que el método `initialize` tiene reservadas dos ranuras en el objeto TxID. Una ranura es para la sesión del gestor de persistencia y la otra ranura es para la transacción del gestor de persistencia. El método `begin` llama al gestor de persistencia para obtener una sesión, la almacena en la ranura SessionSlot indexada y crea una transacción en la sesión y almacena una referencia a esta transacción utilizando la ranura TXSlot indexada.

Método `commit`

El método `commit` se invoca cuando se compromete una transacción ObjectGrid. Todos los cargadores ya se han vaciado. La responsabilidad del plug-in es comunicar este suceso de compromiso a la plataforma.

```
/**
* Esto se invoca cuando la ObjectGrid compromete una transacción.
* Simplemente lo pasamos al gestor de persistencia.
*/
public void commit(TxID id) throws TransactionCallbackException
{
 Transaction tx = (Transaction)id.getSlot(TXslot);
 tx.commit();
}
```

El método busca la transacción del gestor de persistencia almacenada en la ranura y, a continuación, llama al método `commit`.

Método `rollback`

Se invoca este método cuando una transacción ObjectGrid desea retrotraer una transacción. El plug-in lo envía al gestor de transacciones de la plataforma. El siguiente es el fragmento de código:

```
/**
* Esto se invoca cuando la ObjectGrid desea retrotraer una transacción.
* Simplemente lo pasamos al gestor de persistencia.
*/
public void rollback(TxID id) throws TransactionCallbackException
{
 Transaction tx = (Transaction)id.getSlot(TXslot);
   tx.rollback();
}
```

Este método es muy similar al método `commit`. Obtiene una referencia a la transacción del gestor de persistencia desde una ranura y, a continuación, invoca el método `rollback`.

Método `isExternalTransactionActive`

Normalmente, una sesión ObjectGrid funciona en modalidad de compromiso automático o en modalidad de transacción. La modalidad de compromiso automático significa que se crea una transacción implícita alrededor de cada llamada a método para las instancias de ObjectMap de la sesión. Si no hay ninguna transacción activa y una aplicación efectúa una llamada en un método ObjectMap, la infraestructura llama a este método en el plug-in TransactionCallback para comprobar si hay una transacción correcta activa. Si este método devuelve el valor true, entonces la infraestructura efectúa un inicio automático, de lo contrario, un compromiso automático. Este método permite que ObjectGrid se integre en

los entornos en los que la aplicación invoca los métodos `begin`, `commit` o `rollback` en las API de la plataforma, en lugar de las API de ObjectGrid.

Caso de ejemplo: Entorno J2SE (Java 2 Platform, Standard Edition) basado en JDBC (Java Database Connectivity) simple

Este ejemplo utiliza un entorno J2SE en el que la aplicación tiene un cargador basado en JDBC. Existen dos correlaciones, cada una de las cuales tiene un cargador que efectúa un respaldo de cada correlación mediante una tabla diferente de la base de datos. El plug-in TransactionCallback obtiene una conexión JDBC y, a continuación, invoca los métodos `begin`, `commit` y `rollback` en la conexión. La siguiente es la implementación TransactionCallback de ejemplo:

```
public class JDBCTCB implements TransactionCallback
{
 DataSource datasource;
 int connectionSlot;
 public JDBCTCB(DataSource ds)
{
  datasource = ds;
}
 public void initialize(ObjectGrid objectGrid)
throws TransactionCallbackException
{
   connectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
}
public void begin(TxID id) throws TransactionCallbackException
{
try
{
   Connection conn = datasource.getConnection();
   conn.setAutoCommit(false);
   id.putSlot(connectionSlot, conn);
}
  catch(SQLException e)
{
throw new TransactionCallbackException("No se puede iniciar la transacción", e);
}
}
public void commit(TxID id) throws TransactionCallbackException
{
  Connection conn = null;
try
{
   conn = (Connection)id.getSlot(connectionSlot);
   conn.commit();
     conn.close();
}
  catch(SQLException e)
{
throw new TransactionCallbackException("No se puede comprometer la
transacción", e);
}
finally {
   if (conn!=null) {
try {
     conn.close();
}
    catch (SQLException closeE) {
}
}
}
}
public void rollback(TxID id) throws TransactionCallbackException
{
```

```
  Connection conn = null;
try
{
   conn = (Connection)id.getSlot(connectionSlot);
   conn.rollback();
     conn.close();
}
  catch(SQLException e)
{
throw new TransactionCallbackException("No se puede retrotraer la
transacción", e);
}
finally {
   if (conn!=null) {
try {
     conn.close();
}
    catch (SQLException closeE) {
}
}
}
}
 public boolean isExternalTransactionActive(Session session)
{
  return false;
}
 public int getConnectionSlot()
{
  return connectionSlot;
}
}
```

Este ejemplo muestra un plug-in TransactionCallback que convierte los sucesos de la transacción ObjectGrid en una conexión JDBC. Cuando se inicializa el plug-in, reserva una sola ranura para mantener una referencia de conexión JDBC. El método `begin` obtiene a continuación una conexión JDBC para la nueva transacción, desactiva el compromiso automático y luego almacena una referencia a la conexión en la ranura TxID. Los métodos `commit` y `rollback` recuperan la conexión desde la ranura TxID y llaman al método adecuado en la conexión. El método `isExternalTransaction` siempre devuelve false, lo que indica que la aplicación debe utilizar las API de la transacción ObjectGrid de forma explícita para controlar las transacciones. Un cargador emparejado con este plug-in obtiene la conexión JDBC del TxID. Un cargador es similar al ejemplo siguiente:

```
public class JDBCLoader implements Loader
{
 JDBCTCB tcb;
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
 tcb = (JDBCTCB)session.getObjectGrid().getTransactionCallback();
}
 public List get(TxID txid, List keyList, boolean forUpdate)
throws LoaderException
{
  Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
  // implementar get aquí
   return null;
}
 public void batchUpdate(TxID txid, LogSequence sequence)
 throws LoaderException, OptimisticCollisionException
{
```

```
  Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
  // Implementar la actualización por lotes TODO aquí
}
}
```

El cargador obtiene una referencia a la instancia de JDBCTCB cuando se invoca el método `initialize`. A continuación, obtiene la conexión que se ha obtenido mediante JDBCTCB cuando es necesario en los métodos `get` y `batchUpdate`. Las implementaciones de TransactionCallback y de los cargadores se suelen escribir en pares que colaboran entre sí. La implementación de TransactionCallback maneja la transacción y almacena los objetos que necesitan los cargadores en las ranuras de TxID. A continuación, los cargadores implementan los métodos `get` y `batchUpdate` en el contexto de una transacción gestionada por TransactionCallback utilizando recursos obtenidos generalmente por TCB.

Caso de ejemplo: entorno del motor de servlets

En este caso de ejemplo, ObjectGrid utiliza un cargado basado en JDBC pero en un motor de servlets gestionado. El contenedor espera que utilicemos el método `UserTransaction` para iniciar y comprometer las transacciones. Esto es ligeramente diferente del caso de J2SE ya que no es necesario almacenar una referencia a la conexión JDBC en una ranura TxID. El contenedor gestiona la conexión JDBC. Cuando una transacción de contenedor está activa, una conexión que se busca utilizando un origen de datos da como resultado la misma conexión cada vez ya que el contenedor recuerda las conexiones que utiliza esta transacción y devuelve la misma conexión cada vez que se llama al método `DataSource.getConnection`. Presuponga que la referencia de origen de datos se ha configurado como `Shareable` en el ejemplo siguiente:

```
public class ManagedJDBCTCB implements TransactionCallback {
 UserTransaction tx;
 public void initialize(ObjectGrid objectGrid)
throws TransactionCallbackException
{
try
{
   InitialContext ic = new InitialContext();
tx = (UserTransaction)ic.lookup("java:comp/UserTransaction");
}
  catch(NamingException e)
{
throw new TransactionCallbackException("No se puede
encontrar UserTransaction", e);
}
}
public void begin(TxID id) throws TransactionCallbackException
{
try
{
   tx.begin();
}
  catch(SystemException e)
{
throw new TransactionCallbackException("No puede empezar tx", e);
}
  catch(NotSupportedException e)
{
throw new TransactionCallbackException("No puede empezar tx", e);
}
}
public void commit(TxID id) throws TransactionCallbackException
{
try
```

```
{
 tx.commit();
}
  catch(SystemException e)
{
throw new TransactionCallbackException("No puede comprometer tx", e);
}
  catch(HeuristicMixedException e)
{
throw new TransactionCallbackException("No puede comprometer tx", e);
}
  catch(RollbackException e)
{
throw new TransactionCallbackException("No puede comprometer tx", e);
}
  catch(HeuristicRollbackException e)
{
throw new TransactionCallbackException("No puede comprometer tx", e);
}
}
public void rollback(TxID id) throws TransactionCallbackException
{
try
{
   tx.rollback();
}
  catch(SystemException e)
{
throw new TransactionCallbackException("No puede comprometer tx", e);
}
}
 public boolean isExternalTransactionActive(Session session) {
  return false;
}
}
```

Este ejemplo obtiene una referencia al método `UserTransaction` en el método `initialize` y, a continuación, correlaciona `begin`, `commit` y `rollback` en los métodos `UserTransaction` adecuados. Las ranuras no son necesarias debido a que el contenedor verifica que se ha recuperado la información de conexión correcta para esta transacción. El siguiente es el cargador JDBC que funciona con esta implementación de TransactionCallback:

```
public class ManagedJDBCLoader implements Loader
{
 DataSource myDataSource;
 ManagedJDBCLoader(DataSource ds)
{
  myDataSource = ds;
}
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
}
 public List get(TxID txid, List keyList, boolean forUpdate)
throws LoaderException
{
try
{
   Connection conn = myDataSource.getConnection();
   // aquí se implementa get de TODO con esta conexión
   return null;
}
  catch(SQLException e)
{
throw new LoaderException("No puede obtener objetos", e);
```

```
}
}
 public void batchUpdate(TxID txid, LogSequence sequence)
 throws LoaderException, OptimisticCollisionException
{
try
{
   Connection conn = myDataSource.getConnection();
   // aquí se implementa update de TODO con esta conexión
}
  catch(SQLException e)
{
throw new LoaderException("No puede actualizar objetos", e);
}
}
}
```

Este ejemplo puede resultar más sencillo que la versión JDBC básica debido a que el contenedor gestiona las conexiones y verifica que dentro de la misma transacción, el método `DataSource.getConnection` siempre devuelve la misma conexión cuando se llama con la misma transacción activa cada vez. No intente guardar en antememoria la conexión en una ranura como resultado, aunque la aplicación puede guardar en antememoria la conexión si así lo decide.

Interfaz OptimisticCallback

Puede proporcionar un objeto Callback optimista que se pueda conectar que implemente la interfaz com.ibm.websphere.objectgrid.plugins.OptimisticCallback.

Finalidad

La interfaz OptimisticCallback se utiliza para proporcionar operaciones de comparación optimista para los valores de una correlación. Es necesario OptimisticCallback cuando se utiliza la estrategia de bloqueo optimista como se describe en “Bloqueo optimista” en la página 137. ObjectGrid proporciona una implementación de OptimisticCallback por omisión. No obstante, generalmente la aplicación debe conectar su propia implementación de la interfaz OptimisticCallback.

Conecte un objeto OptimisticCallback suministrado por la aplicación

El siguiente ejemplo muestra cómo una aplicación puede conectar un objeto OptimisticCallback para la correlación de respaldo de empleados de la instancia de 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 );
```

El objeto EmployeeOptimisticCallbackImpl del ejemplo anterior debe implementar la interfaz OptimisticCallback. La aplicación también puede utilizar un archivo XML para conectar su objeto OptimisticCallback como se muestra en el ejemplo siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="grid1">
<backingMap name="employees" pluginCollectionRef="employees"
lockStrategy="OPTIMISTIC" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="employees">
<bean id="OptimisticCallback"
className="com.xyz.EmployeeOptimisticCallbackImpl" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Implementación por omisión

La infraestructura de ObjectGrid proporciona una implementación por omisión de la interfaz OptimisticCallback que se utiliza si la aplicación no conecta un objeto OptimisticCallback suministrado por la aplicación, como se ha mostrado en el apartado anterior. La implementación por omisión siempre devuelve el valor especial de NULL_OPTIMISTIC_VERSION como el objeto de versión para el valor y nunca actualiza el objeto de versión. Esta acción convierte la comparación optimista en una función de tipo ″sin operación″. En la mayor parte de los casos, no desea que la función ″sin operación″ se lleve a cabo cuando está utilizando la estrategia de bloqueo optimista. Las aplicaciones deben implementar la interfaz OptimisticCallback y conectar sus propias implementaciones de OptimisticCallback, de modo que no se utilice la implementación por omisión. No obstante, al menos existe un caso de ejemplo en el que resulta útil la implementación por omisión de OptimisticCallback suministrada. Tenga en cuenta la situación siguiente:

- Un cargador se conecta para la correlación de respaldo.
- El cargador sabe cómo realizar la comparación optimista sin la ayuda de un plug-in OptimisticCallback.

¿Cómo puede saber el cargador de qué modo ha de tratar las versiones optimistas 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 un valor de versión optimista. Por ejemplo, presuponga que se utiliza la siguiente interfaz para el objeto de valor de la correlación de empleados:

```
public interface Employee
{
// Número de secuencia secuencial utilizado para las versiones optimistas.
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 versión actual de un objeto de valor Employee. Aumenta el valor devuelto para generar un número de versión nuevo antes de actualizar el almacenamiento persistente con el valor nuevo de Employee. Para un cargador JDBC (Java Database Connectivity), se utiliza el número de secuencia actual de la cláusula where de una sentencia SQL update sobrecualificada y ésta utiliza el nuevo número de secuencia generado para establecer la columna de número de secuencia en el valor del número de secuencia nuevo. Otra posibilidad es que el cargador utilice alguna función suministrada por el programa de fondo

que actualice automáticamente una columna oculta que pueda utilizarse para las versiones optimistas. En algunos casos, se puede utilizar un procedimiento almacenado o un desencadenante que ayude a mantener una columna que contiene información de versiones. Si el cargador utiliza una de estas técnicas para mantener la información de versiones optimista, entonces la aplicación no necesita proporcionar una implementación de OptimisticCallback. En este caso se puede utilizar el valor por omisión de OptimisticCallback porque el cargador puede manejar las versiones optimistas sin la ayuda de un objeto OptimisticCallback.

Implementar la interfaz OptimisticCallback

La interfaz OptimisticCallback contiene los métodos y valores especiales siguientes:

```
public interface OptimisticCallback
{
 final static Byte NULL_OPTIMISTIC_VERSION;
 Object getVersionedObjectForValue(Object value);
 void updateVersionedObjectForValue(Object value);
 void serializeVersionedValue(Object versionedValue,
 ObjectOutputStream stream) throws IOException;
 Object inflateVersionedValue(ObjectInputStream stream) throws
 IOException, ClassNotFoundException;
}
```

La lista siguiente proporciona una descripción de las consideraciones para cada uno de los métodos de la interfaz OptimisticCallback:

NULL_OPTIMISTIC_VERSION
: Este valor especial lo devuelve el método getVersionedObjectForValue si se utiliza la implementación por omisión de OptimisticCallback en lugar de una implementación de OptimisticCallback suministrada por la aplicación.

Método getVersionedObjectForValue
: Este método puede devolver una copia del valor o puede devolver un atributo del valor que se puede utilizar para fines de versiones. Se llama a este método cuando se asocia un objeto a una transacción. Cuando no se conecta un cargador a una correlación de respaldo, la correlación de respaldo utiliza este valor en el momento en que se compromete para realizar una comparación de versiones optimista. La correlación de respaldo utiliza la comparación de versiones optimista para asegurarse de que la versión no se ha modificado desde que esta transacción ha accedido por primera vez a la entrada de correlación. Si otra transacción ya ha modificado la versión de esta entrada de correlación, la comparación de versiones fallará y la correlación de respaldo visualizará una excepción OptimisticCollisionException para forzar que se retrotraiga la transacción. Si se conecta un cargador, la correlación de respaldo no utiliza la información de versiones optimista. En su lugar, el cargador es el responsable de la comparación de versiones optimista y actualiza la información de versión cuando resulta necesario. Generalmente, el cargador obtiene el objeto de versión inicial de LogElement que se ha pasado al método batchUpdate del cargador, que se invoca cuando se efectúa una operación de vaciado o se compromete 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() );
}
}
```

Como se ha demostrado en el ejemplo anterior, el atributo sequenceNumber se devuelve en un objeto java.lang.Long como espera el cargador, lo que implica que la misma persona que ha escrito el cargador ha escrito la implementación EmployeeOptimisticCallbackImpl o ha colaborado con la persona que ha implementado EmployeeOptimisticCallbackImpl - por ejemplo, han acordado el valor que devuelve el método getVersionedObjectForValue.

Como se ha descrito anteriormente, por omisión, OptimisticCallback devuelve el valor especial NULL_OPTIMISTIC_VERSION como el objeto de versión.

Método updateVersionedObjectForValue

Se invoca ese método cuando una transacción ha actualizado un valor y es necesario un nuevo objeto de versión nueva. Si getVersionedObjectForValue devuelve un atributo del valor, este método actualiza generalmente el valor del atributo con un objeto de versión nuevo. Si getVersionedObjectForValue devuelve una copia del valor, generalmente, este método no hace nada. El valor por omisión de OptimisticCallback no hace nada ya que la implementación por omisión de getVersionedObjectForValue devuelve siempre el valor especial NULL_OPTIMISTIC_VERSION como el objeto de versión.

A continuación, se muestra la implementación que utiliza el objeto EmployeeOptimisticCallbackImpl utilizado en el apartado de OptimisticCallback:

```
public void  updateVersionedObjectForValue(Object value)
{
 if ( value != null )
{
  Employee emp = (Employee) value;
  long next = emp.getSequenceNumber() + 1;
  emp.updateSequenceNumber( next );
}
}
```

Como se ha mostrado en el ejemplo anterior, el atributo sequenceNumber se aumenta en uno, de modo que la próxima vez que se invoca el método getVersionedObjectForValue, el valor java.lang.Long devuelto tiene un valor long que es el valor de número de secuencia original más uno, por ejemplo, es el valor de versión siguiente para esta instancia de Employee. Una vez más, este ejemplo implica que la misma persona que escribe el cargador escribe también EmployeeOptimisticCallbackImpl o colabora con la persona que implementa EmployeeOptimisticCallbackImpl.

Método serializeVersionedValue

Este método escribe el valor de versión de la corriente especificada. Dependiendo de la implementación, se puede utilizar el valor de versión para identificar las colisiones de actualización optimistas. En algunas implementaciones, el valor de versión es una copia del valor original. Es posible que otras implementaciones tengan un número de secuencia u otro tipo de objeto para indicar la versión del valor. Dado que la implementación

real se desconoce, se proporciona este método para realizar la serialización correcta. La implementación por omisión efectúa una llamada writeObject.

Método inflateVersionedValue
Este método toma la versión serializada del valor de versión y devuelve el objeto de valor de versión real. Dependiendo de la implementación, se puede utilizar el valor de versión para identificar las colisiones de actualización optimistas. En algunas implementaciones, el valor de versión es una copia del valor original. Es posible que otras implementaciones tengan un número de secuencia u otro tipo de objeto para indicar la versión del valor. Debido a que se desconoce la implementación real, se proporciona este método para realizar la deserialización correcta. La implementación por omisión efectúa readObject.

Programación de duplicación

La duplicación se configura asociando un MapSet con un ReplicationGroup y atributos de política de duplicación. El ReplicationGroup define los miembros de servidor que se utilizan para el tipo primario, y las duplicaciones y los tipos reposo asociados. También define el número mínimo y máximo de duplicaciones que se necesitan para esta configuración. Los atributos de política de duplicación indican si se necesita la duplicación síncrona o asíncrona, si se debe permitir el acceso de lectura a las duplicaciones y si se debe utilizar la compresión cuando se envían datos de duplicación a las duplicaciones. La duplicación tiene un impacto mínimo en el modelo de programación. El principal impacto se ejerce en las aplicaciones que precargan datos en las correlaciones.

Precarga de correlaciones

Puede asociar un cargador con cada correlación. El cargador se utiliza para capturar objetos cuando no se pueden encontrar en la correlación, así como para escribir cambios en un programa de fondo cuando se compromete una transacción. Los cargadores también se pueden utilizar para precargar los datos en una correlación. El método de precarga de la interfaz del cargador se invoca cuando Java Virtual Machine (JVM) se convierte en una unidad primaria del grupo de duplicación. El método de precarga no se invoca en las duplicaciones o las unidades en reposo. El método de precarga intenta cargar todos los datos referenciados previstos desde el programa de fondo en la correlación utilizando la sesión proporcionada. La correlación que se va a utilizar la identifica el argumento BackingMap que se pasa el método de precarga.

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

Precarga en un MapSet particionado

Las correlaciones pueden particionarse en *N* particiones. Las correlaciones pueden almacenarse en varios servidores, donde cada entrada está identificada por una clave que sólo está almacenada en uno de esos servidores. Las correlaciones muy grandes se pueden mantener en una ObjectGrid, teniendo en cuenta que la aplicación ya no está limitada por el tamaño de almacenamiento dinámico de una única JVM para alojar todas las entradas de una correlación. Las aplicaciones que deseen precargarse con el método de precarga de la interfaz del cargador deben identificar el subconjunto de los datos que se debe precargar. Siempre existe un número fijo de particiones. El número se puede determinar utilizando el siguiente fragmento de código:

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

Este fragmento de código muestra cómo una aplicación puede identificar el subconjunto de los datos que se van a precargar desde la base de datos. Las aplicaciones deben utilizar siempre estos métodos, aunque la correlación no se haya particionado inicialmente. Estos métodos permiten una mayor flexibilidad: si los administradores particionan la correlación posteriormente, el cargador continúa trabajando correctamente.

La aplicación debe emitir consultas para recuperar el subconjunto myPartition desde el programa de fondo. Si se está utilizando una base de datos, será más fácil tener una columna con el identificador de partición de un registro determinado, a menos que exista alguna consulta natural que permite particionar fácilmente los datos de la tabla.

Rendimiento

La implementación de precarga debe copiar los datos del programa de fondo en la correlación almacenando varios objetos en la correlación en una única transacción. La siguiente cuestión es ″¿Cuántos registros se deben almacenar por transacción?″ y, desgraciadamente, la respuesta es ″Depende″. Cuando una transacción incluye bloques de más de 100 entradas, el aumento de rendimiento disminuye. El número óptimo dependerá de varios factores, incluidos el tamaño y la complejidad de los objetos. Empiece con 100 entradas y, a continuación, aumente el número hasta que no se observen más aumentos del rendimiento. Las transacciones de mayor tamaño dan como resultado un mayor rendimiento de duplicación. Recuerde que sólo la unidad primaria ejecuta el código de precarga. Los datos precargados se duplican desde la unidad primaria a las duplicaciones que estén en línea.

Precarga de MapSets

Si la aplicación utiliza un MapSet con varias correlaciones, cada correlación tiene su propio cargador. Cada cargador tiene un método de precarga. La ObjectGrid carga cada correlación en serie. Será más eficaz cargar todas las correlaciones designando una única correlación como la correlación de precarga. Esto es sólo una convención de aplicación. Por ejemplo, dos correlaciones, departamento y empleado, pueden utilizar el cargador del departamento para precargar las correlaciones de departamento y empleado. Esto garantiza que, transaccionalmente, si una aplicación desea un departamento, los empleados de ese departamento estarán en la antememoria. Obviamente, esto significa que cuando el cargador del departamento precarga un departamento desde el programa de fondo, también captura los empleados del departamento. El objeto de departamento y los objetos de empleado asociados se deben añadir a la correlación utilizando una única transacción para que esto se cumpla.

Precarga recuperable

Algunos clientes tienen conjuntos de datos muy grandes que se deben guardar en la antememoria. La precarga de estos datos puede necesitar mucho tiempo. A veces, la precarga debe finalizar para que la aplicación pueda ir en línea. Esto significará que, en algunos casos, le interesará que la precarga sea recuperable. Supongamos que existe un millón de registros a precargar. La unidad primaria los está precargando y falla en el registro 800.000. Normalmente, la duplicación elegida como la nueva unidad primaria borra los estados duplicados y empieza

desde el principio. ObjectGrid puede obtener un mejor resultado utilizando un ReplicaPreloadController. El cargador de la aplicación también tendrá que implementar la interfaz ReplicaPreloadController. Esto añade un método al cargador:

```
Status checkPreloadStatus(Session session, BackingMap bmap);
```

Este método lo invoca el tiempo de ejecución de ObjectGrid antes de invocar normalmente el método de precarga de la interfaz de cargador. ObjectGrid comprueba el resultado de este método (Estado) para determinar su comportamiento siempre que se promociona una duplicación a unidad primaria.

Valor de estado devuelto	Comportamiento de ObjectGrid como reacción
Status.PRELOADED_ALREADY	ObjectGrid no llama al método de precarga ya que este valor de estado indica que la correlación está completamente precargada.
Status.FULL_PRELOAD_NEEDED	ObjectGrid borra la correlación e invoca normalmente el método de precarga.
Status.PARTIAL_PRELOAD_NEEDED	ObjectGrid deja la correlación tal cual e invoca la precarga. Esta estrategia permite al cargador de la aplicación continuar la precarga desde ese punto en adelante.

Evidentemente, cuando una unidad primaria está precargando la correlación, debe dejar algún estado en una correlación del MapSet que se está duplicando, para que la duplicación pueda determinar qué estado debe devolver. Puede utilizar una correlación adicional denominada, por ejemplo, RecoveryMap. Esta RecoveryMap debe formar parte del mismo MapSet que se está precargando. Esto garantiza que se duplique coherentemente con los datos que se están precargando.

A continuación, se propone una implementación. Cuando la precarga compromete cada bloque de registros, también debe actualizar un contador/valor en la RecoveryMap como parte de esa transacción. Esto significa que los datos precargados y los datos de RecoveryMap se duplican atómicamente en las duplicaciones. Cuando la duplicación se promociona a unidad primaria, puede comprobar la RecoveryMap para ver qué ha pasado. La RecoveryMap puede mantener simplemente una única entrada con la clave 'state'. Si no existe ningún objeto para esta clave, necesitaremos una precarga completa (checkPreloadStatus devuelve FULL_PRELOAD_NEEDED). Si existe un objeto para esta clave 'state', si el valor es 'COMPLETE', la precarga ha finalizado y checkPreloadStatus devuelve PRELOADED_ALREADY. De lo contrario, el objeto de valor indica dónde se debe reiniciar la precarga y el método checkPreloadStatus debe devolver PARTIAL_PRELOAD_NEEDED. El cargador puede almacenar el punto de recuperación en una variable de instancia del cargador, de forma que cuando se invoque la precarga, conozca el punto de partida. La RecoveryMap también puede mantener una entrada por correlación, si cada correlación se precarga de forma independiente.

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

El tiempo de ejecución de ObjectGrid está diseñado para no perder datos comprometidos cuando falla la unidad primaria. En el apartado siguiente se

muestran los algoritmos utilizados para ello. Estos algoritmos se aplican sólo cuando un grupo de duplicación utiliza la duplicación síncrona. El cargador es opcional.

El tiempo de ejecución de ObjectGrid se puede configurar para duplicar de forma síncrona todos los cambios de una unidad primaria a las duplicaciones. Cuando una JVM se promociona para ser una duplicación, la unidad primaria envía primero una instantánea de la correlación a la duplicación. Una vez la duplicación ha procesado esta instantánea, la unidad primaria empieza a enviar todos los cambios (transacciones completas) desde la generación de la instantánea. Al final, la duplicación alcanzará a la unidad primaria. Este proceso de duplicación inicial es asíncrono. Cuando una duplicación alcanza la unidad primaria, el par entra en modalidad de igual y, por último, empieza la duplicación síncrona. A partir de este punto, cada transacción comprometida en la unidad primaria se enviará a todas las duplicaciones en modalidad de igual y la unidad primaria esperará un mensaje de reconocimiento. Esto ralentiza la unidad primaria, comparado con el caso de ejemplo de duplicación asíncrona, debido a la latencia implicada en la recepción de mensajes de reconocimiento. Una secuencia de compromiso síncrona en la unidad primaria será parecida a la siguiente:

Paso con el cargador	**Paso sin el cargador**
Obtener bloqueos para las entradas	igual
Desechar cambios en el cargador	NO
Guardar los cambios en la antememoria	igual
Enviar los cambios a duplicaciones y esperar el acuse de recibo	igual
Comprometer los cambios en el cargador mediante el plug-in TransactionCallback	El compromiso del plug-in TransactionCallBack continúa invocándose, pero normalmente no hace nada.
Liberar bloqueos para las entradas	igual

Observe que los cambios se envían a la duplicación antes de comprometerlos en el cargador. ¿Cuándo se comprometen los cambios en la duplicación? Revise esta secuencia:

Durante la inicialización, inicializar las listas tx en la unidad primaria.

- Set CommitedTx = {}, RolledBackTx = {}

Durante el proceso de compromiso síncrono:

Paso con el cargador	**Paso sin el cargador**
Obtener bloqueos para las entradas	igual
Desechar cambios en el cargador	NO
Guardar los cambios en la antememoria	igual
Enviar cambios con una transacción comprometida y una transacción retrotraída en la duplicación y esperar el acuse de recibo	igual
Borrar la lista de transacciones comprometidas y transacciones retrotraídas	igual
Comprometer el cargador mediante el plug-in TransactionCallBack	El compromiso del plug-in TransactionCallBack continúa invocándose, pero normalmente no hace nada

Paso con el cargador	Paso sin el cargador
Si el compromiso es satisfactorio, añadir la transacción a las transacciones comprometidas; de lo contrario, añadirla a las transacciones retrotraídas	NO
Liberar bloqueos para las entradas	igual

Proceso de la duplicación

- Recibir cambios
- Comprometer todas las transacciones recibidas en la lista de transacciones comprometidas
- Retrotraer todas las transacciones recibidas en la lista de transacciones retrotraídas
- Iniciar una transacción o sesión
- Aplicar cambios a la transacción o sesión
- Guardar la transacción o sesión en la lista de pendientes
- Devolver una respuesta

Observe que en la duplicación, no existen interacciones de cargador mientras está en modalidad de duplicación. La unidad primaria debe pasar todos los cambios a través del cargador. La duplicación es como un zángano.

Un efecto secundario de este algoritmo es que la duplicación siempre tiene las transacciones, pero no se comprometen hasta que la siguiente transacción primaria envía el estado de compromiso a dichas transacciones. Es entonces cuando se comprometen o se retrotraen en la duplicación. Pero hasta ese momento, las transacciones no están comprometidas. Se puede añadir un temporizador en la unidad primaria que enviará el resultado de la transacción después de un breve periodo de tiempo (unos pocos segundos). Esto limitará la obsolescencia en esa ventana de tiempo, pero no la eliminará completamente. La obsolescencia sólo es un problema cuando se utiliza la modalidad de lectura de duplicaciones. De lo contrario, no es visible y no tiene ningún impacto en la aplicación.

Cuando la unidad primaria falla, es probable que se hayan comprometido/ retrotraído algunas transacciones en ella, pero que el mensaje nunca haya llegado a la duplicación con estos resultados. Cuando se promociona una duplicación a la nueva unidad primaria, una de sus primeras acciones es manejar esta condición. Cada transacción pendiente se vuelve a procesar con el conjunto de correlaciones de la nueva unidad primaria. Si hay un cargador, cada transacción se proporciona al cargador. Estas transacciones se aplican en orden FIFO estricto. Si una transacción falla, se ignora. Si hay 3 transacciones pendientes (A, B y C), A se puede comprometer, B se puede retrotraer y C se puede comprometer también. Ninguna transacción tendrá ningún efecto en las demás. Se supone que son independientes.

Si lo desea, un cargador puede utilizar una lógica ligeramente diferente cuando está en modalidad de 'recuperación de sustitución por anomalía', por oposición a la modalidad 'normal'. El cargador puede saber fácilmente cuándo está en modalidad de recuperación de sustitución por anomalía implementando la interfaz ReplicaPreloadController. El método checkPreloadStatus sólo se invoca cuando finaliza la recuperación de sustitución por anomalía. Por lo tanto, si se invoca el método de aplicación de la interfaz del cargador antes que checkPreloadStatus, será una transacción de recuperación. Cuando se invoque el método

checkPreloadStatus, la recuperación de sustitución por anomalía habrá terminado.

Singletons con estado utilizando la duplicación

WebSphere Extended Deployment ha añadido el soporte de singletons en su primer release con Partitioning Facility. Esto permite a las aplicaciones crear singletons en un clúster. El tiempo de ejecución de ObjectGrid habilita una característica parecida utilizando los MapSets duplicados. Aunque el patrón singleton de ObjectGrid ofrece muchas ventajas, también supone algunas desventajas. Partitioning Facility proporciona un suceso a la aplicación cuando el singleton/la aplicación se activa localmente; esto se comunica utilizando el método partitionLoad del recurso de particiones. Un MapSet duplicado también tiene un singleton, la unidad primaria. El método ReplicaPreloadController#checkPreloadStatus del cargador notifica a la aplicación cuando se convierte en la unidad primaria. Esto se puede utilizar de forma parecida a Partitioning Facility, pero tiene la ventaja de que es portátil entre las distintas versiones de WebSphere Application Server o servidores de aplicaciones de la competencia.

Partitioning Facility tiene un suceso de desactivación, pero el tiempo de ejecución de ObjectGrid no ofrece esta posibilidad. Una unidad primaria en la ObjectGrid normalmente se ejecuta hasta que falla. No se puede trasladar. Esta es una ventaja de Partitioning Facility frente a ObjectGrid. A continuación, se muestra una tabla con las posibilidades:

Tabla 16.

Posibilidad	Partitioning Facility	Singletons de ObjectGrid
Suceso de inicio de singleton	Sí	Sí
Suceso de detención de singleton	Sí	No
Duplicación de estado de singleton	No	Sí
Calidad de servicio (QoS) variable para la duplicación	No	Sí
Colocación flexible de singletons	Sí	No
Puede trasladar el singleton en el tiempo de ejecución	Sí	No
Direccionamiento IIOP de trabajo a singleton	Sí	No
Requiere un servidor J2EE (Java 2 Platform, Enterprise Environment)	Sí	No
Requiere una versión completa de WebSphere Extended Deployment	Sí	No
Requiere EJB (Enterprise JavaBeans)	Sí	No
La aplicación se puede trasladar a otros servidores de aplicaciones	No	Sí

Estado de singleton

Partitioning Facility no ofrecía soporte incorporado para la gestión de estados. Las aplicaciones se dejaban en sus propios dispositivos si el singleton necesitaba un estado. Normalmente, esto significaba que el estado se pasaba a una base de datos. Si la partición fallaba, el servidor que se había elegido para alojar y recuperar la partición debía recuperar el estado de la base de datos. En cambio, si una aplicación utiliza ObjectGrid, el singleton puede mantener su estado en la correlación asociada con el ReplicaPreloadController que gestiona el singleton. Si la unidad primaria o el singleton falla, la unidad de duplicación elegida para ser la nueva unidad primaria ya tiene el estado localmente gracias a la duplicación. Utilice la duplicación síncrona a menos que la pérdida de datos sea aceptable para la aplicación.

Colocación flexible de singletons

Partitioning Facility utiliza el mecanismo de políticas de High Availability Manager para determinar dónde se alojará una partición, y estas políticas se pueden modificar en el tiempo de ejecución con un efecto inmediato. Las políticas del grupo de duplicación de ObjectGrid no son tan flexibles como las de High Availability Manager y no se pueden cambiar sin reiniciar todos los servidores. Si utiliza ObjectGrid, perderá la capacidad de mover singletons en el tiempo de ejecución.

Duplicación de QoS variable

Partition Facility no ofrece gestión de estados. ObjectGrid ofrece una amplia variedad de enfoques de duplicación:

- Sin duplicación
- Duplicación asíncrona
- Duplicación síncrona

La política de duplicación del MapSet asociado con la correlación que está utilizando para el estado determina la política. La duplicación síncrona garantiza que no haya pérdida de datos, pero es más lenta. La duplicación asíncrona es más rápida, pero implica que una o varias transacciones comprometidas en la unidad primaria se pueden perder si la unidad primaria falla.

Equilibrio de carga entre duplicaciones

ObjectGrid, a menos que se configure de otra forma, envía todas las peticiones de lectura y grabación al servidor primario de un determinado grupo de duplicación. Esto significa que el servidor primario por sí solo debe ofrecer servicio a todas las peticiones de los clientes. Si lo desea, puede dejar que las peticiones de lectura se envíen a las duplicaciones del servidor primario. Esto permite compartir la carga de las peticiones de lectura entre varias JVM; no obstante, enviar peticiones de lectura a las duplicaciones perjudica a la coherencia.

Esto se utiliza normalmente sólo cuando los clientes almacenan en antememoria datos que están cambiando todo el tiempo y cuando los clientes utilizan el bloqueo pesimista.

Si los datos están cambiando continuamente e invalidándose en antememorias cercanas de cliente, la unidad primaria observará como resultado una tasa de peticiones get relativamente alta de los clientes. De la misma forma, en la

modalidad de bloqueo pesimista, no existe ninguna antememoria local, por lo que todas las peticiones se envían a la unidad primaria.

Si los datos son relativamente estáticos o si no se utiliza la modalidad pesimista, la lectura de duplicaciones no tendrá un gran impacto en el rendimiento, ya que la frecuencia de las peticiones get de los clientes con antememorias en caliente no será muy alta.

No obstante, cuando un cliente se inicia por primera vez, su antememoria cercana está vacía y las peticiones de antememoria a dicha memoria vacía se envían a la unidad primaria. La antememoria de cliente obtiene datos con el tiempo, lo que disminuye esta carga de peticiones. Si existe un gran número de cliente y muchos de ellos se inician simultáneamente, esta carga puede ser significativa y las lecturas de duplicaciones serán una opción de rendimiento recomendable.

Lecturas de duplicaciones y duplicación asíncrona

Si los datos en el grupo de duplicación no cambian a menudo, esta suele ser una buena opción. Las peticiones get de los clientes se pueden direccionar a los datos en las duplicaciones que estén en línea. Puede que se envíe una petición get a una duplicación que no tenga una copia, y puede que el par de clave/valor no se haya duplicado todavía en la duplicación. Si los datos no están en la duplicación, la petición get se redireccionará a la unidad primaria.

Si los datos cambian, es muy probable que las peticiones get de las duplicaciones devuelvan datos obsoletos. Esto puede ser aceptable o no para la aplicación. Si no es aceptable, no habilite las lecturas desde las duplicaciones.

Lecturas de duplicaciones en modalidad de duplicación síncrona

La duplicación síncrona intenta mantener la duplicación exactamente igual que la unidad primaria. Si la unidad primaria falla, se garantiza que todos los datos comprometidos en la unidad primaria estarán disponibles en todas las duplicaciones que estuvieran en modalidad de igual cuando se produjo la anomalía. Este es el caso cuando se producen anomalías; no obstante, si se permiten lecturas de las duplicaciones, se exponen algunos efectos secundarios de los algoritmos utilizados.

Cuando la unidad primaria va a comprometer una transacción, se envía una copia de los cambios a la duplicación y la duplicación compromete esta transacción en los dos casos siguientes:

- La unidad primaria falla
- Se envía la siguiente transacción en la unidad primaria

Cuando falla la unidad primaria, se comprometen todas las transacciones pendientes en la duplicación.

Las transacciones pendientes sólo se comprometen cuando se compromete una transacción posterior en la unidad primaria. La unidad primaria concatena en este mensaje de duplicación el resultado del compromiso. Cuando la duplicación recibe uno de estos mensajes, compromete o retrotrae las transacciones pendientes que tuvieran resultados especificados en ese mensaje.

Las transacciones pendientes sólo están visibles para la lectura en una duplicación cuando están comprometidas. Obviamente, si la unidad primaria se carga e incluye modificaciones regulares, estas transacciones pendientes se comprometen muy

rápidamente. Si la carga de la modificación en la unidad primaria es lenta, habrá periodos en los que no se comprometerán transacciones pendientes, hasta que se realice la siguiente modificación primaria.

De esta forma, la duplicación de una unidad primaria que acepta modificaciones normalmente estará al menos una transacción por detrás de la unidad primaria desde el punto de vista de la lectura. No se pierden datos; estas transacciones están físicamente en la duplicación, simplemente no se comprometen hasta que se envía el resultado de las transacciones pendientes desde la unidad primaria. Este compromiso se realiza cuando se ejecuta la siguiente transacción de lectura y grabación.

Resumen

Si la lectura de la duplicación está habilitada, la aplicación debe estar preparada para tolerar algunas peticiones get que devuelvan datos obsoletos. Este problema se produce tanto si se utiliza la duplicación síncrona como asíncrona.

Particiones

Utilice particiones cuando los objetos del MapSet requieran más memoria de la que hay disponible en una única máquina virtual Java (JVM), o si la JVM no puede proporcionar la productividad necesaria para las actualizaciones.

¿Dónde se mantienen las entradas?

Un algoritmo hash determina qué servidor mantiene cada entrada. El administrador especifica el número de particiones que se utilizan con la definición de PartitionSet. Esta configuración no se puede modificar después de que se inicien las JVM. Se obtiene un valor hash simple a partir de la clave de una entrada y el resultado de este valor modula (%) el número de particiones e indica qué servidor ″posee″ dicha entrada.

Normalmente, se utiliza el método hashCode de Java en el objeto de clave. Altere temporalmente este valor alterando temporalmente la implementación de hashCode.

A veces, puede que una aplicación no desee modificar el valor del algoritmo hash normal y prefiera utilizar otro algoritmo hash para la distribución de entradas. La interfaz com.ibm.websphere.objectgrid.plugins.PartitionableKey permite esta situación. Esta interfaz tiene un único método:

```
Object ibmGetPartition();
```

Si la clave implementa esta interfaz, el tiempo de ejecución de ObjectGrid utiliza el algoritmo hash del objeto que devuelve este método, en lugar del algoritmo hash del objeto de clave.

Particiones en el tiempo de ejecución

La interfaz com.ibm.websphere.objectgrid.PartitionManager proporciona varias API para que una aplicación pueda determinar información sobre las particiones en el tiempo de ejecución. Una aplicación puede obtener una referencia a una instancia de esta interfaz utilizando el método getPartitionManager de la interfaz BackingMap. La referencia a la BackingMap de una correlación se puede obtener utilizando el método getMap(String) de la interfaz ObjectGrid en cualquier instancia de ObjectGrid. O bien, se pasa como parámetro en algunos de los retornos de

llamada de plug-in como, por ejemplo, preload(Session, BackingMap) de la interfaz de cargador.

Métodos de la interfaz PartitionManager

La instancia de PartitionManager permite a una aplicación determinar los siguientes hechos sobre las particiones:

Nombre de método	Descripción
int getNumOfPartitions()	Devuelve el número de particiones en que se divide la correlación.
int getPartition(Object key)	Devuelve el número de particiones basadas en 0 que se utiliza para la entrada con la clave especificada.
List /*Integer*/ getPartitions(List /*Object*/ keys)	Este método es el mismo que el método getPartition, pero trabaja en una lista de claves. La lista de enteros devuelta contiene el número de particiones de cada clave de entrada correspondiente.
List /*List Integer*/ getPartitionLists(List /* Object */ keys)	Este método es el mismo que el método getPartitions, pero devuelve una lista ordenada de listas de particiones. Por ejemplo, la primera entrada en la lista devuelta contiene una lista de las claves de entrada correspondientes a la partición 0. La siguiente entrada contendrá una lista de las claves de entrada correspondientes a la partición 1, etc.
List /*LogSequence*/ partitionLogSequence(LogSequence ls)	Este método divide una LogSequence en una lista de LogSequences para las particiones especificadas. Se examina la LogSequence de entrada y se determina la partición correspondiente para cada LogElement que contiene. Después de examinar la secuencia, para cada partición que tenga un LogElement, se devuelve una LogSequence de dichos LogElements.

Limitaciones de las particiones

Una transacción sólo puede modificar entradas en una única partición por transacción. Si una transacción modifica varias entradas en un MapSet y dichas entradas aplican hash a distintas particiones, la transacción se retrotrae cuando se intenta comprometer. Una transacción puede leer objetos de distintas particiones. No obstante, una transacción sólo puede modificar entradas en una única partición.

Sucesos de aplicación cuando se elige la unidad primaria de una partición

Si se proporciona un cargador para una correlación y el cargador también implementa ReplicaPreloadController, la aplicación puede utilizar el retorno de llamada checkPreloadStatus para recibir un suceso que indique que la JVM que recibe dicha llamada de método es ahora la unidad primaria de la partición. El ID de partición se puede identificar utilizando el método getPartitionId de la interfaz BackingMap. Consulte “Cargadores” en la página 202 para obtener más información sobre la precarga.

Particiones en un cliente comparado con la ejecución en un servidor

Las particiones sólo funcionan cuando la aplicación utiliza una ObjectGrid que se obtiene con los métodos de conexión de la interfaz ObjectGrid. Si la ObjectGrid se proporciona a la aplicación utilizando un retorno de llamada en un plug-in, se trata de una ObjectGrid local que no realiza ningún direccionamiento. Si ejecuta un servidor y desea aprovechar las posibilidades de partición de forma transparente, utilice la ObjectGrid que se obtiene utilizando un método de conexión para todas las transacciones. No obstante, el rendimiento disminuye si se compara con el uso de la interfaz ObjectGrid local que proporciona la infraestructura. Si no necesita la posibilidad de particiones, utilice la referencia local que se proporciona en el plug-in siempre que sea posible.

Indexación

La característica de indexación se pueden utilizar para crear uno o varios índices en una BackingMap. Un índice se crea a partir de un atributo de un objeto en la BackingMap. Esta característica proporciona a las aplicaciones una forma de buscar más rápidamente determinados objetos. Si el índice, las aplicaciones tienen que localizar los objetos por sus claves. La característica de indexación permite a las aplicaciones buscar objetos con un valor específico o en un rango de valores. Esto es parecido a la consulta de EJB (Enterprise JavaBeans) que puede localizar objetos EJB consultando con un criterio especificado. La indexación ofrece a las aplicaciones la comodidad de buscar más fácilmente los objetos y un aumento del rendimiento en el proceso de búsqueda de objetos.

Existen dos tipos de indexación: estática y dinámica. Con la indexación *estática*, debe configurar el plug-in de índice en la BackingMap antes de inicializar la instancia de ObjectGrid. Puede realizar esta configuración con XML o puede realizar la configuración mediante programación de la BackingMap. La indexación estática empieza creando un índice durante la inicialización de ObjectGrid. El índice está sincronizado siempre con la BackingMap y preparado para utilizarse. Una vez iniciado el proceso de indexación estática, el mantenimiento del índice forma parte del proceso de gestión de transacciones de ObjectGrid. Cuando las transacciones comprometen cambios, estos cambios actualizan también el índice estático. Los cambios de índice se retrotraen si se retrotrae la transacción.

La indexación dinámica permite crear un índice en una BackingMap antes o después de la inicialización de la instancia de ObjectGrid que lo contiene. Las aplicaciones tienen un control de ciclo de vida sobre el proceso de indexación dinámica. Un índice dinámico se puede eliminar cuando ya no sea necesario. Cuando una aplicación crea un índice dinámico, puede que el índice no esté preparado para su uso inmediato debido al tiempo que tarda en terminar el proceso de creación del índice. Como la cantidad de tiempo depende de la cantidad de datos indexados, se proporciona la interfaz DynamicIndexCallback para las aplicaciones que deseen recibir notificaciones cuando se produzcan determinados sucesos de indexación. Estos sucesos incluyen preparado, error y destruir. Las aplicaciones pueden implementar esta interfaz de retorno de llamada y registrarse con el proceso de indexación dinámica.

La característica de indexación se representa mediante el plug-in MapIndexPlugin, o Índice de forma abreviada. El MapIndexPlugin es un plug-in de BackingMap. Una BackingMap puede tener varios plug-ins Index configurados, siempre que sigan las normas de configuración de Index.

Si una BackingMap tiene configurado un plug-in de índice, el objeto de proxy de índice se puede recuperar de la ObjectMap correspondiente. Si se invoca el método getIndex en la ObjectMap y se pasa el nombre del plug-in de índice, se devuelve el objeto de proxy de índice. El objeto de proxy de índice se tiene que convertir en una interfaz de índice de aplicación adecuada como, por ejemplo, MapIndex, MapRangeIndex o una interfaz de índice personalizada.

Actualmente, la característica de indexación sólo está soportada en la antememoria local, no en la antememoria distribuida. Si se intenta realizar una operación de indexación en una antememoria distribuida, se genera la excepción `UnsupportedOperationException`.

Implementación del plug-in de índice

La clase HashIndex en el paquete com.ibm.websphere.objectgrid.plugins.index es la implementación de plug-in de índice incorporada que puede dar soporte a las dos interfaces de índice de aplicación incorporadas: MapIndex y MapRangeIndex.

Las aplicaciones pueden proporcionar su propia implementación de plug-in de índice para permitir la programación de índices más complejos. La clase de implementación de índice debe implementar la interfaz com.ibm.websphere.objectgrid.plugins.index.MapIndexPlugin. El MapIndexPlugin tiene la siguiente definición:

```
/**
* Una implementación de índice debe implementar esta interfaz para que las modificaciones
* en la correlación se propaguen de forma que pueda mantener el índice mientras
* se comprometen las transacciones. Sólo se pueden elegir atributos que implementen
* {@link java.lang.Comparable} para indexarlos.
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapIndex
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex
*/
public interface MapIndexPlugin
{
/**
* Debe ser el nombre del atributo que se va a indexar. Si el objeto
* tiene un atributo denominado EmployeeName, el índice invocará el
* método "getEmployeeName". El nombre de atributo debe ser el
* el nombre que aparece en el método get y el atributo debe implementar
* la interfaz {@link java.lang.Comparable}.
*
* @param attributeName
* El nombre del atributo que se va a establecer.
*/
public void setAttributeName(String attributeName);
/**
* El nombre de este índice.
*
* @return El nombre del índice.
*
* @see com.ibm.websphere.objectgrid.ObjectMap#getIndex
*/
 String getName();
/**
* Obtiene un objeto de proxy de índice para realizar operaciones de búsqueda de índice. El
* emisor debe convertir el objeto devuelto a un objeto MapIndex o MapRangeIndex
* para realizar las operaciones de búsqueda.
*
* @param map El objeto MapIndexInfo necesario para mantener el índice.
* .
* @return un proxy a un objeto que implementa MapIndex o MapRangeIndex.
*/
```

```
Object getIndexProxy(MapIndexInfo map);
/**
* Lo invoca el principal para que el índice pueda actualizarse como resultado de
* los cambios aplicados a la correlación durante el ciclo de compromiso de una transacción.
* Utilice el método {@link LogElement#getType()} para determinar qué operación se
* necesita para actualizar el índice. Utilice {@link LogElement#getBeforeImage()} para
* obtener el objeto de valor que existía antes de comprometer la transacción aplicando
* un cambio en la correlación y {@link LogElement#getAfterImage()} para obtener el objeto
* de valor después de que la transacción de compromiso aplique el cambio a la entrada
* de correlación.
*
* Tenga en cuenta que el método {@link #undoBatchUpdate(TxID, LogSequence)} se puede
* invocar posteriormente para deshacer estos cambios si se produce una excepción
* que provoca la retrotracción de las transacciones de compromiso.
*
* @param txid La transacción de los cambios.
* @param sequence La secuencia de anotaciones que contiene los cambios de la transacción.
*
* @throws ObjectGridRuntimeException si se produce una anomalía que necesita la
* retrotracción de la transacción.
*/
void doBatchUpdate(TxID txid, LogSequence sequence) throws
ObjectGridRuntimeException;
/**
* Lo invoca el principal para deshacer todos los cambios realizados en el índice como
* resultado de una llamada previa al método {@link #doBatchUpdate(TxID, LogSequence)}.
* Se invoca cuando se produce una excepción o un error que requiere retrotraer
* todos los cambios realizados por una transacción. Por este motivo, la implementación
* de este método debe capturar todos los objetos Throwable y continuar con el
* siguiente LogElement de LogSequence hasta que se procesen todos los LogElements de
* forma que se deshagan tantos cambios en el índice como sea posible. Sólo se debe
* generar una ObjectGridException después de procesar toda la LogSequence y si este
* método no ha podido deshacer correctamente 1 o más cambios en la LogSequence.
*
* Utilice el método {@link LogElement#getUndoType()} para determinar qué operación
* se necesita para deshacer un cambio realizado en el índice. Utilice
* {@link LogElement#getBeforeImage()} para obtener el objeto de valor que existía
* antes de comprometer la transacción aplicando un cambio en la correlación y {@link
* LogElement#getAfterImage()} para obtener el objeto de valor después de que la
* transacción de compromiso aplique el cambio a la entrada de correlación.
*
* @param txid La transacción de los cambios.
* @param sequence La secuencia de anotaciones que contiene los cambios de la
transacción.
*
*/
void undoBatchUpdate( TxID txid, LogSequence sequence) throws ObjectGridException;
}
```

Los métodos setAttributeName y getName son directos y los revelan sus nombres. Los otros métodos requieren más atención.

Método getIndexProxy

El método getIndexProxy debe devolver un objeto de proxy de índice que implementa la interfaz MapIndex, la interfaz MapRangeIndex o una interfaz Index personalizada. La implementación del objeto de proxy de índice es la parte principal del plug-in de índice.

Se pasa un objeto MapIndexInfo a este método para proporcionar información de cambios transaccionales. Estos son los datos que están visibles sólo para la transacción actual que invoca el método getIndexProxy. El objeto de proxy de índice puede utilizar este objeto MapIndexInfo para buscar los datos transaccionales.

A continuación se proporciona la definición de la interfaz MapIndexInfo:

```
/**
* Esta interfaz se utiliza para proporcionar a un índice información detallada de
* los cambios para una correlación específica en una transacción.
*/
public interface MapIndexInfo
{
/**
* Un índice contiene los valores de clave de un conjunto de entradas de correlación
* que tienen un valor de atributo específico. Este método devuelve la ObjectMap al
* que el índice hace referencia a la ObjectMap con la que está asociado el índice.
*
* @return ObjectMap con la que está asociado este índice.
*/
ObjectMap getMap();
/**
* Devuelve el conjunto de todos los cambios realizados por la transacción actual
* a la ObjectMap que devuelve el método {@link #getMap()}.
*
* @param includeRemoved debe establecerse en true para incluir tipos LogElement.DELETE
* en la lista devuelta por este método.
*
* @return Lista de LogElement creados para cada entrada de ObjectMap que ha
* insertado, actualizado o eliminado la transacción actual.
*
* @throws ObjectGridRuntimeException
*/
List getTransactionChanges(boolean includeRemoved) throws
ObjectGridRuntimeException;
/**
* Devuelve el conjunto de cambios que se aplican a un determinado conjunto de claves
* en la transacción actual para la ObjectMap que devuelve el método
* {@link #getMap()}. Si no se ha hecho referencia a una clave en la
* transacción, se devuelve nulo.
*
* @param keys La lista de claves para las que se necesitan los datos.
* @return Lista de LogElement correspondiente a las claves o nulo si no se hace
* referencia a las claves.
*
* @throws ObjectGridRuntimeException
*
* @see com.ibm.websphere.objectgrid.plugins.LogElement
* @see com.ibm.websphere.objectgrid.ObjectMap
*/
List getTransactionChanges(List keys) throws ObjectGridRuntimeException;
}
```

El método getIndexProxy está diseñado para dar soporte al método getIndex(String name) de la interfaz ObjectMap. El objeto de proxy de índice devuelto será el objeto devuelto por el método getIndex de ObjectMap. Por ejemplo, la aplicación invoca el método getIndex de ObjectMap que, a continuación, invoca este método getIndexProxy y devuelve el objeto que devuelve este método getIndexProxy. La aplicación tiene que convertir el objeto de proxy de índice devuelto a una interfaz de índice de aplicación como, por ejemplo, MapIndex, MapRangeIndex u otra interfaz de índice personalizada.

El siguiente ejemplo de código muestra algunas implementaciones de objeto de proxy de índice que puede devolver el método getIndexProxy:

```
/**
* Una clase utilizada para devolver un proxy a este índice de correlación
* de forma que las aplicaciones puedan realizar operaciones de consulta
* utilizando la interfaz MapIndex.
*/
class Proxy implements MapIndex
```

```
{
/**
* El objeto MapIndexInfo asociado con este objeto de proxy de índice.
*/
protected MapIndexInfo ivMap;
/**
* Número máximo de reintentos cuando varias transacciones concurrentes
* modifican el índice durante una operación de consulta.
*/
protected static final int RETRY_LIMIT = 10;
/**
* Comparador EQUAL que se utiliza.
*/
final protected ProxyEQComparator ivEQ = new ProxyEQComparator();
final protected ProxyGTComparator ivGT = new ProxyGTComparator();
final protected ProxyRangeComparator ivRange = new ProxyRangeComparator();
/**
* Construir un objeto de proxy para una determinada ObjectMap.
*
* @param map
* es el objeto MapIndexInfo.
*/
Proxy(MapIndexInfo map)
{
ivMap = map;
}
/**
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapIndex#findAll
*/
public Iterator findAll(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"el attributeValue debe ser una referencia no nula" );
}
// Utilizar el comparador mayor que para la comprobación de rango.
ivEQ.ivAttribute = (Comparable) attributeValue;
ArrayList resultList = null;
int retryCount = 0;
boolean retry;
do
{
// Variables que se deben reinicializar cada vez mediante bucle.
retry = false;
resultList = new ArrayList();
// Utilice índice para obtener el conjunto (Set) de claves para correlacionar
// entradas que contengan el valor de atributo especificado.
Set s = (Set) index.get( attributeValue );
Set keySet = processSet( s, ivEQ );
if ( keySet != null )
{
resultList.addAll( keySet );
}
else
{
// Otra transacción ha modificado el conjunto (Set) obtenido del índice
// mientras la anterior iteraba el conjunto para ejecutar la
// operación addAll. Por lo tanto, se debe volver a intentar empezando
// por el principio con la obtención del conjunto del índice para elegir
// los cambios en la transacción que acaba de modificar el conjunto.
++retryCount;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
```

```
retry = true;
}
} while ( retry );
// Devolver el iterador de lista de resultados creada por el bucle anterior.
Iterator result = resultList.iterator();
  return result;
}
/**
* Procesar un conjunto obtenido del índice para determinar si las claves son
* para entradas de correlación que cumplen los criterios de selección de consulta.
*
* @param s
* es el conjunto de valores de clave para las entradas en BackingMap
* sobre el que se crea este índice. Una referencia nula indica sólo
* que se deben procesar los cambios de la transacción actual.
*
* @param comparator
* es el comparador que se utiliza para realizar la comprobación de rango.
*
* @return Conjunto de claves que cumplen los criterios de selección o una referencia
* nula si se produce una excepción al iterar sobre el conjunto.
*
* @throws FinderException
* si una condición de error no permite la ejecución del
* proceso del conjunto.
*/
protected Set processSet(Set s, ProxyComparator comparator)
throws FinderException {
HashSet resultSet = new HashSet();
//...
//procesar el conjunto, utilizar el comparador y preparar resultSet.
//...
return resultSet;
}
} // finalizar proxy de clase
/**
* Una clase utilizada para devolver un proxy a este índice de correlación para que
* las aplicaciones puedan realizar operaciones de consulta con la interfaz MapRangeIndex.
*/
class RangeProxy extends Proxy implements MapRangeIndex
{
/**
* Varios comparadores que necesita el proxy para realizar la
* comprobación de rango del valor de atributo.
*/
final private ProxyLTComparator ivLT = new ProxyLTComparator();
final private ProxyLEComparator ivLE = new ProxyLEComparator();
final private ProxyGEComparator ivGE = new ProxyGEComparator();
/**
* El índice es una SortedMap sincronizada.
*/
final SortedMap ivIndexSortedMap;
/**
* Construir un proxy MapRangeIndex.
*/
RangeProxy(MapIndexInfo map)
{
super( map );
ivIndexSortedMap = (SortedMap) index;
}
/**
* Ejecutar operación de consulta en una correlación especificada y un objeto ProxyComparator.
*
* @param map
* subconjunto del índice en que realizar la operación de finder.
* @param proxyComparator
* comparador utilizado para realizar la comprobación de rango en el valor de atributo.
```

```
*
* @return Conjunto de claves que debe devolver el método finder.
*
* @throws FinderException
* es una anomalía que se produce durante la ejecución de la consulta.
*/
private Set executeQuery(Map map, ProxyComparator proxyComparator)
throws FinderException {
HashSet resultList = null;
int retryCount = 0;
boolean retry;
do
{
// Variables que se deben reinicializar cada vez mediante bucle.
retry = false;
resultList = new HashSet();
// Utilice índice para obtener el conjunto de claves para correlacionar
// entradas que contengan el valor de atributo especificado.
SortedMap treeMap = (SortedMap) index;
Collection values = map.values();
if ( values.isEmpty() )
{
// El índice no contiene nada actualmente en el rango, por lo que sólo
// tenemos que comprobar los cambios de la transacción actual.
Set keySet = processSet( null, proxyComparator );
if ( keySet != null )
{
resultList.addAll( keySet );
}
}
else
{
// El índice contiene algunas claves en el rango, por lo que debemos
// consultar ambas entradas de índice así como los cambios de transacción actuales.
Iterator iter = values.iterator();
 while ( iter.hasNext() )
{
Set keySet;
try
{
Set s = (Set) iter.next();
keySet = processSet( s, proxyComparator );
}
catch (ConcurrentModificationException e)
{
// Indicar que no se puede obtener keySet.
keySet = null;
}
if ( keySet != null )
{
resultList.addAll( keySet );
}
else
{
++retryCount;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
retry = true;
}
}
}
} while ( retry );
return resultList;
}
/*
```

```
* (no Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findGreater
*/
public Iterator findGreater(Object attributeValue)
throws FinderException {
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"el attributeValue debe ser una referencia no nula" );
}
// Utilizar el comparador mayor que para la comprobación de rango.
ivGT.ivAttribute = (Comparable) attributeValue;
SortedMap tailMap = ivIndexSortedMap.tailMap( attributeValue );
Set resultSet = executeQuery( tailMap, ivGT );
Iterator result = resultSet.iterator();
  return result;
}
/*
* (no Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findGreaterEqual
*/
public Iterator findGreaterEqual(Object attributeValue)
throws FinderException {
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"el attributeValue debe ser una referencia no nula" );
}
// Utilizar el comparador mayor que para la comprobación de rango.
ivGE.ivAttribute = (Comparable) attributeValue;
SortedMap tailMap = ivIndexSortedMap.tailMap( attributeValue );
Set resultSet = executeQuery( tailMap, ivGE );
Iterator result = resultSet.iterator();
  return result;
}
/*
* (no Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findLess
*/
public Iterator findLess(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"el attributeValue debe ser una referencia no nula" );
}
// Utilizar el comparador mayor que para la comprobación de rango.
ivLT.ivAttribute = (Comparable) attributeValue;
SortedMap headMap = ivIndexSortedMap.headMap( attributeValue );
Set resultSet = executeQuery( headMap, ivLT );
Iterator result = resultSet.iterator();
  return result;
}
/*
* (no Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findLessEqual
*/
public Iterator findLessEqual(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"el attributeValue debe ser una referencia no nula" );
```

```
}
// Utilizar el comparador mayor que para la comprobación de rango.
ivLE.ivAttribute = (Comparable) attributeValue;
Set resultSet;
int retryCount = 0;
boolean retry;
do
{
// reinicializarse para cada reintento que se produzca.
retry = false;
SortedMap headMap = ivIndexSortedMap.headMap( attributeValue );
resultSet = executeQuery( headMap, ivLE );
Set s = (Set) ivIndexSortedMap.get( attributeValue );
ivEQ.ivAttribute = (Comparable) attributeValue;
Set equalSet = processSet( s, ivEQ );
if ( equalSet != null )
{
    if ( ! equalSet.isEmpty() )
{
resultSet.addAll( equalSet );
}
}
else
{
// Otra transacción ha modificado el índice mientras se ejecutaba
// processSet. Por lo tanto, se debe reintentar la consulta completa.
++retryCount;
retry = true;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
}
} while ( retry );
// Devolver el iterador de lista de resultados creada por el bucle anterior.
Iterator result = resultSet.iterator();
  return result;
}
/*
* (no Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findRange
*/
public Iterator findRange(Object lowAttributeValue, Object highAttributeValue)
throws FinderException {
if ( lowAttributeValue == null )
{
throw new IllegalArgumentException(
"el lowAttributeValue debe ser una referencia no nula" );
}
if ( highAttributeValue == null )
{
throw new IllegalArgumentException(
"el highAttributeValue debe ser una referencia no nula" );
}
// Utilizar el comparador mayor que para la comprobación de rango.
ivRange.ivLowAttribute = (Comparable) lowAttributeValue;
ivRange.ivHighAttribute = (Comparable) highAttributeValue;
SortedMap subMap = ivIndexSortedMap.
subMap( lowAttributeValue, highAttributeValue );
Set resultSet = executeQuery( subMap, ivRange );
Iterator result = resultSet.iterator();
  return result;
}
}
/**
* Clase base abstracta utilizada para determinar si el valor de atributo está en el rango.
```

```
*/
abstract class ProxyComparator
{
abstract boolean inRange(Object attribute);
}
/**
* Ejecuta la comprobación de rango menor que.
*/
class ProxyLTComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
  return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) < 0 );
}
}
}
/**
* Ejecuta la comprobación de rango menor o igual que.
*/
class ProxyLEComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
  return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) <= 0 );
}
}
}
/**
* Ejecuta la comprobación de rango igual que.
*/
class ProxyEQComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
  return false;
}
else
{
return ( ivAttribute.compareTo( attribute ) == 0 );
}
}
}
/**
* Ejecuta la comprobación de rango mayor que.
*/
class ProxyGTComparator extends ProxyComparator
{
Comparable ivAttribute;
```

```
boolean inRange(Object attribute)
{
if ( attribute == null )
{
  return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) > 0 );
}
}
}
/**
* Ejecuta la comprobación de rango mayor o igual que.
*/
class ProxyGEComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
  return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) >= 0 );
}
}
}
/**
* Ejecuta la comprobación de rango lowAttribute <= attribute < highAttribute.
*/
class ProxyRangeComparator extends ProxyComparator
{
Comparable ivLowAttribute;
Comparable ivHighAttribute;
boolean inRange(Object o)
{
if ( o == null )
{
  return false;
}
Comparable attribute = (Comparable) o;
if ( attribute.compareTo( ivLowAttribute ) < 0 )
{
return false; // attribute < ivLowAttribute
}
else
{
// ivLowAttribute <= attribute
if ( attribute.compareTo( ivHighAttribute ) < 0 )
{
return true; // ivLowAttribute <= attribute < ivHighAttribute
}
else
{
return false; // attribute >= ivHighAttribute
}
}
}
}
```

Métodos doBatchUpdate y undoBatchUpdate

Los métodos doBatchUpdate y undoBatchUpdate son métodos fundamentales en la interfaz MapIndexPlugin. El método doBatchUpdate se invoca como resultado de los cambios aplicados a la correlación durante el ciclo de compromiso de una transacción. El método undoBatchUpdate se utiliza para deshacer todos los cambios realizados en el índice como resultado de una llamada anterior al método doBatchUpdate. Se invoca cuando se produce una excepción o una condición de error que requiere la retrotracción de todos los cambios realizados por la transacción. A ambos métodos se les proporciona el TxID actual y una lista de cambios para esta correlación. Deben iterar los cambios y procesarlos.

El siguiente ejemplo de código muestra cómo implementar estos dos métodos y los métodos a los que dan soporte.

```
/**
* Correlación (Map) sincronizada que se utiliza como implementación de índice donde
* el objeto de valor de atributo es la clave y un conjunto Java (Java Set) es el valor.
* Un miembro del conjunto es la clave de una entrada BackingMap que coincide con el
valor de atributo.
*/
Map index; //<Object attribute, Set keys>
public void doBatchUpdate(TxID txid, LogSequence sequence)
throws ObjectGridRuntimeException
{
 Iterator iter = sequence.getAllChanges();
 while ( iter.hasNext() )
{
LogElement elem = (LogElement) iter.next();
  Object key = elem.getCacheEntry().getKey();
LogElement.Type doType = elem.getType();
if ( doType == LogElement.INSERT )
{
Object newAttribute = getAttribute( elem.getAfterImage() );
insertIntoIndex( key, newAttribute );
}
else if ( doType == LogElement.UPDATE )
{
Object newAttribute = getAttribute( elem.getAfterImage() );
Object oldAttribute = getAttribute( elem.getBeforeImage() );
updateIndex( key, oldAttribute, newAttribute );
}
else if ( doType == LogElement.DELETE )
{
Object oldAttribute = getAttribute( elem.getBeforeImage() );
removeFromIndex( key, oldAttribute );
}
else if ( doType == LogElement.EVICT )
{
Object beforeImage = elem.getBeforeImage();
if ( beforeImage != null )
{
Object oldAttribute = getAttribute( beforeImage );
removeFromIndex( key, oldAttribute );
}
}
}
}
public void undoBatchUpdate(TxID txid, LogSequence sequence)
throws ObjectGridException
{
int errors = 0;
 Iterator iter = sequence.getAllChanges();
 while ( iter.hasNext() )
{
```

```
try
{
LogElement elem = (LogElement) iter.next();
  Object key = elem.getCacheEntry().getKey();
LogElement.Type undoType = elem.getUndoType();
if ( undoType == LogElement.INSERT )
{
Object newAttribute = getAttribute( elem.getBeforeImage() );
insertIntoIndex( key, newAttribute );
}
else if ( undoType == LogElement.UPDATE )
{
Object oldAttribute = getAttribute( elem.getAfterImage() );
Object newAttribute = getAttribute( elem.getBeforeImage() );
updateIndex( key, oldAttribute, newAttribute );
}
else if ( undoType == LogElement.DELETE )
{
Object oldAttribute = getAttribute( elem.getAfterImage() );
removeFromIndex( key, oldAttribute );
}
}
catch ( Throwable t )
{
++errors;
}
}
if ( errors > 0 )
{
throw new ObjectGridException( errors
+ " se han producido excepciones durante la retrotracción de los cambios
de índice.");
}
}
/**
* Extrae el atributo de un objeto de valor especificado.
*
* @param value El objeto de valor.
*
* @return atributo del objeto de valor, que puede ser una referencia nula.
*
* @throws ObjectGridRuntimeException se genera si se produce una excepción
* al intentar extraer el valor de atributo del objeto de valor.
*/
private Object getAttribute(Object value) throws ObjectGridRuntimeException
{
try
{
Object attribute = null;
 if ( value != null )
{
Method m = getAttributeMethod( value );
attribute = getAttributeMethod.invoke( value, emptyArray );
}
return attribute;
}
catch ( InvocationTargetException e )
{
Throwable t = e.getTargetException();
throw new ObjectGridRuntimeException( "Throwable inesperado capturado", t );
}
catch ( Throwable t )
{
throw new ObjectGridRuntimeException( "Throwable inesperado capturado", t );
}
}
private void updateIndex(Object key, Object oldAttribute, Object newAttribute)
```

```
{
// ¿La actualización ha cambiado el atributo?
if ( newAttribute != null && oldAttribute != null &&
oldAttribute.equals( newAttribute ) )
{
// No, por lo tanto nada se debe cambiar en el índice.
return;
}
// Si no se restringe el cargador a acceder sólo a tablas con columnas sin posibilidad
// de nulos, debemos considerar la posibilidad de que el atributo sea nulo.
Set oldKeys = null;
if ( oldAttribute != null )
{
// Eliminar oldAttribute de entrada de índice.
oldKeys = (Set) index.get( oldAttribute );
if ( oldKeys != null )
{
oldKeys.remove( key );
if ( oldKeys.isEmpty() )
{
index.remove( oldAttribute );
}
}
}
// Si no se restringe el cargador a acceder sólo a tablas con columnas sin
posibilidad
// de nulos, debemos considerar la posibilidad de que el atributo sea nulo.
Set keys = null;
if ( newAttribute != null )
{
keys = (Set) index.get( newAttribute );
// Añadir newAttribute a índice.
if ( keys == null )
{
// Como distintas transacciones pueden estar actualizando distintas entradas
// BackingMap y varias entradas de correlación pueden tener el mismo valor de
// atributo, debemos utilizar un objeto de conjunto sincronizado para garantizar
// que sólo una transacción pueda realizar cambios cada vez en el conjunto.
keys = Collections.synchronizedSet( new HashSet() );
index.put( newAttribute, keys );
}
// Añadir clave de esta entrada de correlación al conjunto de claves del nuevo valor
de atributo.
keys.add( key );
}
}
private void insertIntoIndex( Object key, Object newAttribute )
{
// Si no se restringe el cargador a acceder sólo a tablas con columnas sin
posibilidad
// de nulos, debemos considerar la posibilidad de que el atributo sea nulo.
if ( newAttribute != null )
{
Set keys = (Set) index.get( newAttribute );
if ( keys == null )
{
// Como distintas transacciones pueden estar actualizando distintas entradas
// Map y varias entradas de correlación pueden tener el mismo valor de
// atributo, debemos utilizar un objeto de conjunto sincronizado para garantizar
// que sólo una transacción pueda realizar cambios cada vez en el conjunto.
keys = Collections.synchronizedSet( new HashSet() );
index.put( newAttribute, keys );
}
// Añadir clave de esta entrada de correlación al conjunto de claves del nuevo valor
de atributo.
keys.add( key );
}
```

```
}
private void removeFromIndex(Object key, Object oldAttribute )
{
// Extraer el valor de atributo anterior
Set oldKeys = null;
// Si no se restringe el cargador a acceder sólo a tablas con columnas sin
posibilidad
// de nulos, debemos considerar la posibilidad de que el atributo sea nulo.
if ( oldAttribute != null )
{
oldKeys = (Set) index.get( oldAttribute );
if ( oldKeys != null )
{
oldKeys.remove( key );
if ( oldKeys.isEmpty() )
{
index.remove( oldAttribute );
}
}
}
}
```

Interfaces de índice de aplicación

Las interfaces de índice de aplicación están diseñadas para dar soporte a métodos de consulta. Actualmente, hay definidas dos interfaces de índice de aplicación: MapIndex y MapRangeIndex.

MapIndex

MapIndex es un índice sencillo para buscar objetos por el valor de atributo. Permite indexar cualquier valor de atributo en una correlación. De esta forma, la aplicación puede buscar rápidamente todos los objetos de la correlación que tengan un valor de atributo específico. A continuación se proporciona la definición de la interfaz MapIndex:

```
/**
* Índice abstracto que se puede crear en una correlación vacía. El índice
* se puede utilizar para realizar búsquedas eficaces y probablemente otras
* operaciones, como operaciones relacionales en un atributo de una correlación.
* MapIndex se proporciona con todos los sucesos de actualización y mantiene un
* índice que se puede utilizar para emitir consultas sencillas en el índice
* posteriormente. El índice puede utilizar un retorno de llamada definido por el
* índice para realizar un índice en los atributos compuestos.
*/
public interface MapIndex
{
/**
* Devuelve las claves de las entradas que tienen el valor de atributo
* especificado.
*
* @param attributeValue
* una referencia no nula al valor de atributo que se debe buscar.
*
* @return Una lista de las claves de las entradas con ese atributo.
*
* @throws IllegalArgumentException si el argumento attributeValue es nulo.
* @throws FinderException se genera si se alcanza una excepción o el límite de
* reintentos cuando varias transacciones concurrentes que actualizan el índice
* impiden que finalice findAll.
*/
Iterator findAll(Object attributeValue) throws FinderException;
}
```

MapRangeIndex

MapRangeIndex es un índice sencillo para buscar objetos con un valor de atributo en un determinado rango. Permite indexar cualquier valor de

atributo en una correlación. Se diferencia de MapIndex en que permite utilizar rangos de valores y operaciones de comparación de valores en las consultas. De esta forma, las consultas pueden buscar todos los objetos con un valor de atributo menor o mayor que un valor específico. A continuación se proporciona la definición de la interfaz MapRangeIndex:

```
/**
* Índice que permite búsquedas de tipo de comparación.
*/
public interface MapRangeIndex extends MapIndex
{
/**
* Busca todas las claves con entradas con un atributo mayor que
* el valor especificado.
*
* @param attributeValue es el punto final bajo del rango, excluyendo el
* valor de atributo bajo.
*
* @return El conjunto de claves con valores mayores que el atributo.
*
* @throws IllegalArgumentException si el argumento attributeValue es nulo.
* @throws FinderException se genera si se alcanza una excepción o el
* límite de reintentos
* cuando varias transacciones concurrentes que actualizan el índice
* impiden que finalice findAll.
*/
Iterator findGreater(Object attributeValue) throws FinderException;
/**
* Busca todas las claves con entradas con un atributo mayor o igual que
* el valor especificado.
*
* @param attributeValue es el punto final bajo del rango, incluido el
* valor de atributo bajo.
*
* @return El conjunto de claves con atributos que cumplen los criterios
*
* @throws IllegalArgumentException si el argumento attributeValue es nulo.
* @throws FinderException se genera si se alcanza una excepción o el
* límite de reintentos
* cuando varias transacciones concurrentes que actualizan el índice
* impiden que finalice findAll.
*/
Iterator findGreaterEqual(Object attributeValue) throws FinderException;
/**
* Busca todas las claves con entradas con un atributo menor que
* el valor especificado.
*
* @param attributeValue es el punto final alto del rango, excluyendo el
* valor de punto final alto.
*
* @return El conjunto de claves con atributos que cumplen los criterios
*
* @throws IllegalArgumentException si el argumento attributeValue es nulo.
* @throws FinderException se genera si se alcanza una excepción o el límite
* de reintentos
* cuando varias transacciones concurrentes que actualizan el índice
* impiden que finalice findAll.
*/
Iterator findLess(Object attributeValue) throws FinderException;
/**
* Busca todas las claves con entradas con un atributo menor o igual que
* el valor especificado.
*
* @param attributeValue es el punto final alto del rango, incluido el
* valor de punto final alto.
*
* @return El conjunto de claves con atributos que cumplen los criterios
```

```
*
* @throws IllegalArgumentException si el argumento attributeValue es nulo.
* @throws FinderException se genera si se alcanza una excepción o el límite
* de reintentos
* cuando varias transacciones concurrentes que actualizan el índice
* impiden que finalice findAll.
*/
Iterator findLessEqual(Object attributeValue) throws FinderException;
/**
* Devuelve todas las claves de las entradas con el atributo incluido en
* el rango especificado, de forma que lowAttributeValue <= attribute
* < highAattributeValue.
*
* @param lowAttributeValue es el punto final bajo del rango, incluido el
* valor de atributo bajo.
* @param highAttributeValue es el punto final alto del rango, excluyendo el
* valor de atributo alto.
*
* @return La lista de claves con entradas en el rango, en orden ascendente.
*
* @throws IllegalArgumentException si el argumento lowAttributeValue o
* highAttributeValue
* es nulo o lowAttributeValue > highAttributeValue.
* @throws FinderException se genera si se alcanza una excepción o el límite de
* reintentos cuando varias transacciones concurrentes que actualizan el índice
* impiden que finalice findAll.
*/
Iterator findRange(Object lowAttributeValue, Object highAttributeValue)
throws FinderException;
}
```

Las aplicaciones deben convertir el objeto de índice obtenido del método getIndex de la instancia de ObjectMap en la interfaz de índice de aplicación deseada. Si el plug-in de índice está diseñado para dar soporte a la interfaz MapRangeIndex, el objeto de índice se puede convertir en el tipo MapRangeIndex; de lo contrario, se debe convertir en el tipo MapIndex.

Puede definir una interfaz de índice de aplicación personalizada. Implemente el índice de aplicación personalizada como el objeto de proxy de índice que puede devolver el método getIndexProxy del MapIndexPlugin. Convierta el objeto de índice obtenido del método getIndex de la instancia de ObjectMap en esta interfaz de índice de aplicación personalizada y utilícela.

Adición de plug-ins de índice estático

Existen dos enfoques para añadir plug-ins de índice estático a una configuración de BackingMap: configuración XML y configuración mediante programación. El siguiente ejemplo ilustra el enfoque de configuración 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="nombre de índice" />
<property name="RangeIndex" type="boolean" value="true" description="true
para MapRangeIndex" />
<property name="AttributeName" type="java.lang.String" value="employeeCode"
description="nombre de atributo" />
</bean>
</backingMapPluginCollection>
```

La interfaz BackingMap tiene dos métodos que se pueden utilizar para añadir plug-ins de índice estático: addMapIndexPlugin y setMapIndexPlugins. A continuación, se proporciona la definición de estos dos métodos.

```
/**
* Este método añade un plug-in de índice a esta correlación. Se supone que
* la implementación de índice se ha construido con el nombre
* del atributo que se va a indexar. El nombre del índice se especifica cuando
* se construye el índice.
*
* Para evitar un {@link IllegalStateException}, se debe invocar este método
* antes que el método {@link ObjectGrid#initialize()}. Asimismo, recuerde que
* el método {@link ObjectGrid#getSession()} llama implícitamente al método
* {@link ObjectGrid#initialize()} si todavía no lo ha invocado la
* aplicación.
*
* @param index La implementación del índice.
*
* @throws IndexAlreadyDefinedException Este índice ya existe.
* @throws IllegalStateException si se llama a este método después de
* llamar al método {@link ObjectGrid#initialize()}.
*/
public void addMapIndexPlugin(MapIndexPlugin index)
throws IndexAlreadyDefinedException;
/**
* Este método establece la lista de objetos MapIndexPlugin para esta BackingMap.
* Si la BackingMap ya tiene una lista de objetos MapIndexPlugin,
* esa lista se sustituirá por la lista pasada como
* argumento en la invocación actual de este método.
*
* Para evitar un {@link IllegalStateException}, se debe invocar este método
* antes que el método {@link ObjectGrid#initialize()}. Asimismo, recuerde que
* el método {@link ObjectGrid#getSession()} llama implícitamente al método
* {@link ObjectGrid#initialize()} si todavía no lo ha invocado la
* aplicación.
*
* @param indexList Una referencia no nula a una lista de objetos {@link
* MapIndexPlugin}.
*
* @throws IllegalArgumentException si genera si indexList es nula
* o la indexList contiene un objeto que no es una
* instancia de {@link MapIndexPlugin}.
*/
public void setMapIndexPlugins(List indexList );
```

El siguiente fragmento de código ilustra el enfoque de configuración mediante programación:

```
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 la clase de plug-in de índice.
HashIndex mapIndexPlugin = new HashIndex();
mapIndexPlugin.setName("CODE");
mapIndexPlugin.setAttributeName("EmployeeCode");
mapIndexPlugin.setRangeIndex(true);
personBackingMap.addMapIndexPlugin(mapIndexPlugin);
```

Utilización de índices estáticos

Después de añadir un plug-in de índice estático a una configuración de BackingMap e inicializar la instancia de ObjectGrid que lo contiene, las aplicaciones

pueden obtener el objeto de índice por el nombre de la instancia de ObjectMap de la BackingMap. Convierta el objeto de índice en la interfaz de índice de aplicación. Ahora se pueden ejecutar las operaciones de índice soportadas por la interfaz de índice de aplicación. A continuación, se proporciona la definición del método getIndex de la interfaz ObjectMap:

```
/**
* Devuelve una referencia al índice denominado que se puede utilizar con esta correlación.
* Este índice no se puede compartir entre las hebras y trabaja con las mismas normas que
* una Session. El valor devuelto se debe convertir a la interfaz de índice correcta como,
* por ejemplo, MapIndex o MapRangeIndex, o a una interfaz de índice personalizada
* como el índice geoespacial.
*
* @param name El nombre del índice
*
* @return Una referencia al índice; se debe convertir a la interfaz de índice
* correspondiente.
*
* @throws IndexUndefinedException si el índice no está definido en la BackingMap
* @throws IndexNotReadyException si el índice no está preparado
* @throws UnsupportedOperationException si la correlación es una correlación distribuida
*/
Object getIndex(String name)
throws IndexUndefinedException, IndexNotReadyException,
UnsupportedOperationException;
```

El siguiente fragmento de código ilustra la forma de obtener y utilizar 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 y eliminación de índices dinámicos

Los índices dinámicos se pueden crear y eliminar en una instancia de BackingMap mediante programación en cualquier momento. Un índice dinámico se diferencia de un índice estático en que el índice dinámico se puede crear después de que se haya inicializado la instancia de ObjectGrid que lo contiene. A diferencia de la indexación estática, la indexación dinámica es un proceso asíncrono y debe estar en estado preparado previamente para poder utilizarse. La forma de obtener y utilizar los índices dinámicos es la misma que para los índices estáticos. Si un índice dinámico ya no es necesario, se puede eliminar. La interfaz BackingMap tiene métodos para crear y eliminar índices dinámicos. A continuación, se proporciona la definición de estos métodos:

```
/**
* Crear un índice dinámico en la BackingMap
*
* @param name El nombre del índice. El nombre no puede ser nulo.
* @param isRangeIndex Indica si se debe crear un MapRangeIndex o un MapIndex.
* Si se establece en true, el índice será del tipo MapRangeIndex.
* @param attributeName El nombre del atributo que se va a indexar.
* attributeName no puede ser nulo.
* @param dynamicIndexCallback El retorno de llamada que se invocará en los sucesos
* de índice dinámico.
* dynamicIndexCallback es opcional y puede ser nulo.
*
* @throws IndexAlreadyDefinedException si ya existe un MapIndexPlugin con el
* nombre especificado.
```

```
* @throws UnsupportedOperationException si la correlación es una correlación
distribuida.
*
*/
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb)
throws IndexAlreadyDefinedException, UnsupportedOperationException;
/**
* Crear un índice dinámico en la BackingMap.
*
* @param index La implementación del índice. El índice no puede ser nulo.
* @param dynamicIndexCallback El retorno de llamada que se invocará en los sucesos
* de índice dinámico.
* dynamicIndexCallback es opcional y puede ser nulo.
*
* @throws IndexAlreadyDefinedException si ya existe un MapIndexPlugin con el
* nombre especificado.
* @throws UnsupportedOperationException si la correlación es una correlación
distribuida.
*/
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback
dynamicIndexCallback)
throws IndexAlreadyDefinedException, UnsupportedOperationException;
/**
* Eliminar un índice dinámico de la BackingMap
*
* @param name El nombre del índice. El nombre no puede ser nulo.
*
* @throws IndexUndefinedException si no existe un MapIndexPlugin con el
* nombre especificado.
* @throws OperationNotSupportedException si la correlación es una correlación
distribuida.
*/
public void removeDynamicIndex(String name) throws IndexUndefinedException;
```

El siguiente fragmento de código ilustra el enfoque de creación, utilización y eliminación de un índice dinámico mediante programación:

```
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 el 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) {
//...
}
//Una vez preparado el índice, las aplicaciones pueden intentar obtener la
//instancia de interfaz de índice de aplicación.
//Las aplicaciones tienen que encontrar una forma de garantizar que el índice
//esté preparado para su uso, si no se utiliza la interfaz DynamicIndexCallback.
//A continuación, se demuestra la forma de esperar a que el índice esté preparado
//El tiempo de espera total debe tener en cuenta el tamaño de la correlación
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("El índice no está preparado. Continúe esperando.");
try {
Thread.sleep(3000);
} catch (Throwable tt) {
//...
}
} catch (Throwable t) {
//excepción inesperada
t.printStackTrace();
}
}
if(!ready){
System.out.println("El índice no está preparado. Debe manejar esta
situación.");
}
//utilizar el índice para realizar consultas
//Consulte en la interfaz MapIndex o MapRangeIndex las operaciones soportadas.
//El atributo de objeto en el que se ha creado el índice es el EmployeeCode.
//Se supone que el atributo EmployeeCode es del tipo Integer, que debe ser
//el tipo de datos del parámetro pasado en las operaciones de índice.
Iterator iter = codeIndex.findLessEqual(new Integer(15));
//eliminar el índice dinámico cuando ya no sea necesario
bm.removeDynamicIndex("CODE");
```

Interfaz DynamicIndexCallback

La interfaz DynamicIndexCallback está diseñada para las aplicaciones que deseen obtener notificaciones en los sucesos de indexación preparado, error o destruir. Es un parámetro opcional para el método createDynamicIndex() de la BackingMap. Con una instancia de DynamicIndexCallback registrada, las aplicaciones pueden ejecutar la lógica empresarial al recibir la notificación de un suceso de indexación. Por ejemplo, el suceso preparado significa que el índice está preparado para su uso. Cuando se recibe una notificación de este suceso, una aplicación puede intentar obtener la instancia de la interfaz de índice de aplicación y utilizarla. A continuación, se muestra la definición de la interfaz DynamicIndexCallback:

```
/**
* Esta es la interfaz de retorno de llamada del proceso de indexación dinámica.
* Si las aplicaciones desean obtener notificaciones en los sucesos preparado, error
* o destruir, pueden implementar esta interfaz de retorno de llamada y registrarse con
* el proceso de indexación dinámica cuando se cree el índice dinámico.
*
*/
public interface DynamicIndexCallback {
/**
* Se invocará este método de retorno de llamada cuando el índice dinámico esté preparado.
*
* @param indexName
* El nombre del índice
*
*/
public void ready(String indexName);
/**
* Se invoca cuando el proceso de indexación dinámica encuentra un error inesperado.
*
* @param indexName El nombre del índice
```

```
* @param t Objeto Throwable que provoca la situación de error en el proceso
* del índice dinámico.
*/
public void error(String indexName, Throwable t);
/**
* Se invocará este método de retorno de llamada cuando se haya eliminado
* el índice dinámico
*
* @param indexName
* El nombre del índice
*/
public void destroy(String indexName);
}
```

El siguiente fragmento de código ilustra el 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);
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);
}
}
```

Consideraciones sobre el rendimiento

Aunque uno de los principales objetivos de la característica de indexación es mejorar el rendimiento general de BackingMap, existen algunos factores a tener en cuenta antes de utilizar esta característica. Si la indexación no se utiliza correctamente, el rendimiento de la aplicación puede disminuir.

- El número de transacciones de escritura concurrentes. El proceso de indexación se puede producir cada vez que una transacción escribe datos en una BackingMap. El rendimiento disminuirá si hay muchas transacciones que escriben datos en la correlación de forma concurrente cuando una aplicación intenta operaciones de consulta de índice.
- El tamaño del conjunto de resultados devuelto por una operación de consulta. Cuando el tamaño del conjunto de resultados aumenta, el rendimiento de las consultas disminuye. Un experimento ha demostrado que el rendimiento disminuye cuando el tamaño del conjunto de resultados es mayor o igual a un 15% de la BackingMap.
- El número de índices creados en la misma BackingMap. Cada índice consume recursos del sistema. Cuando el número de índices creados en la BackingMap aumenta, el rendimiento disminuye.

La conclusión es que la función de indexación puede aumentar significativamente el rendimiento de la BackingMap en determinados entornos. Un caso ideal para la indexación es cuando la BackingMap es mayormente de lectura, el conjunto de resultados de la búsqueda es un porcentaje pequeño de las entradas de BackingMap y sólo se crean unos pocos índices en la BackingMap.

Configuración de ObjectGrid

ObjectGrid se puede configurar para ejecutarse en un entorno distribuido o como una antememoria local que sólo está disponible en una única JVM. Una ObjectGrid local puede configurarse mediante programación o con un archivo XML de ObjectGrid. El archivo XML de ObjectGrid es el lugar donde se definen ObjectGrids, BackingMaps y sus respectivos plug-ins.

Una ObjectGrid local se puede migrar a un entorno distribuido. Para configurar una ObjectGrid distribuida, se debe proporcionar un XML de clúster conjuntamente con el XML de ObjectGrid. El archivo XML de clúster define los servidores en la topología ObjectGrid y cómo se particionan y se duplican los datos de ObjectGrid entre los servidores. En este apartado se describe cómo configurar ObjectGrids locales y distribuidas.

Configuración de ObjectGrid local

Para configurar una ObjectGrid local, consulte “Configuración de ObjectGrid local”.

ObjectGrid distribuida

Para configurar una ObjectGrid distribuida, consulte “Configuración de ObjectGrid distribuida” en la página 274.

Configuración de ObjectGrid local

Una ObjectGrid local puede configurarse mediante programación o con XML. ObjectGridManager es el punto de entrada para ambos medios de configuración.

Existen varios métodos en ObjectGridManager que se pueden utilizar para crear una ObjectGrid local. Para obtener una descripción completa de cada método, consulte “Interfaz ObjectGridManager” en la página 93.

Utilice los temas siguientes para configurar una ObjectGrid local:

- En “Configuración de ObjectGrid básica” se describe cómo crear un archivo XML muy sencillo con una ObjectGrid y una BackingMap definidas.
- En “Configuración de ObjectGrid completa” en la página 262 se define cada elemento y atributo del archivo XML y se describe cómo obtener el mismo resultado que con el archivo XML mediante programación.
- En “Configuración de ObjectGrid en modalidad mixta” en la página 273 se describe cómo utilizar una combinación de métodos de configuración programados XML.

Configuración de ObjectGrid básica

Este tema muestra cómo crear un archivo XML de ObjectGrid muy sencillo, el archivo `bookstore.xml`, con una ObjectGrid y una BackingMap definidas.

Las primeras líneas del archivo son la cabecera necesaria para cada archivo XML de ObjectGrid. El siguiente XML define la ObjectGrid *bookstore* con la BackingMap *book*:

bookstore.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="bookstore">
<backingMap name="book" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

El archivo XML se envía a la interfaz ObjectGridManager para crear una instancia de ObjectGrid basada en el archivo. El siguiente fragmento de código valida el archivo `bookstore.xml` con el esquema XML y crear una ObjectGrid denominada *bookstore*. La instancia de ObjectGrid recién creada no se guarda en la antememoria.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore",
new URL("file:etc/test/bookstore.xml"), true, false);
```

El código siguiente realiza la misma tarea sin XML. Utilice este código para definir mediante programación una BackingMap en una ObjectGrid. Este código crea una BackingMap book en la ObjectGrid bookstoreGrid:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
```

Configuración de ObjectGrid completa

Este tema es una guía completa para configurar una ObjectGrid. Se define cada elemento y atributo del archivo XML. Se proporcionan archivos XML de ejemplo junto con código que realiza la misma tarea mediante programación.

En este tema se hace referencia al siguiente archivo XML: `bookstore.xml`. Después de este ejemplo se describen detalladamente los elementos y atributos de este archivo.

Archivo *bookstore.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="bookstore">
<bean id="ObjectGridEventListener"
className="com.company.organization.MyObjectGridEventListener" />
<backingMap name="books" pluginCollectionRef="collection1" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
classname="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
<property name="maxSize" type="int" value="321" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Elemento ObjectGridConfig

Número de apariciones: una

Elementos hijo: el elemento objectGrids y el elemento backingMapPluginCollections

El elemento objectGridConfig es el elemento de nivel superior del archivo XML. Debe escribirse en el documento XML como se muestra en el ejemplo anterior. Este elemento establece el espacio de nombres del archivo y la ubicación del esquema. El esquema se define en el archivo `objectGrid.xsd`. ObjectGrid busca este archivo en el directorio raíz de los archivos JAR (Java Archive) de ObjectGrid.

Elemento objectGrids

Número de apariciones: una

Elemento hijo: elemento objectGrid

El elemento objectGrids es un contenedor para todos los elementos de objectGrid. En el archivo `sample1.xml`, el elemento objectGrids contiene una objectGrid con el nombre `bookstore`.

Elemento objectGrid

Número de apariciones: de una a varias

Elementos hijo: el elemento bean y el elemento backingMap

Utilice el elemento objectGrid para definir una ObjectGrid en un archivo XML. Cada uno de los atributos del elemento objectGrid corresponde a un método de la interfaz ObjectGrid.

```
<objectGrid
(1) name="objectGridName"
(2) securityEnabled="true|false"
(3) authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS|
AUTHORIZATION_MECHANISM_CUSTOM"
(4) permissionCheckPeriod="permission check period"
(5) txTimeout="seconds"
/>
```

Atributos:

1. Atributo **name** (necesario): Especifica el nombre que se asigna a ObjectGrid. Si falta este atributo, la validación XML fallará.
2. Atributo **securityEnabled** (opcional, el valor por omisión es `false`): Si se establece este atributo en `true` se habilita la seguridad para ObjectGrid. Habilitar la seguridad a nivel de ObjectGrid significa habilitar las autorizaciones de acceso en los datos de la correlación. Por omisión, la seguridad está inhabilitada.
3. Atributo **authorizationMechanism** (opcional, el valor por omisión es `AUTHORIZATION_MECAHNISM_JAAS`): Establece el mecanismo de autorización para esta ObjectGrid. Este atributo puede establecerse en uno de los dos valores: `AUTHORIZATION_MECHANISM_JAAS` o `AUTHORIZATION_MECHANISM_CUSTOM`. Establézcalo en `AUTHORIZATION_MECHANISM_CUSTOM` cuando utilice un plug-in MapAuthorization personalizado. Este valor entra en vigor si el atributo securityEnabled se establece en `true`.

4. **permissionCheckPeriod** (opcional, el valor por omisión es `0`): Especifica un valor entero en segundos que indica la frecuencia con que se ha de comprobar el permiso que se utiliza para permitir un acceso de cliente. Si el valor del atributo es `0`, cada llamada al método get, put, update, remove o evict solicita al mecanismo de autorización, ya sea la autorización JAAS o la autorización personalizada, que compruebe si el sujeto actual tiene permiso. Un valor mayor que `0` indica el número de segundos en que se guarda en antememoria un conjunto de permisos antes de regresar al mecanismo de autorización para que lo renueve. Este valor entra en vigor si el atributo securityEnabled se establece en `true`. Para obtener más información, consulte “Seguridad de ObjectGrid” en la página 140.
5. **txTimeout** (opcional, toma por omisión el valor `0`): la cantidad de tiempo, en segundos, que está permitida la finalización de una transacción. Si una transacción no finaliza en esta cantidad de tiempo, la transacción se marca para la retrotracción y se genera una excepción `TransactionTimeoutException`. Si el valor se establece en `0`, las transacciones nunca exceden el tiempo de espera.

El siguiente archivo XML de ejemplo, el archivo `bookstoreObjectGridAttr.xml`, muestra un modo de configurar los atributos de una ObjectGrid. En este ejemplo, la seguridad está habilitada, el mecanismo de autorización se ha establecido en JAAS y el período de comprobación de permisos se ha establecido en 45 segundos.

Archivo *bookstoreObjectGridAttr.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="bookstore" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS"
permissionCheckPeriod="45">
</objectGrid>
</objectGrids>
</objectGridConfig>
```

El siguiente código muestra el método programado para obtener la misma configuración que la del archivo `bookstoreObjectGridAttr.xml` en el ejemplo anterior.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory
.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore", false);
bookstoreGrid.setSecurityEnabled();
bookstoreGrid.setAuthorizationMechanism(
SecurityConstants.AUTHORIZATION_MECHANISM_JAAS);
bookstoreGrid.setPermissionCheckPeriod(45);
```

Elemento backingMap

Número de apariciones: de cero a varias

Elementos hijo: ninguno

El elemento backingMap se utiliza para definir una BackingMap de una ObjectGrid. Cada uno de los atributos del elemento backingMap se corresponde a un método de la interfaz BackingMap.

```
<backingMap
(1) name="backingMapName"
(2) readOnly="true|false"
(3) pluginCollectionRef="referencia a backingMapPluginCollection"
(4) numberOfBuckets="número de cubetas"
(5) preloadMode="true|false"
(6) lockStrategy="OPTIMISTIC|PESSIMISTIC|NONE"
(7) numberOfLockBuckets="número de cubetas de bloqueo"
(8) lockTimeout="tiempo de espera de bloqueo"
(9) copyMode="COPY_ON_READ_AND_COMMIT|COPY_ON_READ|
COPY_ON_WRITE|NO_COPY"
(10) valueInterfaceClassName="nombre de clase de interfaz de valor"
(11) copyKey="true|false"
(12) nullValuesSupported="true|false"
(13) ttlEvictorType="CREATION_TIME|LAST_ACCESS_TIME|NONE"
(14) timeToLive="tiempo de duración"
/>
```

Atributos:

1. Atributo **name** (necesario): Especifica el nombre que se asigna a BackingMap. Si falta este atributo, la validación XML fallará.
2. Atributo **readOnly** (opcional, el valor por omisión es `false`): Si se establece este atributo en `true` BackingMap será de sólo lectura. Si se establece el atributo en `false` BackingMap será de lectura y grabación. Si se especifica un valor, el resultado será el valor por omisión de lectura y grabación.
3. Atributo **pluginCollectionRef** (opcional): Especifica una referencia a un plug-in backingMapPluginCollection. El valor de este atributo debe coincidir con el atributo de ID de un plug-in backingMapCollection. La validación falla si no existe ningún ID coincidente. Esta referencia está diseñada para que resulte un modo fácil de volver a utilizar los plug-in BackingMap.
4. Atributo **numberOfBuckets** (opcional, el valor por omisión es `503`): El número de cubetas que utilizará BackingMap. BackingMap utiliza una correlación hash para su implementación. Si existen muchas entradas en BackingMap el rendimiento será mejor si el número de cubetas es mayor porque el riesgo de colisiones es menor a medida que aumenta el número de cubetas. Cuanto más elevado sea el número de cubetas, mayor será la concurrencia.
5. Atributo **preloadMode** (opcional, el valor por omisión es `false`): Establece la modalidad de carga previa si se ha establecido un plug-in de cargador para este BackingMap concreto. Si el atributo se establece en `true`, el método Loader.preloadMap(Session, BackingMap) se invoca de modo asíncrono. De lo contrario, bloquea la ejecución del método cuando carga los datos, de modo que la antememoria no está disponible hasta que se completa la carga previa. La carga previa se produce durante la inicialización de ObjectGrid.
6. El atributo **lockStrategy** (opcional, el valor por omisión es `OPTIMISTIC`): Establece LockStrategy que se utiliza para BackingMap. La estrategia de bloqueo determina si el gestor de bloqueo ObjectGrid interno se utiliza cuando una transacción accede a una entrada de correlación. Este atributo se puede establecer en uno de los tres valores: `OPTIMISTIC`, `PESSIMISTIC` o `NONE`.

   `OPTIMISTIC` se utiliza generalmente para una correlación que no tiene un plug-in del cargador, la correlación se suele leer y el bloqueo no lo proporcionan ni el gestor de persistencia utilizando objectGrid como una antememoria ni la aplicación. Para la estrategia de bloqueo optimista, se obtiene un bloqueo exclusivo en una entrada de correlación que se inserta, actualiza o suprime durante el compromiso. El bloqueo asegura que otra aplicación no puede modificar la información de versión mientras la transacción que se está comprometiendo esté realizando una comprobación de versión optimista.

PESSIMISTIC se utiliza generalmente para una correlación que no tiene un plug-in del cargador y el bloqueo no lo proporcionan ni el gestor de persistencia utilizando objectGrid como una antememoria ni el plug-in del cargador ni la aplicación. La estrategia de bloqueo pesimista se utiliza cuando el método optimista falla con demasiada frecuencia debido a que las transacciones de actualización colisionan frecuentemente en la misma entrada de correlación. El método optimista puede fallar cuando la correlación no se lee por lo general, o cuando un gran número de clientes acceden a una correlación pequeña.

NONE indica que no es necesario el uso interno de LockManager debido a que el control de concurrencia se proporciona de forma externa a ObjectGrid, ya sea mediante el gestor de persistencia utilizando ObjectGrid como una antememoria lateral, mediante la aplicación o mediante el plug-in del cargador que utiliza los bloqueos de base de datos para controlar la concurrencia.

7. Atributo **numberOfLockBuckets** (opcional, el valor por omisión es 383): Establece el número de cubetas de bloqueo que utiliza el gestor de bloqueos para este BackingMap. Cuando el atributo 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, cuantas más cubetas de bloqueo haya, mejor será el rendimiento, ya que el riesgo de colisiones disminuye a medida que crece el número de cubetas. Cuanto más elevado sea el número de cubetas de bloqueo, mayor será la concurrencia. Cuando el atributo lockStrategy se establece en NONE, no se utiliza ningún gestor de bloqueos para este BackingMap. En este caso, no es necesario establecer el atributo numberOfLockBuckets.
8. Atributo **lockTimeout** (opcional, el valor por omisión es 15): Establece el tiempo de espera de bloqueo que utiliza el gestor de bloqueos para este BackingMap. Cuando el atributo lockStrategy se establece en OPTIMISTIC o PESSIMISTIC, se crea un gestor de bloqueos para BackingMap. Para impedir que se produzcan puntos muertos, el gestor de bloqueos tiene un valor de tiempo de espera por omisión que espera a que se conceda un bloqueo. Si se supera el límite de este tiempo de espera, se produce una excepción LockTimeoutException. El valor por omisión de 15 segundos es suficiente para la mayor parte de las aplicaciones pero en un sistema con mucha carga, se puede agotar el tiempo de espera cuando no existe un punto muerto. En este caso, se puede utilizar este método para aumentar el valor por omisión de tiempo de espera de bloqueo para impedir de este modo que se produzcan excepciones de tiempo de espera falsas. Cuando la estrategia de bloqueo es NONE, este BackingMap no utiliza ningún gestor de bloqueos. En este caso, no es necesario establecer el atributo lockTimeout.
9. Atributo **copyMode** (opcional, el valor por omisión es COPY_ON_READ_AND_COMMIT): El atributo copyMode determina si una operación get de una entrada de BackingMap devuelve el valor real, una copia del valor o un proxy para el valor. El atributo copyMode se puede establecer para uno de los cuatro valores: COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE o NO_COPY.

   La modalidad COPY_ON_READ_AND_COMMIT garantiza que una aplicación no tenga nunca una referencia al objeto de valor que está en BackingMap y que, a su vez, la aplicación trabaje siempre con una copia de valor que está en BackingMap.

   La modalidad COPY_ON_READ mejora el rendimiento en comparación con la modalidad COPY_ON_READ_AND_COMMIT ya que elimina la copia que se produce cuando se compromete una transacción. Para conservar la integridad de los

datos de BackingMap, la aplicación promete destruir cada una de las referencias que tiene para una entrada después de que se haya comprometido la transacción. Esta modalidad da como resultado un método ObjectMap.get que devuelve una copia del valor, en lugar de una referencia al valor para garantizar que si la aplicación cambia el valor, esto no afecte al valor de BackingMap hasta que se haya comprometido la transacción.

La modalidad `COPY_ON_WRITE` mejora el rendimiento en comparación con la modalidad `COPY_ON_READ_AND_COMMIT` ya que elimina la copia que se produce cuando una transacción llama al método ObjectMap.get por primera vez para una clave determinada. En su lugar, 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 menos que la aplicación llame a un método set sobre la interfaz del valor.

La modalidad `NO_COPY` permite que una aplicación prometa que nunca modificará un objeto de valor que se ha obtenido utilizando un método ObjectMap.get a cambio de mejoras de rendimiento. Si se utiliza esta modalidad, no se copiará el valor.

10. Atributo **valueInterfaceClassName** (opcional): Cuando se establece el atributo copyMode en `COPY_ON_WRITE`, es necesario un atributo valueInterfaceClassName. Se ignora para todas las demás modalidades. El valor COPY_ON_WRITE utiliza un proxy cuando se efectúan llamadas al método ObjectMap.get. El proxy garantiza que no se realice una copia del valor a menos que la aplicación llame a un método set en la clase que se ha especificado como el atributo valueInterfaceClassName.
11. Atributo **copyKey** (opcional, el valor por omisión es `false`): Este atributo determina si se ha de copiar la clave cuando se crea una entrada de correlación. Al copiar el objeto de clave la aplicación también puede utilizar el mismo objeto de clave para cada operación ObjectMap. Si se establece en `true` se copia el objeto de clave cuando se crea una entrada de correlación.
12. Atributo **nullValuesSupported** (opcional, el valor por omisión es `true`): Dar soporte a valores nulos significa que se puede colocar un valor nulo en una correlación. Si se establece en `true`, se soportan los valores nulos en ObjectMap, de lo contrario, no se soportan. Si soportan valores nulos, una operación `get` que devuelva `null` puede significar que el valor es nulo o que la correlación no contiene la clave que se ha pasado.
13. Atributo **ttlEvictorType** (opcional, el valor por omisión es `NONE`): El atributo ttlEvictorType determina de qué modo se calcula la hora de caducidad de una entrada BackingMap. Este atributo se puede establecer en uno de los tres valores: `CREATION_TIME`, `LAST_ACCESS_TIME` o `NONE`.

    `NONE` indica que una entrada no tiene hora de caducidad y puede permanecer activa en BackingMap hasta que la aplicación la suprima explícitamente o invalide la entrada.

    `CREATION_TIME` indica que una hora de caducidad de entrada es la suma de la hora de la creación más el valor del atributo timeToLive.

    `LAST_ACCESS_TIME` indica que una hora de caducidad de entrada es la suma de la hora del último acceso de la entrada más el valor del atributo timeToLive.
14. Atributo **timeToLive** (opcional, el valor por omisión es `0`): El período de tiempo, en segundos, en que está activa cada una de las entradas de correlación. El valor por omisión de `0` significa que la entrada de correlación está activa siempre o hasta que la aplicación suprima explícitamente o invalide la entrada. Si el atributo no es `0`, se utiliza el desalojador TTL para desalojar la entrada de correlación basada en este valor.

El siguiente archivo XML, el archivo `bookstoreBackingMapAttr.xml`, muestra una configuración backingMap de ejemplo. Este ejemplo utiliza todos los atributos opcionales, excepto el atributo pluginCollectionRef. Para obtener un ejemplo que muestre cómo se utiliza pluginCollectionRef, consulte “Elemento backingMapPluginCollection” en la página 272.

Archivo *bookstoreBackingMapAttr.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="bookstore">
<backingMap name="book" readOnly="true" numberOfBuckets="641"
preloadMode="false" lockStrategy="OPTIMISTIC"
numberOfLockBuckets="409" lockTimeout="30" copyMode="COPY_ON_WRITE"
valueInterfaceClassName=
"com.ibm.websphere.samples.objectgrid.CounterValueInterface"
copyKey="true" nullValuesSupported="false"
ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3000" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

El siguiente código de ejemplo muestra el método programado para obtener la misma configuración que la del archivo `bookstoreBackingMapAttr.xml` en el ejemplo siguiente:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
bookMap.setReadOnly(true);
bookMap.setNumberOfBuckets(641);
bookMap.setPreloadMode(false);
bookMap.setLockStrategy(LockStrategy.OPTIMISTIC);
bookMap.setNumberOfLockBuckets(409);
bookMap.setLockTimeout(30);
// cuando se establece la modalidad de copia en COPY_ON_WRITE,
// es necesaria una clase valueInterface
bookMap.setCopyMode(CopyMode.COPY_ON_WRITE,
com.ibm.websphere.samples.objectgrid.CounterValueInterface.class);
bookMap.setCopyKey(true);
bookMap.setNullValuesSupported(false);
bookMap.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);
bookMap.setTimeToLive(3000); // establecer el tiempo de actividad en 50 minutos
```

Elemento bean

Número de apariciones (en el elemento objectGrid): de cero a varias

Número de apariciones (en el elemento backingMapPluginCollection): de cero a varias

Elemento hijo: elemento de propiedad

Utilice el elemento bean para definir los plug-ins. Los plug-ins se pueden asociar a ObjectGrids y BackingMaps.

Los plug-ins de ObjectGrid:

- Plug-in TransactionCallback

- Plug-in ObjectGridEventListener
- Plug-in SubjectSource
- Plug-in MapAuthorization
- Plug-in SubjectValidation

Los plug-ins de BackingMap:
- Plug-in del cargador
- Plug-in ObjectTransformer
- Plug-in OptimisticCallback
- Plug-in Evictor
- Plug-in MapEventListener
- Plug-in MapIndex

Atributos del elemento bean

```
<bean
(1) id="TransactionCallback|ObjectGridEventListener|SubjectSource|
MapAuthorization|SubjectValidation|Loader|ObjectTransformer|
OptimisticCallback|Evictor|MapEventListener|MapIndexPlugin"
(2) className="class name"
/>
```

1. Atributo **id** (necesario): Especifica el tipo del plug-in que se ha de crear. Para un bean que es un elemento hijo del elemento objectGrid, los valores válidos son los plug-ins `TransactionCallback`, `ObjectGridEventListener`, `SubjectSource`, `MapAuthorization` y `SubjectValidation`. Para un bean que es un elemento hijo del elemento backingMapPluginCollection, los valores válidos son los plug-ins `Loader`, `ObjectTransformer`, `OptimisticCallback`, `Evictor` y `MapEventListener`. Cada uno de los valores válidos del atributo id representa una interfaz.
2. Atributo **className** (necesario): Especifica el nombre de la clase de la instancia para crear el plug-in. La clase debe implementar la interfaz del tipo de plug-in.

El archivo `bean.xml` siguiente muestra cómo utilizar el elemento de bean para configurar los plug-ins. En este archivo XML, se añade un plug-in ObjectGridEventListener a la ObjectGrid de bookstore. El atributo className de este bean es la clase com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener. Esta clase implementa la interfaz com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener según sea necesario.

También se define un plug-in de BackingMap en el siguiente ejemplo del archivo `bookstoreBean.xml`. Se añade un plug-in del desalojador al BackingMap de bookstore. Dado que el id del bean es *Evictor*, el atributo className debe especificar una clase que implemente la interfaz com.ibm.websphere.objectgrid.plugins.Evictor. La clase com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor implementa esta interfaz. Backingmap hace referencia a sus plug-ins utilizando el atributo pluginCollectionRef. Consulte “Interfaz BackingMap” en la página 112 para obtener más información acerca de cómo añadir los plug-ins a una BackingMap.

Archivo *bookstoreBean.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="bookstore">
<bean id="ObjectGridEventListener"
className="com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener" />
<backingMap name="book" pluginCollectionRef="bookPlugins" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="bookPlugins">
<bean id="Evictor"
classnName="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

El código siguiente muestra el método programado de obtener la misma configuración que la del archivo `bookstoreBean.xml` anterior.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
TranPropListener tranPropListener = new TranPropListener();
bookstoreGrid.addEventListener(tranPropListener);
BackingMap bookMap = bookstoreGrid.defineMap("book");
Evictor lruEvictor = new
 com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
bookMap.setEvictor(lruEvictor);
```

Elemento property

Número de apariciones: de cero a varias

Elemento hijo: ninguno

El elemento property se utiliza para añadir propiedades a los plug-ins. El nombre de la propiedad corresponde a un método set del atributo className del bean que contiene la propiedad.

Atributos del elemento property

```
<property
(1) name="name"
(2) type="java.lang.String|boolean|java.lang.Boolean|int|
java.lang.Integer|double|java.lang.Double|byte|
java.lang.Byte|short|java.lang.Short|long|
java.lang.Long|float|java.lang.Float|char|
java.lang.Character"
(3) value="value"
(4) description="description"
/>
```

1. Atributo **name** (necesario): Especifica el nombre de la propiedad. El valor que se asigna a este atributo debe corresponder a un método set de la clase que se proporciona como el atributo className del elemento bean. Por ejemplo, si el atributo className del bean se establece en com.ibm.MyPlugin y el nombre de la propiedad suministrado es `size`, entonces la clase com.ibm.MyPlugin debe tener un método setSize.
2. Atributo **type** (necesario): Especifica el tipo de la propiedad. Es el tipo del parámetro que se ha pasado al método set identificado mediante el atributo

name. Los valores válidos son primitivos Java, los correspondientes a `java.lang` y `java.lang.String`. Los atributos de nombre y tipo deben corresponder a una signatura de método del atributo className del bean. Por ejemplo, si el nombre es `size` y el tipo es `int`, entonces debe existir un método `setSize(int)` en la clase que se especifica como el atributo className para el bean.

3. Atributo **value** (necesario): Especifica el valor de la propiedad. Este valor se convierte en el tipo que se especifica mediante el atributo de tipo y se utiliza como un parámetro en la llamada al método set que se identifica mediante los atributos de nombre y tipo. El valor de este atributo no se valida de ningún modo. El implementador del plug-in debe verificar que el valor que se ha pasado es válido. El implementador puede visualizar una excepción `IllegalArgumentException` en el método set si el parámetro no es válido.
4. Atributo **description** (opcional): Utilice este atributo para escribir una descripción de la propiedad.

El siguiente archivo `bookstoreProperty.xml` muestra cómo añadir un elemento de propiedad a un bean. En este ejemplo, se añade a un desalojador una propiedad con el nombre `maxSize` y el tipo `int`. El desalojador com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor tiene una signatura de método que coincide con el método setMaxSize(int). Se pasa un valor entero de `499` al método setMaxSize(int) en la clase com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor.

Archivo bookstoreProperty.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="bookstore">
<backingMap name="book" pluginCollectionRef="bookPlugins" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="bookPlugins">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
<property name="MaxSize" type="int" value="449"
description="Tamaño máximo del desalojador LRU" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

El código siguiente obtiene la misma configuración que el archivo `bookstoreProperty.xml`:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
LRUEvictor lruEvictor =
new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
// Si se utiliza el archivo XML en su lugar, la propiedad que se
// ha añadido puede hacer que se efectúe la llamada siguiente
lruEvictor.setMaxSize(449);
bookMap.setEvictor(lruEvictor);
```

Elemento backingMapPluginCollections

Número de apariciones: de cero a varias

Elemento hijo: elemento backingMapPluginCollection

El elemento backingMapPluginCollections es un contenedor de todos los elementos backingMapPluginCollection. En el archivo `bookstore.xml`, el elemento backingMapPluginCollections contiene un elemento backingMapPluginCollection con el id `collection1`.

Elemento backingMapPluginCollection

Número de apariciones: de cero a varias

Elemento hijo: elemento bean

El elemento backingMapPluginCollection define los plug-ins de BackingMap. Cada elemento backingMapPluginCollection se identifica mediante su atributo de id. Cada elemento backingMap debe hacer referencia a sus plug-ins utilizando el atributopluginCollectionRef en el elemento backingMap. Si existen varios BackingMaps que deban tener configurados los plug-ins de modo similar, cada uno de ellos puede hacer referencia al mismo elemento backingMapPluginCollection.

Atributos del elemento backingMapPluginCollection

```
<backingMapPluginCollection
(1) id="id"
/>
```

1. Atributo **id** (necesario): El identificador de backingMapPluginCollection. Cada id debe ser exclusivo. El id al que hace referencia el atributo pluginCollectionRef del elemento backingMap. Si el valor de un atributo pluginCollectionRef no coincide con el id de un elemento backingMapPluginCollection, la validación XML fallará. Cualquier número de elementos backingMap puede hacer referencia a un solo elemento backingMapPluginCollection.

El siguiente archivo `bookstoreCollection.xml` muestra cómo utilizar el elemento backingMapPluginCollection. En este archivo, se definen tres elementos backingMap. Las BackingMap `book` y `customer` utilizan backingMapPluginCollection `collection1`. Cada una de estas dos BackingMaps tienen su propio desalojador LRUEvictor. BackingMap `employee` hace referencia a backingMapPluginCollection `collection2`. Este BackingMap tiene establecido un desalojador LFUEvictor como un plug-in de desalojador y la clase EmployeeOptimisticCallbackImpl establecida como un plug-in OptimisticCallback.

Archivo bookstoreCollection.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="bookstore">
<backingMap name="book" pluginCollectionRef="collection1" />
<backingMap name="customer" pluginCollectionRef="collection1" />
<backingMap name="employee" pluginCollectionRef="collection2" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
```

```
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
</backingMapPluginCollection>
<backingMapPluginCollection id="collection2">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor" />
<bean id="OptimisticCallback"
className="com.ibm.websphere.samples.objectgrid.
EmployeeOptimisticCallBackImpl" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

El código siguiente muestra como se obtiene mediante programación la misma configuración que la del archivo `bookstoreCollection.xml`.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
LRUEvictor bookEvictor = new LRUEvictor();
bookMap.setEvictor(bookEvictor);
BackingMap customerMap = bookstoreGrid.defineMap("customer");
LRUEvictor customerEvictor = new LRUEvictor();
customerMap.setEvictor(customerEvictor);
BackingMap employeeMap = bookstoreGrid.defineMap("employee");
LFUEvictor employeeEvictor = new LFUEvictor();
employeeMap.setEvictor(employeeEvictor);
OptimisticCallback employeeOptCallback =
 new EmployeeOptimisticCallbackImpl();
employeeMap.setOptimisticCallback(employeeOptCallback);
```

Configuración de ObjectGrid en modalidad mixta

Se puede configurar ObjectGrid utilizando una combinación de una configuración XML y una configuración programada.

Para realizar una configuración mixta, cree en primer lugar un archivo XML que pasar a la interfaz ObjectGridManager. Después de crear una ObjectGrid basado en el archivo XML, se puede manipular la ObjectGrid mediante programación, siempre que no se haya llamado al método ObjectGrid.initialize(). El método ObjectGrid.getSession() llama implícitamente al método ObjectGrid.initialize() si no lo ha llamado la aplicación.

Ejemplo

La siguiente es una demostración de cómo obtener una configuración en modalidad mixta. Se utiliza el archivo `mixedBookstore.xml` siguiente.

Archivo mixedBookstore.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="bookstore">
<backingMap name="book" readOnly="true" numberOfBuckets="641"
pluginCollectionRef="bookPlugins" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="bookPlugins">
<bean id="Evictor"
```

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

El siguiente fragmento de código que muestra el XML que se ha pasado a ObjectGridManager y la ObjectGrid que se acaba de crear, se manipula adicionalmente.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore",
new URL("file:etc/test/document/mixedBookstore.xml"), true, false);
// llegado este punto, tenemos la ObjectGrid que se ha definido en el XML
// ahora modifique la BackingMap que se ha creado y configurado
BackingMap bookMap = bookstoreGrid.getMap("book");
// el XML ha establecido readOnly en true
// aquí el atributo readOnly se modifica a false
bookMap.setReadOnly(false);
// el XML no ha establecido nullValuesSupported, por lo tanto,
// toma el valor por omisión true. Aquí el valor
// se establece en false
bookMap.setNullValuesSupported(false);
// obtener el desalojador que se ha establecido en el XML
// y establecer su maxSize
LFUEvictor lfuEvictor = (LFUEvictor) bookMap.getEvictor();
lfuEvictor.setMaxSize(443);
bookstoreGrid.initialize();
// no se permite una configuración adicional
// de esta ObjectGrid después de la llamada de inicialización
```

Configuración de ObjectGrid distribuida

Para crear una ObjectGrid distribuida, se debe crear un archivo XML de clúster y emparejar con un archivo XML de ObjectGrid.

Con los archivos XML de clúster y XML de ObjectGrid, puede iniciar un servidor ObjectGrid.

Antes de crear un archivo XML de clúster, cree un archivo XML de ObjectGrid como lo haría para una ObjectGrid local. Para obtener más información sobre cómo construir un archivo XML de ObjectGrid, consulte “Configuración de ObjectGrid local” en la página 261. El siguiente archivo `university.xml` se utiliza como el archivo XML de ObjectGrid para los ejemplos de “Configuración de clúster” en la página 275.

Archivo *university.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="academics">
<backingMap name="faculty" />
<backingMap name="student" />
<backingMap name="course" />
</objectGrid>
<objectGrid name="athletics">
<backingMap name="athlete" />
<backingMap name="equipment" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

A continuación se muestra el archivo XML de clúster `universityCluster.xml` que se puede utilizar con el archivo `university.xml` para iniciar un servidor ObjectGrid. El archivo `universityCluster.xml` es un archivo XML de clúster muy básico donde se han eliminado todos los atributos XML opcionales.

Archivo *universityCluster.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
</cluster>
<objectgridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

En el apartado “Configuración de clúster” se muestra un ejemplo de uso de muchos de los atributos y elementos XML opcionales.

Configuración de clúster

En este apartado se describe cada uno de los elementos y atributos del XML de clúster. También se proporcionan ejemplos que muestran cómo utilizar el XML de clúster con el XML de ObjectGrid para obtener una configuración. El archivo `university.xml` se utiliza como XML de ObjectGrid para estos ejemplos.

Elemento clusterConfig

Número de apariciones: una

Elementos hijo: los elementos cluster, objectgridBinding, partitionSet y replicationGroup

El elemento clusterConfig es el elemento de nivel superior del archivo XML de clúster. Debe estar en la parte superior del archivo, tal como se demuestra en el archivo `universityCluster.xml`. Este elemento establece el espacio de nombres del archivo y la ubicación del esquema. El esquema se define en el archivo `objectGridCluster.xsd`.

Elemento cluster

Número de apariciones: una

Elementos hijo: los elementos serverDefinition, authenticator y adminAuthorization

El elemento cluster se utiliza para definir un clúster ObjectGrid. Cada uno de los servidores del clúster se define en el elemento cluster. El elemento cluster también se utiliza para definir atributos de seguridad y relacionados con la red.

```
<cluster
(1) name="clusterName"
(2) securityEnabled="true|false"
(3) statisticsEnabled="true|false"
(4) statisticsSpec="statisticsSpecification"
(5) singleSignOnEnabled="true|false"
(6) loginSessionExpirationTime="seconds"
(7) adminAuthorizationEnabled="true|false"
(8) adminAuthorizationMechanism=""
(9) clientMaxRetries="numberOfRetries"
(10) clientMaxForwards="numberOfForwards"
(11) clientStartupRetries="numberOfRetries"
(12) clientRetryInterval="seconds"
(13) tcpConnectionTimeout="seconds"
(14) tcpMinConnections="numberOfConnections"
(15) tcpMaxConnections="numberOfConnections"
(16) tcpInactivityTimeout="seconds"
(17) tcpMaxWaitTime="seconds"
(18) peerHeartbeatInterval="seconds"
(19) peerTransportBufferSize="sizeInMBs"
(20) threadPoolMinSize="minThreads"
(21) threadPoolMaxSize="maxThreads"
(22) threadPoolInactivityTimeout="seconds"
(23) managementTimeout="seconds"
(24) threadsPerClientConnect="numberOfThreads"
/>
```

Atributos:

1. Atributo **name** (necesario): es el nombre que se asigna al clúster. Si falta este atributo, la validación XML fallará.
2. Atributo **securityEnabled** (opcional, el valor por omisión es **false**): habilita la seguridad del clúster cuando se establece en true. Si se establece en false, la seguridad a nivel de clúster se inhabilita. Para obtener más información, consulte el “Seguridad de ObjectGrid” en la página 140.
3. Atributo **statisticsEnabled** (opcional, el valor por omisión es **false**): habilita las estadísticas del clúster cuando se establece en true. Cuando las estadísticas están habilitadas, el atributo statisticsSpec se utiliza para establecer la especificación de estadísticas.
4. Atributo **statisticsSpec** (opcional): especifica la serie que se utiliza para establecer la especificación de estadísticas. Esta serie determina qué estadísticas se recopilan.
5. Atributo **singleSignOnEnabled** (opcional, el valor por omisión es false): si se establece el atributo singleSignOnEnabled en true, el cliente podrá conectarse a cualquier servidor una vez autenticado con uno de los servidores. Cuando este atributo se establece en false, el cliente debe autenticarse con cada servidor para poder conectarse.
6. Atributo **loginSessionExpirationTime** (opcional): la cantidad de tiempo en segundos que debe transcurrir antes de que caduque la sesión de inicio de sesión. Si la sesión de inicio de sesión caduca, el cliente debe reautenticarse.

7. Atributo **adminAuthorizationEnabled** (opcional, el valor por omisión es false): este valor se utiliza para habilitar la autorización administrativa. Si el valor es true, todas las tareas administrativas necesitan autorización. El mecanismo de autorización que se utiliza viene especificado por el valor del atributo adminAuthorizationMechanism.
8. Atributo **adminAuthorizationMechanism** (opcional, el valor por omisión es AUTHORIZATION_MECHANISM_JAAS): este atributo indica qué mecanismo de autorización se utiliza. ObjectGrid da soporte a dos mecanismos de autorización: JAAS (Java Authentication and Authorization Service) y personalizada. El mecanismo de autorización JAAS utiliza el enfoque basado en la política JAAS estándar. Para especificar JAAS como mecanismo de autorización, establezca el valor en AUTHORIZATION_MECHANISM_JAAS. El mecanismo de autorización personalizada utiliza una implementación de AdminAuthorization de plug-in de usuario. Para especificar un mecanismo de autorización personalizada, establezca el valor en AUTHORIZATION_MECHANISM_CUSTOM. Para obtener más información sobre cómo se utilizan estos dos mecanismos, consulte “Seguridad de ObjectGrid” en la página 140.
9. Atributo **clientMaxRetries** (opcional, el valor por omisión es 4): el número máximo de veces que se puede reintentar automáticamente una petición de un servidor cuando el servicio no está disponible.
10. Atributo **clientMaxForwards** (opcional, el valor por omisión es 5): el número máximo de veces que se reenvía una petición anómala a otro servidor.
11. Atributo **clientStartupRetries** (opcional, el valor por omisión es 8): el número máximo de veces que se reintenta automáticamente una petición mientras se espera que finalice el arranque del servidor. Como High Availability Manager requiere una gran cantidad de tiempo para iniciarse, establezca este número en un valor lo suficientemente alto. Si el número no es lo bastante alto, las peticiones de cliente que se sometan antes de que se haya iniciado completamente el servidor fallarán.
12. Atributo **clientRetryInterval** (opcional, el valor por omisión es 10): el intervalo de tiempo, en segundos, entre los reintentos del cliente. Se utiliza para los atributos clientMaxRetries y clientStartupRetries.
13. Atributo **tcpConnectionTimeout** (opcional, el valor por omisión es 180): el atributo tcpConnectionTimeout es el tiempo de espera de conexión de socket. El valor se especifica en segundos.
14. Atributo **tcpMinConnections** (opcional, el valor por omisión es 2): el número mínimo de conexiones de la agrupación de conexiones.
15. Atributo **tcpMaxConnections** (opcional, el valor por omisión es 20): el número máximo de conexiones de la agrupación de conexiones.
16. Atributo **tcpInactivityTimeout** (opcional, el valor por omisión es infinito): el número de segundos de inactividad que debe transcurrir en una conexión antes de que la conexión se elimine de la agrupación de conexiones.
17. Atributo **tcpMaxWaitTime** (opcional, el valor por omisión es 120): el número máximo de segundos que un sistema espera una conexión disponible cuando todas las conexiones están en uso y el número de conexiones ha alcanzado el valor del atributo tcpMaxConnections.
18. Atributo **peerHeartbeatInterval** (opcional, el valor por omisión es 120): el atributo peerHeartbeatInterval es el intervalo de pulsaciones que utiliza High Availability Manager. El valor se especifica en segundos.

19. Atributo **peerTransportBufferSize** (opcional, el valor por omisión es 10): el atributo peerTransportBufferSize representa el tamaño de almacenamiento intermedio de mensajes de transporte que utiliza High Availability Manager. Este valor se especifica en megabytes.
20. Atributo **threadPoolMinSize** (opcional, el valor por omisión es 6): especifica el tamaño mínimo de la agrupación de hebras de High Availability Manager.
21. Atributo **threadPoolMaxSize** (opcional, el valor por omisión es 20): especifica el tamaño máximo de la agrupación de hebras de High Availability Manager.
22. Atributo **threadPoolInactivityTimeout** (opcional, el valor por omisión es 6000): especifica el tiempo de espera de inactividad de hebras de la agrupación de hebras de High Availability Manager. El valor de tiempo de espera se especifica en segundos.
23. Atributo **managementTimeout** (opcional, el valor por omisión es 30): varias de las funciones de MBean de ObjectGrid envían mensajes a los servidores del clúster para recopilar información de los servidores o realizar operaciones en ellos. El valor managementTimeout indica cuánto tiempo intenta el cliente recibir un mensaje del servidor. Si existen problemas de comunicación entre el cliente y el servidor, o si el servidor está ocupado, el cliente reintenta la operación la cantidad de tiempo especificada por el valor managementTimeout. El valor de managementTimeout se especifica en segundos.
24. Atributo **threadsPerClientConnect** (opcional, el valor por omisión es 5): el número de hebras que se crean por ClientClusterContext. Cada llamada al método de conexión en la interfaz ObjectGridManager genera un nuevo ClientClusterContext.

El siguiente archivo `universityClusterAttr.xml` es una configuración de ejemplo que utiliza los distintos atributos opcionales en el elemento cluster. En este ejemplo, la seguridad está inhabilitada. También se modifican los distintos atributos de cliente, tcp, igual y hebra. `universityClusterAttr.xml` no es una recomendación de qué valores se asignan a los atributos. Es un ejemplo de cómo establecer los valores de los atributos.

Archivo *universityClusterAttr.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster" securityEnabled="false" statisticsEnabled="true"
statisticsSpec="map.all=enabled" singleSignOnEnabled="false"
loginSessionExpirationTime="1800" adminAuthorizationEnabled="false"
adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_JAAS" clientMaxRetries="2"
clientMaxForwards="2" clientStartupRetries="2" clientRetryInterval="5"
tcpConnectionTimeout="160" tcpMinConnections="2" tcpMaxConnections="15"
tcpInactivityTimeout="3600" tcpMaxWaitTime="160" peerHeartbeatInterval="130"
peerTransportBufferSize="15" threadPoolMinSize="8" threadPoolMaxSize="25"
threadPoolInactivityTimeout="6050" managementTimeout="60">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectGridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
```

```
</objectGridBinding>
<objectGridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectGridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

Elemento serverDefinition

Número de apariciones: de una a varias

Elementos hijo: ninguno

El elemento serverDefinition se utiliza para definir un servidor ObjectGrid. Cada servidor se ejecuta en su propia JVM (Java Virtual Machine) y requiere dos puertos.

Atributos:

```
<serverDefinition
(1) name="serverName"
(2) host="hostname"
(3) clientAccessPort="portNumber"
(4) peerAccessPort="portNumber"
(5) traceSpec="traceSpecification"
(6) systemStreamToFileEnabled="true|false"
(7) workingDirectory="logsDirectory"
/>
```

1. Atributo **name** (necesario): es el nombre que se asigna al servidor. Si falta este atributo, la validación XML fallará.
2. Atributo **host** (necesario): el nombre de sistema principal de la máquina donde se ejecuta la JVM del servidor. Cada máquina puede alojar varios servidores ObjectGrid. Si falta este atributo, la validación XML fallará.
3. Atributo **clientAccessPort** (necesario): el puerto en el servidor que se utiliza para las conexiones de cliente. Si falta este atributo, la validación XML fallará.
4. Atributo **peerAccessPort** (necesario): el puerto en el servidor que se utiliza para la comunicación entre servidores ObjectGrid. Si falta este atributo, la validación XML fallará.
5. Atributo **traceSpec** (opcional, el valor por omisión es *=all=disabled): si se establece el atributo traceSpec, se habilita el rastreo del servidor utilizando la serie especificada.
6. Atributo **systemStreamToFileEnabled** (opcional, el valor por omisión es **true**): si este atributo se establece en true, System.out, System.err y las corrientes de salida de rastreo se dirigen a un archivo. Cuando este atributo se establece en false, System.out se dirige a la corriente stdout y System.err se dirige a la corriente stderr. Si el rastreo está habilitado, la salida del rastreo se dirige a un archivo, independientemente del valor del atributo systemStreamToFileEnabled.
7. Atributo **workingDirectory** (opcional): el atributo workingDirectory especifica dónde se escriben los atributos de anotaciones cronológicas. Si no se especifica un atributo workingDirectory, las anotaciones cronológicas se escriben en el directorio actual.

El archivo `universityClusterServerAttr.xml` demuestra el uso de los atributos serverDefinition. En este archivo XML, el servidor server1 está configurado para ejecutarse en el sistema principal lion.ibm.com. El puerto 12501 se utiliza para el acceso del cliente al servidor y el puerto 12502 se utiliza para las comunicaciones de servidor a servidor. Como el atributo systemStreamToFileEnabled se establece en true, System.out, System.err y el rastreo se dirigen a un archivo en el directorio especificado con el atributo workingDirectory. En este ejemplo, los archivos están en el directorio `/objectgrid/`. Como el atributo traceSpec se establece en ″ObjectGrid=all=enabled″, todo el rastreo relacionado con ObjectGrid se captura y se dirige a un archivo.

Archivo *universityClusterServerAttr.xml*

```
<?xml version="1.0" encoding="UTF-8" ?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" systemStreamToFileEnabled="true"
workingDirectory="/objectgrid/" traceSpec="ObjectGrid=all=enabled" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectGridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectGridBinding>
<objectGridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectGridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

Elemento objectgridBinding

Número de apariciones: de una a varias

Elemento hijo: elemento mapSet

El elemento objectgridBinding se utiliza para enlazar los elementos objectGrid del XML de ObjectGrid con la topología definida en el XML de clúster. El valor asignado al atributo ref debe coincidir con el atributo name de uno de los elementos objectGrid en el XML de ObjectGrid. Sólo se puede hacer referencia a un elemento objectGrid del XML de ObjectGrid en un objectgridBinding del XML de clúster.

Atributos

```
<objectgridBinding
(1) ref="objectGridReference"
(2) minThreadPoolSize="minSize"
(3) maxThreadPoolSize="maxSize"
/>
```

1. Atributo **ref** (necesario): el atributo ref se utiliza para hacer referencia a un elemento objectGrid definido en el archivo XML de ObjectGrid. Cada elemento objectgridBinding debe hacer referencia a uno de los elementos objectGrid del XML de ObjectGrid. El atributo ref debe coincidir con el atributo name de uno de los elementos objectGrid en el XML de ObjectGrid.
2. Atributo **minThreadPoolSize** (opcional, el valor por omisión es 3): el atributo minThreadPoolSize es el número mínimo de hebras que están permitidas en la agrupación de hebras para cada miembro del grupo de duplicación. El número de hebras está controlado por un gestor de agrupaciones de hebras, pero no puede ser menor que el valor de minThreadPoolSize. En general, cuanto más hebras, el cliente puede recibir una respuesta más rápida del servidor. Un número mayor de hebras genera también una mayor contención. No obstante, las máquinas más rápidas pueden manejar hebras concurrentes adicionales de forma eficaz.
3. Atributo **maxThreadPoolSize** (opcional, el valor por omisión es 10): el atributo maxThreadPoolSize es el número máximo de hebras que están permitidas en la agrupación de hebras para cada miembro del grupo de duplicación. El número de hebras está controlado por un gestor de agrupaciones de hebras, pero no puede ser mayor que el valor de maxThreadPoolSize. En general, cuanto más hebras, el cliente puede recibir una respuesta más rápida del servidor. Un número mayor de hebras genera también una mayor contención. No obstante, las máquinas más rápidas pueden manejar hebras concurrentes adicionales de forma eficaz.

El archivo `universityClusterOGBinding.xml` demuestra cómo utilizar el elemento objectgridBinding y sus atributos. En este ejemplo, se define un elemento objectgridBinding. El elemento objectgridBinding hace referencia a la ObjectGrid ″academics″ definida en el archivo `university.xml`. Observe que aunque la ObjectGrid ″athletics″ está en el archivo `university.xml`, ningún elemento objectgridBinding hace referencia a la ObjectGrid athletics en el archivo `universityClusterOGBinding.xml`. La ObjectGrid ″athletics″ no está agrupado en clúster ya que no está incluido en el archivo `universityClusterOGBinding.xml`. En este caso, sólo se crea y agrupa en clúster la ObjectGrid ″academics″, ya que está en el archivo `university.xml` y se hace referencia a ella en el archivo `universityClusterOGBinding.xml`.

Los atributos minThreadPoolSize y maxThreadPoolSize también se establecen en este ejemplo. El valor de minThreadPoolSize se establece en 2, y el valor de maxThreadPoolSize se establece en 11. El gestor de agrupaciones de hebras de cada miembro del grupo de duplicación mantiene el número de hebras dentro de estos límites para todas las correlaciones de esta ObjectGrid.

Archivo *universityClusterOGBinding.xml*

```
<?xml version="1.0" encoding="UTF-8" ?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
</cluster>
```

```
<objectgridBinding ref="academics" minThreadPoolSize="2" maxThreadPoolSize="11">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

Elemento mapSet

Número de apariciones: de una a varias

Elemento hijo: elemento de correlación

El elemento mapSet se utiliza para agrupar correlaciones. Las correlaciones de un mapSet se particionan de forma parecida. En una ObjectGrid distribuida, cada correlación debe pertenecer a uno y sólo un mapSet.

Atributos

```
<mapSet
(1) name="mapSetName"
(2) partitionSetRef="partitionSetReference"
(3) synchronousReplication="true|false"
(4) replicaReadEnabled="true|false"
(5) replicaDeliveryRate="deliveryRate"
(6) compression="true|false"
/>
```

1. Atributo **name** (necesario): especifica el nombre que se asigna a mapSet.
2. Atributo **partitionSetRef** (necesario): cada mapSet se debe asociar con un partitionSet mediante el atributo partitionSetRef. El valor de partitionSetRef debe coincidir con el valor del atributo name de uno de los elementos partitionSet. Utilizando el atributo partitionSetRef y su partitionSet correspondiente, se particionan las correlaciones del mapSet.
3. Atributo **synchronousReplication** (opcional, el valor por omisión es **false**): cuando este atributo se establece en true, la duplicación entre los miembros del grupo de duplicación se produce de forma síncrona. Cuando se establece en false, la duplicación se produce de forma asíncrona.
4. Atributo **replicaReadEnabled** (opcional, el valor por omisión es **false**): si el valor de synchronousReplication se establece en false y el valor de replicaReadEnabled es true, los clientes pueden leer datos de las duplicaciones. Se ha hecho el máximo esfuerzo en distribuir las peticiones de lectura entre el miembro primario y sus duplicaciones. Si el atributo synchronousReplication se establece en true, el atributo replicaReadEnabled se ignora.
5. **replicaDeliveryRate** (opcional, el valor por omisión es **1000**): el valor de replicaDeliveryRate representa el número máximo de registros por LogSequence que se entregan en cada duplicación.
6. Atributo **compressReplicationEnabled** (opcional, el valor por omisión es **true**): cuando compressReplicationEnabled se establece en true, los mensajes de duplicación se comprimen.

El archivo `universityClusterMapSet.xml` es un poco más complejo que los ejemplos de archivo XML anteriores. En este archivo, la ObjectGrid academics está dividido en dos conjuntos de correlaciones. El conjunto de correlaciones academicsMapSet1 contiene las correlaciones faculty y course. Estas dos correlaciones se particionan de acuerdo con el partitionSet partitionSet1. Los valores de duplicación de estas correlaciones también son los mismos porque están en el mismo mapSet.

El objectgridBinding academics también contiene el mapSet academicsMapSet2. Este mapSet contiene sólo la correlación student. La correlación student está particionada de forma diferente a las correlaciones del mapSet academicsMapSet1. La correlación student está particionada de acuerdo con el partitionSet studentPSet. Como academicsMapSet2 no ha establecido explícitamente valores para los atributos relacionados con la duplicación, incluidos los atributos synchronousReplication, replicaReadEnabled, replicaDeliveryRate y compressReplicationEnabled, se le asignan los valores por omisión. Esta es otra forma en la que varía el comportamiento de los dos mapSets dentro del objectgridBinding academics.

El objectgridBinding athletics contiene el mapSet athleticsMapSet. Como el mapSet academicsMapSet1 en el objectgridBinding academics, se particiona de acuerdo con el partitionSet partitionSet1. Los atributos relacionados con la duplicación de este mapSet se establecen en los valores por omisión, ya que no se indican explícitamente.

Archivo *universityClusterMapSet.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectgridBinding ref="academics"
<mapSet name="academicsMapSet1" partitionSetRef="partitionSet1"
synchronousReplication="true" replicaReadEnabled="true"
replicaDeliveryRate="1500" compressReplicationEnabled="true">
<map ref="faculty" />
<map ref="course" />
</mapSet>
<mapSet name="academicsMapSet2" partitionSetRef="studentPSet">
<mapRef="student" />
</mapSet>
</objectGridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetREf="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<partitionSet name="studentPSet">
<partition name="studentPartition1" replicationGroupRef="replicationGroup1" />
<partition name="studentPartition2" replicationGroupRef="replicationGroup2" />
</partitionSet>
<replicationGroup name="replicationGroup1">
```

```
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
<replicationGroup name="replicationGroup2" minReplicas="1" maxReplicas="2">
<replicationGroupMember serverRef="server1" priority="2" />
<replicationGroupMember serverRef="server2" priority="1" />
</replicationGroup>
</clusterConfig>
```

Elemento de correlación

Número de apariciones: de una a varias

Elementos hijo: ninguno

Cada correlación en un mapSet hace referencia a uno de los elementos backingMap definidos en el archivo XML de ObjectGrid. Cuando se define una ObjectGrid distribuida, cada backingMap dentro del elemento objectGrid del XML de ObjectGrid debe estar referenciada por una correlación en el XML de clúster. En una ObjectGrid distribuida, cada correlación debe pertenecer a uno y sólo un mapSet.

Atributos

```
<map
(1) ref="backingMapReference"
/>
```

1. Atributo **ref** (necesario): una referencia a una backingMap en el XML de ObjectGrid. Cada correlación de un mapSet debe hacer referencia a una backingMap del archivo XML de ObjectGrid. El valor asignado a ref debe coincidir con el atributo name de uno de los elementos backingMap en el XML de ObjectGrid.

Consulte el archivo `universityClusterMapSet.xml` para ver un ejemplo de uso del elemento map. Cada backingMap de la objectGrid academics en `university.xml` está referenciada por una correlación en uno y sólo un mapSet en `universityClusterMapSet.xml`. Lo mismo se cumple para la objectGrid athletics. Se genera una excepción `ObjectGridException` si un objectgridBinding hace referencia a una objectGrid del XML de ObjectGrid, pero no incluye todas sus correlaciones en un mapSet.

Elemento partitionSet

Número de apariciones: de una a varias

Elementos hijos: partition

El elemento partitionSet se utiliza para definir particiones de un mapSet. Cada partición del mapSet se particiona entre las particiones de un partitionSet. Un mapSet se asocia con un partitionSet con el atributo partitionSetRef en el elemento mapSet. Cuando sólo está definida una partición en un partitionSet, los datos contenidos en las correlaciones de un mapSet asociado no se particionan.

Atributos

```
<partition-set
(1) name="partitionSetName"
/>
```

1. Atributo **name** (necesario): este atributo se utiliza para asignar un nombre a un partitionSet. El atributo partitionSetRef del mapSet hace referencia al nombre del partitionSet.

Consulte el archivo `universityClusterMapSet.xml` para ver un ejemplo de uso de partitionSet. En `universityClusterMapSet.xml`, hay definidos dos partitionSets: partitionSet1 y studentPSet. El partitionSet partitionSet1 sólo tiene una partición definida. Como sólo hay una partición definida, los mapSet que utilicen el partitionSet partitionSet1 tendrán sus datos particionados. En el archivo `universityClusterMapSet.xml` existen dos mapSets que están particionados de acuerdo con el partitionSet partitionSet1. Mediante la partitionSetRef del elemento mapSet, los mapSets academicsMapSet1 y athleticsMapSet se enlazan con el partitionSet partitionSet1.

El partitionSet studentPSet contiene dos particiones. Los mapSet que utilicen este partitionSet tendrán sus datos de correlación divididos en dos particiones. En el archivo `universityClusterMapSet.xml`, el mapSet academicsMapSet2 utiliza el partitionSet studentPSet.

Elemento partition

Número de apariciones: de una a varias

Elementos hijo: ninguno

El elemento partition se utiliza para definir particiones en un partitionSet. Las particiones se utilizan para dividir los datos en las correlaciones de un mapSet.

Atributos

```
<partition
(1) name="partitionName"
(2) replicationGroupRef="replicationGroupReference"
/>
```

1. Atributo **name** (necesario): el atributo name se utiliza para asignar un nombre a una partición. El nombre de una partición debe ser exclusivo en su partitionSet.
2. Atributo **replicationGroupRef** (necesario): el atributo replicationGroupRef se utiliza para asociar un replicationGroup con una partición. replicationGroupRef debe coincidir con el atributo name de uno de los elementos replicationGroup.

Consulte el archivo `universityClusterMapSet.xml` para ver un ejemplo de uso del elemento partition. En el archivo `universityClusterMapSet.xml`, hay definidas varias particiones. El partitionSet partitionSet1 tiene una partición denominada partition1. El partitionSet studentPSet contiene dos particiones: studentPartition1 y studentPartition2.

Cada partición está asociada con un replicationGroup mediante el atributo replicationGroupRef. En el archivo `universityClusterMapSet.xml`, la partición studentPartition1 está duplicada a través del replicationGroup replicationGroup1. La partición studentPartition2 está duplicada a través del replicationGroup replicationGroup2.

Elemento replicationGroup

Número de apariciones: de una a varias

Elemento hijo: elemento replicationGroupMember

Un replicationGroup se utiliza para definir cómo se duplican una correlación o sus particiones. Las particiones de una correlación se duplican entre los miembros del grupo de duplicación dentro de un replicationGroup.

Atributos

```
<replicationGroup
(1) name="replicationGroupName"
(2) minReplicas="minNumberOfReplicas"
(3) maxReplicas="maxNumberOfReplicas"
```

1. Atributo **name** (necesario): el atributo name se utiliza para asignar un nombre a un replicationGroup.
2. Atributo **minReplicas** (opcional, el valor por omisión es **0** cuando sólo hay un replicationGroupMember en el replicationGroup; el valor por omisión es **1** cuando hay más de un replicationGroupMember en el replicationGroup): el atributo minReplicas se utiliza para indicar cuántos replicationGroupMembers debe haber disponibles para que el acceso de escritura pueda correlacionar datos en este replicationGroup. Si el número de duplicaciones disponibles es menor que el número de minReplicas especificado, sólo se permite el acceso de lectura a las correlaciones. Si minReplicas se establece en 0, el acceso de escritura continúa estando permitido en el miembro primario, aunque las duplicaciones no estén disponibles.

   Para activar la duplicación, al menos dos miembros de grupo de duplicación deben estar disponibles, y el atributo minReplicas debe ser al menos 1. Es importante conocer cómo se comporta un replicationGroup durante la fase de ″activación″ de un clúster ObjectGrid. Si desea correlacionar datos para que estén disponibles después de iniciar sólo un servidor, defina un replicationGroup con sólo un replicationGroupMember. En un replicationGroup con sólo un replicationGroupMember, los datos no se duplican.

   A continuación, se proporcionan algunas normas para establecer el valor de minReplicas.

   ```
   minReplicas >= 0
   minReplicas <= maxReplicas
   minReplicas <= número de miembros en el replicationGroup -1
   ```

3. Atributo **maxReplicas** (opcional, el valor por omisión es **0** cuando sólo hay un replicationGroupMember en el replicationGroup; el valor por omisión es **1** cuando hay más de un replicationGroupMember en el replicationGroup): el atributo maxReplicas representa el número máximo de duplicaciones que se activan en el replicationGroup. En un replicationGroup, la duplicación se produce entre el número de maxReplicas especificadas si hay el mismo número miembros disponibles. Si maxReplicas es menor que el número de miembros de grupo de duplicación en el grupo, los miembros adicionales están en espera; esto es, están en estado suspendido hasta que una de las duplicaciones deje de estar disponible.

   A continuación, se proporcionan algunas normas para establecer el valor de maxReplicas.

   ```
   maxReplicas >= 0
   maxReplicas >= minReplicas
   ```

Consulte el archivo `universityClusterMapSet.xml` para ver un ejemplo de uso del elemento partition. En el archivo `universityClusterMapSet.xml` hay dos grupos de duplicación definidos: replicationGroup1 y replicationGroup2. El grupo de duplicación replicationGroup1 contiene sólo un replicationGroupMember. Las particiones enlazadas con este grupo de duplicación no se duplican ya que se necesita más de un replicationGroupMember para la duplicación.

El grupo de duplicación replicationGroup2 contiene dos replicationGroupMembers. La partición studentPartition2 del conjunto de particiones studentPSet utiliza este grupo de duplicación. Por lo tanto, la partición studentPartition2 está duplicada a través de los dos replicationGroupMembers. El grupo de duplicación replicationGroup2 también tiene establecidos los atributos minReplicas y maxReplicas. Como minReplicas está establecido en 1, los datos de correlación que se alojan en este grupo de duplicación son de sólo lectura hasta que un miembro primario y al menos una duplicación estén disponibles. Un valor de maxReplicas igual a 1 indica que el miembro primario de este grupo de duplicación duplica los datos en una duplicación como máximo. En el caso del grupo de duplicación replicationGroup2, no se puede exceder una duplicación porque el grupo sólo contiene dos miembros. Uno es el miembro primario y el otro es la duplicación.

Elemento replicationGroupMember

Número de apariciones: de una a varias

Elementos hijo: ninguno

Un elemento replicationGroupMember se utiliza para hacer referencia a una definición de servidor. Cada elemento replicationGroupMember también tiene una prioridad asociada. Esta prioridad se utiliza para determinar qué replicationGroupMember es el servidor primario y qué miembros son duplicaciones.

Atributos

```
<replicationGroupMember
(1) serverRef="serverDefinitionReference"
(2) priority="priority"
/>
```

1. Atributo **serverRef** (necesario): el atributo serverRef se utiliza para asociar una definición de servidor con un elemento replicationGroupMember. El atributo serverRef asocia cada replicationGroupMember con un servidor específico.
2. Elemento **priority** (necesario): el atributo priority se utiliza para determinar qué miembros de grupo de duplicación son los primarios. El rango de valores de prioridad va de `1` al número de miembros de grupo de duplicación, donde `1` es la prioridad máxima. ObjectGrid realiza el esfuerzo de reconocer la prioridad de cada miembro de grupo de duplicación. El elemento replicationGroupMember con una prioridad `1` es el miembro primario, a menos que las circunstancias lo impidan. Si todos los servidores y sus replicationGroupMembers están disponibles al mismo tiempo, se reconocen los valores de prioridad. No obstante, si un replicationGroupMember con una prioridad `2` está disponible mucho antes que cualquier otro repliationGroupMember, se convierte en el miembro primario.

   Si el miembro primario se inicia satisfactoriamente y falla después de un periodo de tiempo, se debe seleccionar un nuevo miembro primario. El elemento replicationGroupMember con la siguiente prioridad máxima será el nuevo miembro primario. No obstante, se puede seleccionar una duplicación diferente como nuevo miembro primario si la duplicación con la siguiente prioridad máxima está determinada a ir detrás en su duplicación.

Consulte el archivo `universityClusterMapSet.xml` para ver un ejemplo de uso del elemento partition. En el archivo `universityClusterMapSet.xml`, el grupo de duplicación replicationGroup1 tiene sólo un replicationGroupMember. Debido a esta definición, este replicationGroup sólo tiene un miembro primario. No hay

duplicaciones en este grupo. El replicationGroupMember definido está activado en el servidor server1 como indica el valor serverRef.

El grupo de duplicación replicationGroup2 tiene más de un replicationGroupMember. El primer replicationGroupMember de la lista se activa en el servidor server1. Este miembro tiene una prioridad 2. El segundo replicationGroupMember de la lista se activa en el servidor sever2. Como el segundo miembro tiene una prioridad 1, será el miembro primario de este grupo de duplicación si los miembros de grupo están disponibles aproximadamente al mismo tiempo. El primer replicationGroupMember que aparece en la lista sirve como duplicación porque tiene una prioridad 2.

También es importante entender cómo el valor de minReplicas afecta al grupo de duplicación replicationGroup2. Considere un ejemplo en el que se están ejecutando los servidores server1 y server2. En este caso, los dos replicationGroupMembers están disponibles. Por lo tanto, los valores de minReplicas y maxReplicas se cumplen ambos y los datos se duplican entre el miembro primario y la duplicación de este grupo. Si el server1 deja de estar disponible, uno de los replicationGroupMembers deja de estar disponible. En este caso, los datos del grupo de duplicación replicationGroup2 pasan a ser de sólo lectura, porque el valor de minReplicas ya no se cumple.

Elemento authenticator

Número de apariciones: de cero a varias

Elemento hijo: elemento de propiedad

El elemento authenticator se utiliza para autenticar clientes en servidores ObjectGrid del clúster. La clase especificada por el atributo className debe implementar la interfaz com.ibm.websphere.objectgrid.security.plugins.Authenticator. El autenticador puede utilizar propiedades para invocar métodos en la clase especificada por el atributo className. Consulte “Elemento property” en la página 290 para obtener más información sobre el uso de las propiedades.

Atributos

```
<authenticator
(1) className="authenticatorClassName"
/>
```

1. Atributo **className** (necesario): el atributo className se utiliza para especificar una clase que implementa la interfaz com.ibm.websphere.objectgrid.security.plugins.Authenticator. Esta clase se utiliza para autenticar clientes en los servidores del clúster ObjectGrid.

El siguiente archivo `universityClusterSecurity.xml` muestra cómo utilizar el elemento authenticator. En este ejemplo, la clase com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator se especifica como autenticador. Esta clase implementa la interfaz com.ibm.websphere.objectgrid.security.plugins.Authenticator.

Archivo *universityClusterSecurity.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster" securityEnabled="true"
```

```
singleSignOnEnabled="true"
loginSessionExpirationTime="1800" adminAuthorizationEnabled="true"
adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
<serverDefinition name="server1" host="lion.ibm.com"
clientAccessPort="12501" peerAccessPort="12502" />
<authenticator
className ="com.ibm.websphere.objectgrid.security.plugins.builtins.
KeyStoreLoginAuthenticator" />
<adminAuthorization className= "com.ibm.MyAdminAuthorization">
<property name="interval" type="int" value="60"
description="Establecer el
intervalo en 60 segundos" />
</adminAuthorization>
</cluster>
<objectgridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

Elemento adminAuthorization

Número de apariciones: de cero a varias

Elemento hijo: elemento de propiedad

Un elemento adminAuthorization se utiliza para configurar el acceso administrativo al clúster ObjectGrid. Las tareas administrativas se pueden ejecutar una vez se ha proporcionado el acceso de administración.

Atributos

```
<adminAuthorization
(1) className="adminAuthClassName"
/>
```

1. Atributo **className** (necesario): el atributo className se utiliza para especificar una clase que implementa la interfaz com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization.

Consulte el archivo `universityClusterSecurity.xml` del apartado authenticator para ver un ejemplo de uso del elemento adminAuthorization. En el archivo universityClusterSecurity.xml, se utiliza una adminAuthorization personalizada. La clase com.ibm.MyAdminAuthorization se utiliza como la clase adminAuthorization. Para utilizar una adminAuthorization personalizada, el atributo securityEnabled debe ser `true`, adminAuthorizationMechanism debe establecerse en `AUTHORIZATION_MECHANISM_CUSTOM` y se debe proporcionar un elemento adminAuthorization. Este elemento adminAuthorization también utiliza una

propiedad. Para obtener más información sobre cómo utilizar las propiedades, consulte “Elemento property”.

Elemento property

Número de apariciones: de cero a varias

Elementos hijo: ninguno

El elemento property se utiliza para invocar métodos set en los elementos authenticator y adminAuthorization. El nombre de la propiedad corresponde a un método set del className del elemento authenticator o adminAuthorization que contiene la propiedad.

Atributos

```
<property
(1) name="propertyName"
(2) type="java.lang.String|boolean|java.lang.Boolean|int|
java.lang.Integer|double|java.lang.Double|byte|
java.lang.Byte|short|java.lang.Short|long|
java.lang.Long|float|java.lang.Float|char|
java.lang.Character"
(3) value="propertyValue"
(4) description="description"
/>
```

1. Atributo **name** (necesario): el nombre de la propiedad. El valor asignado a este atributo debe corresponder a un método set de la clase que se proporciona como el className del autenticador o adminAuthorization. Por ejemplo, si el className del autenticador se establece en com.ibm.MyAuthenticator y el nombre de la propiedad suministrado es `interval`, entonces la clase com.ibm.MyAuthenticator debe tener un método setInterval.
2. Atributo **type** (necesario): el tipo de la propiedad. Es el tipo del parámetro que se ha pasado al método set identificado mediante el atributo name. Los valores válidos son primitivos Java, los correspondientes a java.lang y java.lang.String. El nombre y el tipo deben corresponder a una signatura de método del className del bean. Por ejemplo, si el nombre es interval y el tipo es int, entonces debe existir un método setInterval(int) en la clase que se especifica como el className del autenticador o adminAuthorization.
3. Atributo **value** (necesario): el valor de la propiedad. Este valor se convierte en el tipo que se especifica mediante el atributo de tipo y se utiliza como un parámetro en la llamada al método set que se identifica mediante los atributos de nombre y tipo. Es importante saber que el valor de este atributo no se valida de ningún modo. El implementador del plug-in debe verificar que el valor que se ha pasado es válido. El implementador puede visualizar una excepción `IllegalArgumentException` en el método set si el parámetro no es válido.
4. Atributo **description** (opcional): Utilice este atributo para escribir una descripción de la propiedad.

Consulte el archivo `universityClusterSecurity.xml` para ver un ejemplo de uso del elemento adminAuthorization. El elemento property se puede utilizar en los elementos authenticator y adminAuthorization del XML de clúster. En el archivo `universityClusterSecurity.xml`, property se utiliza para invocar un método set en adminAuthorization. En este caso, se invoca un método setInterval en la clase com.ibm.MyAdminAuthorization. Se pasa un valor entero igual a `60`.

Capítulo 10. Integración de ObjectGrid con WebSphere Application Server

Utilice ObjectGrid con las características suministradas con WebSphere Application Server para mejorar las aplicaciones con las posibilidades de ObjectGrid.

Instale WebSphere Application Server y WebSphere Extended Deployment. Una vez instalado WebSphere Extended Deployment, puede añadir funciones de ObjectGrid a las aplicaciones J2EE (Java 2 Platform, Enterprise Edition).

Se puede utilizar la API de ObjectGrid API en una aplicación J2EE destinada para WebSphere Application Server. El archivo `wsobjectgrid.jar` está en el directorio `\base\lib` una vez instalado WebSphere Extended Deployment. Además de la integración de la API de ObjectGrid con el modelo de programación de aplicaciones J2EE, puede beneficiarse del soporte de propagación de transacciones distribuidas. Con este soporte, puede configurar las instancias de ObjectGrid para coordinar los resultados del compromiso de las transacciones en un clúster de WebSphere Application Server.

1. Efectúe los pasos de programación básicos para habilitar una aplicación J2EE con ObjectGrid. Para obtener más información, consulte “Integración de ObjectGrid en un entorno Java 2 Platform, Enterprise Edition”.
2. Supervise los datos de rendimiento de las aplicaciones ObjectGrid. Para obtener más información, consulte “Supervisión del rendimiento de ObjectGrid con PMI (Performance Monitoring Infrastructure) de WebSphere Application Server” en la página 295.
3. Cuando se intercala ObjectGrid, es posible que las transacciones se inicien y finalicen mediante un coordinador de transacciones externo. Para obtener más información, consulte “ObjectGrid y la interacción con transacciones externas” en la página 302.
4. Utilice Partitioning Facility con ObjectGrid. La característica ObjectGrid proporciona la capacidad de almacenamiento en antememoria de los pares de clave y valor de modo transaccional y Partitioning Facility proporciona la capacidad de direccionamiento basado en contexto según las características de los objetos. Para obtener más información, consulte “Integración de ObjectGrid y Partition Facility” en la página 305.
5. Puede utilizar beans de CMP (persistencia gestionada por contenedor) en WebSphere Application Server Versión 6.0.2 y posteriores, beneficiándose de ObjectGrid como antememoria externa en lugar de una antememoria integrada. Para obtener más información, consulte “Configuración de ObjectGrid para trabajar con beans gestionados por contenedor” en la página 326.

También puede utilizar ObjectGrid con JMS para distribuir los cambios entre los diferentes niveles o en entornos de plataformas combinadas. Para obtener más información, consulte “Java Message Service para distribuir cambios de transacción” en la página 344.

Integración de ObjectGrid en un entorno Java 2 Platform, Enterprise Edition

ObjectGrid da soporte a modelos de programación de servlet y EJB (Enterprise JavaBeans) en el entorno J2EE (Java 2 Platform, Enterprise Edition).

Este tema describe los pasos de programación comunes para habilitar una aplicación J2EE con ObjectGrid.

Caso de ejemplo de ObjectGrid local

1. Definir una configuración de ObjectGrid. Definir una configuración de ObjectGrid con archivos XML, mediante una interfaz de programación o con un uso combinado de archivos XML y de configuración programada. Para obtener más información, consulte el tema Configuración de ObjectGrid.
2. Crear un objeto de URL. Si la configuración de ObjectGrid es un archivo XML, cree un objeto de URL que apunte a dicho archivo XML. Puede utilizar este objeto de URL para crear instancias de ObjectGrid utilizando la API ObjectGridManager. Si el archivo XML de configuración de ObjectGrid se incluye en un archivo WAR (Web Archive) o en un archivo JAR (Java Archive) EJB (Enterprise JavaBeans), se puede acceder al mismo como un recurso para el cargador de clases del módulo Web y EJB. Por ejemplo, si el archivo XML de configuración de ObjectGrid está en la carpeta WEB–INF del archivo WAR del módulo Web, los servlets que estén en dicho archivo WAR pueden crear un objeto URL con el patrón siguiente:

   ```
   URL url =className.class.getClassLoader().
   getResource("META-INF/objectgrid-definition.xml");
   URL objectgridUrl = ObjectGridCreationServlet.class.getClassLoader().
   getResource("WEB-INF/objectgrid-definition.xml");
   ```

3. Crear u obtener instancias de ObjectGrid. Utilice la API de ObjectGridManager para obtener y crear instancias de ObjectGrid. Con la API de ObjectGridManager, puede crear instancias de ObjectGrid con XML y utilizar los métodos del programa de utilidad para crear rápidamente una instancia de ObjectGrid sencilla. Las aplicaciones deben utilizar la API de ObjectGridManagerFactory para obtener una referencia a la API de ObjectGridManager. Consulte el siguiente ejemplo de código:

   ```
   import com.ibm.websphere.objectgrid.ObjectGridManager;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory ;
   ...
   ObjectGridManager objectGridManager = ObjectGridManagerFactory.
   getObjectGridManager();
   ObjectGrid ivObjectGrid = objectGridManager.
   createObjectGrid(objectGridName, objectgridUrl, true, true);
   ```

   Para obtener más información acerca de la API de ObjectGridManager, consulte el tema Interfaz ObjectGridManager.
4. Inicializar las instancias de ObjectGrid. Utilice el método initiliatize de la interfaz ObjectGrid para iniciar la rutina de carga de ObjectGrid y las instancias de Session. Este método initialize se considera opcional porque la primera llamada al método getSession efectúa una inicialización implícita. Una vez invocado este método, la configuración de ObjectGrid se considera completa y lista para su uso en tiempo de ejecución. Cualquier invocación del método de configuración adicional como, por ejemplo, la llamada al método defineMap(String mapName), da como resultado una excepción.
5. Obtener una sesión y una instancia de ObjectMap. Una sesión es un contenedor de instancias de ObjectMap. Una hebra debe obtener su propio objeto de sesión para interactuar con el núcleo principal de ObjectGrid. Puede considerar esta técnica como una sesión que solo puede utilizar una hebra cada vez. La sesión se puede compartir entre hebras si utiliza solamente una hebra cada vez. No obstante, si se utiliza una conexión J2EE o una infraestructura de transacciones, el objeto de sesión no se puede compartir entre las hebras. Una buena analogía para este objeto es una conexión JDBC (Java Database Connectivity) con una base de datos.

Una correlación ObjectMap es un manejador para una correlación con nombre. Las correlaciones deben tener claves y valores homogéneos. Una instancia de ObjectMap sólo la puede utilizar la hebra asociada actualmente con la sesión utilizada para obtener esta instancia de ObjectMap. Los objetos Session y ObjectMap no los pueden compartir varias hebras al mismo tiempo. Las palabras clave se aplican en una transacción. Una retrotracción de una transacción retrotrae cualquier asociación de palabra clave que se aplica durante esta transacción.El ejemplo de codificación es el siguiente:

```
Session ivSession = ivObjectGrid.getSession();
ObjectMap ivEmpMap = ivSession.getMap("employees");
ObjectMap ivOfficeMap = ivSession.getMap("offices");
ObjectMap ivSiteMap = ivSession.getMap("sites");
ObjectMap ivCounterMap = ivSession.getMap("counters");
```

6. Iniciar una sesión, leer o grabar objetos y comprometer o retrotraer la sesión. Las operaciones de correlación deben estar dentro de un contexto de transacción. El método begin del objeto Session se utiliza para iniciar un contexto de transacción explícito. Después de iniciar la sesión, las aplicaciones pueden iniciar las operaciones de correlación. Las operaciones más comunes son las llamadas a los métodos get, `update`, insert y remove para objetos en correlaciones. Al final de las operaciones de correlación, se invoca el método commit o rollback del objeto Session para comprometer un contexto de transacción explícito o retrotraer un contexto de transacción explícito. El siguiente es un ejemplo de programación:

```
ivSession.begin();
Integer key = new Integer(1);
if (ivCounterMap.containsKey(key) == false) {
 ivCounterMap.insert(key, new Counter(10));
}
ivSession.commit();
```

Caso de ejemplo de ObjectGrid distribuida

Este caso de ejemplo de ObjectGrid distribuida difiere del caso de ejemplo de ObjectGrid local sólo en la forma de obtener la instancia de ObjectGrid. Los siguientes ejemplos de código demuestran cómo obtener una instancia de ObjectGrid distribuida:

```
//Utilice ObjectGridManagerFactory para obtener la referencia a la API de ObjectGridManager
ObjectGridManager objectGridManager = ObjectGridManagerFactory.
getObjectGridManager();
//Obtenga el ClientClusterContext que representa el clúster ObjectGrid de
//ObjectGridManager
//En el supuesto de que el servidor WebSphere también aloja un servidor ObjectGrid
//que es miembro del clúster ObjectGrid.
ClientClusterContext context = objectGridManager.connect(null, null);
//Obtenga la instancia de ObejctGrid del ObjectGridManager, un
//ClientClusterContext específico.
ObjectGrid ivObjectGrid= objectGridManager.getObjectGrid(context,
"objectgridName");
```

Ha realizado los pasos de programación básicos para habilitar una aplicación J2EE con ObjectGrid.

Para obtener más información, consulte Creación de aplicaciones J2EE (Java 2 Platform, Enterprise Edition) habilitadas para ObjectGrid y Consideraciones para la integración de las aplicaciones J2EE (Java 2 Platform, Enterprise Edition) y ObjectGrid.

Creación de aplicaciones J2EE (Java 2 Platform, Enterprise Edition) habilitadas para ObjectGrid

Utilice esta tarea para configurar la vía de acceso de construcción o la vía de acceso de clase, de las aplicaciones J2EE (Java 2 Platform, Enterprise Edition) habilitadas para Java. La vía de acceso de clase debe incluir el archivo `wsobjectgrid.jar` situado en el directorio `$install_root/lib`.

Desarrollo de una aplicación J2EE habilitada para ObjectGrid. Consulte Integración de ObjectGrid en un entorno J2EE para obtener más información.

Esta tarea muestra cómo se establece la vía de acceso de construcción para incluir el archivo `wsobjectgrid.jar` en IBM Rational Software Development Platform Versión 6.0.

1. En la vista del Explorador de proyectos de la perspectiva J2EE, pulse con el botón derecho del ratón en proyecto **WEB** o Enterprise JavaBeans (**EJB**) y seleccione **Propiedades**. Se visualizará la ventana Propiedades.
2. Seleccione **Vía de acceso de construcción java** en el panel de la izquierda, pulse la pestaña **Bibliotecas** en el panel de la derecha y pulse **Añadir variable**. Se muestra la ventana **Nueva entrada de variable classpath**.
3. Pulse **Configurar variables** para abrir la ventana **Preferencia**.
4. Añada una entrada de variable nueva.
   a. Pulse **Nuevo**.
   b. Escriba `OBJECTGRID_JAR` en el campo **nombre**. Pulse **Archivo** para abrir la ventana **Selección de archivos JAR**.
   c. Vaya al directorio `/lib`, pulse el archivo **wsobjectgrid.jar** y pulse **Abrir** para cerrar la ventana **Selección de archivos JAR**.
   d. Pulse **Aceptar** para cerrar la ventana **Nueva entrada de variable**.

   La variable `OBJECTGRID_JAR` se visualiza en la lista de variables classpath.
5. Pulse **Aceptar** para cerrar la ventana **Preferencia**.
6. Seleccione la variable `OBJECTGRID_JAR` en la lista de variables y pulse **Aceptar** para cerrar la ventana **Nueva entrada de variable classpath**. Se muestra la variable `OBJECTGRID_JAR` en el panel **Bibliotecas**.
7. Pulse **Aceptar** para cerrar la ventana **Propiedades**.

La vía de acceso de construcción se ha establecido de modo que se incluye el archivo `wsobjectgrid.jar` en IBM Rational Software Development Platform Versión 6.0.

Consideraciones acerca de la integración de las aplicaciones Java 2 Platform, Enterprise Edition y ObjectGrid

Tenga en cuenta estas consideraciones cuando integre una aplicación J2EE (Java 2 Platform, Enterprise Edition) con ObjectGrid.

Beans de arranque y ObjectGrid

Puede utilizar los beans de arranque en una aplicación para la rutina de carga de una instancia ObjectGrid cuando se inicia la aplicación y para destruir la instancia de ObjectGrid cuando se detiene la aplicación. Un bean de arranque es un bean de sesión sin estado con un com.ibm.websphere.startupservice.AppStartUpHome inicial remoto y una interfaz com.ibm.websphere.startupservice.AppStartUp remota. Cuando WebSphere Application Server ve un EJB (Enterprise JavaBean), reconoce

el bean de arranque. La interfaz remota tiene dos métodos, el método de inicio y el método de detención. Utilice el método de inicio para la rutina de carga de la ObjectGrid e invoque el método destroy de ObjectGrid con el método de detención. La aplicación puede mantener una referencia a la ObjectGrid utilizando el método ObjectGridManager.getObjectGrid para obtener una referencia cuando sea necesario. Para obtener más información, consulte el tema Interfaz ObjectGridManager.

Cargadores de clases e instancias de ObjectGrid

Debe prestar atención cuando comparta una sola instancia de ObjectGrid entre módulos de aplicación que utilizan cargadores de clase diferentes. Los módulos de aplicación que utilizan cargadores de clases diferentes no funcionan y dan como resultado excepciones de conversión de clases en la aplicación. Una ObjectGrid sólo debe compartirse por módulos de aplicaciones que utilicen el mismo cargador de clases o cuando los objetos de aplicación, por ejemplo, los plug-in, claves y valores estén en un cargador de clases común.

Gestionar el ciclo de vida de las instancias de ObjectGrid en un servlet

Puede gestionar el ciclo de vida de las instancias de ObjectGrid con el método init y el método destroy de un servlet. Utilice el método init para crear e inicializar las instancias de ObjectGrid que necesite la aplicación. Después de que se hayan creado y guardado en la antememoria las instancias de ObjectGrid, puede obtener las instancias por su nombre con la API ObjectGridManager. Utilice el método destroy para destruir estas instancias de ObjectGrid y liberar recursos del sistema. Para obtener más información, consulte el tema Interfaz ObjectGridManager.

Supervisión del rendimiento de ObjectGrid con PMI (Performance Monitoring Infrastructure) de WebSphere Application Server

ObjectGrid da soporte a PMI (Performance Monitoring Infrastructure) cuando se ejecuta en un servidor de aplicaciones WebSphere Application Server o WebSphere Extended Deployment. PMI recopila datos de rendimiento en aplicaciones de tiempo de ejecución y proporciona interfaces que dan soporte a aplicaciones externas para supervisar los datos de rendimiento.

Para obtener más información acerca de las estadísticas que proporciona ObjectGrid, consulte el tema Estadísticas de ObjectGrid.

ObjectGrid utiliza la característica PMI personalizada de WebSphere Application Server para añadir su propia instrumentación PMI. Con este método, puede habilitar e inhabilitar PMI en ObjectGrid mediante la interfaz de la consola administrativa o JMX (Java Management Extensions). Asimismo, puede acceder a las estadísticas de ObjectGrid con las interfaces PMI y JMX estándar que utilizan las herramientas de supervisión, incluido Tivoli Performance Viewer.

1. Habilitar PMI en ObjectGrid. Debe habilitar PMI para ver las estadísticas de PMI. Para obtener más información, consulte “Habilitación de PMI en ObjectGrid” en la página 298.
2. Recuperar las estadísticas PMI de ObjectGrid. Vea el rendimiento de las aplicaciones de ObjectGrid con Tivoli Performance Viewer. Para obtener más información, consulte “Recuperación de las estadísticas PMI de ObjectGrid” en la página 301.

Estadísticas de ObjectGrid

ObjectGrid proporciona dos módulos PMI (Performance Monitoring Infrastructure): el módulo objectGridModule y el módulo mapModule.

Módulo objectGridModule

El módulo objectGridModule contiene una estadística de tiempo: el tiempo de respuesta de las transacciones. Una transacción ObjectGrid se define como el período de tiempo entre la llamada al método Session.begin y la llamada al método Session.commit. Este período de tiempo se considera el tiempo de respuesta de la transacción.

El elemento raíz del módulo objectGridModule, el elemento ObjectGrids, hace las veces de punto de entrada para las estadísticas ObjectGrid. Este elemento raíz tiene como hijos instancias de ObjectGrid que tienen como hijos tipos de transacciones. La estadística de tiempo de respuesta se asocia a cada tipo de transacción. En el siguiente diagrama se muestra la estructura del módulo objectGridModule:

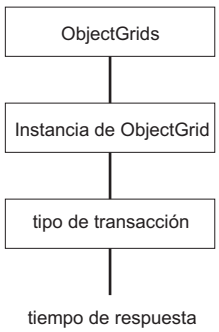


Figura 19. Estructura del módulo ObjectGridModule

El diagrama siguiente muestra un ejemplo de la estructura del módulo PMI de ObjectGrid. En este ejemplo, existen dos instancias de ObjectGrid en el sistema: objectGrid1 y objectGrid2. La instancia objectGrid1 tiene dos tipos de transacciones: update y read y la instancia objectGrid2 sólo tiene el tipo de transacción update.

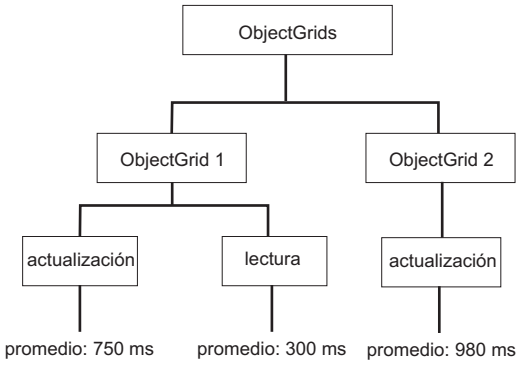


Figura 20. Estructura del módulo PMI de ObjectGrid

Los tipos de transacciones los definen los desarrolladores de aplicaciones debido a que saben 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 las transacciones futuras.
*
* Después de llamar a este método, todas las transacciones futuras tienen el mismo
* tipo hasta que se establece otro tipo de transacción. Si no se establece ningún
* tipo de transacción, se utiliza el tipo de transacción TRANSACTION_TYPE_DEFAULT
* por omisión.
*
* Los tipos de transacciones se utilizan principalmente para fines de seguimiento
* de datos  estadísticos. Los usuarios pueden definir previamente los tipos de
* transacciones que 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 de una transacción 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 siguiente es `updatePrice`. Existe una estadística `updatePrice` bajo la instancia ObjectGrid que se corresponde con 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 las estadísticas agregadas para todos los tipos de transacciones en la ObjectGrid especificada.

Módulo mapModule

El módulo PMI mapModule contiene tres estadísticas que están relacionadas con las correlaciones de ObjectGrid:

- **Proporción de coincidencias de la correlación:** Esta estadística 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**: Esta estadística CountStatistic efectúa un seguimiento del número de entradas de la correlación.

- **Tiempo de respuesta de la actualización por lotes del cargador**: Esta estadística TimeStatistic efectúa un seguimiento del tiempo de respuesta que se utiliza para la operación de actualización por lotes del cargador.

El elemento raíz del módulo mapModule, el elemento de correlación de ObjectGrid, sirve como punto de entrada para las estadísticas de correlación de ObjectGrid. Este elemento raíz tiene como hijos instancias de ObjectGrid que tienen como hijos instancias de correlación. Cada instancia de correlación tiene tres estadísticas listadas. En el diagrama siguiente se muestra la estructura de mapModule:
El diagrama siguiente muestra un ejemplo de la estructura de mapModule:

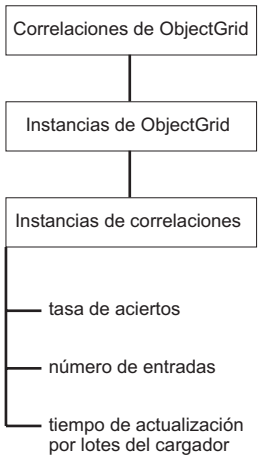


Figura 21. Estructura del módulo mapModule

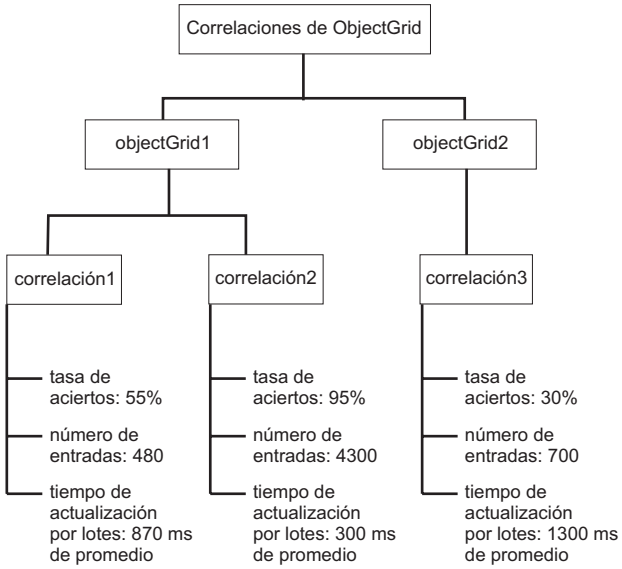


Figura 22. Ejemplo de la estructura del módulo mapModule

Habilitación de PMI en ObjectGrid

Puede utilizar PMI (Performance Monitoring Infrastructure) de WebSphere Application Server para habilitar o inhabilitar las estadísticas a cualquier nivel. Por ejemplo, puede elegir que simplemente 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. En este tema se muestra cómo se utiliza la consola administrativa y los scripts wsadmin para habilitar PMI en ObjectGrid.

Utilice PMI de WebSphere Application Server para proporcionar un mecanismo granular en el que pueda habilitar o inhabilitar las 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. Este apartado muestra cómo utilizar la consola administrativa y los scripts wsadmin para habilitar PMI en ObjectGrid.

1. Abra la consola administrativa, por ejemplo, http://localhost:9060/ibm/console.
2. Pulse **Supervisión y ajuste > Performance Monitoring Infrastructure > *nombre_servidor***.
3. Compruebe que se haya seleccionado **Habilitar PMI (Infraestructura de supervisión del rendimiento)**. Este valor está habilitado por omisión. Si no se habilita este valor, seleccione el recuadro de selección y, a continuación, reinicie el servidor.
4. Pulse **Personalizado**. En el árbol de configuración, seleccione **ObjectGrid** y el módulo **Correlaciones de ObjectGrid**. Habilite las estadísticas para cada módulo.

La categoría de tipo de transacción para las estadísticas de ObjectGrid se crea durante la ejecución. Sólo puede ver las subcategorías de las estadísticas de ObjectGrid y de a correlación en el panel de tiempo de ejecución.

Por ejemplo, puede realizar los pasos siguientes para habilitar las estadísticas de PMI para la aplicación de ejemplo:

1. Inicie la aplicación utilizando la dirección Web `http://host:port/ObjectGridSample`, en el que el sistema principal y el puerto son el nombre de sistema principal y el número de puerto HTTP del servidor en el que está instalado el ejemplo.
2. En la aplicación de ejemplo, pulse **ObjectGridCreationServlet** y, a continuación, pulse los botones de acción 1, 2, 3, 4 y 5 para generar algunas acciones en ObjectGrid y en las correlaciones. No cierre esta página del servlet en este momento.
3. Vuelva a 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. Amplíe el módulo **Correlaciones de ObjectGrid** en el árbol de tiempo de ejecución y, a continuación, pulse el enlace **clusterObjectGrid**. En el grupo **Correlaciones de ObjectGrid**, existe una instancia de ObjectGrid que se denomina clusterObjectGrid y bajo este grupo clusterObjectGrid existen cuatro correlaciones: contadores, empleados, oficinas y sitios. En la instancia de **ObjectGrids**, existe una instancia de clusterObjectGrid y bajo dicha instancia existe un tipo de transacción denominado `DEFAULT`.
6. Puede habilitar las estadísticas que le interesen. Para fines de demostración puede habilitar el **número de entradas de correlación** para la correlación de empleados y el **tiempo de respuesta de la transacción** para el tipo de transacción DEFAULT.

Puede automatizar la tarea de habilitar PMI con scripts. Consulte Habilitación de PMI en ObjectGrid con scripts para obtener más información.

Habilitación de PMI en ObjectGrid con scripts

Automatice la tarea de habilitar PMI en ObjectGrid con la herramienta wsadmin.

El servidor de aplicaciones se debe haber iniciado y debe tener instalada una aplicación habilitada para ObjectGrid. También debe poder iniciar la sesión y utilizar la herramienta wsadmin. Para obtener más información acerca de la herramienta wsadmin, consulte Utilización de scripts (wsadmin) en el centro de información de WebSphere Extended Deployment Versión 6.0.x.

Utilice esta tarea para automatizar la habilitación de PMI. Para habilitar PMI con la consola administrativa, consulte Habilitación de PMI en ObjectGrid.

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 de tiempo de ejecución PMI de ObjectGrid. Compruebe si se ha habilitado PMI para el servidor con los mandatos siguientes:

   ```
   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 no se ha habilitado PMI, ejecute los mandatos siguientes para habilitar PMI:

   ```
   wsadmin>$AdminConfig modify $pmi {{enable true}}
   wsadmin>$AdminConfig save
   ```

   Si tiene que habilitar PMI, reinicie el servidor.
3. Establezca las variables para modificar el conjunto de estadísticas en un conjunto personalizado. Ejecute los mandatos siguientes:

   ```
   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 como personalizadas: Ejecute el mandato siguiente:

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
   ```
5. Establezca las variables para habilitar la estadísticas PMI en objectGridModule. Ejecute los mandatos siguientes:

   ```
   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. Ejecute el mandato siguiente:

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params $sigs
   ```
7. Establezca las variables para habilitar la estadísticas PMI en mapModule. Ejecute los mandatos siguientes:

```
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
```

8. Establezca la serie de estadísticas. Ejecute el mandato siguiente:

```
wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params2 $sigs2
```

Estos pasos habilitan PMI en tiempo de ejecución para ObjectGrid pero no modifican la configuración de PMI. Si reinicia el servidor de aplicaciones, se pierden los valores de PMI que no sean la habilitación PMI principal.

Después de habilitar PMI, puede ver las estadísticas de PMI con la consola administrativa o mediante scripts. Consulte “Recuperación de las estadísticas PMI de ObjectGrid” y “Recuperación de estadísticas PMI de ObjectGrid con scripts” para obtener más información.

Recuperación de las estadísticas PMI de ObjectGrid

Consulte las estadísticas de rendimiento de sus aplicaciones ObjectGrid.

Una vez habilitadas las estadísticas de ObjectGrid, puede recuperarlas. Para habilitar PMI en ObjectGrid, consulte Habilitación de PMI en ObjectGrid.

Utilice esta tarea para ver las estadísticas de rendimiento de sus aplicaciones ObjectGrid.

1. Abra la consola administrativa. Por ejemplo, http://localhost:9060/ibm/console.
2. Pulse **Supervisión y ajuste > Performance Viewer > Actividad actual**.
3. Pulse el servidor que desea supervisar con Tivoli Performance Viewer y habilite la supervisión.
4. Pulse el servidor para ver la página de Performance Viewer.
5. Expanda el árbol de configuración. Pulse **Correlaciones de ObjectGrid > clusterObjectGrid** y seleccione **employees**. Expanda **ObjectGrids > clusterObjectGrid**, y seleccione **DEFAULT**.
6. En la aplicación de ejemplo de ObjectGrid, vuelva al servlet ObjectGridCreationServlet, pulse el botón 1, **llenar correlaciones**. Puede ver las estadísticas en el visor.

Puede ver las estadísticas de ObjectGrid en Tivoli Performance Viewer.

Puede automatizar la tarea de recuperar las estadísticas utilizando JMX (Java Management Extensions) o con la herramienta wsadmin. Consulte “Recuperación de estadísticas PMI de ObjectGrid con scripts”

Recuperación de estadísticas PMI de ObjectGrid con scripts

Utilice esta tarea para recuperar las estadísticas de rendimiento de las aplicaciones ObjectGrid.

Habilite PMI (Performance Monitoring Infrastructure) en el entorno de servidor de aplicaciones. Para obtener más información, consulte Habilitación de PMI en ObjectGrid o Habilitación de PMI en ObjectGrid con scripts. También debe poder iniciar la sesión y utilizar la herramienta wsadmin. Para obtener más información acerca de la herramienta wsadmin, consulte Utilización de scripts (wsadmin) en el centro de información de WebSphere Extended Deployment Versión 6.0.x.

Utilice esta tarea para obtener las estadísticas de rendimiento del entorno de servidor de aplicaciones. Para obtener más información acerca de las estadísticas de ObjectGrid que puede recuperar, consulte “Estadísticas de ObjectGrid” en la página 296.

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. Establezca las variables para el entorno. Ejecute los mandatos siguientes:

```
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 las estadísticas mapModule. Ejecute los mandatos siguientes:

```
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 las estadísticas de mapModule. Ejecute el mandato siguiente:

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params $sigs
```

5. Establezca las variables para obtener las estadísticas de objectGridModule. Ejecute los mandatos siguientes:

```
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. Ejecute el mandato siguiente:

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params2 $sigs2
```

Consulte el tema “Estadísticas de ObjectGrid” en la página 296 para obtener más información acerca de las estadísticas devueltas.

ObjectGrid y la interacción con transacciones externas

Generalmente las transacciones ObjectGrid comienzan con el método session.begin y terminan con el método session.commit. No obstante, cuando se intercala ObjectGrid, es posible que las transacciones se inicien y finalicen mediante un coordinador de transacciones externo. En este caso, no es necesario que llame al método session.begin y finalice con el método session.commit.

Coordinación de transacciones externas

El plug-in TransactionCallback de ObjectGrid se amplía con el método isExternalTransactionActive(Session session) que asocia la sesión ObjectGrid con una transacción externa. La cabecera del método es la siguiente

```
public synchronized boolean isExternalTransactionActive(Session session)
```

Por ejemplo, se puede establecer ObjectGrid de modo que se integre con WebSphere Application Server y WebSphere Extended Deployment. La clave de esta integración transparente es la utilización de la API ExtendedJTATransaction de WebSphere Application Server Versión 5.x y Versión 6.x. No obstante, si está utilizando WebSphere Application Server Versión 6.0.2, debe aplicar APAR PK07848 para dar soporte a este método. Utilice el código de ejemplo siguiente para asociar una sesión ObjectGrid con un ID de transacción de WebSphere Application Server:

```
/**
* Es método debe asociar una sesión objectGrid con un ID de transacción
* de WebSphere.
*/
Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
 // recuerde que este localid significa que la sesión se guarda para más tarde.
 localIdToSession.put(new Integer(jta.getLocalId()), session);
  return true;
}
```

Recuperar una transacción externa

Es posible que algunas veces tenga que recuperar un objeto de servicio de transacciones externo para que lo utilice el plug-in TransactionCallback de ObjectGrid. En el servidor WebSphere Application Server, debe buscar el objeto ExtendedJTATransaction de su espacio de nombre 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("No se puede registrar la devolución de llamada de jta", e);
}
 catch(NamingException e){
throw new RuntimeException("No se puede obtener el objeto de transacción");
}
}
```

Para otros productos, puede utilizar un método diferente para recuperar el objeto de servicio de transacciones.

Control de compromiso mediante retorno de llamada externo

El plug-in TransactionCallback debe recibir una señal externa para comprometer o retrotraer la sesión ObjectGrid. Para recibir esta señal externa, utilice el retorno de llamada del servicio de transacciones externo. Tiene que implementar la interfaz de retorno de llamada externa y registrarla con el servicio de transacciones externo. Por ejemplo, en el caso de WebSphere Application Server, tiene que implementar la interfaz SynchronizationCallback, 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("No se puede registrar la devolución de llamada de jta", e);
}
 catch(NamingException e)
{
throw new RuntimeException("No se puede obtener el objeto de transacción");
}
}
public synchronized void afterCompletion(int localId, byte[] arg1,
boolean didCommit)
{
 Integer lid = new Integer(localId);
 // buscar la sesión para el localId
 Session session = (Session)localIdToSession.get(lid);
 if(session != null)
{
try
{
   // si WebSphere Application Server se compromete cuando
   // se protege la transacción para backingMap.
   // Ya se ha realizado un vaciado en beforeCompletion
    if(didCommit)
{
session.commit();
}
else
{
    // de lo contrario, retrotraer
session.rollback();
}
}
  catch(NoActiveTransactionException e)
{
   // imposible en teoría
}
  catch(TransactionException e)
{
   // siempre que ya haya realizado un vaciado, esto no deberá fallar
}
finally
{
   // borre siempre 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 oficialmente
   // un modo de indicar que la transacción
   // ha fallado, por lo tanto, efectúe lo siguiente
throw new RuntimeException("Error en el vaciado de antememoria", e);
}
}
}
}
```

Utilice las API de ObjectGrid con el plug-in de TransactionCallback

Este plug-in, cuando se utiliza como el plug-in TransactionCallback para una ObjectGrid, inhabilita autocommit. El patrón de uso normal de ObjectGrid 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, ObjectGrid presupone que la aplicación utiliza ObjectGrid cuando hay una transacción gestionada por contenedor presente. El fragmento de código anterior modifica el código siguiente 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 caso de aplicación Web. La aplicación utiliza la interfaz UserTransaction normal para las transacciones de tipo begin, commit y rollback. ObjectGrid automáticamente se inicia y compromete en torno a la transacción del contenedor. Si el método es un método EJB (Enterprise JavaBeans) que utiliza el atributo TX_REQUIRES, a continuación, traslade la referencia UserTransaction y las llamadas a las transacciones begin y commit y el método funcionará del mismo modo. En este caso, el contenedor es responsable de iniciar y finalizar la transacción.

Integración de ObjectGrid y Partition Facility

Utilice la aplicación de ejemplo ObjectGridPartitionCluster para obtener información acerca de las funciones combinadas de ObjectGrid y Partitioning Facility, WPF.

Consulte “ObjectGrid y Partitioning Facility” en la página 306 para obtener información resumida acerca de cómo funcionan conjuntamente ObjectGrid y Partitioning Facility.

Para utilizar ObjectGrid con Partitioning Facility, debe tener instalado WebSphere Extended Deployment en su entorno.

El ejemplo de ObjectGridPartitionCluster muestra las funciones combinadas de ObjectGrid y Partitioning Facility, WPF. La característica ObjectGrid proporciona la capacidad de almacenamiento en antememoria de los pares de clave y valor por transacción y Partitioning Facility proporciona la capacidad de direccionamiento basado en contexto según las características de los objetos.

- Instale y ejecute la aplicación de ejemplo ObjectGridPartitionCluster. Para obtener más información, consulte “Instalación y ejecución de la aplicación de ejemplo ObjectGridPartitionCluster” en la página 308.
- Si desea ver o modificar el código fuente de la aplicación de ejemplo, puede cargar el archivo EAR (Enterprise Archive) en la herramienta de desarrollo. Para obtener más información, consulte “Creación de una aplicación de ObjectGrid y Partitioning Facility integrada” en la página 311.
- Obtenga información acerca de la aplicación de ejemplo. Consulte “Ejemplo: Programación de ObjectGrid y Partitioning Facility” en la página 315 para obtener una descripción del código que contiene la aplicación de ejemplo.

Consulte el Capítulo 10, “Integración de ObjectGrid con WebSphere Application Server”, en la página 291 para obtener más información acerca de cómo integrar ObjectGrid con otras características de WebSphere Application Server. Para obtener más información acerca del modelo de programación de ObjectGrid, consulte el Capítulo 9, “Visión general de la interfaz de programación de la aplicación ObjectGrid”, en la página 93.

ObjectGrid y Partitioning Facility

Las características ObjectGrid y Partitioning Facility, WPF, pueden funcionar conjuntamente para proporcionar almacenamiento en antememoria de los pares de clave y valor y el direccionamiento basado en contexto de las características de objetos.

El ejemplo ObjectGridPartitionCluster muestra las funciones combinadas de ObjectGrid y Partitioning Facility, WPF. ObjectGrid y Partitioning Facility son dos características del producto WebSphere Extended Deployment. La característica ObjectGrid proporciona la capacidad de almacenamiento en antememoria de los pares de clave y valor de modo transaccional y Partitioning Facility proporciona la capacidad de direccionamiento basado en contexto según las características de los objetos.

Además de mostrar las características de plug-in TransactionCallback y del cargador, este ejemplo también muestra cómo utilizar los plug-in ObjectGridEventListener, ObjectTransformer y OptimisticCallback. En particular, el ejemplo muestra cómo propagar las transacciones ObjectGrid locales y cómo invalidar los objetos modificados de un servidor a otros servidores con y sin la comprobación de versión optimista.

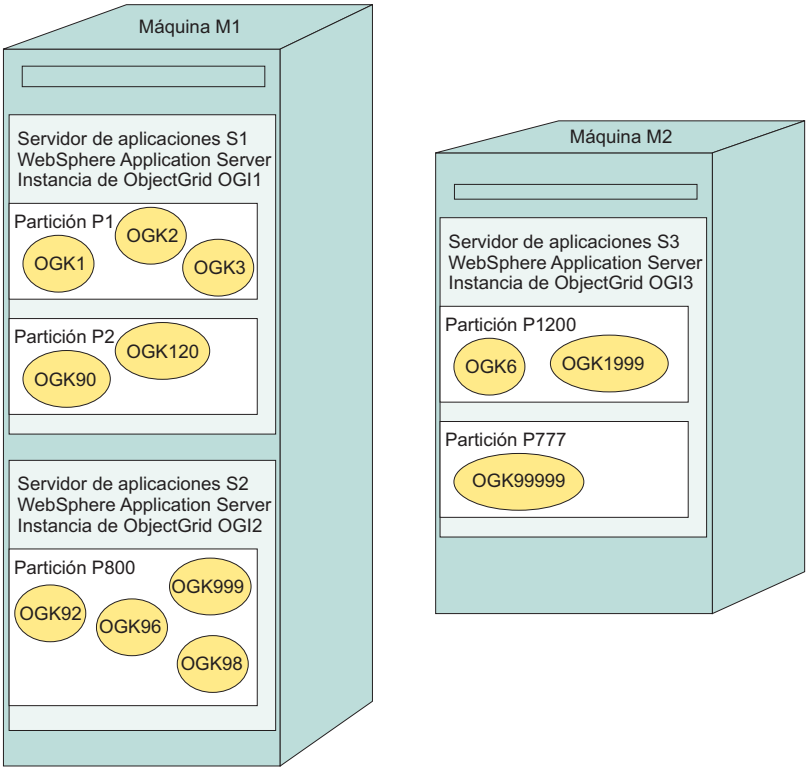
Debe utilizar la característica de direccionamiento basado en contexto de Partitioning Facility para asegurarse de que las peticiones de actualización, inserción y supresión de objetos para la misma clave se direccionen a la misma máquina virtual Java (JVM) y para que se puedan distribuir las peticiones de recuperación de objetos a todas las JVM de ObjectGrid con la gestión de carga de trabajo. Utilizando Partitioning Facility se mantiene la integridad de los datos a través de las diferentes instancias de ObjectGrid de los miembros del clúster.

Para mantener la integridad y la coherencia de ObjectGrid, puede utilizar Partitioning Facility para distribuir una ObjectGrid de gran tamaño entre las ObjectGrid particionadas y el direccionamiento basado en contexto de Partitioning Facility direccionará las peticiones según las claves de ObjectGrid. Por ejemplo, necesita que ObjectGrid maneje un gran número de objetos que no caben en una ObjectGrid de JVM. Puede utilizar Partitioning Facility para cargar los datos en diferentes servidores con el método partitionLoadEvent como carga previa y el direccionamiento basado en contexto de Partitioning Facility le encontrará la ObjectGrid correcta.

El ejemplo crea un conjunto de particiones basadas en hash y contextos de direccionamiento de clúster de particiones:

- Puede particionar y correlacionar claves ObjectGrid con las particiones WPF con una estrategia de muchos a muchos.
- Las particiones WPF se pueden alojar en el clúster WebSphere Application Server en una estrategia de muchos a muchos.

El diagrama siguiente muestra los valores típicos y la configuración del ejemplo ObjectGridPartitionCluster:



En el diagrama anterior, la máquina M1 y la máquina M2 se utilizan para desplegar el ejemplo ObjectGridPartitionCluster. Cada máquina física puede alojar uno o más WebSphere Application Server. Por ejemplo, la máquina M1 aloja dos servidores de aplicaciones: el servidor de aplicaciones S1 y el servidor de aplicaciones S2. La máquina M2 aloja un servidor, que es el servidor de aplicaciones S3. Cada servidor tiene una instancia de ObjectGrid: una instancia de OGI1ObjectGrid para el servidor de aplicaciones S1, la instancia OGI2 de ObjectGrid para el servidor de aplicaciones S2 y la instancia OGI3 de ObjectGrid para el servidor de aplicaciones S3.

Cada servidor de aplicaciones puede alojar muchas particiones. Por ejemplo, el servidor S1 aloja la partición P1 y la partición P2 y el servidor S3 aloja la partición P1200 y la partición P777.

Cada partición puede alojar muchas claves ObjectGrid. Por ejemplo, la partición P1 aloja las claves OGK1, OGK2 y OGK3 de ObjectGrid y la partición P800 aloja las particiones OGK92, OGK96, OGK98 y OGK9999.

Todas las peticiones de actualización, inserción y supresión de ObjectGrid se direccionan según las claves de ObjectGrid. Tiene dos opciones para recuperar objetos: desde cualquier servidor en una estrategia gestionada de carga de trabajo, o desde la partición de servidor concreta para esta clave.

Instalación y ejecución de la aplicación de ejemplo ObjectGridPartitionCluster

Utilice esta tarea para instalar y ejecutar la aplicación de ejemplo ObjectGridPartitionCluster para comprobar las funciones entre ObjectGrid y Partitioning Facility.

Instale WebSphere Extended Deployment. Consulte la pagina de biblioteca de WebSphere Extended Deployment para obtener las instrucciones.

Un entorno adecuado para ejecutar el ejemplo ObjectGridPartitionCluster es instalar WebSphere Extended Deployment en dos máquinas físicas o crear dos nodos y federarlos con el gestor de despliegue.

1. Para mostrar correctamente las características de este ejemplo, configure un clúster que tenga tres o más miembros del clúster.
2. Instale el archivo `D_ObjectGridPartitionClusterSample.ear`. El archivo `D_ObjectGridPartitionClusterSample.ear` desplegado por Partitioning Facility, WPF, está listo para su instalación y ejecución. Si modifica el código fuente de ejemplo, siga las instrucciones de creación y despliegue de wpf para crear y desplegar el archivo EAR (Enterprise Archive).

   La forma común de instalar los archivos EAR de aplicación es utilizar la consola administrativa. Siga el procedimiento de instalación de la aplicación de empresa para instalar el archivo `D_ObjectGridPartitionClusterSample.ear`. Para acceder a esta parte de la consola administrativa, pulse **Aplicaciones > Instalar nueva aplicación**. No despliegue el archivo EAR durante la instalación. Utilice los valores por omisión, con la excepción del paso en el que se le solicita que seleccione una ubicación de instalación. En este paso, seleccione el clúster que ha definido, en lugar del servidor por omisión, server1.
3. Ejecute el cliente ObjectGridPartitionClusterSample.
   a. Inicie el clúster. En la consola administrativa, pulse **Servidores > Clústeres**. Seleccione el clúster y pulse **Inicio**.
   b. Ejecute el mandato de script **WAS_INSTALL_ROOT\bin\wpfadmin balance**. Compruebe que la partición tiene un estado activo utilizando el mandato **WAS_INSTALL_ROOT\bin\wpfadmin list**. Para obtener más información acerca del script wpfadmin y sus mandatos, consulte la guía de Partitioning Facility en el centro de información de WebSphere Extended Deployment.
   c. Para ejecutar el cliente ObjectGridPartitionClusterSample, ejecute el mandato siguiente:

```
WAS_INSTALL_ROOT/bin/launchClient.bat|sh \
WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear \
-CCBootstrapPort=PORT
```

Donde PORT es el puerto RMI del servidor que puede encontrar en el archivo SystemOut.log del servidor después de iniciar el servidor. Generalmente, este valor del puerto es uno de los valores siguientes: 9810, 9811, 9812.

Por ejemplo, puede ejecutar el mandato siguiente:

```
WAS_INSTALL_ROOT/bin/launchClient.bat|sh
WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear
  -CCBootstrapPort=9811
```

Para un uso más avanzado de este script, consulte “Opciones de cliente de aplicación ObjectGridPartitionClusterSample” en la página 310.

4. Cambie el número de particiones. Cambie el número de particiones que crea el enterprise bean de sesión ObjectGridPartitionCluster: el número de particiones que crea el bean de sesión PFClusterObjectGridEJB se decide mediante la variable de entrada de entorno NumberOfPartitions que se encuentra en el archivo META-INF\ejb-jar.xml. El valor por omisión es 10. Puede cambiar el valor de esta variable de entorno y volver a instalar la aplicación para crear un número de particiones diferente. Establezca el número de particiones en un número inferior a 999999.
5. Cambie las opciones del receptor distribuido. Puede cambiar las siguientes opciones del receptor distribuido ObjectGrid:

Tabla 17. Opciones del receptor distribuido

Nombre de la variable	Descripción
enableDistribution,	Puede habilitar un receptor distribuido ObjectGrid con la variable de entrada de entorno enableDistribution que se encuentra en el descriptor de despliegue EJB. El valor por omisión es true, el cual está habilitado. Establezca el valor en false para desactivar el receptor distribuido.
propagationMode	Puede cambiar la modalidad de propagación con la variable de entrada de entorno propagationMode que está en el descriptor de despliegue EJB. El valor por omisión es update. Puede cambiar el valor por invalidate si no desea el valor por omisión.
propagationVersionOption,	Puede cambiar la opción de versión de propagación con la variable de entrada de entorno propagationVersionOption que se encuentra en el descriptor de despliegue EJB. El valor por omisión es enable. Puede establecer el valor en disable.
compressionMode	Puede cambiar la modalidad de compresión con la variable de entrada de entorno compressionMode que se encuentra en el descriptor de despliegue EJB. El valor por omisión es enable. Puede establecer el valor en disable.

El valor por omisión es propagar las actualizaciones con la comprobación de versión. Es posible que desee establecer el valor en la modalidad invalidate sin la comprobación de versión.

Ha instalado y ejecutado la aplicación de ejemplo ObjectGridPartitionCluster.

Opciones de cliente de aplicación ObjectGridPartitionClusterSample

Utilice estas opciones para un uso avanzado de la ejecución del archivo D_ObjectGridPartitionClusterSample.ear.

Uso avanzado de ejemplo

Consulte “Instalación y ejecución de la aplicación de ejemplo ObjectGridPartitionCluster” en la página 308 para obtener más información acerca de cómo instalar y ejecutar el archivo D_ObjectGridPartitionClusterSample.ear.

Para un uso avanzado del ejemplo, consulte la siguiente guía completa sobre su uso:

```
WAS_INSTALL_ROOT/bin/launchClient.bat|sh
WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear
-CCproviderURL=corbaloc::NOMBRE_SIS_PPAL:PUERTO_RMI_SERVIDOR [-loop BUCLE] [-threads
NÚMERO_DE_HEBRAS] [-add NÚMERO_DE_STOCKS_POR_PARTICIÓN] [-waitForPropagation
SEGUNDOS_DE_ESPERA_PARA_PROPAGACIÓN] [-getIteration
NÚMERO_DE_ITERACIONES_POR_CLAVEOG]
```

Cumplimente las variables siguientes:

- NOMBRE_SIS_PPAL: especifica el nombre de sistema principal del servidor de aplicaciones que se está ejecutando.
- PUERTO_RMI_SERVIDOR: especifica el puerto de la rutina de carga del servidor de aplicaciones.
- BUCLE: especifica el número de bucles que ejecuta el cliente. Este parámetro es opcional. El valor por omisión es 1.
- NÚMERO_DE_HEBRAS: especifica el número de hebras que ejecuta el cliente. Este parámetro es opcional. El valor por omisión es 1.
- NÚMERO_DE_STOCKS_POR_PARTICIÓN: especifica el número de stocks que ha de añadir cada partición. Este parámetro es opcional. El valor por omisión es 3.
- SEGUNDOS_DE_ESPERA_PARA_PROPAGACIÓN: especifica el número de segundos que ha de esperar para que los objetos de ObjectGrid que se acaban de añadir o actualizar se propaguen a los demás servidores. El valor por omisión es de 2 segundos.
- NÚMERO_DE_ITERACIONES_POR_CLAVEOG: especifica el número de iteraciones para recuperar objetos en ObjectGrid de un modo gestionado por la carga de trabajo. El valor por omisión es 6. Cuando se especifican más iteraciones, se muestra un patrón claro de los objetos de la misma clave en los diferentes servidores WebSphere Application Server.

Salida de ejemplo

La salida de este mandato es similar al ejemplo siguiente:

```
C:\dev\xd6\bin>launchClient
D_ObjectGridPartitionClusterSample.ear -CCBootstrapPort=9812
IBM WebSphere Application Server, Release 6.0
J2EE Application Client Tool
Copyright IBM Corp., 1997-2004
WSCL0012I: Procesando argumentos de línea de mandatos.
WSCL0013I: Inicializando J2EE Application Client Environment.
WSCL0035I: La inicialización de J2EE Application Client Environment ha finalizado.
WSCL0014I: Invocando la clase de Cliente de aplicaciones
class com.ibm.websphere.samples.objectgrid.partitionclust
er.client.PartitionObjectGrid
```

```
ObjectGrid Partition Sample has 10 partitions
PARTITION: ObjectGridHashPartition000007->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000003->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000005->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000010->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000006->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000009->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000008->clusterdevNode01/s1
PARTITION: ObjectGridHashPartition000002->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000001->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000004->clusterdevNode01/s2
********** Partition=ObjectGridHashPartition000004**********
------ObjectGrid Operations: Stock Ticket=Stock000104 --------
get on partition for ticket: Stock000104->clusterdevNode02/s2
update: Stock000104->clusterdevNode02/s2
sleep 2 seconds.....
Iteration 1 : Stock000104->clusterdevNode01/s2
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 2 : Stock000104->clusterdevNode01/s1
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 3 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 4 : Stock000104->clusterdevNode01/s2
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 5 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 6 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
------ObjectGrid Operations: Stock Ticket=Stock000114 --------
get on partition for ticket: Stock000114->clusterdevNode01/s2
update: Stock000114->clusterdevNode02/s2
sleep 2 seconds.....
Iteration 1 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 2 : Stock000114->clusterdevNode01/s2
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 3 : Stock000114->clusterdevNode01/s1
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 4 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 5 : Stock000114->clusterdevNode01/s2
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 6 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
------ObjectGrid Operations: Stock Ticket=Stock000124 --------
get on partition for ticket: Stock000124->clusterdevNode02/s2
update: Stock000124->clusterdevNode01/s2
sleep 2 seconds.....
Iteration 1 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 2 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 3 : Stock000124->clusterdevNode01/s2
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 4 : Stock000124->clusterdevNode01/s1
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 5 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 6 : Stock000124->clusterdevNode01/s2
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
C:\dev\xd6\bin>
```

Creación de una aplicación de ObjectGrid y Partitioning Facility integrada

Abra, modifique e instale la aplicación de ejemplo de partición de ObjectGrid.

Utilice estos pasos para modificar, exportar e instalar el archivo ObjectGridPartitionSample.ear en un entorno de WebSphere Extended Deployment. Si no desea realizar cambios en el archivo de ejemplo, puede utilizar el archivo D_ObjectGridPartitionClusterSample.ear desplegado y habilitado para Partitioning Facility, WPF. Si utiliza el archivo D_ObjectGridPartitionClusterSample.ear, puede instalar y ejecutar el archivo sin realizar los pasos siguientes. Los dos archivos EAR (Enterprise Archive) están en el directorio WAS_INSTALL_ROOT/installableApps.

1. Configure el archivo ObjectGridPartitionSample.ear en el entorno de creación como, por ejemplo, IBM Rational Application Developer Versión 6.0.x o Application Server Toolkit Versión 6.0.x. Para obtener más información, consulte "Guía de iniciación para la creación de una aplicación de ObjectGrid y Partitioning Facility".
2. Modifique el código fuente del ejemplo.
3. Exporte la aplicación ObjectGridPartitionClusterSample desde el entorno de creación como un archivo EAR. Para obtener más información, consulte "Exportación del archivo ObjectGridPartitionClusterSample.ear en IBM Rational Application Developer" en la página 314.
4. Despliegue la aplicación, de modo que pueda funcionar conjuntamente con Partitioning Facility. Para obtener más información, consulte "Despliegue el archivo ObjectGridPartitionClusterSample.ear para trabajar con Partitioning Facility" en la página 314.
5. Instale el archivo ObjectGridPartitionClusterSample.ear en WebSphere Extended Deployment. La forma común de instalar archivos EAR de la aplicación es utilizar la consola administrativa WebSphere Application Server. Siga el procedimiento de instalación de aplicaciones de empresa de la consola administrativa para instalar el archivo D_ObjectGridPartitionClusterSample.ear. No despliegue el archivo durante la instalación, utilice el valor por omisión. Utilice el valor por omisión para cada paso, excepto cuando se le solicite que seleccione dónde debe instalarlo. En este paso, seleccione el clúster que ha definido, del servidor server1 por omisión.

Ha instalado el archivo ObjectGridPartitionClusterSample.ear en un entorno de WebSphere Extended Deployment.

Para obtener más información acerca de la programación con ObjectGrid, Partitioning Facility y las aplicaciones de ejemplo, consulte "Ejemplo: Programación de ObjectGrid y Partitioning Facility" en la página 315.

Guía de iniciación para la creación de una aplicación de ObjectGrid y Partitioning Facility

Utilice Application Server Toolkit Versión 6.0.x o IBM Rational Application Developer Versión 6.0.x para volver a crear la aplicación de ejemplo.

El archivo ObjectGridPartitionClusterSample.ear del directorio WAS_INSTALL_ROOT/installableApps contiene todo el código fuente. Puede utilizar Application Server Toolkit Versión 6.0.x o IBM Rational Application Developer Versión 6.0.x para volver a crear esta aplicación de ejemplo. Esta tarea utiliza Rational Application Developer como un ejemplo para establecer el entorno de creación para el archivo ObjectGridPartitionClusterSample.ear. También puede utilizar Application Server Toolkit; una herramienta de ensamblaje gratuita que se incluye con WebSphere Application Server en un CD diferente.

El archivo EAR (Enterprise Archive) desplegado y habilitado para WPF, el archivo `D_ObjectGridPartitionClusterSample.ear` que también está en el directorio `WAS_INSTALL_ROOT/installableApps`, está preparado para su instalación y ejecución.

1. Importe el archivo `ObjectGridPartitionClusterSample.ear` a Rational Application Developer.
   a. Inicie Rational Application Developer.
   b. **Opcional:** Abra la perspectiva J2EE (Java 2 Platform, Enterprise Edition) para trabajar con proyectos J2EE. Pulse **Ventana > Abrir perspectiva > Otros > J2EE**.
   c. **Opcional:** Abra la vista del Explorador de proyectos. Pulse **Ventana > Mostrar vista > Explorador de proyectos**. Otra vista útil es la vista del navegador: **Ventana > Mostrar vista > Navegador**.
   d. Importe el archivo `ObjectGridPartitionClusterSample.ear`. Pulse **Archivo > Importar > Archivo EAR**, a continuación, pulse **Siguiente**.
   e. Seleccione el archivo `ObjectGridPartitionClusterSample.ear` del directorio `WAS_INSTALL_ROOT/installableApps`.
   f. **Opcional:** Pulse **Nuevo** para abrir el asistente de tiempo de ejecución del nuevo servidor y siga las instrucciones.
   g. En el campo de servidor de destino, seleccione el tipo de tiempo de ejecución de servidor **WebSphere Application Server V6.0**.
   h. Pulse **Finalizar**.

   Los proyectos ObjectGridPartitionClusterSample, ObjectGridPartitionClusterSampleEJB y ObjectGridPartitionClusterSampleClient se deben crear y deben visualizarse en la vista del Explorador de proyectos.
2. Configure el proyecto ObjectGridPartitionClusterSampleEJB.
   a. En la vista del Explorador de proyectos de la perspectiva J2EE, pulse con el botón derecho del ratón el proyecto **ObjectGridPartitionClusterSampleEJB** de los proyectos EJB y seleccione **Propiedades**. Se visualizará la ventana Propiedades.
   b. Pulse **Vía de acceso de construcción java** en el panel de la izquierda, pulse la pestaña **Bibliotecas** en el panel de la derecha y pulse **Añadir variable**. Se muestra la ventana Nueva entrada de variable classpath.
   c. Pulse **Configurar variables** para abrir la ventana Preferencia.
   d. Pulse **Nuevo** para abrir la ventana Nueva entrada de variable.
   e. Escriba `ObjectGridPartitionCluster_JAR` como **Nombre** y pulse **Archivo** para abrir la ventana Selección de archivos JAR.
   f. Examine el directorio `WAS_INSTALL_ROOT/lib` y seleccione **wsobjectgrid.jar**. Pulse **Abrir** para cerrar la ventana Selección de archivos JAR.
   g. Pulse **Aceptar** para cerrar la ventana Nueva entrada de variable. La variable ObjectGridPartitionCluster_JAR se visualiza en la lista de variables classpath.
   h. Pulse **Aceptar** para cerrar la ventana Preferencia.
   i. Seleccione la variable **ObjectGridPartitionCluster_JAR** en la lista de variables y pulse **Aceptar** para cerrar la ventana Nueva entrada de variable classpath. La variable ObjectGridPartitionCluster_JAR se visualiza en el panel Bibliotecas.
   j. Repita este procedimiento para añadir el archivo `wpf.jar` al entorno.
   k. Compruebe que el archivo `wpf.jar` y el archivo `wsobjectgrid.jar` estén en la vía de acceso de clases de creación.

Una vez configurado el entorno de creación, puede modificar el código fuente y aplicar otros cambios. Para obtener más información, consulte “Creación de una aplicación de ObjectGrid y Partitioning Facility integrada” en la página 311.

Exportación del archivo ObjectGridPartitionClusterSample.ear en IBM Rational Application Developer

Una vez modificado el archivo de ejemplo, puede exportar la aplicación ObjectGridPartitionClusterSample para crear un archivo EAR (Enterprise Archive) que puede instalar en servidores WebSphere Extended Deployment.

Debe haber importado el archivo `ObjectGridPartitionSample.ear` a las herramientas de desarrollo, de modo que pueda realizar cambios en el fuente. Para obtener más información, consulte “Guía de iniciación para la creación de una aplicación de ObjectGrid y Partitioning Facility” en la página 312. Antes de exportarla, efectúe los cambios en la aplicación de ejemplo.

Puede exportar el archivo `ObjectGridPartitionClusterSample.ear` desde el proyecto ObjectGridPartitionClusterSample de las aplicaciones de empresa de IBM Rational Application Developer. Puede instalar el archivo `ObjectGridPartitionClusterSample.ear` exportado en cualquier servidor WebSphere Extended Deployment Versión 6.0 después de desplegar Partitioning Facility.

1. En la vista del Explorador de proyectos de la perspectiva J2EE (Java 2 Platform, Enterprise Edition), pulse con el botón derecho del ratón la aplicación **ObjectGridPartitionClusterSample** que se encuentra bajo las aplicaciones de empresa. Pulse **Exportar > Archivo EAR**. Se muestra la ventana Exportar.
2. Pulse **Examinar** para abrir la ventana **Guardar como**. Localice el directorio de salida de destino, especifique el nombre de archivo como `ObjectGridPartitionClusterSample` y pulse **Guardar**.
3. Pulse **Examinar** para abrir la ventana Guardar como. Localice el directorio de salida de destino y especifique el nombre de archivo como `ObjectGridPartitionClusterSample`. Pulse **Guardar**.

Se creará el archivo `ObjectGridPartitionClusterSample.ear` en el directorio de salida de destino especificado.

Después de desplegar el archivo `ObjectGridPartitionClusterSample.ear` para Partitioning Facility, WPF, puede ejecutar el archivo en WebSphere Extended Deployment. Para obtener más información, consulte “Despliegue el archivo ObjectGridPartitionClusterSample.ear para trabajar con Partitioning Facility”.

Despliegue el archivo ObjectGridPartitionClusterSample.ear para trabajar con Partitioning Facility

Si piensa instalar el archivo `ObjectGridPartitionClusterSample.ear` en WebSphere Extended Deployment, debe realizar una operación wpf-deploy en el archivo.

Debe tener un archivo `ObjectGridPartitionClusterSample.ear` existente. Para modificar el archivo existente, consulte “Creación de una aplicación de ObjectGrid y Partitioning Facility integrada” en la página 311.

Efectúe la operación wpf-deploy para preparar el archivo EAR (Enterprise Archive) en un entorno de WebSphere Extended Deployment.

1. Cree un directorio `DEST_DIR`.
2. Copie el archivo `ObjectGridPartitionClusterSample.ear` en el directorio `DEST_DIR`. Cambie el nombre del archivo

ObjectGridPartitionClusterSample.ear por el archivo old_ObjectGridPartitionClusterSample.ear.

3. Ejecute el mandato siguiente, donde WORKING_DIR es el directorio de trabajo para la herramienta ejbdeloy, por ejemplo, el directorio c:\temp.

```
WAS_HOME\bin\ejbdeploy.bat|ejbdeploy.sh
DEST_DIR\old_ObjectGridPartitionClusterSample.ear WORKING_DIR
DEST_DIR\ObjectGridPartitionClusterSample.ear
```

4. Ejecute el mandato siguiente, en el que TEMP_DIR es un directorio temporal para la herramienta. Si se especifica el argumento -keep, no se suprimirán los directorios temporales que se crean mediante el programa de utilidad wpfStubUtil.

```
WAS_HOME\bin\wpfStubUtil.cmd|wpfStubUtil.sh
DEST_DIR\ObjectGridPartitionClusterSample.ear
 ObjectGridPartitionClusterSampleEJB.jar com/ibm/websphere/samples/
objectgrid/partitioncluster/ejb/PFClusterObjectGridEJB.class
TEMP_DIR [-stubDebug|-keep]
```

El archivo ObjectGridPartitionClusterSample.ear está preparado para ejecutarse en un entorno de WebSphere Extended Deployment. El archivo de ejemplo D_ObjectGridPartitionClusterSample.ear que se incluye con WebSphere Extended Deployment ya se ha desplegado. No es necesario que despliegue este archivo antes de instalar la aplicación si no ha modificado el código fuente.

Instale el archivo ObjectGridPartitionClusterSample.ear en el entorno de WebSphere Extended Deployment con la consola de administración. Para obtener más información, consulte “Creación de una aplicación de ObjectGrid y Partitioning Facility integrada” en la página 311.

Ejemplo: Programación de ObjectGrid y Partitioning Facility

Este ejemplo muestra cómo utilizar las funciones combinadas de ObjectGrid y Partitioning Facility en un entorno Java 2 Platform, Enterprise Edition en WebSphere Extended Deployment.

Finalidad

Además de demostrar las funciones ObjectGrid y Partitioning Facility, WPF, combinadas, en este ejemplo se muestra también la propagación del receptor distribuido de ObjectGrid y su invalidación.

Las peticiones de actualización, inserción y supresión de objetos se dirigen a los servidores específicos en los que se alojan las particiones para las claves de ObjectGrid correspondientes. Las peticiones del método get de ObjectGrid se gestionan como carga de trabajo entre todos los servidores.

En este ejemplo también se muestra cómo se realiza una partición de una ObjectGrid grande en ObjectGrids más pequeñas y cómo utilizar el método partitionLoadEvent para la carga previa de datos, de modo que la ObjectGrid particionada puede alojar un número ilimitado de objetos.

Visión general

El archivo ObjectGridPartitionClusterSampler.ear crea un objeto stock que ilustra cómo funcionan juntos ObjectGrid y Partitioning Facility. El objeto stock contiene las propiedades siguientes:

- ticket

- company
- serialNumber
- description
- lastTransaction
- price

Donde la propiedad lastTransaction es la hora en la que se ha modificado el stock. Utilice la propiedad lastTransaction para indicar si son recientes los objetos de ObjectGrid de las máquinas virtuales Java (JVM).

En el ejemplo, se ha creado la instancia de ObjectGrid en el método setContext de EJB (Enterprise JavaBeans) con la clase ObjectGridFactory.

Defina un conjunto de particiones basadas en hash. El valor por omisión es de 10 particiones, pero puede modificar el número de particiones. Puede aplicar hash a los tickets de stock en estas particiones utilizando el archivo `SampleUtility.java`. Cada partición puede alojar muchos pares de clave y valor de ObjectGrid.

El ejemplo demuestra cómo las peticiones de inserción, actualización y supresión de ObjectGrid se direccionan a un servidor particionado específico y cómo las peticiones del método get de ObjectGrid se direccionan a un servidor determinado para su clase o a cualquier servidor de un clúster. El ejemplo compara los valores de objeto de una clave de servidores diferentes después de que se haya modificado el valor debido a una operación de actualización, inserción o supresión de esta clave en un servidor determinado.

Ubicación

Utilice este ejemplo en un entorno de clúster en el que cada servidor pueda alojar muchas particiones y en el que cada partición puede alojar muchos objetos con claves diferentes.

Existen dos archivos de ejemplo de clúster de partición de ObjectGrid en el directorio `<raíz_instalación>\installableApps\`:

- El archivo `ObjectGridPartitionClusterSampler.ear` contiene el código fuente. Para ver el origen, amplíe el archivo EAR en el sistema de archivos o importe el origen a un entorno de desarrollo. Para obtener más información, consulte “Creación de una aplicación de ObjectGrid y Partitioning Facility integrada” en la página 311.
- El archivo `D_ObjectGridPartitionClusterSample.ear` ya se ha desplegado para Partitioning Facility. Lea el archivo readme y las instrucciones para que este archivo se ejecute rápidamente.

Explicación

Los apartados siguientes incluyen descripciones sobre la aplicación de ejemplo de clúster de partición de ObjectGrid:

- “Carga previa de la partición de ObjectGrid” en la página 325

Interfaz EJB de operación de ObjectGrid

En este artículo se muestra la interfaz EJB (Enterprise JavaBeans) de operación de ObjectGrid que efectúa operaciones get, get desde el servidor particionado, server, insert, update, y remove.

Finalidad

La interfaz de operación de ObjectGrid efectúa operaciones get, get desde el servidor particionado, insert, update y remove. El método get desde el servidor particionado se direcciona a una partición que corresponde a la clave que solicita. El método get se direcciona mediante una estrategia gestionada de carga de trabajo a cualquier servidor.

La interfaz PFClusterObjectGridEJB

El contenido de la interfaz PFClusterObjectGridEJB es el siguiente:

```
/**
* Interfaz remota para Enterprise Bean: PFClusterObjectGridEJB
*/
public interface PFClusterObjectGridEJB extends javax.ejb.EJBObject {
public String PARTITION_PREFIX = "ObjectGridHashPartition";
/**
 * Obtener todas las particiones
*
 * @return Matriz de series
 * @throws java.rmi.RemoteException
*/
 public String [] getAllPartitions() throws java.rmi.RemoteException;
/**
 * Obtener dónde está alojada la partición
*
* @param partition
* @return String
 * @throws java.rmi.RemoteException
*/
 public String getServer(String partition)
 throws java.rmi.RemoteException;
/**
 * Obtener el objeto Stock y la información de su servidor
 * (ServerIDResult) para un ticket de tipo stock
 * desde cualquier servidor de un clúster (que tenga gestión de carga de trabajo)
*
* @param ticket
* @return
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult getStock(String ticket)
 throws java.rmi.RemoteException;
/**
 * Obtener el objeto stock y la información de su servidor particionado
 * para un ticket de tipo stock
 * desde la partición desde la que se aplica hash a esta clave de ticket
*
* @param ticket
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult getStockOnPartitionedServer(String ticket)
 throws java.rmi.RemoteException;
/**
 * Actualizar stock en un servidor determinado en el que la partición esté
```

```
 * activa para esta clave de ticket de stock.
*
* @param stock
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult updateStock(Stock stock)
 throws java.rmi.RemoteException;
/**
 * Suprimir el stock en un servidor determinado en el que la partición está
 * activa para esta clave de ticket de stock.
*
* @param ticket
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult removeStock(String ticket)
 throws java.rmi.RemoteException;
/**
 * Insertar stock en un servidor determinado en el que la partición está
 * activa para esta clave de ticket de stock.
*
* @param stock
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult insertStock(Stock stock)
 throws java.rmi.RemoteException;
/**
 * Recuperar datos de todos los servidores y comparar los valores
*
 * @param server
 * @return ServerObjectGridVerification
 * @throws java.rmi.RemoteException
*/
 public ServerObjectGridVerification verifyObjectGrid(String server)
 throws java.rmi.RemoteException;
}
```

Clase PartitionKey

La clase PartitionKey controla el comportamiento del direccionamiento basado en contexto de Partitioning Facility.

El código siguiente ilustra la clase de clave de partición de ejemplo. Cuando el método devuelve `not null`, se direcciona con el direccionador de Partitioning Facility, WPF. Cuando el método devuelve `null`, se envía al direccionador de gestión de carga de trabajo, WLM.

```
/**
* PartitionKey para el bean de sesión sin estado particionado WPFKeyBasedPartition
*
*/
public class PFClusterObjectGridEJB_PartitionKey {
/**
 * Número de particiones
*
 * El valor por omisión es 10.
*
*/
 static int numOfPartitions=10;
/**
 * Sólo una vez para getPartitionNumbers
*/
 static boolean getNumOfPartitions=true;
/**
 * Obtener el número de particiones
```

```
*
*/
 static void getPartitionNumbers(){
 //obtener sólo una vez
  if (getNumOfPartitions){
try {
    InitialContext ic = new InitialContext();
    PFClusterObjectGridEJBHome home =
    (PFClusterObjectGridEJBHome) PortableRemoteObject.narrow(
ic.lookup("java:comp/env/ejb/PFClusterObjectGridEJB"),
    PFClusterObjectGridEJBHome.class);
    final PFClusterObjectGridEJB session = home.create();
    String[] PARTITIONS = session.getAllPartitions();
numOfPartitions=PARTITIONS.length;
getNumOfPartitions=false;
}
   catch (ClassCastException e) {
    e.printStackTrace();
numOfPartitions=10;
}
   catch (RemoteException e) {
   e.printStackTrace();
numOfPartitions=10;
}
   catch (NamingException e) {
   e.printStackTrace();
numOfPartitions=10;
}
   catch (CreateException e) {
   e.printStackTrace();
numOfPartitions=10;
}
}
}
/**
* Devolver la clave de la partición
*
* @param partition
* @return String
*/
public static String getStock(String key) {
 return null;
}
/**
* Devolver la clave de la partición
*
* @param key
* @return String
*/
public static String getServer(String key) {
 return key;
}
/**
* Recuperar los datos de ObjectGrid de un servidor particionado en el que
* se modifican datos (la calidad e integridad más elevada).
*
* @param ticket
* @return hashcode de ticket de stock
*/
public static String getStockOnPartitionedServer(String ticket) {
if (ticket==null){
 return null;
}
getPartitionNumbers();
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
```

```
* Devolver la clave de la partición
*
* @param stock
* @return hashcode de ticket de stock
*/
public static String updateStock(Stock stock) {
getPartitionNumbers();
 String ticket=null;
 if (stock!=null){
  ticket=stock.getTicket();
}
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Devolver la clave de la partición
*
* @param stock
* @return hashcode de ticket de stock
*/
public static String insertStock(Stock stock) {
getPartitionNumbers();
 String ticket=null;
 if (stock!=null){
  ticket=stock.getTicket();
}
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Devolver la clave de la partición
*
 * @param server
* @return String
*/
public static String verifyObjectGrid(String server) {
 return server;
}
/**
* Devolver la clave de la partición
*
* @param stock
* @return hashcode de ticket de stock
*/
public static String removeStock(String ticket) {
if (ticket==null){
 return null;
}
getPartitionNumbers();
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Devolver la clave de la partición
*
* @param partition
* @return
*/
public static String getAllPartitions() {
 return null;
}
}
```

Cada método remoto debe tener un método correspondiente que devuelva una serie válida o un valor nulo.

Clase SampleUtility y correlación de particiones

Utilice el archivo `SampleUtility.java` para manipular claves, tickets de tipo stock, hash y particiones. También puede utilizar este archivo para correlacionar las

claves de ObjectGrid con particiones. Puede desarrollar una clase de programa de utilidad para correlacionar las claves de ObjectGrid con particiones que se ajusten a las necesidades de su empresa. Para utilizar Partitioning Facility con ObjectGrid, debe correlacionar claves diferentes en particiones diferentes.

Clase SampleUtility

La siguiente es la clase del programa de utilidad para el ejemplo ObjectGridPartitionCluster:

```
/**
* Clase del programa de utilidad para el ejemplo ObjectGridPartitionCluster
*
*
*/
public class SampleUtility {
/**
 * Contenedor para el registro de particiones.
*/
 static Map serverPartitions= new HashMap();
/**
 * Prefijo del nombre de la partición
*/
public static String PARTITION_PREFIX = "ObjectGridHashPartition";
/**
 * Prefijo del nombre de stock
*/
public static String STOCK_PREFIX="Stock";
/**
 * Recuperar el número de partición del nombre de la partición
*
* @param partition
 * @return int
*/
 public static int getIntFromPartition(String partition){
  int result=-1;
  int pre=PARTITION_PREFIX.length();
  int p=partition.length();
  String num=partition.substring(pre, p);
  result=Integer.parseInt(num);
  return result;
}
/**
 * Recuperar el número de la partición del ticket de tipo stock
*
* @param ticket
* @return
*/
 public static int getIntFromStockTicket(String ticket){
  int result=-1;
  int pre=STOCK_PREFIX.length();
  int p=ticket.length();
  String num=ticket.substring(pre, p);
  result=Integer.parseInt(num);
  return result;
}
/**
 * Aplicar hash al ticket de tipo stock en base a un hash determinado.
*
* @param ticket
 * @param base
 * @return int
*/
 public static int hashTicket(String ticket, int base){
  if (base<1){
   return 0;
```

```
}
  int hash=0;
  int num=getIntFromStockTicket(ticket);
  hash= num % base;
  return hash;
}
/**
 * Aplicar hash a la clave de stock para una partición
*
* @param ticket
 * @param base
 * @return String - nombre de la partición
*/
 public static String hashStockKeyToPartition(String ticket, int base){
  String p=null;
  int hashcode=hashTicket(ticket, base)+1;
p=PARTITION_PREFIX+ padZeroToString(hashcode+"", 6);
  return p;
}
/**
 * Registrar el servidor/partición
*
 * @param server
* @param partition
*/
 public static void addServer(String server, String partition){
  serverPartitions.put(server, partition);
}
/**
 * Eliminar servidor/partición
*
 * @param server
*/
 public static void removeServer(String server){
  serverPartitions.remove(server);
}
/**
 * Obtener todos los servidores en los que están activas las particiones.
*
 * @return Iterator - String
*/
 public static Iterator getAllServer(){
  return serverPartitions.values().iterator();
}
}
```

Debe utilizar la misma base global de hash y analizar las variables a las que aplicar hash según la base de hash. Tenga en cuenta el ejemplo siguiente:

```
myKey.hashCode % hashBase
```

Debe analizar myKey como una variable de hash y debe conservar la misma base de hash entre diferentes servidores. En el ejemplo anterior, se busca la misma variable desde el entorno Java. No puede utilizar `key1 % 100`, pero puede utilizar `key2 % 90`.

Creación de ObjectGrid en el método setContext de enterprise bean

Cree la instancia de ObjectGrid en el método setContext de enterprise bean como en el archivo `PFClusterObjectGridEJBBean.java` y recupere los datos de carga previa.

```
/**
* setSessionContext
*
* con la instancia de ObjectGrid
```

```
*/
public void setSessionContext(javax.ejb.SessionContext ctx) {
 mySessionCtx = ctx;
try {
  InitialContext ic = new InitialContext();
//obtener PartitionManager
  ivManager = (PartitionManager)
ic.lookup("java:comp/websphere/wpf/PartitionManager");
  // obtener la configuración de enableDistribution
  boolean enableDistribution = ((Boolean)
ic.lookup("java:comp/env/enableDistribution")).booleanValue();
System.out.println("***** enableDistribution="+ enableDistribution);
  // obtener la configuración de propagationMode
String propagationMode = (String) ic.lookup("java:comp/env/propagationMode");
System.out.println("***** pMode="+ propagationMode);
  String pMode=null;
  if (propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_DEFAULT_KEY)||
   propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_INVALID_KEY) ){
pMode=propagationMode;
}
  // obtener la configuración de propagationVersionOption
  String propagationVersionOption = (String)
ic.lookup("java:comp/env/propagationVersionOption");
System.out.println("***** pVersionOption="+ propagationVersionOption);
  String pVersion=null;
  if (propagationVersionOption.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_VERS_KEY)||
   propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_NOVERS_KEY) ){
pVersion=propagationVersionOption;
}
  // obtener la configuración de compressionMode
String compressionMode = (String) ic.lookup("java:comp/env/compressionMode");
System.out.println("***** compressMode="+ compressionMode);
  String compressMode=null;
  if (compressionMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_COMPRESS_DISABLED)||
   propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_COMPRESS_ENABLED) ){
compressMode=compressionMode;
}
  // si se ha habilitado la carga previa
  bPreload = ((Boolean)
ic.lookup("java:comp/env/preload")).booleanValue();
System.out.println("***** enablePreload="+ bPreload);
  //si se ha habilitado la supresión
  bRemove = ((Boolean)
ic.lookup("java:comp/env/remove")).booleanValue();
System.out.println("***** enableRemove="+ bRemove);
  // si se ha habilitado el cargador
  boolean bLoader = ((Boolean)
ic.lookup("java:comp/env/loader")).booleanValue();
System.out.println("***** enableLoader="+ bLoader);
  // obtener la vía de acceso y el nombre de archivo
  String filePathandName = (String)
ic.lookup("java:comp/env/filePathandName");
System.out.println("***** fileName="+ filePathandName);
  //obtener la instancia de ObjectGrid
  og=ObjectGridFactory.getObjectGrid(ogName,
enableDistribution,  pMode, pVersion,
  compressMode, bLoader,
  filePathandName);
  if (og==null){
   throw new RuntimeException
("La instancia de ObjectGrid es nula en ObjectGridPartitionClusterSample");
```

```
}
System.out.println("Contexto de Bean, getObjectGrid=" + og + " para name="+ ogName);
  if (bPreload && !lock){
System.out.println("Carga previa de datos");
   PersistentStore store=PersistentStore.getStore(filePathandName);
   store.preload(10);
   store.verify(10);
lock=true;
   preloadData=store.getAllRecords();
}
}
catch (Exception e) {
  logger.logp(Level.SEVERE, CLASS_NAME,
"setSessionContext", "Excepción: " + e);
  throw new EJBException(e);
}
}
```

Clase ObjectGridFactory de singleton

Se crea una instancia de ObjectGrid con una fábrica personalizada que guarda en antememoria la instancia de ObjectGrid con valores personalizados.

El siguiente es un ejemplo de cómo crear una instancia de ObjectGrid mediante programación, cómo establecer el objeto ObjectGridTransformer, configurar el receptor de sucesos de propagación y establecer este receptor en la instancia de ObjectGrid. También puede utilizar un archivo XML para realizar esta configuración.

```
/**
*
* Crear una instancia de ObjectGrid y configurarla.
*
*
*/
public class ObjectGridFactory {
/**
 * Nombre de ObjectGrid
*/
static String ogName="WPFObjectGridSample";
/**
 * Instancia de ObjectGrid
*/
 static ObjectGrid og=null;
/**
 * Sesión de ObjectGrid
*/
 static Session ogSession=null;
/**
 * Nombre de la correlación
*/
static String mapName="SampleStocks";
/**
 * Antememoria de ObjectGrid
*/
 static Map ogCache= new HashMap();
/**
 * Obtener la instancia de ObjectGrid
*
 * @param ogn
 * @param enableDist
 * @param pMode
 * @param pVersion
 * @param compressMode
* @return
*/
public static synchronized ObjectGrid getObjectGrid(String ogn,
 boolean enableDist,
```

```
 String pMode,
 String pVersion,
 String compressMode,
 boolean loader,
 String fileName){
  if (ogn!=null){
ogName=ogn;
}
else {
throw new IllegalArgumentException ("el nombre de ObjectGrid indicado es nulo");
}
  if (ogCache.containsKey(ogName)){
   return (ObjectGrid) ogCache.get(ogName);
}
try {
   ObjectGridManager manager= ObjectGridManagerFactory.
getObjectGridManager();
og=manager.createObjectGrid(ogName);
if (enableDist){
    TranPropListener tpl=new TranPropListener();
     if (pMode!=null){
      tpl.setPropagateMode(pMode);
}
     if (pVersion!=null){
       tpl.setPropagateVersionOption(pVersion);
}
     if (compressMode!=null) {
      tpl.setCompressionMode(compressMode);
}
     og.addEventListener(tpl);
}
   // Definir BackingMap y establecer el cargador
  BackingMap bm = og.defineMap(mapName);
ObjectTransformer myTransformer=
  new MyStockObjectTransformer();
  bm.setObjectTransformer(myTransformer);
OptimisticCallback myOptimisticCallback=
  new MyStockOptimisticCallback();
  if (loader){
   TransactionCallback tcb=new MyTransactionCallback();
   Loader myLoader= new MyCacheLoader(fileName, mapName);
   og.setTransactionCallback(tcb);
   bm.setLoader(myLoader);
}
  og.initialize();
  ogCache.put(ogName, og);
}
catch (Exception e) {
}
 return og;
}
}
```

Carga previa de la partición de ObjectGrid

En este tema se describe cómo realizar la carga previa de una instancia de ObjectGrid.

Utilice el método partitionLoadEvent para cargar objetos que estén relacionados con esta partición sólo cuando se activa la partición. Cuando se cargan objetos al activar la partición, se particiona ObjectGrid de modo que ObjectGrid pueda manejar un número elevado de objetos.

```
/**
* Se llama cuando se asigna una partición específica a este proceso de servidor.
* @param partitionName
* @return
```

```
*/
public boolean partitionLoadEvent(String partitionName) {
//carga previa de datos
 preloadDataForPartition(partitionName);
logger.logp(
Level.FINER,
CLASS_NAME,
"partitionLoadEvent",
"Loading "+ partitionName );
  return true;
}
/**
*
* carga previa de datos
*
* @param partition
*/
private synchronized void preloadDataForPartition(String partition){
 if (bPreload && (preloadData!=null)){
  Iterator itr=preloadData.keySet().iterator();
  while (itr.hasNext()){
   String ticket= (String) itr.next();
   String p=SampleUtility.
   hashStockKeyToPartition(ticket, numOfPartitions);
   if (partition.equals(p)){
    Stock stock= (Stock) preloadData.get(ticket);
System.out.println("carga previa en la partición=" +
partition + " con ticket de datos="+ ticket);
    insertStock(stock);
}
}
}
}
```

Es posible que tenga que inhabilitar las actualizaciones distribuidas si utiliza la carga previa particionada de su ObjectGrid de gran tamaño para particionarlo. La versión actual de las actualizaciones distribuidas no se puede particionar. El direccionamiento basado en contexto de PartitioningFacility, WPF, busca los datos correctos en la partición correcta.

Configuración de ObjectGrid para trabajar con beans gestionados por contenedor

Con WebSphere Application Server Versión 6.0.2 y posteriores, puede utilizar los beans de CMP (persistencia gestionada por contenedor) con un producto con antememoria externa.

Utilice esta tarea para usar los beans de CMP y beneficiarse de ObjectGrid como antememoria externa en lugar de una antememoria integrada. El motor de permanencia de WebSphere Application Server es el que facilita esta función.

1. Defina los argumentos de JVM para definir el adaptador CacheFactoryManager y la ubicación del archivo de configuración XML de ObjectGrid. CacheFactoryManager es un adaptador entre el motor de permanencia y ObjectGrid.
   a. Pulse **Servidores** > **Servidores de aplicaciones** > *nombre_servidor* > **Java y gestión de procesos** > **Definición de proceso** > **Máquina virtual Java** > **Argumentos de JVM genéricos**.
   b. Añada las propiedades siguientes:
      - `-Dcom.ibm.ws.pmcache.manager=com.ibm.ws.objectgrid.adapter.pm.CacheFactoryManager`

- -Dcom.ibm.ws.pmcache.config=file:/d:/temp/objectGrid.xml

  La propiedad -Dcom.ibm.ws.pmcache.config especifica el archivo de configuración para la ObjectGrid. El valor es un URL para el archivo de configuración de ObjectGrid.

2. Configure el archivo de configuración XML de ObjectGrid La configuración se encuentra en el archivo objectGrid.xml. Considere el ejemplo siguiente. Se necesita la información sobre aplicaciones y módulos para la aplicación J2EE (Java 2 Platform, Enterprise Edition). La información se refleja en el archivo objectGrid.xml. Una aplicación Accounts tiene tres Enterprise JavaBeans de CMP: Savings, Checkin y MoneyMarket. Estos Enterprise JavaBeans están contenidos en el módulo PersonalBankingEJB. El nombre de visualización es *Accounts* y *PersonalBankingEJB* es el módulo EJB del descriptor de despliegue de la aplicación. Savings, Checkin y MoneyMarket son los nombres especificados en el elemento ejb-name del descriptor de despliegue de Enterprise Java Bean para los beans de entidad gestionados por contenedor (CMP). A continuación se muestra un fragmento de ejemplo para esta configuración:

   ```
   <ObjectGrids>
   <ObjectGrid name="Accounts">
   <BackingMap name="PersonalBankingEJB.jar#Savings" readOnly="true"
   pluginCollectionRef="default" />
   <BackingMap name="PersonalBankingEJB.jar#Checkin" readOnly="true"
   pluginCollectionRef="default" />
   <BackingMap name="PersonalBankingEJB.jar#MoneyMarket" readOnly="true"
   pluginCollectionRef="default" />
   </ObjectGrid>
   </ObjectGrids>
   ```

   El archivo PersonalBankingEJB.jar se especifica en los distintivos EJB del descriptor de despliegue de aplicaciones, como se muestra en el ejemplo siguiente:

   ```
   <module id="module_1">
   <ejb>PersonalBankingEJB.jar</ejb>
   </module>
   ```

3. Habilite el valor LifeTimeInCache del gestor de persistencia para cada bean en la aplicación que debe utilizar la antememoria externa. ObjectGrid precisa que se habilite este valor del descriptor de despliegue, pero ignora el valor LifeTimeInCache. La configuración de ObjectGrid tiene prioridad.

4. Configure explícitamente una backingMap para desalojar objetos de la antememoria. A continuación, se ofrece un fragmento de código XML del archivo objectGrid.xml:

   ```
   <backingMapPluginCollection id="TotalTimeToLive">
   <bean id="Evictor"
   className="com.ibm.websphere.objectgrid.plugins.builtins.TTLEvictor">
   <property name="pruneSize" type="int" value="2"
   description="establece el tamaño máximo del desalojador de TTL" />
   <property name="numberOfHeaps" type="int" value="1"
   description="establece el número de almacenamientos dinámicos de TTL" />
   <property name="sleepTime" type="int" value="1"
   description="tiempo de inactividad de la hebra de desalojador" />
   </bean>
   </backingMapPluginCollection>
   <backingMapPluginCollection id="LifeTimeInCache">
   <bean id="Evictor"
   className="com.ibm.websphere.objectgrid.plugins.builtins.TTLEvictor">
   <property name="lifeTime" type="int" value="3"
   description="el tiempo de vida de la entrada de correlación es de 3 segundos" />
   <property name="pruneSize" type="int" value="2"
   description="establece el tamaño máximo del desalojador de TTL" />
   ```

```
<property name="numberOfHeaps" type="int" value="1"
description="establece el número de almacenamientos dinámicos de TTL" />
<property name="sleepTime" type="int" value="1"
description="tiempo de inactividad de la hebra de desalojador" />
</bean>
</backingMapPluginCollection>
```

La propiedad lifeTime controla el desalojador.

Consulte el Capítulo 10, “Integración de ObjectGrid con WebSphere Application Server”, en la página 291 para obtener más información acerca de cómo utilizar ObjectGrid con WebSphere Application Server.

Capítulo 11. Procedimientos recomendados para el rendimiento de ObjectGrid

Puede mejorar el rendimiento de una correlación de ObjectGrid con los siguientes procedimientos recomendados. Éstos se implementan sólo en el contexto de la aplicación y de la arquitectura.

Todas las aplicaciones y entornos utilizan una solución distinta para el rendimiento. ObjectGrid proporciona personalizaciones incorporadas para mejorar el rendimiento, pero también se puede mejorar el rendimiento en la arquitectura de la aplicación. Las siguientes áreas ofrecen mejoras de rendimiento:

- "Procedimientos recomendados para el rendimiento del bloqueo"

  Seleccione entre las distintas estrategias de bloqueo que pueden afectar al rendimiento de las aplicaciones.
- "Procedimientos recomendados para el método copyMode" en la página 330

  Seleccione entre las distintas modalidades de copia que se pueden utilizar para cambiar la manera en que ObjectGrid mantiene y copia entradas.
- "Procedimientos recomendados para la interfaz ObjectTransformer" en la página 334

  Utilice la interfaz ObjectTransformer para permitir que la devolución de llamadas a la aplicación proporcione implementaciones personalizadas de operaciones comunes y caras como la serialización de objetos y copias exactas de objetos.
- "Procedimientos recomendados para el rendimiento del plug-in desalojador" en la página 336

  Seleccione entre las estrategias de desalojo LFU (menos frecuentemente utilizado) y LRU (menos recientemente utilizado).
- "Procedimientos recomendados para el desalojador por omisión" en la página 338

  Propiedades del desalojador TTL (tiempo de vida) por omisión, el desalojador por omisión que se crea con cada backingMap.

Procedimientos recomendados para el rendimiento del bloqueo

Las estrategias de bloqueo pueden afectar al rendimiento de la aplicación.

Para obtener más detalles sobre las estrategias de bloqueo siguientes, consulte el tema "Bloqueo" en la página 131.

Estrategia de bloqueo pesimista

Puede utilizar la estrategia de bloqueo pesimista para operaciones de correlación de lectura y grabación donde suele haber conflicto de claves. La estrategia de bloqueo pesimista es la que mayor impacto tiene en el rendimiento.

Estrategia de bloqueo optimista

El bloqueo optimista es la configuración por omisión. Esta estrategia mejora el rendimiento y la escalabilidad frente a la estrategia pesimista. Utilice esta estrategia si las aplicaciones pueden tolerar algunos errores de actualización optimista mientras se sigue obteniendo un mejor resultado que con una estrategia pesimista. Esta estrategia funciona muy bien para aplicaciones de actualización poco frecuente, en su mayor parte, de lectura.

Estrategia de ningún bloqueo

Utilizar la estrategia de ningún bloqueo resulta adecuado para aplicaciones que son de sólo lectura. La estrategia de ningún bloqueo no obtiene ningún bloqueo. Por lo tanto, ofrece el rendimiento y la escalabilidad de mayor simultaneidad.

Procedimientos recomendados para el método copyMode

ObjectGrid hace una copia del valor basándose en la configuración de CopyMode. Puede utilizar el método setCopyMode(CopyMode, valueInterfaceClass) de la API de 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.

Si una aplicación utiliza la interfaz ObjectMap para obtener una referencia a un valor de correlación, se recomienda utilizar únicamente esa referencia dentro de la transacción de ObjectGrid que obtuvo la referencia. El uso de la referencia en otra transacción de ObjectGrid puede llevar a errores. Por ejemplo, si utiliza la estrategia de bloqueo pesimista con BackingMap, una llamada al método get o getForUpdate adquirirá un bloqueo S (shared, compartido) o U (update, actualización), respectivamente. El método get devolverá la referencia al valor y cuando se complete la transacción se liberará el bloqueo obtenido. Para bloquear la entrada de correlación de otra transacción, la transacción debe llamar al método get o getForUpdate. Cada transacción debe obtener su propia referencia al valor, para ello, se llamará al método get o getForUpdate en lugar de reutilizar la misma referencia al valor en varias transacciones.

Utilice la siguiente información para seleccionar entre las modalidades de copia con esta información:

Modalidad COPY_ON_READ_AND_COMMIT

Esta modalidad es la que toma por omisión. Cuando se utiliza esta modalidad se omite el argumento valueInterfaceClass. Esta modalidad garantiza que una aplicación no contendrá una referencia al objeto de valor que se encuentra en BackingMap y, en su lugar, la aplicación siempre trabajará con una copia del valor que se encuentra en BackingMap. La modalidad COPY_ON_READ_AND_COMMIT garantiza que la aplicación nunca puede dañar inadvertidamente los datos que se almacenan en antememoria de la BackingMap. Cuando una transacción de aplicación llama a un método ObjectMap.get con una clave determinada de argumento y es el primer acceso de la entrada de ObjectMap para esa clave, se devuelve una copia del valor. Cuando se compromete una transacción, los cambios comprometidos por la aplicación se copian a la BackingMap para garantizar que la aplicación no tiene una referencia al valor comprometido en BackingMap.

Modalidad COPY_ON_READ

La modalidad COPY_ON_READ mejora el rendimiento respecto a la modalidad COPY_ON_READ_AND_COMMIT, dado que elimina la copia que se produce cuando se compromete una transacción. Cuando se utiliza esta modalidad se omite el argumento valueInterfaceClass. Para conservar la integridad de los datos de BackingMap, la aplicación garantiza que todas las referencias que haya para una entrada se destruirán después de que se comprometa la transacción. Con esta modalidad, el método ObjectMap.get devuelve una copia del valor en lugar de una referencia a éste para garantizar que los cambios realizados por la aplicación en el valor no influyan en el valor de BackingMap hasta que no se comprometa la

transacción. No obstante, cuando se compromete la transacción, no se realiza una copia de los cambios. Si no que se almacena en BackingMap la copia devuelta por el método ObjectMap.get. La aplicación destruirá todas las referencias de entrada de correlación después de que se comprometa la transacción. Si la aplicación no puede destruirlos, podrían dañarse los datos almacenados en la antememoria de BackingMap. Si una aplicación utiliza esta modalidad y tiene problemas, cambie a la modalidad COPY_ON_READ_AND_COMMIT para comprobar si existe todavía el problema. Si desaparece el problema, entonces el error se debía a que la aplicación no ha podido destruir todas las referencias después de comprometerse la transacción.

Modalidad COPY_ON_WRITE

La modalidad COPY_ON_WRITE mejora el rendimiento con respecto a la modalidad COPY_ON_READ_AND_COMMIT, que elimina la copia que se produce la primera vez que una transacción llama al método ObjectMap.get con una clave determinada. El método ObjectMap.get devuelve el valor al proxy en lugar de una referencia directa al objeto de valor. El proxy garantiza que no se realiza ninguna copia del valor, a no ser que la aplicación llame a un método set en la interfaz del valor que se especifica en el argumento valueInterfaceClass. El proxy proporciona una implementación de una *COPY_ON_WRITE*. Cuando se compromete una transacción, BackingMap examina el proxy para determinar si se ha realizado alguna copia como consecuencia de una llamada al método set. Si se ha efectuado una copia, se almacena en BackingMap la referencia a esa copia. La gran ventaja de esta modalidad es que nunca se copiará un valor en una lectura o en un compromiso si la transacción nunca llama al método set para cambiar el valor.

Las modalidades COPY_ON_READ_AND_COMMIT y COPY_ON_READ realizan una copia exacta cuando se recupera un valor de la ObjectMap. Si la aplicación sólo actualiza algunos de los valores que se recuperan de una transacción, entonces esta modalidad no es óptima. La modalidad COPY_ON_WRITE admite este comportamiento de un modo eficaz pero requiere que la aplicación utilice un patrón sencillo. Son necesarios los objetos de valor para admitir una interfaz. La aplicación debe utilizar los métodos de esta interfaz cuando interactúa con el valor dentro de una sesión de ObjectGrid. Si este es el caso, entonces la ObjectGrid crea los proxy para los valores que se devuelven a la aplicación. El proxy tiene una referencia de lo que es un valor real. Si simplemente no se lee la aplicación, siempre se ejecuta con la copia real. Si la aplicación modifica un atributo en el objeto, el proxy hace una copia del objeto real y realiza la modificación en la copia. El proxy utiliza entonces la copia de ahí en adelante. Esto permite que la aplicación evite completamente la operación de copia para los objetos que son de sólo lectura. Todas las operaciones de modificación deben comenzar con el prefijo set. Los Enterprise JavaBeans se suelen codificar para utilizar este estilo de denominación de métodos para los métodos que modifican atributos de objetos. Debe seguirse esta convención. Todos los objetos modificados se copian en el momento en que los modifica la aplicación. Este es el caso de ejemplo de lectura y grabación más eficaz admitido por ObjectGrid. Puede configurar una correlación para que utilice COPY_ON_WRITE como se detalla a continuación. En este ejemplo, en la aplicación se pretende almacenar objetos Person con clave utilizando el nombre de la correlación. El objeto Person tiene un código similar al del 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 actúa conjuntamente con valores que se recuperan de un ObjectMap. Modifique el objeto para que utilice 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 tiene entonces que configurar la BackingMap con el fin de utilizar la modalidad COPY_ON_WRITE, como en este ejemplo:

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// utilizar COPY_ON_WRITE para esta correlación con
// IPerson como la clase valueProxyInfo
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// La aplicación utilizará entonces el siguiente
// patrón cuando utilice la correlación PERSON.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// la aplicación convierte el valor devuelto en IPerson y no en Person
IPerson p = (IPerson)person.get("Billy");
p.setAge(p.getAge()+1);
...
// crear una nueva Person y agregar 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 FUNCIONA. Causará una ClassCastException
sess.begin();
// el error está en que se utiliza Person en lugar de
// IPerson
Person a = (Person)person.get("Bobby");
sess.commit();
```

En el primer apartado se muestra que la aplicación recupera un valor que se llamaba Billy en la correlación. La aplicación convierte el valor devuelto al objeto IPerson y 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 convertir el proxy a dos tipos. La última parte del fragmento de código anterior muestra que no se permite en la modalidad COPY_ON_WRITE. La aplicación recupera el registro de Bobby e intenta convertirlo a un objeto Person. Esta acción produce un error con una excepción de conversión de clase porque el proxy que se devuelve no es un objeto Person. El proxy devuelto implementa el objeto IPerson y ValueProxyInfo.

Interfaz ValueProxyInfo y soporte de actualización parcial

Esta interfaz permite que una aplicación recupere el valor de sólo lectura comprometido al que hace referencia el proxy o el conjunto de atributos que se han 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 representan los atributos que la aplicación ha modificado durante esta transacción. El caso de uso principal de ibmGetDirtyAttributes es en un cargador basado en conectividad JDBC (Java Database Connectivity) o basado en CMP. Sólo es necesario actualizar los atributos que aparecen en la lista, ya sea en la sentencia SQL o en el objeto correlacionado con la tabla, con lo que el cargador generará código SQL más eficaz. Cuando se compromete una transacción CopyOnWrite y si el cargador está conectado, éste puede convertir los valores de los objetos modificados a la interfaz ValueProxyInfo para obtener esta información.

Control del método equals cuando se utiliza la modalidad COPY_ON_WRITE o los proxy.

Por ejemplo, el código siguiente crea un objeto Person y lo inserta en una ObjectMap. A continuación, recupera el mismo objeto con el método ObjectMap.get. El valor se convierte a la interfaz. Si el valor se convierte 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 produce un error cuando se utiliza la operación == porque no pueden ser el mismo objeto.

```
session.begin();
// operación new del objeto Person
Person p = new Person(...);
personMap.insert(p.getName, p);
// recuperarlo de nuevo, recordar utilizar la interfaz para la conversión
IPerson p2 = personMap.get(p.getName());
if(p2 == p)
{
// son iguales
}
```

```
else
{
// no lo son
}
```

Otra consideración es cuando debe alterar temporalmente el método equals. Como se ilustra en el siguiente fragmento de código, el método equals debe verificar que el argumento es un objeto que implementa la interfaz IPerson y convierte el argumento para que sea una interfaz IPerson. Dado que el argumento podría ser un proxy que implementa la interfaz IPerson, debe utilizar los métodos getAge y getName cuando realiza comparaciones de igualdad de variables.

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

Modalidad NO_COPY

La modalidad NO_COPY permite que una aplicación garantice que nunca modificará un objeto de valor que se obtenga con un método `ObjectMap.get` a cambio por mejoras de rendimiento. Cuando se utiliza esta modalidad se omite el argumento valueInterfaceClass. Si se utiliza esta modalidad, nunca se realiza una copia del valor. Si la aplicación modifica valores, se dañarán los datos de BackingMap. La modalidad NO_COPY se utiliza fundamentalmente para correlaciones de sólo lectura donde la aplicación nunca modifica los datos. Si la aplicación utiliza esta modalidad y tiene problemas, cambie a la modalidad COPY_ON_READ_AND_COMMIT para comprobar si aún existe el problema. Si ya no aparece el problema, entonces la aplicación modifica el valor devuelto por el método ObjectMap.get, durante la transacción o después de que se ha comprometido la transacción.

Procedimientos recomendados para la interfaz ObjectTransformer

ObjectTransformer utiliza la devolución de llamadas a la aplicación para proporcionar implementaciones personalizadas de operaciones comunes y caras como la serialización y la copia exacta de objetos.

Para obtener detalles específicos sobre la interfaz ObjectTransformer, consulte el tema “Plug-in ObjectTransformer” en la página 214. Desde un punto de vista de rendimiento y teniendo en cuenta la información del método CopyMode del tema “Procedimientos recomendados para el método copyMode” en la página 330, resulta claro que la ObjectGrid copia los valores en todos los casos excepto en la modalidad NO_COPY. El mecanismo de copia por omisión que se utiliza dentro de la ObjectGrid es la serialización, que se sabe que es una operación cara. En esta situación se puede utilizar la interfaz ObjectTransformer. La interfaz ObjectTransformer utiliza devoluciones de llamada a la aplicación para proporcionar una implementación personalizada de operaciones comunes y caras como la serialización de objetos y copias exactas de objetos.

Una aplicación puede proporcionar una implementación de la interfaz ObjectTransformer a una correlación. La ObjectGrid delega entonces en los

métodos de este objeto y confía en la aplicación para proporcionar una versión optimizada de todos los métodos de la interfaz. A continuación se detalla la interfaz ObjectTransformer:

```
public interface ObjectTransformer
{
 void serializeKey(Object key, ObjectOutputStream stream)
 throws IOException;
 void serializeValue(Object value, ObjectOutputStream stream)
 throws IOException;
 Object inflateKey(ObjectInputStream stream)
 throws IOException, ClassNotFoundException;
 Object inflateValue(ObjectInputStream stream)
 throws IOException, ClassNotFoundException;
 Object copyValue(Object value);
 Object copyKey(Object key);
}
```

Puede asociar una interfaz ObjectTransformer a una BackingMap mediante este 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

La serialización de objetos suele ser el mayor problema de rendimiento con ObjectGrid. Si la aplicación no facilita un plug-in ObjectTransformer, ObjectGrid utiliza el mecanismo serializable por omisión. 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 un 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. Los métodos serialize serializan el objeto y proporcionan una corriente de datos. El método serializa el método en la corriente de datos proporcionada. Los métodos inflate proporcionan la corriente de entrada y esperan que la aplicación cree el objeto, lo infle con datos de la corriente de datos 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 recibe un objeto de una ObjectMap, la ObjectGrid realiza una copia exacta del valor de objeto para garantizar que la copia de la correlación BaseMap permanezca a salvo. 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 copiado de nuevo el objeto en la fase de confirmación para crear una copia privada, pero en este caso, el coste del rendimiento de esta acción se ha compensado indicando al programador de la aplicación que no utilice el valor después de que se confirma la transacción. El mecanismo de copia de objeto por omisión 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 la utilización de serialize e inflate supone un problema para la aplicación, proporcione un plug-in ObjectTransformer e

implemente los métodos `copyValue` y `copyKey` mediante una copia de objeto más eficaz.

Procedimientos recomendados para el rendimiento del plug-in desalojador

Si utiliza plug-in desalojadores, no estarán activos hasta que no los cree e indique a la correlación de reserva que los utilice. Utilice estos procedimientos recomendados y sugerencias de rendimiento para los desalojadores LFU (menos frecuentemente utilizado) y LRU (menos recientemente utilizado).

Desalojador LFU (menos frecuentemente utilizado)

El concepto de desalojador LFU es eliminar entradas de la correlación que no se utilizan frecuentemente. Las entradas de la correlación se reparten a lo largo de un conjunto de almacenamientos dinámicos binarios. A medida que aumenta el uso de una entrada de antememoria concreta, pasa a una posición superior en el orden de almacenamiento dinámico. Cuando el desalojador intenta un conjunto de desalojos, sólo elimina las entradas de antememoria que están situadas en orden inferior a un punto específico en el almacenamiento dinámico binario. Como consecuencia, se desalojan las entradas menos frecuentemente utilizadas.

Desalojador LRU (menos recientemente utilizado)

El desalojador LRU sigue los mismos conceptos que el desalojador LFU con unas pequeñas diferencias. La diferencia principal es que LRU utiliza una cola FIFO (el primero en entrar es el primero en salir) en lugar de un conjunto de almacenamientos dinámicos binarios. Cada vez que se accede a una entrada de antememoria, se desplaza a la cabeza de la cola. Como consecuencia, el principio de la cola contiene las entradas de correlación utilizadas con más frecuencia y el final contiene las entradas de correlación menos recientemente utilizadas. Por ejemplo, la entrada de antememoria A se utiliza 50 veces y la entrada de antememoria B se utiliza sólo una vez inmediatamente después de la entrada de antememoria A. En esta situación, la entrada de antememoria B estará al principio de la cola porque se ha utilizado más recientemente y la entrada de antememoria A estará al final de la cola. El desalojador LRU desaloja las entradas de antememoria que se encuentran al final de la cola, que son las entradas de correlación menos recientemente utilizadas.

Propiedades de LFU y LRU y procedimientos recomendados para mejorar el rendimiento

Número de almacenamientos dinámicos

Cuando utiliza el desalojador LFU, todas las entradas de antememoria de una correlación concreta se ordenan a lo largo del número de almacenamientos dinámicos especificado, lo que mejora notablemente el rendimiento e impide que se sincronicen todos los desalojos en un almacenamiento dinámico binario que contiene toda la ordenación de la correlación. Más almacenamientos dinámicos también agilizan el tiempo que se requiere para reordenar los almacenamientos dinámicos porque cada almacenamiento dinámico tiene menos entradas. Establezca el número de almacenamientos dinámicos en el 10% del número de entradas de la BaseMap.

Número de colas

Cuando utiliza el desalojador LRU, todas las entradas de antememoria para

una correlación concreta se ordenan a lo largo del número de colas LRU especificadas, lo que mejora notablemente el rendimiento e impide que se sincronicen todos los desalojos en una cola que contiene toda la ordenación de la correlación. Establezca el número de colas en el 10% del número de entradas de la BaseMap.

Propiedad MaxSize

Cuando un desalojador LFU o LRU empieza a desalojar entradas, utiliza la propiedad MaxSize del desalojador para determinar cuántos almacenamientos dinámicos binarios o elementos de cola LRU se van a desalojar. Por ejemplo, supongamos que establece el número de almacenamientos dinámicos o colas para que haya 10 entradas de correlación en cada cola de correlación. Si la propiedad MaxSize se establece en 7, el desalojador desalojará 3 entradas de cada almacenamiento dinámico u objeto de cola para proporcionar de nuevo un tamaño de almacenamiento dinámico de 7. El desalojador sólo desaloja entradas de correlación de un almacenamiento dinámico o cola cuando el almacenamiento dinámico o cola contiene más elementos que el valor indicado en la propiedad MaxSize. Establezca el MaxSize en el 70% del tamaño de almacenamiento dinámico o cola. Para este ejemplo, se establece el valor en 7. Puede obtener un tamaño aproximado de cada almacenamiento intermedio o cola dividiendo el número de entradas de BaseMap por el número de almacenamientos dinámicos o colas que se utilizan.

Propiedad SleepTime

Un desalojador no elimina constantemente entradas de la correlación. Si no que permanece inactivo durante un período de tiempo establecido y sólo se activa cada número *n* de segundos, donde *n* se refiere a la propiedad SleepTime. Esta propiedad también afecta positivamente al rendimiento: si se ejecuta muy a menudo un barrido de desalojo disminuirá el rendimiento debido a los recursos que son necesarios para procesarlos. No obstante, si no se utiliza el desalojador lo suficiente puede que se quede con una correlación de entradas innecesarias. Una correlación llena de entradas innecesarias puede afectar negativamente a los requisitos de memoria y al proceso de recursos que se requieren para la correlación. Para la mayoría de las correlaciones, se recomienda establecer los barridos de desalojo en quince segundos. Si se graba frecuentemente en la correlación y se utiliza a una alta velocidad de transacción, quizá sea de más utilidad establecer el valor en un tiempo menor. No obstante, si se accede con muy poca frecuencia a la correlación, puede establecer el tiempo en un valor mayor.

Ejemplo

En el ejemplo siguiente se define una correlación, se crea un nuevo desalojador LFU, se establecen las propiedades del desalojador y se establece la correlación para que utilice el desalojador:

```
//Utilizar ObjectGridManager para crear u obtener la ObjectGrid. Consulte el
// apartado de ObjectGridManager
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");
//Establecer propiedades suponiendo 50.000 entradas de correlación
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

El uso del desalojador LRU es muy similar al del desalojador LFU. A continuación figura un ejemplo:

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
//Establecer propiedades suponiendo 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 2 líneas distintas con respecto al ejemplo de desalojador LFU.

Procedimientos recomendados para el desalojador por omisión

Procedimientos recomendados para el desalojador por omisión basado en el *tiempo de vida*.

Además de los plug-in desalojadores que se describen en el tema “Procedimientos recomendados para el rendimiento del plug-in desalojador” en la página 336, se crea un desalojador TTL con cada correlación de reserva. El desalojador por omisión elimina entradas basándose en un concepto de *tiempo de vida*. El atributo ttlType define este comportamiento. Existen tres atributos ttlType:

- **Ninguna** (NONE): especifica que nunca caducan las entradas 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 hayan creado.
- **Hora de último acceso** (LAST_ACCESED_TIME): especifica que las entradas se desalojan en función de cuándo se haya accedido por última vez a ellas.

Propiedades del desalojador por omisión y procedimientos recomendados para el rendimiento

Propiedad TimeToLive

Esta propiedad, junto con la propiedad ttlType, es la más crucial desde una perspectiva de rendimiento. Si utiliza el ttlType CREATION_TIME, el desalojador desaloja las entradas cuando la hora de creación es igual al valor del atributo TimeToLive. Si establece el valor del atributo TimeToLive en 10 segundos, todo en la correlación completa se desaloja transcurridos diez segundos. Es importante tener precaución a la hora de establecer este valor para el ttlType CREATION_TIME. Se recomienda utilizar este desalojador cuando existen razonablemente altas cantidades de agregaciones a la antememoria que sólo se utilizan durante un tiempo establecido. Con esta estrategia, todo lo que se crea se eliminará cuando haya transcurrido el tiempo establecido.

A continuación figura un ejemplo de dónde resulta de utilidad un tipo TTL de CREATION_TIME. Supongamos que utiliza una aplicación Web que obtiene acciones bursátiles y no es crítico obtener las acciones más recientes. En este caso, las acciones bursátiles se almacenan en la antememoria de una ObjectGrid durante 20 minutos. Transcurridos 20 minutos, caducan las entradas de correlación de ObjectGrid y se desalojan. Cada veinte minutos más o menos, la correlación de ObjectGrid utiliza el plug-in del cargador para renovar los datos de correlación de la base de datos. La base de datos se actualiza cada 20 minutos con las acciones

bursátiles más recientes. Por lo tanto, para esta aplicación resulta ideal utilizar un valor de TimeToLive de 20 minutos.

Si utiliza el atributo de ttlType LAST_ACCESSED_TIME, establezca TimeToLive en un número inferior que si utilizara el ttlType CREATION_TIME, porque cada vez que se accede al atributo TimeToLive éste se restablece. En otras palabras, si el atributo TimeToLive es igual a 15 y ha existido una entrada durante 14 segundos pero entonces se accede a ésta, no caducará de nuevo hasta que no hayan transcurrido otros 15 segundos. Si establece TimeToLive en un número relativamente elevado, quizá muchas entradas nunca se desalojen. No obstante, si establece el valor en algo similar a 15 segundos, podrían eliminarse entradas cuando no se accede a éstas a menudo.

A continuación figura un ejemplo de dónde resulta de utilidad un tipo TTL de LAST_TIME. Se utiliza una correlación de ObjectGrid para retener datos de sesión de un cliente. Los datos de la sesión se deben destruir si el cliente no los utiliza durante un período de tiempo. Por ejemplo, los datos de la sesión caducan después de 30 minutos de no actividad en el cliente. En este caso, lo que se necesita exactamente para esta aplicación es utilizar un tipo TTL de LAST_ACCESSED_TIME con el atributo TimeToLive establecido en 30 minutos.

Ejemplo

En el ejemplo siguiente se crea una correlación de reserva, se establece su atributo ttlType de desalojador por omisión y se establece la propiedad TimeToLive.

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
bMap.setTtlEvictorType(TTLType.LAST_ACCESSED_TIME);
bMap.setTimeToLive(15);
```

La mayoría de los valores del desalojador deberían establecerse antes de la inicialización de ObjectGrid. Para obtener un conocimiento exhaustivo sobre los desalojadores, consulte “Desalojadores” en la página 192.

Capítulo 12. Distribución de cambios entre máquinas virtuales Java de igual

Los objetos LogSequence y LogElement comunican los cambios que han aparecido en una transacción de ObjectGrid con un plug-in.

Para obtener más información sobre cómo se puede utilizar JMS (Java Message Service) para distribuir cambios transaccionales, consulte “Java Message Service para distribuir cambios de transacción” en la página 344.

Un requisito previo es que la instancia de ObjectGrid debe almacenarla en antememoria ObjectGridManager. Consulte “Métodos createObjectGrid” en la página 93 para obtener más información. El valor booleano de cacheInstance debe establecerse en true.

Los objetos proporcionan un medio para que una aplicación publique fácilmente los cambios que han ocurrido en una infraestructura de objetos mediante un transporte de mensajes a ObjectGrids de igual en máquinas virtuales Java (JVM) remotas y luego aplique esos cambios en esa JVM. La clase LogSequenceTransformer es fundamental para habilitar este soporte. En este artículo se examina cómo se escribe un receptor con un sistema de mensajería JMS (Java Message Service) para propagar los mensajes.

ObjectGrid admite la transmisión de LogSequences que se producen de un compromiso de transacción de ObjectGrid entre miembros de clúster de WebSphere Application Server con un plug-in proporcionado por IBM. Por omisión, no se habilita esta función, pero se puede configurar para que esté operativa. No obstante, si el consumidor o el productor no es WebSphere Application Server, quizá sea necesario utilizar un sistema de mensajería JMS externo.

1. Inicialice el plug-in. La ObjectGrid llama al método `initialize` del plug-in, parte del contrato de la interfaz ObjectGridEventListener, cuando se inicia la ObjectGrid. El método `initialize` debe obtener sus recursos JMS, incluidas las conexiones, las sesiones y los editores e iniciar la hebra que es el receptor JMS. El método initialize es similar al del ejemplo siguiente:

```
public void initialize(Session session)
{
 mySession = session;
 myGrid = session.getObjectGrid();
try
{
  if(mode == null)
{
throw new ObjectGridRuntimeException("No se ha especificado ninguna modalidad");
}
  if(userid != null)
{
   connection = topicConnectionFactory.createTopicConnection(
   userid, password);
}
else
   connection = topicConnectionFactory.createTopicConnection();
  // es necesario iniciar la conexión para recibir mensajes.
  connection.start();
  // la sesión jms no es transaccional (false).
  jmsSession = connection.createTopicSession(false,
  javax.jms.Session.AUTO_ACKNOWLEDGE);
  if(topic == null)
   if(topicName == null)
```

```
{
throw new ObjectGridRuntimeException("No se ha especificado ningún tema");
}
else
{
    topic = jmsSession.createTopic(topicName);
}
  publisher = jmsSession.createPublisher(topic);
// iniciar la hebra de receptor.
listenerRunning = true;
  listenerThread = new Thread(this);
  listenerThread.start();
}
 catch(Throwable e)
{
throw new ObjectGridRuntimeException("No se puede inicializar", e);
}
}
```

En el código para iniciar la hebra se utiliza una hebra J2SE (Java 2 Platform, Standard Edition). Si ejecuta un servidor WebSphere Application Server Versión 6.x o WebSphere Application Server Versión 5.x Enterprise, utilice la API (interfaz de programas de aplicación) de beans asíncronos para iniciar esta hebra del daemon. También puede utilizar las API comunes. A continuación figura un fragmento de código de sustitución de ejemplo que muestra la misma acción con un gestor de trabajo:

```
// iniciar la hebra de receptor.
listenerRunning = true;
workManager.startWork(this, true);
```

El plug-in debe implementar también la interfaz Work en lugar de la interfaz Runnable. También tendrá que añadir un método release para establecer la variable listenerRunning en false. Debe proporcionarse el plug-in con una instancia de WorkManager en su constructor o mediante inyección si se utiliza un contenedor de IoC (Inversión de control).

2. Transmitir los cambios. A continuación figura un ejemplo del método transactionEnd para publicar los cambios locales que se realizan en una ObjectGrid. Se utiliza JMS aunque, desde luego, puede utilizar cualquier transporte de mensajes que facilite una mensajería de publicación y suscripción fiable.

```
// Este método está sincronizado para asegurarse de que
// los mensajes se publican en el orden en que la transacción
// se ha comprometido. Si se empezaron a publicar los mensajes
// en paralelo, los receptores podrían dañar la correlación
// dado que podrían llegar supresiones antes que inserciones, etc.
public synchronized void transactionEnd(String txid,
boolean isWriteThroughEnabled,
boolean committed, Collection changes)
{
try
{
  // debe habilitarse WriteThrough y comprometerse.
  if(isWriteThroughEnabled && committed)
{
   // grabar las secuencia en un byte []
   ByteArrayOutputStream bos = new ByteArrayOutputStream();
   ObjectOutputStream oos = new ObjectOutputStream(bos);
   if (publishMaps.isEmpty()) {
    // serializar la colección completa
    LogSequenceTransformer.serialize(changes, oos, this, mode);
}
else {
```

```
    // filtrar LogSequences basándose en el contenido de publishMaps
    Collection publishChanges = new ArrayList();
    Iterator iter = changes.iterator();
    while (iter.hasNext()) {
     LogSequence ls = (LogSequence) iter.next();
     if (publishMaps.contains(ls.getMapName())) {
     publishChanges.add(ls);
}
}
   LogSequenceTransformer.serialize(publishChanges, oos, this,
mode);
}
  // crear un mensaje de objeto para los cambios
  oos.flush();
  ObjectMessage om = jmsSession.createObjectMessage(
  bos.toByteArray());
  // establecer propiedades
  om.setStringProperty(PROP_TX, txid);
  om.setStringProperty(PROP_GRIDNAME, myGrid.getName());
  // transmitirlo.
  publisher.publish(om);
}
}
 catch(Throwable e)
{
throw new ObjectGridRuntimeException("No se pueden enviar los cambios", e);
}
}
```

Este método utiliza varias variables de instancias:

- Variable **jmsSession**: sesión JMS que se utiliza para publicar mensajes. Se crea cuando se inicializa el plug-in.
- Variable **mode**: la modalidad de distribución.
- Variable **publishMaps**: un conjunto que contiene los nombres de todas las correlaciones con cambios para publicar. Si la variable está vacía, se publicarán todas las correlaciones.
- Variable **publisher**: objeto TopicPublisher que se crea durante el método initialize del plug-in.

3. Reciba y aplique mensajes de actualización. A continuación figura el método run. Este método se ejecuta en un bucle hasta que la aplicación detiene el bucle. En cada iteración del bucle se intenta recibir un mensaje JMS y aplicarlo a la ObjectGrid.

```
private synchronized boolean isListenerRunning()
{
 return listenerRunning;
}
public void run()
{
try
{
System.out.println("Iniciado el receptor");
  // obtener una sesión jms para recibir los mensajes.
  // No transaccional.
  TopicSession myTopicSession;
  myTopicSession = connection.createTopicSession(false,
  javax.jms.Session.AUTO_ACKNOWLEDGE);
  // obtener un suscriptor para el tema, true indica que no se reciben
  // mensajes transmitidos con editores en esta conexión.
  // De lo contrario, se recibirían las actualizaciones propias.
  TopicSubscriber subscriber = myTopicSession.createSubscriber(topic,
  null, true);
System.out.println("Receptor iniciado");
```

```
  while(isListenerRunning())
{
   ObjectMessage om = (ObjectMessage)subscriber.receive(2000);
   if(om != null)
{
    // Utilizar la sesión que se ha transferido en la inicialización...
    // es muy importante utilizar aquí NoWriteThrough
    mySession.beginNoWriteThrough();
    byte[] raw = (byte[])om.getObject();
    ByteArrayInputStream bis = new ByteArrayInputStream(raw);
    ObjectInputStream ois = new ObjectInputStream(bis);
    // inflar las LogSequences
    Collection collection = LogSequenceTransformer.inflate(ois,
     myGrid);
    Iterator iter = collection.iterator();
    while (iter.hasNext()) {
     // procesar todos los cambios de correlaciones de acuerdo a la modalidad
     // cuando la LogSequence se ha serializado
     LogSequence seq = (LogSequence)iter.next();
     mySession.processLogSequence(seq);
}
    mySession.commit();
   } // si había un mensaje
  } // bucle while
  // detener la conexión
  connection.close();
}
 catch(IOException e)
{
System.out.println("Excepción de E/S: " + e);
}
 catch(JMSException e)
{
System.out.println("Excepción de JMS: " + e);
}
 catch(ObjectGridException e)
{
System.out.println("Excepción de ObjectGrid: " + e);
System.out.println("Causada por: " + e.getCause());
}
 catch(Throwable e)
{
System.out.println("Excepción : " + e);
}
System.out.println("Receptor detenido");
}
```

La clase LogSequenceTransformer y las API: ObjectGridEventListener, LogSequence y LogElement permiten que se utilice cualquier publicación y suscripción fiable para distribuir los cambios y filtrar las correlaciones que desea distribuir. En los fragmentos de código de esta tarea se muestra cómo se utilizan estas API con JMS para generar ObjectGrid de igual a igual que comparten aplicaciones que se alojan en un conjunto de plataformas distintas que comparten un transporte de mensajes común.

Java Message Service para distribuir cambios de transacción

Utilice JMS (Java Message Service) para cambios distribuidos entre distintos niveles o en entornos de plataformas mixtas.

JMS es un protocolo ideal para cambios distribuidos entre distintos niveles o en entornos de plataformas mixtas. Por ejemplo, algunas aplicaciones que utilizan la ObjectGrid se podrían desarrollar en Gluecode o Tomcat, en tanto que otras

aplicaciones podrían ejecutarse en WebSphere Application Server Versión 6.0. JMS es ideal para cambios distribuidos entre iguales de ObjectGrid en estos distintos entornos. El transporte de mensajes de High Availability Manager es muy rápido, pero sólo se pueden distribuir cambios a las JVM que se encuentran en un solo grupo central. JMS es más lento, pero permite que un conjunto de clientes de aplicaciones mayor y más variado comparta una ObjectGrid. JMS es ideal, por ejemplo, cuando se comparten datos de una ObjectGrid entre un cliente Swing plano y una aplicación desplegada en WebSphere Extended Deployment.

Visión general

Se ha implementado JMS para distribuir cambios de transacción mediante un objeto Java que se comporta como un receptor ObjectGridEventListener. Este objeto puede propagar el estado de las cuatro maneras siguientes:

Invalidate
Todas las entradas que se desalojen, actualicen o supriman se eliminarán de todas las máquinas virtuales Java (JVM) de igual cuando reciban el mensaje.

Invalidate conditional
Sólo se desalojará la entrada si la versión local es la versión del editor o anterior a ésta.

Push Todas las entradas que se hayan desalojado, actualizado, suprimido o insertado se agregarán o escribirán encima en todas las JVM de igual cuando reciban el mensaje JMS.

Push conditional
Sólo se actualizará o agregará la entrada en el destinatario si la entrada local es menos reciente que la versión que se está publicando.

Escucha de cambios para publicación

El plug-in implementa la interfaz ObjectGridEventListener para interceptar el evento transactionEnd. Cuando la ObjectGrid invoca a este método, el plug-in intenta convertir la lista LogSequence para cada correlación a la que afecta la transacción en un mensaje JMS y luego lo publica. Se puede configurar el plug-in con el fin de publicar cambios para todas las correlaciones o para un subconjunto de correlaciones. Se procesan los objetos LogSequence para las correlaciones que tienen habilitada la publicación. La clase LogSequenceTransformer de ObjectGrid serializa una LogSequence filtrada para cada correlación a una corriente de datos. Después de que se serializan todas las LogSequences a la corriente de datos se crea un ObjectMessage JMS y se publica a un tema bien conocido.

Escucha de mensajes JMS y aplicación de éstos a la ObjectGrid local

El mismo plug-in también inicia una hebra que recorre un bucle, que recibe todos los mensajes que se publican al tema bien conocido. Cuando llega un mensaje, se pasa su contenido a la clase LogSequenceTransformer para convertirlo a un conjunto de objetos LogSequence. A continuación, se inicia una transacción NoWriteThrough. Se proporcionan todos los objetos LogSequence al método Session.processLogSequence, que actualiza las correlaciones locales con los cambios. El método processLogSequence reconoce el modo de distribución. Se compromete la transacción y ahora la antememoria local refleja los cambios.

Para obtener más información sobre cómo se utiliza JMS para distribuir cambios de transacción, consulte el Capítulo 12, “Distribución de cambios entre máquinas virtuales Java de igual”, en la página 341.

Capítulo 13. Integración de contenedores basados en inyección

Puede utilizar la infraestructura IoC (Inversión de control) para configurar completamente la ObjectGrid. Por consiguiente, no tiene que utilizar la infraestructura de configuración XML de ObjectGrid.

Contenedores basados en inyección

Los contenedores basados en inyección, también conocidos como IoC (Inversión de control), son patrones comunes que utilizan las aplicaciones en el cliente y en el servidor. Existen varias implementaciones origen abiertas de tales contenedores. La nueva especificación Enterprise JavaBeans (EJB) Versión 3.0 también toma prestados algunos de estos conceptos. La mayoría de estas infraestructuras son contenedores de Enterprise JavaBean y aceptan la responsabilidad de crear una instancia de un bean concreto. Estas infraestructuras también inicializan los beans con conjuntos de propiedades y conectan otros Enterprise JavaBeans que requieren utilizando los pares getter y setter en atributos de Enterprise JavaBean. Las API (interfaz de programas de aplicación) de ObjectGrid están diseñadas para funcionar bien con tales contenedores. Comenzando con los métodos ObjectGridManager.createObjectGrid, puede configurar estos contenedores para ejecutar una rutina de carga de una ObjectGrid de modo que la aplicación tenga una referencia a una ObjectGrid en funcionamiento o puede solicitar que el contenedor proporcione una sesión de ObjectGrid cuando sea necesario.

Patrones admitidos

En los apartados siguientes se describen los pasos para verificar que una aplicación que utiliza una infraestructura IoC puede utilizar limpiamente las API de ObjectGrid.

Utilice ObjectGridManager para crear ObjectGrids

Las API de ObjectGrid están diseñadas para funcionar bien con infraestructuras IoC. El singleton raíz utilizado por tal infraestructura es la interfaz ObjectGridManager, que tiene varios métodos de fábrica `createObjectGrid` que devuelven una referencia a una ObjectGrid con nombre. Puede establecer esta referencia a ObjectGrid como un singleton de la infraestructura IoC, de modo que solicitudes posteriores del bean devuelvan la misma instancia de ObjectGrid.

Plug-ins de ObjectGrid

Entre los plug-ins de ObjectGrid están:

- TransactionCallback
- ObjectGridEventListener
- SubjectSource
- MapAuthorization
- SubjectValidation

Cada plug-in es un JavaBean sencillo que implementa una interfaz. Puede utilizar la infraestructura IoC para crear estos plug-ins y conectarlos a los atributos adecuados en la instancia de ObjectGrid. Todos los plug-ins tienen un método set correspondiente en la interfaz ObjectGrid para la integración limpia con la infraestructura IoC.

Crear correlaciones

Se puede utilizar el método de fábrica createMap en la interfaz ObjectGrid para crear una correlación recién denominada. Todos los plug-ins que la BackingMap (el objeto devuelto por createMap) requiere se crean utilizando la infraestructura IoC y luego se conectan a la BackingMap mediante el nombre de atributo adecuado. Dado que ningún otro objeto hace referencia a BackingMap, las infraestructuras IoC no lo crean automáticamente. Como una solución, puede conectar cada BackingMap a un atributo ficticio en la bean de aplicación principal. Utilice el método setMaps() para borrar las BackingMaps que se han definido previamente en esta ObjectGrid y sustitúyalas por la lista de BackingMaps que se proporcionan.

Plug-ins de correlación de respaldo

Los plug-ins de la BackingMap se comportan del mismo modo que los de la ObjectGrid. Cada plug-in tiene un método `set` correspondiente en la interfaz de BackingMap. Los plug-ins de BackingMap son:

- Loader
- ObjectTransformer
- OptimisticCallback
- Evictor
- MapEventListener

Patrones de uso

Cuando la infraestructura IoC tenga configurado el archivo de configuración para producir una ObjectGrid, cree un enterprise bean llamado `gridName_Session` o similar. Defínalo como un Enterprise JavaBean que se obtiene al llamar al método `getSession` en el Enterprise JavaBean del singleton de ObjectGrid. La aplicación utilizará entonces una infraestructura IoC para obtener una referencia a un objeto gridName_Session cada vez que un objeto requiera una nueva sesión.

Resumen

El uso de ObjectGrid en entornos que ya utilizan una infraestructura IoC para la creación y configuración de instancias de bean es directo. Puede utilizar la infraestructura IoC para configurar completamente la ObjectGrid y, por consiguiente, no tiene que utilizar la infraestructura de configuración XML. La ObjectGrid funciona de manera similar con la infraestructura IoC existente.

Capítulo 14. Resolución de problemas

En este apartado se describen casos de ejemplo para solucionar problemas que se producen por errores de la aplicación o por cuestiones de diseño de la aplicación. Si sospecha que ObjectGrid tiene algún defecto, quizá tenga que habilitar el rastreo de ObjectGrid, como se describe en el apartado de rastreo de ObjectGridManager.

Errores recurrentes y desconocidos

Los intentos de una aplicación por mejorar el rendimiento utilizando la modalidad de copia: COPY_ON_READ, COPY_ON_WRITE o NO_COPY se describen en el apartado CopyMode. En la aplicación aparecen problemas recurrentes cuando cambia el síntoma del problema y el problema es desconocido o inesperado. Los problemas recurrentes no se producen cuando se cambia la modalidad de copia a la modalidad COPY_ON_READ_AND_COMMIT.

Problema

El problema podría deberse a datos dañados en la correlación de ObjectGrid, que es una consecuencia de que la aplicación viola el contrato de programación de la modalidad de copia que se está utilizando. Al dañarse los datos se podrían provocar errores imprevisibles de modo recurrente o de una manera desconocida o inesperada.

Solución

La solución es que la aplicación cumpla el contrato de programación establecido para la modalidad de copia que se está utilizando. Para las modalidades de copia COPY_ON_READ y COPY_ON_WRITE, esto significa que la aplicación utiliza una referencia a un objeto de valor fuera del ámbito de la transacción donde se ha obtenido la referencia al valor. Para utilizar estas modalidades, la aplicación debe aceptar destruir la referencia al objeto de valor después de que se completa la transacción y obtener una nueva referencia al objeto de valor en todas las transacciones que tengan que acceder al objeto de valor. Para la modalidad de copia NO_COPY, la aplicación debe aceptar no cambiar nunca el objeto de valor. En este caso, debe modificarse la aplicación de modo que no cambie el objeto de valor o debe utilizar otra modalidad de copia. Consulte el apartado CopyMode para obtener detalles adicionales con respecto al valor de la modalidad de copia.

Técnica general de tratamiento de excepciones

Si se conoce la causa raíz de un objeto Throwable resulta de utilidad aislar el origen de un problema. A continuación figura un ejemplo de un método de programa de utilidad que un controlador de excepciones puede utilizar para encontrar la causa raíz del objeto Throwable que se ha producido.

Ejemplo

```
static public Throwable findRootCause( Throwable t )
{
// Empezar con el Throwable que se ha producido como root.
Throwable root = t;
// Seguir la cadena de cause hasta encontrar el último Throwable de la cadena.
Throwable cause = root.getCause();
while ( cause != null )
{
root = cause;
```

```
cause = root.getCause();
}
// Devolver el último Throwable de la cadena como la cause root.
return root;
}
```

Técnicas específicas para el tratamiento de excepciones

Inserción duplicada

Normalmente, este problema sólo aparecerá en entornos de propagación de transacciones distribuidas. No suele producirse.

Mensaje

```
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl < processLogSequence Exit
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl > commit for:
TX:08FE0C67-0105-4000-E000-1540090A5759 Entry
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl > rollbackPMapChanges for:
TX:08FE0C67-0105-4000-E000-1540090A5759
as result of Throwable: com.ibm.websphere.objectgrid.plugins.
CacheEntryException:
Duplicate key on an insert!
Entry com.ibm.websphere.objectgrid.plugins.CacheEntryException:
Duplicate key on an insert!
at com.ibm.ws.objectgrid.map.BaseMap.applyPMap(BaseMap.java:528)
at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:405)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.commitPropagatedLogSequence
(TranPropWorkerThread.java:553)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.processCommitRequest
(TranPropWorkerThread.java:449)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.run
(TranPropWorkerThread.java:200)
at java.lang.Thread.run(Thread.java:568)
```

Problema

Cuando la secuencia de anotaciones cronológicas filtradas se propaga de una JVM a otra, se procesa la secuencia de anotaciones cronológicas externa en la segunda JVM. Puede que exista la entrada de esta clave o los dos códigos de operación de secuencia de anotaciones cronológicas deberían ser distintos. Este problema se produce de vez en cuando.

Impacto y solución

Cuando se produce este problema, no se actualiza la entrada en otra JVM, lo que puede provocar una incoherencia en ObjectGrid. No obstante, existe un método alternativo para impedir este problema. Puede utilizar WPF (WebSphere Partitioning Facility) en la recuperación de objetos además de insertar, actualizar o eliminar objetos. Consulte el apartado Integración de ObjectGrid y Partitioning Facility para obtener más información sobre esta técnica.

Excepción de colisión optimista

Puede recibir una excepción OptimisticCollisionException directamente o al recibir una excepción ObjectGridException. El código siguiente es un ejemplo de cómo se detecta una excepción y a continuación se muestra su mensaje:

```
try {
...
} catch (ObjectGridException oe) {
System.out.println(oe);
}
```

Causa de la excepción

Las excepciones OptimisticCollisionException se crean en situaciones en las que dos clientes distintos intentan actualizar la misma entrada de correlación relativamente al mismo tiempo. Se compromete una sesión del cliente y se actualiza la entrada de correlación. No obstante, el otro cliente ya ha leído los datos antes del compromiso y contiene datos antiguos o incorrectos. El otro cliente entonces intentará comprometer los datos incorrectos, que es cuando se crea la excepción.

Recuperación de la clave que ha desencadenado la excepción

Podría resultar de utilidad, a la hora de solucionar problemas de tal excepción, recuperar la clave correspondiente a la entrada que ha desencadenado la excepción. La ventaja de la excepción OptimisticCollisionException es que tiene un método getKey incorporado que devuelve el objeto que representa esa clave. A continuación figura un ejemplo de cómo se recupera e imprime la clave cuando se detecta la excepción OptimisticCollisionException:

```
try {
...
} catch (OptimisticCollisionException oce) {
 System.out.println(oce.getKey());
}
```

Una excepción OptimisticCollisionException podría ser la causa de una excepción ObjectGridException. Si este es el caso, puede utilizar el código siguiente para determinar el tipo de excepción e imprimir la clave. En el código siguiente se utiliza el método findRootCause de programa de utilidad como se describe en el apartado Técnica general de tratamiento de excepciones.

```
try {
...
}
catch (ObjectGridException oe) {
 Throwable Root = findRootCause( oe );
 if (Root instanceof OptimisticCollisionException) {
  OptimisticCollisionException oce = (OptimisticCollisionException)Root;
 System.out.println(oce.getKey());
}
}
```

Excepción LockTimeoutException

Mensaje

Puede obtener una excepción LockTimeoutException directamente o al capturar una excepción ObjectGridException. En el fragmento de código siguiente se muestra cómo obtener la excepción y mostrar el mensaje.

```
try {
...
}
catch (ObjectGridException oe) {
System.out.println(oe);
}
```

El resultado es:

```
com.ibm.websphere.objectgrid.plugins.LockTimeoutException: %Message
```

%Message representa la serie que se pasa como parámetro cuando se crea la excepción y las propiedades y métodos de la excepción se utilizan para mostrar un mensaje de error exacto. Lo más probable es que describa el tipo de bloqueo que se solicitó y sobre qué entrada de correlación actuaba la transacción.

Causa de la excepción

Cuando una transacción o cliente solicita que se otorgue un bloqueo para una entrada de correlación concreta, normalmente tendrá que esperar a que el cliente actual libere el bloqueo. Si la solicitud de bloqueo permanece inactiva durante un período prolongado y nunca se otorga un bloqueo, se crea una excepción `LockTimeoutException`. Esto es para impedir un punto muerto, que se describe con más detalle en el apartado siguiente. Es más probable que aparezca esta excepción cuando utiliza una estrategia de bloqueo pesimista porque nunca se libera el bloqueo hasta que no se compromete la transacción.

Más detalles acerca de la solicitud de bloqueo y la excepción

La excepción `LockTimeoutException` tiene un método incorporado denominado getLockRequestQueueDetails que devuelve una serie que contiene una descripción más a fondo de la situación que desencadenó la excepción. A continuación figura un ejemplo de un fragmento de código que almacena en antememoria la excepción y muestra un mensaje de error.

```
try {
...
}
catch (LockTimeoutException lte) {
 System.out.println(lte.getLockRequestQueueDetails());
}
```

El resultado de salida es:

```
lock request queue
->[TX:163C269E-0105-4000-E0D7-5B3B090A571D, state =
 Granted 5348 milli-seconds ago, mode = U]
->[TX:163C2734-0105-4000-E024-5B3B090A571D, state =
 Waiting for 5348 milli-seconds, mode = U]
->[TX:163C328C-0105-4000-E114-5B3B090A571D, state =
 Waiting for 1402 milli-seconds, mode = U]
```

Si obtiene la excepción en un bloqueo de captura `ObjectGridException`, el código siguiente determina la excepción e imprime los detalles de cola. Se utiliza el método findRootCause de programa de utilidad descrito en el apartado Técnica general de tratamiento de excepciones.

```
try {
...
}
catch (ObjectGridException oe) {
 Throwable Root = findRootCause( oe );
 if (Root instanceof LockTimeoutException) {
  LockTimeoutException lte = (LockTimeoutException)Root;
 System.out.println(lte.getLockRequestQueueDetails());
}
}
```

Solución posible

La excepción `LockTimeoutException` es para impedir posibles puntos muertos en la aplicación. Se iniciará una excepción de este tipo cuando haya esperado un tiempo establecido. El tiempo que espera, no obstante, se puede establecer con el método `setLockTimeout(int)` que está disponible para BackingMap. El uso del método setLockTimeout elimina la excepción `LockTimeoutException`. Si no existe realmente un bloqueo en la aplicación, puede ayudarle a impedir la excepción LockTimeoutException ajustar el tiempo de espera de bloqueo.

En el código siguiente se muestra cómo crear una ObjectGrid, definir una correlación y establecer su valor de LockTimeout en 30 segundos

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("MapName");
// Esto establecerá el tiempo que una
// solicitud de bloqueo esperará antes de que se inicie una excepción
bMap.setLockTimeout(30);
```

Se puede utilizar el código anterior para sobregrabar ObjectGrid y las propiedades de correlación. Si crea una ObjectGrid a partir de un archivo XML, se puede establecer la propiedad LockTimeout dentro del distintivo backingMap. A continuación figura un ejemplo de un distintivo backingMap que establece un valor de LockTimeout de correlación en 30 segundos.

```
<backingMap name="MapName" lockStrategy="PESSIMISTIC" lockTimeout="30">
```

LockDeadlockException

Mensaje

Puede obtener una excepción LockDeadLockException directamente u obtenerla al capturar una excepción ObjectGridException. A continuación figura un ejemplo de código que muestra la obtención de la excepción, después el mensaje que se muestra.

```
try {
...
} catch (ObjectGridException oe) {
System.out.println(oe);
}
```

El resultado es:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: %Message
```

%Message representa la serie que se pasa como parámetro cuando se crea y se genera la excepción.

Causa de la excepción

El tipo de punto muerto más común se produce cuando se utiliza la estrategia de bloqueo pesimista y dos clientes independientes poseen cada uno un bloqueo compartido sobre un objeto concreto. Luego, los dos intentan obtener un bloqueo exclusivo sobre ese objeto. A continuación figura un diagrama que muestra tal situación con bloques de transacción que producirían que se generara tal excepción.

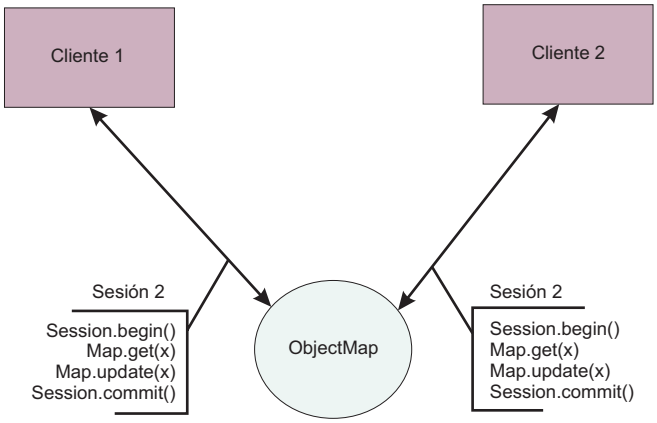


Figura 23. Ejemplo de una posible situación de punto muerto

Esta es una vista abstracta de lo que sucede en el programa cuando se produce una excepción. En una aplicación con muchas hebras actualizando el mismo ObjectMap, es posible encontrarse con esta situación. A continuación figura un ejemplo paso a paso de dos clientes cuando ejecutan los bloques de código de la transacción, que se describe en la figura anterior.

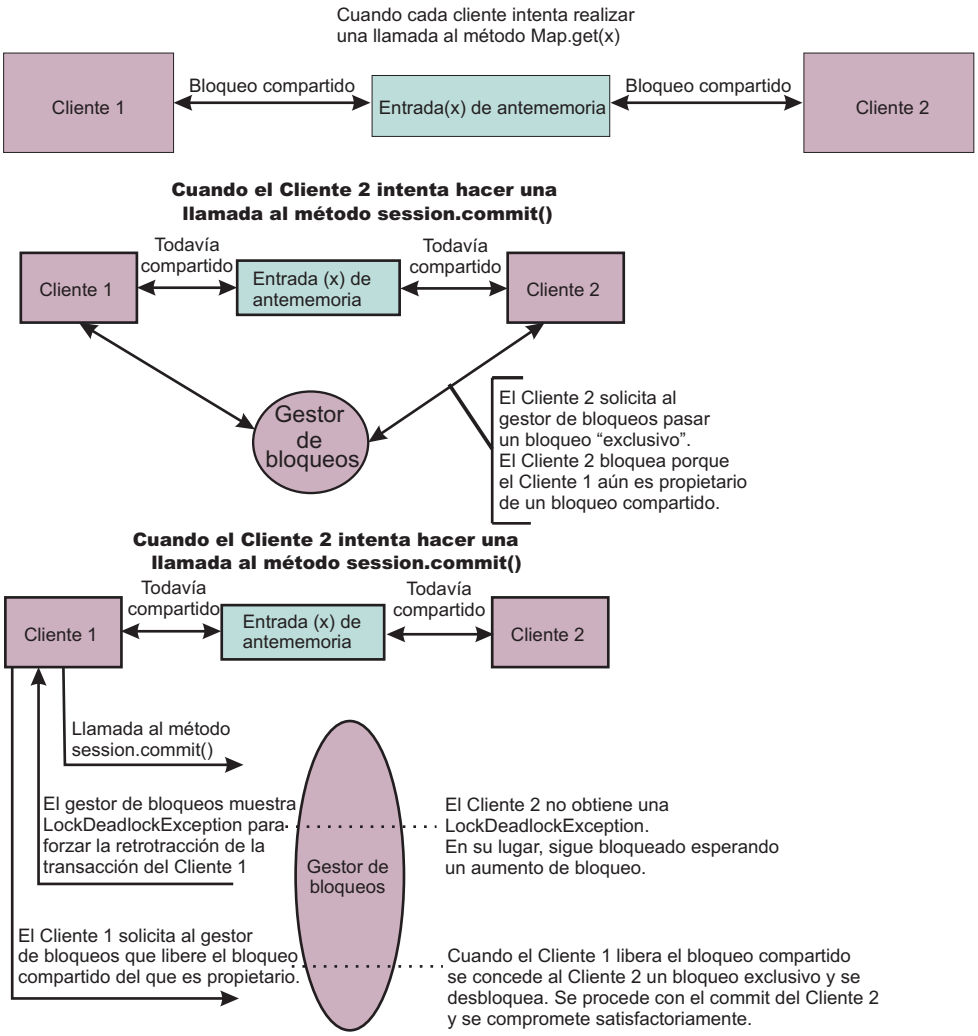


Figura 24. Una situación de punto muerto

Como se muestra, cuando dos clientes intentan obtener bloqueos exclusivos y aún poseen los bloqueos compartidos, es imposible que ninguno de ellos lo obtenga realmente. Siempre esperan a que el otro cliente libere el bloqueo compartido y así se produce una excepción `LockDeadlockException`.

Soluciones posibles

En alguna ocasión es positivo recibir esta excepción. Cuando hay muchas hebras, todas ellas que ejecutan transacciones en una correlación concreta, es posible que se encuentre con la situación descrita anteriormente (Figura 1). Para impedir que se cuelgue el programa, se genera esta excepción. La obtención de esta excepción le permite notificarse a sí mismo y, si lo desea, agregar código al bloque de captura de modo que pueda obtener más detalles de la causa. Dado que sólo se mostrará esta excepción en una estrategia de bloqueo pesimista, una solución sencilla es utilizar simplemente una estrategia de este tipo. Si es necesaria la estrategia pesimista, no obstante, puede utilizar el método getForUpdate en lugar del método get. Esto elimina la obtención de excepciones para la situación descrita anteriormente.

Diagnóstico de problemas de configuración XML

Referencia a una colección de plug-ins no existente

Cuando se utiliza XML para definir plug-ins de BackingMap, el atributo pluginCollectionRef del elemento backingMap debe hacer referencia a un objeto backingMapPluginCollection. El atributo pluginCollectionRef debe coincidir con el ID de uno de los elementos backingMapPluginCollection.

Mensaje

Si el atributo pluginCollectionRef no coincide con ningún atributo de ID de ninguno de los elementos backingMapPluginConfiguration, se mostrará en el archivo de anotaciones cronológicas un mensaje similar al siguiente.

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E:
Este es un mensaje informativo sólo en inglés:
Invalid XML file. Line: 14; URI: null;
Message: Key 'pluginCollectionRef' with
value 'bookPlugins' not found for identity
constraint of element 'objectGridConfig'..
```

A continuación figura un extracto del archivo de anotaciones cronológicas con el rastreo habilitado:

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl
E CWOBJ9002E: Este es un mensaje informativo sólo en inglés:
Invalid XML file. Line: 14; URI: null; Message: Key
'pluginCollectionRef' with
value 'bookPlugins' not found for identity constraint
of element 'objectGridConfig'..
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R com.ibm.websphere.objectgrid.
ObjectGridException:
Invalid XML file: etc/test/document/bookstore.xml
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R at
com.ibm.ws.objectgrid.config.XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R at
com.ibm.websphere.objectgrid.ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R Caused by: org.xml.sax.
SAXParseException: Key 'pluginCollectionRef' with value 'bookPlugins'
not found for identity
constraint of element 'objectGridConfig'.
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.error(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:011 CDT] 686c060e SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:011 CDT] 686c060e SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

Problema

A continuación se muestra el archivo XML que se ha utilizado para producir este error. Observe que el book BackingMap tiene el atributo pluginCollectionRef establecido en bookPlugins y la backingMapPluginCollection única tiene un ID de collection1.

```
<?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="bookstore">
<backingMap name="book" pluginCollectionRef="bookPlugin" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Solución

Para corregir el problema, asegúrese de que el valor de cada pluginCollectionRef coincida con el ID de uno de los elementos backingMapPluginCollection. En este ejemplo, con sólo cambiar el nombre de la pluginCollectionRef por collection1 se impide este error. Otros modos de corregir el problema serían cambiar el ID de la backingMapPluginCollection existente para que coincida con la pluginCollectionRef, o bien, agregar una backingMapPluginCollection adicional con un ID que coincida con la pluginCollectionRef.

Falta un atributo necesario

Muchos de los elementos del archivo XML tienen varios atributos opcionales. Puede incluir o excluir atributos opcionales en el archivo. De cualquier modo, el XML superará la validación. No obstante, hay más atributos necesarios. Si estos atributos necesarios no están presentes cuando se utiliza su elemento asociado, no se supera la validación XML.

Mensaje

Cuando falta un atributo necesario, aparece un mensaje similar al siguiente en el archivo de anotaciones cronológicas. En este ejemplo, falta el atributo type del elemento property.

```
[7/15/05 13:41:41:267 CDT] 6873dcac XmlErrorHandl E CWOBJ9002E:
Este es un mensaje informativo sólo en inglés: Invalid XML file.
Line: 12; URI: null; Message: cvc-complex-type.4:
Attribute 'type' must appear on element 'property'..
```

A continuación figura un extracto del archivo de anotaciones cronológicas con el rastreo habilitado.

```
[7/15/05 14:08:48:506 CDT] 6873dff9 XmlErrorHandl E
CWOBJ9002E: Este es un mensaje informativo sólo en inglés: Invalid XML file.
Line: 12; URI: null; Message: cvc-complex-type.4: Attribute 'type'
must appear on element 'property'..
[7/15/05 14:08:48:526 CDT] 6873dff9 SystemErr R com.ibm.websphere.objectgrid.
ObjectGridException: Invalid XML file: etc/test/document/bookstore.xml
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R at com.ibm.ws.objectgrid.config.
XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R at com.ibm.websphere.objectgrid.
ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R Caused by: org.xml.sax.
SAXParseException: cvc-complex-type.4:
Attribute 'type' must appear on element 'property'.
```

```
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.error(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

Problema

El ejemplo siguiente es el archivo XML que se ha utilizado para producir este error. Observe que la propiedad en el desalojador sólo tiene dos de los tres atributos necesarios. Están presentes los dos atributos name y value, pero falta el atributo type. El atributo que falta provoca que no se supere la validación 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="bookstore">
<backingMap name="book" pluginCollectionRef="collection1" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
<property name="maxSize" value="89" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Solución

Para solucionar este problema, añada el atributo necesario al archivo XML. En el archivo XML de ejemplo que se muestra anteriormente, tendrá que añadir el atributo type y asignarlo el valor integer.

Falta un elemento necesario

El esquema necesita algunos elementos XML. Si no están presentes, el XML no supera la validación.

Mensaje

Cuando falta un elemento necesario, aparece un mensaje similar al siguiente en el archivo de anotaciones cronológicas. En este caso, falta el elemento objectGrid.

```
[7/15/05 14:54:23:729 CDT] 6874d511 XmlErrorHandl E CWOBJ9002E:
Este es un mensaje informativo sólo en inglés: Invalid XML file.
Line: 5; URI: null; Message: cvc-complex-type.2.4.b: The content of
element 'objectGrids' is not complete.
One of '{"http://ibm.com/ws/objectgrid/config":objectGrid}' is expected.
```

Habilite el rastreo para mostrar más información relacionada con este error. El apartado ObjectGridManager cubre información sobre cómo se activa el rastreo.

Problema

El ejemplo siguiente es el archivo XML que se ha utilizado para producir este problema. Observe que el elemento objectGrids no tiene elementos objectGrids hijo. De acuerdo al esquema XML, el elemento objectGrid debe aparecer al menos una vez dentro de los distintivos objectGrid. La falta de este elemento provoca que no se supere la validación 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>
</objectGrids>
</objectGridConfig>
```

Solución

Para corregir este problema, asegúrese de que en el archivo XML se encuentren los elementos necesarios. En el ejemplo anterior, debe colocarse al menos un elemento objectGrid dentro del distintivo objectGrids. Una vez que están presentes los elementos necesarios, puede validar satisfactoriamente el archivo XML.

El archivo XML válido siguiente contiene los elementos necesarios presentes.

```
<?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="bookstore" />
</objectGrids>
</objectGridConfig>
```

El valor XML del atributo no es válido

Mensaje

Algunos de los atributos del archivo XML pueden asignarse sólo a determinados valores. El esquema enumera los valores aceptables de estos atributos. Entre estos atributos se incluyen:

- `authorizationMechanism` • atributo del elemento objectGrid
- `copyMode` atributo del elemento `backingMap`
- `lockStrategy` atributo del elemento `backingMap`
- `ttlEvictorType` atributo del elemento `backingMap`
- `type` atributo del elemento `property`

Si se asigna un valor no válido a uno de estos atributos, no se supera la validación XML.

Cuando se establece un atributo en un valor que no sea uno de los que tiene enumerados, se muestra este mensaje en el archivo de anotaciones cronológicas:

```
[7/19/05 16:45:40:992 CDT] 6870e51b XmlErrorHandl E
CWOBJ9002E: Este es un mensaje informativo sólo en inglés: Invalid XML file.
Line: 6; URI: null; Message:
cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid with
respect to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ,
COPY_ON_WRITE, NO_COPY]'. It must be a value from the enumeration..
```

A continuación figura un extracto del archivo de anotaciones cronológicas con el rastreo habilitado.

```
[7/19/05 16:45:40:992 CDT] 6870e51b XmlErrorHandl E
CWOBJ9002E: Este es un mensaje informativo sólo en inglés: Invalid XML file.
Line: 6; URI: null; Message:
cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid with
respect to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ,
COPY_ON_WRITE, NO_COPY]'. It must be a value from the enumeration..
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R com.ibm.websphere.objectgrid
.ObjectGridException: Invalid XML file: etc/test/document/backingMapAttrBad.xml
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R at com.ibm.ws.objectgrid.config
.XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R at com.ibm.websphere.objectgrid
.ProcessConfigXML$2.run(ProcessConfigXML.java:99)...
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R Caused by:
org.xml.sax.SAXParseException:
cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid
with respect
to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE,
NO_COPY]'.
It must be a value from the enumeration.
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.util
.ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.util
.ErrorHandlerWrapper.error(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.XMLErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.XMLErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.xs.XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.xs.XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

Problema

Se ha establecido incorrectamente un atributo al que se le ha asignado un valor que no está dentro de un conjunto de valores concretos. En este caso, no se ha establecido el atributo copyMode en uno de sus valores enumerados. Se ha establecido en INVALID_COPY_MODE. A continuación figura el archivo XML que se ha utilizado para producir este error.

```
<?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="bookstore" />
<backingMap name="book" copyMode="INVALID_COPY_MODE"/>
</objectGrid>
</objectGrids>
</objectGridConfig>
```

Solución

En este ejemplo, copyMode tiene un valor no válido. Establezca el atributo en uno de estos valores válidos: COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE o NO_COPY.

Validación de XML sin soporte de implementación

El kit de desarrollo de software de IBM (SDK) versión 1.4.2 contiene una implementación de alguna función JAXP que se va a utilizar para la validación XML con un esquema.

Mensaje

Cuando se utiliza un SDK que no contiene esta implementación, los intentos de validación pueden producir un error. Si quisiera validar XML con un SDK que no contiene esta implementación, bájese Xerces e incluya sus archivos Java (JAR) en la classpath.

Si intenta validar XML con un SDK que no tiene la implementación necesaria, aparecerá en el archivo de anotaciones cronológicas el siguiente error.

```
[7/19/05 10:50:45:066 CDT] 15c7850 XmlConfigBuil d XML validation is enabled
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R com.ibm.websphere
.objectgrid[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at
   com.ibm.ws.objectgrid
.ObjectGridManagerImpl.getObjectGridConfigurations
   (ObjectGridManagerImpl.java:182)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid
.ObjectGridManagerImpl.createObjectGrid(ObjectGridManagerImpl.java:309)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid.test.
config.DocTest.main(DocTest.java:128)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R Caused by: java.lang
.IllegalArgumentException: No attributes are implemented
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at org.apache.crimson.jaxp.
DocumentBuilderFactoryImpl.setAttribute(DocumentBuilderFactoryImpl.java:93)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid.config
.XmlConfigBuilder.<init>(XmlConfigBuilder.java:133)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.websphere.objectgrid
.ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
```

Problema

El SDK que se utiliza no contiene una implementación de la función JAXP que es necesaria para validar archivos XML con un esquema.

Solución

Después de bajarse Apache Xerces y de incluir los JAR en la classpath, puede validar el archivo XML satisfactoriamente.

Mensajes de ObjectGrid

Esta información de referencia aporta datos adicionales sobre los mensajes que podrían producirse al utilizar ObjectGrid. Los mensajes se identifican por la clave de mensaje y tienen una descripción y una respuesta del usuario. Podrían ser informativos, de aviso o de error y se indican por la última letra (I, W o E) de la clave de mensaje. La parte de descripción del mensaje explica porqué ha aparecido el mensaje. La parte del mensaje de respuesta del usuario describe qué acción debería tomarse en el caso de un aviso o de un mensaje de error.

CWOBJ0001E: se ha llamado al método {0} después de completar la inicialización.
Descripción: cuando finalice la
inicialización, algunas

invocaciones del método no se aceptan.
Respuesta del usuario: reestructure
el código de forma que la configuración se complete antes de iniciar el uso del
tiempo de ejecución.

ICWOBJ0002W:
el componente ObjectGrid está ignorando una
excepción inesperada: {0}.
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ0005W:
la hebra ha generado una InterruptedException: {0}
Descripción: se ha producido una InterruptedException.
Respuesta del usuario: consulte el mensaje de excepción para ver si se trata de una
excepción esperada o no.

CWOBJ0006W: se ha producido una
excepción: {0}
Descripción: se ha producido una
excepción durante el tiempo de ejecución.
Respuesta del usuario: consulte el
mensaje de excepción para ver si se trata de una
excepción esperada.

CWOBJ0007W: el valor nulo se ha
especificado para {0}, se utiliza
un valor por omisión de {1}.
Descripción: se ha especificado un
valor nulo para la variable. Se
utiliza un valor por omisión.
Respuesta del usuario: establezca
el valor adecuado.
Consulte la documentación de ObjectGrid
para saber los valores válidos para las variables o propiedades.

CWOBJ0008E:
el valor {0} proporcionado para la
propiedad {1} no es válido.
Descripción: se ha especificado un
valor no válido
para la variable.
Respuesta del usuario: establezca el
valor adecuado.
Consulte la documentación de ObjectGrid
para saber los valores válidos para las variables o propiedades.

CWOBJ0010E: falta la clave del
mensaje {0}.
Descripción: falta una clave de
mensaje en el paquete de recursos de mensajes
Respuesta del usuario: CMSG0002

CWOBJ0011W: no se ha podido
deserializar el campo {0} de la clase {1}
utilizando el valor por omisión.
Descripción: durante la
deserialización de un objeto,
no se ha encontrado un campo previsto. Es posible que no se encontrara ese campo
porque ha deserializado el objeto una versión de la clase diferente
a la que lo serializó.
Respuesta del usuario: este aviso
indica un problema potencial.
No se requiere ninguna acción del usuario a menos que se generen otros errores.

CWOBJ0012E: El
código de tipo LogElement, {0} ({1}), no se
reconoce para esta operación.
Descripción: se ha producido un error
interno en el tiempo de ejecución de ObjectGrid.
Respuesta del usuario: CMSG0002

CWOBJ0013E: se ha producido una
excepción al intentar desalojar
entradas de la antememoria: {0}
Descripción: se ha producido un
problema al intentar aplicar
la entradas de desalojo a la antememoria.
Respuesta del usuario: consulte el
mensaje de excepción para ver si se trata de
una excepción esperada.

CWOBJ0014E: el tiempo de ejecución de ObjectGrid ha detectado un intento
de anidar transacciones.
Descripción: no se permite el
anidamiento de transacciones.
Respuesta del usuario: modifique el código para evitar el anidamiento de transacciones.

CWOBJ0015E: se ha producido una
excepción al intentar
procesar una transacción: {0}
Descripción: se ha producido un
problema durante el proceso de la transacción.
Respuesta del usuario: consulte el
mensaje de excepción para ver
si se trata de una excepción esperada.

CWOBJ0016E: no se ha detectado ninguna transacción activa para la
operación actual.
Descripción: es necesaria una transacción activa para
realizar esta operación.
Respuesta del usuario: modifique el
código para iniciar una transacción
antes de realizar esta operación.

CWOBJ0017E: se
ha detectado una excepción de clave duplicada
al procesar la operación ObjectMap: {0}
Descripción: ya existe la clave de la entrada en la antememoria.
Respuesta del usuario: modifique el código para evitar insertar la misma
clave más de una vez.

CWOBJ0018E: no se ha encontrado la
clave al procesar la
operación ObjectMap: {0}
Descripción: no existe la clave de la entrada en la antememoria.
Respuesta del usuario: modifique
el código para garantizar que la entrada existe antes de
intentar la operación.

CWOBJ0019W: no se han encontrado
datos en la ranura de entrada de antememoria reservada
para que la utilice {0} para el nombre de ObjectMap {1}.
Descripción: se ha producido un error
interno en el
tiempo de ejecución de ObjectGrid.
Respuesta del usuario: CMSG0002

CWOBJ0020E: la entrada de antememoria
no está en BackingMap {0}.
Descripción: error interno en el tiempo de ejecución de ObjectGrid.
Respuesta del usuario: CMSG0002

CWOBJ0021E: no se ha encontrado
ninguna instancia utilizable de ObjectTransformer
durante la deserialización del objeto LogSequence para ObjectGrid {0} y
ObjectMap {1}.
Descripción: el extremo receptor de
un objeto LogSequence
no tiene la configuración adecuada para dar soporte a la
instancia de ObjectTransformer adecuada.
Respuesta del usuario: verifique la
configuración de las instancias de ObjectGrid
para los extremos de envío y recepción del objeto LogSequence.

CWOBJ0022E: el llamante no posee
mutex: {0}.
Descripción: se ha producido un error
interno en el tiempo de ejecución de ObjectGrid.
Respuesta del usuario: CMSG0002

CWOBJ0023E: no se reconoce la
modalidad CopyMode ({0}) para esta operación.
Descripción: se ha producido un error
interno en el tiempo de ejecución de ObjectGrid.
Respuesta del usuario: CMSG0002

CWOBJ0024E: no se ha podido
deserializar el campo {0} de la clase {1}
Error de
deserialización.
Descripción: durante la
deserialización de un objeto,
no se ha encontrado un campo obligatorio. Probablemente se trate de un error de tiempo de ejecución de ObjectGrid.
Respuesta del usuario: CMSG0002

CWOBJ0025E: error de serialización
del objeto LogSequence.
El número de objetos LogElement serializados ({0}) no coincide con el número de objetos LogElements leídos ({1}).
Descripción: se ha producido un error
interno en el tiempo de ejecución de ObjectGrid.
Respuesta del usuario: CMSG0002

CWOBJ0026E: El tipo de credencial de
JMX no es correcto.
Debe ser del tipo {0}.
Descripción: el tipo de credencial de
JMX no es correcto.
Si se utiliza la
autenticación básica, el tipo esperado es String[] y su primer elemento
es el nombre de usuario y el segundo, la contraseña.
Si el certificado del cliente se
utiliza, el tipo previsto es Certificate[].
Respuesta del usuario: utilice las
credenciales correctas.

CWOBJ0027E: error de tiempo de
ejecución interno.
El método de clones no está soportado: {0}
Descripción: se ha producido un error
interno en el tiempo de ejecución de ObjectGrid.
CLONE_METHOD_NOT_SUPPORTED_CWOBJ0027.useraction=CMSG0002

CWOBJ0028E: se ha producido un error
en {0} para la correlación {1}. La clave
{2} no se ha encontrado en la correlación. El tipo de LogElement es {3}.
Descripción: se ha producido un error
interno
al intentar desalojar una entrada.
Respuesta del usuario: CMSG0002

CWOBJ0029E: se ha producido un error
en {0} para la correlación {1}. A CacheEntry le
falta un objeto {2} para la clave {3}. El tipo de LogElement es {4}.
Descripción: se ha producido un error
al intentar desalojar una entrada.
Respuesta del usuario: CMSG0002

CWOBJ0900I: el componente de tiempo
de ejecución de ObjectGrid se ha iniciado para el servidor {0}.
Descripción: el componente ObjectGrid
se ha iniciado.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ0901E: se necesita la propiedad
del sistema "{0}" para iniciar el componente ObjectGrid
para el servidor {1}.
Descripción: el componente de
tiempo de ejecución de ObjectGrid necesita que "{0}" se
especifique como propiedad del sistema de la máquina virtual Java.
Respuesta del usuario: consulte el
centro de información para utilizar la consola del administrador de WebSphere
para proporcionar las propiedades personalizadas necesarias de ObjectGrid.

CWOBJ0902W: un error ha impedido
que el componente de tiempo de ejecución de ObjectGrid pudiera iniciarse
en el servidor {0}.
Descripción: un error anterior
ha impedido el inicio del componente ObjectGrid.
Respuesta del usuario: consulte los
mensajes de error anteriores para determinar lo que ha impedido
iniciarse al componente ObjectGrid.

CWOBJ0910I: el componente de tiempo
de ejecución de ObjectGrid se ha detenido para el servidor {0}.
Descripción: el componente ObjectGrid
se ha detenido.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ0911I: iniciando el componente
de tiempo de ejecución de ObjectGrid para el servidor {0}.
Descripción: el componente ObjectGrid
se está iniciando.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ1001I: el servidor ObjectGrid
{0} está preparado para procesar peticiones.
Descripción: el servidor ObjectGrid
está preparado para procesar peticiones.
Respuesta del usuario: los servicios
para este servidor ObjectGrid están disponibles.

CWOBJ1002E: el puerto de servidor {0} ya está siendo utilizado.
Descripción: el servidor ObjectGrid
no puede iniciarse debido a un conflicto de puertos.
Respuesta del usuario: los usuarios
deben elegir otro puerto.

CWOBJ1003I: el servidor de adaptador
DCS está inhabilitado por la configuración,
para habilitarlo, cambie la configuración con un punto final definido.
Descripción: el adaptador DCS está desactivado.
Respuesta del usuario: los usuarios
pueden activar el adaptador DCS modificando la configuración.

CWOBJ1004E: el tema del servidor es nulo
Descripción: el tema del servidor es nulo
Respuesta del usuario: CMSG0002

CWOBJ1005E: la cola de peticiones
entrantes es nula.
Descripción: el manejador de
peticiones de cliente no puede recuperar las peticiones.
Respuesta del usuario: CMSG0002

CWOBJ1006E: la cola de resultados
salientes es nula.
Descripción: el manejador de
peticiones de cliente no puede dar resultados al cliente.
Respuesta del usuario: CMSG0002

CWOBJ1007E: la solicitud del cliente
ObjectGrid es nula.
Descripción: el manejador de

peticiones de cliente no puede manejar una petición que
no contiene ninguna información acerca de la petición.
Respuesta del usuario: compruebe la
petición.

CWOBJ1008E: el TxID de la solicitud
del cliente ObjectGrid es nulo.
Descripción: TXID se utiliza para
hacer coincidir las conexiones y realizar agrupaciones, TXID no puede ser nulo.
Respuesta del usuario: CMSG0002

CWOBJ1009E: el cliente ObjectGrid ha
recibido una respuesta nula del servidor.
Descripción: se ha producido una
respuesta nula del servidor.
Respuesta del usuario: CMSG0002

CWOBJ1010I: se está procesando la
petición de conclusión.
Descripción: los servidores del
clúster están procesando la petición de conclusión.
Respuesta del usuario: ninguna.

CWOBJ1011I: se está enviando la
petición de conclusión.
Descripción: los servidores del
clúster están procesando la petición de conclusión.
Respuesta del usuario: ninguna.

CWOBJ1012I: se lleva a cabo la
petición de conclusión.
Descripción: los servidores del
clúster están procesando la petición de conclusión.
Respuesta del usuario: ninguna.

CWOBJ1110I:
se está iniciando el transporte del clúster ObjectGrid {0} utilizando
dirección IP {1}, puerto {2}, tipo de transporte {3}.
Descripción: se está iniciando el
transporte del miembro del clúster ObjectGrid.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ1111W: la resolución de la
dirección IP del nombre de sistema principal {0} sólo ha encontrado
la dirección de bucle de retorno. La dirección de bucle de retorno se utilizará.
Descripción: puede que haya un
problema con el nombre de sistema principal o la resolución de DNS.
En la implementación relacionada con la producción, se espera una dirección que no es la
del bucle de retorno.
Respuesta del usuario: modifique el
nombre de sistema principal o determine si existe un problema de DNS.

CWOBJ1112E: se ha detectado un error
al buscar la dirección IP
del nombre de sistema principal de un miembro del clúster ObjectGrid.
El nombre de sistema principal es {0}
y el nombre de servidor es {1}. El miembro se excluirá del clúster.
Descripción: no se puede resolver la
dirección IP del sistema principal indicado. El
miembro del clúster ObjectGrid del sistema principal especificado se excluirá.
Respuesta del usuario: corrija el
problema de búsqueda del nombre del sistema principal y vuelva a intentarlo.

CWOBJ1113E: el servicio de
transporte del clúster ObjectGrid en este proceso
no se ha iniciado. Este miembro del clúster no está definido en la configuración.
Descripción: este miembro del clúster
ObjectGrid no es un miembro configurado
del clúster. Si este miembro del clúster debe ser un miembro de un
clúster ObjectGrid, repare la configuración.
Respuesta del usuario: revise la
configuración actual.

CWOBJ1114E: el servicio de transporte
del clúster ObjectGrid en este proceso
no ha podido procesar el mensaje entrante. El mensaje es {0} y la
excepción es {1}.
Descripción: se ha detectado un error
interno imprevisto.
Respuesta del usuario: consulte el
sitio web de soporte en Internet de IBM ObjectGrid
si existe un problema similar o póngase en contacto con el servicio técnico de IBM.

CWOBJ1115E: se ha recibido un suceso
de cambio de vista no reconocido del
transporte del clúster ObjectGrid. El identificador de vista es {0} y el suceso es {1}.
Descripción: el tipo de suceso no se
reconoce.
HA Manager no
sabe cómo responder a ese suceso.
Respuesta del usuario: consulte el
sitio web de soporte en Internet de IBM ObjectGrid
si existe un problema similar o póngase en contacto con el servicio técnico de IBM.

CWOBJ1116E: otro proceso ha intentado
conectarse a este proceso
mediante el transporte del clúster ObjectGrid y ha sido rechazado.
El proceso de conexión
ha proporcionado un nombre de {0}, un destino de {1},
un nombre de miembro de {2} y una dirección IP de {3}.
El mensaje de error es {4}.
Descripción: el transporte del
clúster ObjectGrid ha rechazado el
intento de conexión.
Respuesta del usuario: puede que se
trate de un intento de conexión de una parte no autorizada.

CWOBJ1117E: se ha producido un error
al intentar autenticar una conexión.
La excepción es {0}.
Descripción: el transporte del
clúster ObjectGrid ha rechazado el
intento de conexión.
Respuesta del usuario: puede que se
trate de un intento de conexión de una parte no autorizada.

CWOBJ1118I: inicializando
servidor ObjectGrid [clúster: {0} servidor {1}].
Descripción: el miembro del
clúster ObjectGrid se está inicializando.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ1119I: error del cliente
ObjectGrid al conectarse al sistema principal: {0} puerto: {1}.
Descripción: error al conectarse el
cliente ObjectGrid.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ1120I: el cliente ObjectGrid se
ha conectado satisfactoriamente al sistema principal: {0} puerto: {1}.
Descripción: el cliente ObjectGrid se
ha conectado satisfactoriamente.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ1201E: no se ha definido ningún
punto final de acceso de clientes.
Descripción: no se ha definido ningún
punto final de acceso de clientes.
Respuesta del usuario: defina un
punto final de acceso de clientes válidos.

CWOBJ1202E: el socket de servidor SSL
no ha podido inicializarse. El mensaje
de excepción es {0}
Descripción: el socket de servidor

SSL no ha podido inicializarse.
Los valores de SSL
pueden ser incorrectos o el número de puerto ya está siendo utilizado.
Respuesta del usuario: examine la
excepción para averiguar lo que ha pasado.

CWOBJ1203W: se ha recibido
un suceso de tiempo de espera excedido del servidor para la
transacción: {0}
Descripción: el cliente no ha
recibido el mensaje de respuesta previsto
del servidor dentro del límite de tiempo de espera configurado.
Respuesta del usuario: busque
mensajes anteriores que puedan explicar el tiempo de espera excedido.
Si no encuentra ninguno, pruebe a aumentar el límite de tiempo de espera excedido.

CWOBJ1204W: se ha recibido un mensaje
de un tipo desconocido.
El mensaje es: {0}
Descripción: se ha detectado un error
interno imprevisto.
Respuesta del usuario: consulte el
sitio web de soporte en Internet de IBM ObjectGrid
si existe un problema similar o póngase en contacto con el servicio técnico de IBM.

CWOBJ1205E: no se ha podido
inicializar SSL.
El mensaje
de excepción es {0}
Descripción: no se ha podido
inicializar SSL.
Los valores de SSL
pueden ser incorrectos.
Respuesta del usuario: examine la
excepción para averiguar
lo que ha pasado.

CWOBJ1206W: no se ha podido
inicializar SSL.
El mensaje de excepción es: {0}
Descripción: no se ha podido
inicializar SSL.
Los valores de SSL pueden ser incorrectos.
Respuesta del usuario: examine la
excepción para averiguar lo que ha pasado.

CWOBJ1207W: la propiedad {0} del
plug-in {1} está utilizando un
tipo de propiedad no soportada.
Descripción: los únicos tipos de
propiedades soportados
son las primitivas de Java y sus equivalentes de java.lang. java.lang.String
también tiene soporte.
Respuesta del usuario: compruebe el
tipo de propiedad y cámbielo por
uno de los tipos soportados.

CWOBJ1208W: el tipo de
plug-in especificado, {0}, no es uno de los
tipos de plug-in soportados.
Descripción: este tipo de plug-in no
está soportado.
Respuesta del usuario: añada uno de
los tipos de plug-in
soportados.

CWOBJ1211E: No se ha podido realizar
la creación de la PMI (Performance Monitoring Infrastructure)
de {0}. La excepción es {1}.
Descripción: intento anómalo de
crear una PMI de ObjectGrid.

Respuesta del usuario: examine el
mensaje de excepción
y las anotaciones cronológicas de la captura de datos en primer error (FFDC).

Los mensajes
siguientes se utilizan para reunir el ID de usuario y la contraseña en el panel
o en la entrada estándar.
LOGIN_PANEL_TITLE=Inicio de sesión en el servidor de destino
GENERIC_LOGIN_PROMPT=Entre información de inicio de sesión
USER_ID=Identidad del usuario
PASSWORD=Contraseña
OK=Aceptar
CANCEL=Cancelar

CWOBJ1215I: el receptor de sucesos de
propagación de transacciones ObjectGrid se está
inicializando [ObjectGrid {0}].
Descripción: este mensaje
informativo indica que se está inicializando el escucha de sucesos de propagación de
transacciones de ObjectGrid.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ1216I: el receptor de sucesos de
propagación de transacciones ObjectGrid.
se ha inicializado [ObjectGrid {0}].
Descripción: se ha inicializado el
receptor de sucesos de propagación de transacciones ObjectGrid.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ1217I: se ha inicializado el
punto de servicio de propagación de transacciones ObjectGrid
[ObjectGrid {0}].
Descripción: este mensaje informativo
indica que se ha inicializado el escucha de sucesos de propagación de transacciones de
ObjectGrid.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ1218E: se ha producido un error
en el receptor de sucesos de propagación de transacciones ObjectGrid
[ObjectGrid {0} Mensaje de excepción {1}].
Descripción: el tiempo de
ejecución de ObjectGrid ha detectado un error en la propagación de
transacciones de ObjectGrid.
Respuesta del usuario: examine la
excepción para determinar el error.

CWOBJ1219E: Se ha producido un error
en el punto final de servicio de propagación de transacciones de ObjectGrid
[ObjectGrid {0} Mensaje de excepción {1}].
Descripción: el tiempo de ejecución
de ObjectGrid ha detectado un error en el punto final de servicio de propagación de
transacciones de ObjectGrid.
Respuesta del usuario: examine la
excepción para determinar el error.

CWOBJ1220E: el servicio de
propagación de transacciones de ObjectGrid
no está soportada en este entorno.
Descripción: el servicio de propagación de
transacciones de ObjectGrid
no está soportado en z/OS o en el entorno del servidor ObjectGrid autónomo.
Respuesta del usuario: no utilice el
servicio de propagación de transacciones de ObjectGrid
en z/OS o en el entorno del servidor ObjectGrid autónomo.

CWOBJ1300I: el adaptador ha
inicializado ObjectGrid satisfactoriamente.
Descripción: el adaptador ha
inicializado ObjectGrid satisfactoriamente.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ1301E: el adaptador no ha podido
inicializar ObjectGrid.
Se ha producido una excepción
[Mensaje de excepción {0}].
Descripción: el intento del adaptador
de inicializar ObjectGrid no ha sido satisfactorio.
Respuesta del usuario: examine la
excepción para determinar el error.

CWOBJ1302I: se ha detenido el
adaptador.
Descripción: se ha detenido el
adaptador.
Respuesta del usuario: ninguna. Entrada informativa.

CWOBJ1303I: el adaptador se ha
iniciado.
Descripción:
PMA_CWOBJ1303.explanation=el adaptador se ha iniciado.
Respuesta del usuario: ninguna.
Entrada informativa.

CWOBJ1304I: se ha habilitado la
seguridad de ObjectGrid.
Descripción: se ha habilitado la
seguridad de ObjectGrid.
Respuesta del usuario: ninguna.

CWOBJ1305I: se ha inhabilitado la
seguridad de ObjectGrid.
Descripción: se ha inhabilitado la
seguridad de ObjectGrid.
Respuesta del usuario: ninguna.

CWOBJ1306W: no se pueden recuperar
los certificados de cliente del socket SSL.
Descripción: por alguna razón, el
tiempo de ejecución no puede recuperar los certificados
de cliente del socket SSL.
Respuesta del usuario: compruebe las
configuraciones de SSL.

CWOBJ1307I: se ha habilitado la
seguridad de la instancia de ObjectGrid {0}.
Descripción: se ha habilitado la
seguridad de la instancia de ObjectGrid {0}.
Respuesta del usuario: ninguna.

CWOBJ1308I: se ha inhabilitado la
seguridad de la instancia de ObjectGrid {0}.
Descripción: se ha inhabilitado la
seguridad de la instancia de ObjectGrid {0}.
Respuesta del usuario: ninguna.

CWOBJ1309E: se ha producido un error
imprevisto en la
creación del símbolo de conexión: {0}.
Descripción: se ha producido un error
imprevisto en la
creación del símbolo de conexión.
Respuesta del usuario: compruebe la
configuración de seguridad

CWOBJ1310E: otro proceso ha intentado
conectarse
a este proceso mediante el transporte del grupo principal y ha sido rechazado.
El
proceso de conexión ha proporcionado un nombre de grupo principal de origen {0}, un
destino de {1},
un nombre de miembro de {2} y una dirección IP de {3}.
El mensaje de error es {4}.
Descripción: el gestor de alta
disponibilidad ha

rechazado un intento de conexión.
Respuesta del usuario: puede que se
trate de un intento de conexión
de una parte no autorizada.

CWOBJ1400W: se han detectado varios
archivos JAR de tiempo de ejecución de ObjectGrid en la JVM.
El uso de varios archivos JAR de tiempo de ejecución de ObjectGrid puede acarrear problemas.
Descripción: normalmente sólo de
be haber un archivo JAR de tiempo de ejecución de ObjectGrid
en una JVM.
Respuesta del usuario: utilice el
archivo JAR de tiempo de ejecución de ObjectGrid adecuado para
la configuración.

CWOBJ1401E: se ha detectado un
archivo JAR de tiempo de ejecución de ObjectGrid inadecuado para esta
configuración. La configuración que se ha detectado es {0}. El archivo JAR esperado es {1}.
Descripción: cada archivo JAR de
tiempo de ejecución de ObjectGrid corresponde a una determinada
configuración soportada.
Respuesta del usuario: utilice el
archivo JAR de tiempo de ejecución de ObjectGrid adecuado para la configuración.

CWOBJ1402E: no se ha encontrado el
retorno de llamada del enlace de conexión de ObjectGrid para el id: {0}.
Descripción: error interno en el
tiempo de ejecución de ObjectGrid.
Respuesta del usuario: CMSG0002

CWOBJ1500E: se ha producido una
excepción al intentar crear un
GroupName para el grupo HA ({0}): {1}.
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ1501E: se ha producido una
excepción cuando el miembro ({0}) ha intentado
unirse al grupo HA ({1}): {2}.
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ1503E: no se puede acceder
a ObjectGrid ({0}) para aplicar las actualizaciones
al miembro de la duplicación ({1}).
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ1504E: se ha producido una
excepción al intentar
procesar LogSequences para la duplicación ({0}): {1}.
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ1505E: más de un miembro de
grupo de duplicación
ha informado como si fuera el principal. Sólo puede haber un principal activo. ({0}).
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ1506E: se ha detectado más de un
miembro principal de grupo de duplicación
en este grupo ({1}). Sólo puede haber un principal activo. ({0}).
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ1507W: se ha producido una
excepción al intentar finalizar
un proceso de duplicación para BackingMap ({0}): {1}.
Descripción: al intentar concluir un
miembro de grupo de duplicación principal, se ha producido una excepción durante el
proceso de borrado.
Respuesta del usuario: CMSG0002

CWOBJ1508E: se ha producido una
excepción al intentar enviar el mensaje ({0})
del remitente ({1}) al destinatario ({2}): {3}.
Descripción: se ha producido un
problema al intentar enviar un mensaje entre
miembros del grupo de duplicación.
Respuesta del usuario: CMSG0002

CWOBJ1509E: se ha producido una
excepción al intentar serializar
el mensaje ({0}): {1}.
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ1510E: se ha producido una
excepción al intentar inflar
el mensaje ({0}): {1}.
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ1511I: {0} ({1}) está preparado
para la actividad.
Descripción: el miembro del
grupo de duplicación especificado está preparado
para aceptar peticiones.
Respuesta del usuario: ninguna.

CWOBJ1512W: {0} ya existe en el
grupo de duplicación {1}.
Descripción: el miembro del grupo de
duplicación especificado ya está
activo en este grupo de duplicación.
Respuesta del usuario: ninguna.

CWOBJ1513E: no se ha podido realizar
la duplicación síncrona en {0} ({1}).
Este miembro ya no está activo.
Descripción: se ha detectado un
problema que ha impedido a la
duplicación síncrona completarse satisfactoriamente.
Respuesta del usuario: revise
los mensajes previos en el archivo de anotaciones cronológicas para ayudar
al diagnóstico del problema.
Es posible que deba detener y reiniciar el servidor especificado.

CWOBJ1514I: el principal ({0})
se va a rebajar a
duplicación o reposo.
Descripción: esta no es la
operación habitual, pero el proceso de ObjectGrid
puede continuar.
Respuesta del usuario: CMSG0002

CWOBJ1515I: no se cumplen los
requisitos mínimos de configuración
para el grupo de duplicación ({0}).
Descripción: los necesarios
requisitos de configuración de principal y duplicación
no se cumplen tras el cambio reciente en el miembro de grupo de duplicación.
Respuesta del usuario: espere que los
recursos adicionales se inicien y se
reconozcan para esta configuración.

CWOBJ1516E: se ha producido una
excepción al intentar
activar el proceso de duplicación para ObjectGrid ({0}): {1}.
Descripción: al intentar iniciar un
miembro
de grupo de duplicación principal, se ha producido una excepción durante el proceso de
activación.
Respuesta del usuario: CMSG0002

CWOBJ1517E: no se ha podido
realizar la duplicación síncrona de la
transacción {2} en {0} ({1}). Este miembro ya no está activo.
Descripción: se ha detectado un
problema que ha impedido a la
duplicación síncrona completarse satisfactoriamente.
Respuesta del usuario: revise los
mensajes previos en el archivo de anotaciones cronológicas para ayudar
al diagnóstico del problema.
Es posible que deba detener y reiniciar el
servidor especificado.

CWOBJ1518E: se ha producido una
excepción al intentar
comprometer la transacción de duplicación ({0}) para la transacción principal ({1}) en
Duplicación ({2}): {3}.
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ1519E: se ha producido una
excepción al intentar
retrotraer LogSequences para la duplicación ({0}): {1}.
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ1610W: intente restablecer un
clúster nulo para {0}.
Descripción: los datos de clúster del
grupo de duplicación no están disponibles.
Respuesta del usuario: ninguna.

CWOBJ1611I: el clúster del grupo de
duplicación {0} está preparado para la actividad.
Descripción: ahora el clúster del
grupo de duplicación puede aceptar peticiones.
Respuesta del usuario: ninguna.

CWOBJ1612I: el clúster del grupo de
duplicación {0} no está preparado para la actividad.
Descripción: ahora el clúster del
grupo de duplicación no puede aceptar peticiones.
Respuesta del usuario: ninguna.

CWOBJ1620I: sustituyendo el objetivo
por un error en el direccionamiento de la petición debido a
cambios en el servidor. El nuevo destino es {0}.
Descripción: el destino del
direccionamiento anterior se ha sustituido por un nuevo destino.
Respuesta del usuario: si el grupo de
duplicación deseado no se encuentra en
servicio, debe reactivarlo.

CWOBJ1630I: el grupo de duplicación
no puede satisfacer esta petición {0}.
Descripción: el direccionamiento
se ha rechazado debido a que el servicio no está disponible,
como en una reacción en cadena.
Respuesta del usuario: entrada
informativa.

CWOBJ1632E: la petición original no
tiene un ID válido; no hay forma de reenviar
esta petición.
Descripción: no hay forma de reenviar
esta petición porque la petición original
no tiene un ID válido.
Respuesta del usuario: informe al
soporte técnico de IBM

CWOBJ1634I: el direccionador no puede
encontrar el destino de reenvío; se utilizará el reenvío a ciegas.
Descripción: el direccionador no
puede encontrar el destino de reenvío.
Respuesta del usuario: ninguna.

CWOBJ1660I: el miembro del grupo de
duplicación ha cambiado.
Este servidor
ya no aloja lo que se le está solicitando. La petición original es {0}.
Descripción: el miembro del grupo de
duplicación ha cambiado.
Respuesta del usuario: si el grupo de
duplicación deseado no se encuentra en servicio,
debe reactivarlo.

CWOBJ1661I: los datos de clúster se
han actualizado para el grupo de duplicación:
{0}
Descripción: los datos del
clúster se han actualizado
Respuesta del usuario: ninguna.

CWOBJ1663E: el direccionador del
servidor no puede verificar el direccionamiento del servidor para {0},
porque los datos del clúster para este grupo de duplicación son nulos en el servidor.
Descripción: no hay datos de clúster
del grupo de duplicación
disponibles para verificar.
Respuesta del usuario: informe al
soporte técnico de IBM

CWOBJ1668W: la petición está yendo al
servidor que no se ha iniciado por completo.
Descripción: el inicio del servidor
tarde entre 60 y 120 segundos.
La petición se
reintentará de forma automática si así se ha configurado (la acción por omisión es que la
petición se
reintentará de forma automática).
Respuesta del usuario: ajuste la
configuración o inicie los clientes entre 60 y 120
segundos después de iniciar los servidores.

CWOBJ1680W: El tiempo excedido de la
conexión TCP es inferior al
retryInterval * max(startupRetries, maxRetries), de forma que existe la posibilidad de
que la
conexión excederá el tiempo de espera.
Descripción: el tiempo excedido de la
conexión TCP debe ser mayor
que retryInterval * max(startupRetries, maxRetries).
Respuesta del usuario: ajuste la
configuración.

CWOBJ1682W: el tiempo de espera
configurado para las transacciones es menor que maxForwards *
retryInterval * max(startupRetries, maxRetries), de forma que existe la posibilidad de
que la
transacción excederá el tiempo de espera.
Descripción: el tiempo excedido de la
transacción debe ser mayor que
maxForwards * retryInterval * max(startupRetries, maxRetries).
Respuesta del usuario: ajuste la
configuración.

CWOBJ1700I:
el HAManager autónomo
se ha inicializado con el coregroup {0}.
Descripción: el HAManager autónomo
se ha inicializado satisfactoriamente.
Respuesta del usuario: ninguna.

CWOBJ1701I: el HAManager autónomo ya
está inicializado.
Descripción: el HAManager autónomo
ya se ha inicializado satisfactoriamente.
Respuesta del usuario: ninguna.

CWOBJ1702E: el HAManager autónomo
no se ha inicializado, por lo que no puede iniciarse.
Descripción: el HAManager autónomo no
se ha inicializado.
Respuesta del usuario:
debe inicializarlo antes de iniciarlo.

CWOBJ1710I: el HAManager autónomo
se ha inicializado satisfactoriamente.
Descripción: el HAManager autónomo
se ha inicializado satisfactoriamente.
Respuesta del usuario: ninguna.

CWOBJ1711I: el HAManager autónomo ya
se ha iniciado satisfactoriamente.
Descripción: el HAManager autónomo
ya se ha iniciado satisfactoriamente.
Respuesta del usuario: ninguna.

CWOBJ1712E: el HAManager autónomo
no se ha iniciado.
Descripción: el HAManager autónomo no
se ha iniciado.
Respuesta del usuario: debe
inicializarlo antes de usarlo.

CWOBJ1713E: no se ha podido iniciar
el HAManager autónomo.
Descripción: el
HAManager autónomo no se ha podido iniciar.
Respuesta del usuario: compruebe si
los puertos
ya se están utilizando.

CWOBJ1720I: el controlador HAManager
ha detectado que
el servidor ObjectGrid está en el entorno WebSphere, utilizando HAManager de WebSphere
en lugar de inicializar e iniciar el HAManager autónomo.
Descripción: el servidor ObjectGrid
está ejecutándose en el entorno WebSphere.
Respuesta del usuario: ninguna.

CWOBJ1730I: el controlador HAManager
ha detectado
que el HAManager externo de WebSphere es nulo.
Descripción: no puede obtener el
HAManager externo de WebSphere.
Respuesta del usuario: ninguna.

CWOBJ1790I: debe inicializar e
iniciar el HAManager autónomo.
Descripción: no puede obtener el
HAManager externo de WebSphere.
Debe iniciar el HAManager autónomo.
Respuesta del usuario: ninguna.

CWOBJ1792I: el número máximo de
hebras es {0} y el número mínimo
de hebras es {1}.

Descripción: configure la
agrupación de hebras.
Respuesta del usuario: entrada
informativa.

CWOBJ1800I: se necesita el reenvío
para la petición {0} con respuesta de {1}.
Descripción: se necesita
reenviar el direccionamiento.
Respuesta del usuario: ninguna.
Se resuelve de forma automática.

CWOBJ1810I: se necesita el reenvío
para la respuesta {0}.
Descripción: se necesita el reenvío
para la respuesta.
Respuesta del usuario: ninguna.

CWOBJ1811E: se necesita el reenvío,
pero
no se ha encontrado la petición original.
Descripción: se necesita el reenvío,
pero
no se ha encontrado la petición original.
Respuesta del usuario: ninguna.

CWOBJ1820E: la petición de
reenvío no dispone de un
identificador del grupo de duplicación.
Descripción: no hay ningún
identificador del grupo de duplicación
en esta petición de reenvío.
Respuesta del usuario: póngase en
contacto con el soporte técnico de IBM

CWOBJ1870I: el servicio del servidor
no está disponible para la respuesta {0}.
Descripción: el servicio del servidor
no está disponible a una reacción en cadena u otro tipo de sucesos.
Respuesta del usuario: active al
menos el número mínimo de servidores.

CWOBJ1871E: se ha detectado que un
servicio no está disponible, se ha recibido una respuesta nula,
no hay forma de reintentarlo.
Descripción: se ha recibido una
respuesta nula del servicio que no está disponible
Respuesta del usuario: póngase en
contacto con el soporte técnico de IBM

CWOBJ1872I: el servicio no está
disponible y ha dado una respuesta de {0}.
Descripción: el servicio no está
disponible
Respuesta del usuario: active al
menos el número mínimo de
servidores o compruebe si el inicio del servidor ha sido satisfactorio.

CWOBJ1890I: se ha producido una
petición de redireccionamiento {0} debido a
que un servidor no responde.
Descripción: la petición para el
servidor deseado no se ha podido
completar. La petición se ha redireccionado a otro servidor.
Respuesta del usuario: ninguna. Se
resuelve de forma automática. Si el
grupo de duplicación está fuera de servicio, debe reactivarlo.

CWOBJ1891E: ningún servidor
está disponible en el grupo de duplicación {0}.
Descripción: los servidores no
se habían iniciado o no han

funcionado. No están disponibles.
Respuesta del usuario: si el grupo de duplicación deseado no se encuentra en servicio, debe reactivarlo.

CWOBJ1898W: se necesita el reenvío, pero el direccionador no puede encontrar un nuevo destino disponible para la respuesta {0}
Descripción: el servicio no está disponible.
Respuesta del usuario: haga disponible el servicio.

CWOBJ1899W: se necesita el reenvío, pero el direccionador no puede encontrar un grupo de duplicación correcto para la respuesta {0}
Descripción: se ha perdido el ID del grupo de duplicación.
Respuesta del usuario: póngase en contacto con el soporte técnico de IBM

CWOBJ1900I: se ha inicializado el servicio de llamada de procedimiento remoto del servidor de cliente.
Descripción: se ha inicializado el servicio de llamada de procedimiento remoto del servidor de cliente.
Respuesta del usuario: ninguna.

CWOBJ1901I: se ha iniciado el servicio de llamada de procedimiento remoto del servidor de cliente.
Descripción: se ha iniciado el servicio de llamada de procedimiento remoto del servidor de cliente.
Respuesta del usuario: ninguna.

CWOBJ1902I: las hebras de manejador de llamada de procedimiento remoto del servidor de cliente se han iniciado.
Descripción: las hebras de manejador de llamada de procedimiento remoto del servidor de cliente se han iniciado.
Respuesta del usuario: ninguna.

CWOBJ1903I: se ha inicializado el servicio de red de configuración.
Descripción: se ha inicializado el servicio de red de configuración.
Respuesta del usuario: ninguna.

CWOBJ1904I: se ha iniciado el servicio de red de configuración.
Descripción: se ha iniciado el servicio de red de configuración.
Respuesta del usuario: ninguna.

CWOBJ1905I: se ha iniciado el manejador de configuración.
Descripción: se ha iniciado el manejador de configuración.
Respuesta del usuario: ninguna.

CWOBJ1913I: se ha inicializado el servicio de red de la administración del sistema.
Descripción: se ha inicializado el servicio de red de la administración del sistema.
Respuesta del usuario: ninguna.

CWOBJ1914I: se ha iniciado el servicio de red de la administración del sistema.
Descripción: se ha iniciado el servicio de red de la administración del sistema.
Respuesta del usuario: ninguna.

CWOBJ1915I: se ha iniciado el
manejador de la administración del sistema.
Descripción: se ha iniciado el
manejador de la administración del sistema.
Respuesta del usuario: ninguna.

CWOBJ2000E: no hay ningún miembro en
este grupo de duplicación {0}.
Descripción: no hay ningún miembro en
este grupo de duplicación.
Respuesta del usuario: compruebe si
los servidores se han iniciado o si los datos están disponibles

CWOBJ2001W: no hay ningún miembro
disponible en este grupo de duplicación {0}.
Descripción: no hay ningún miembro
disponible en este grupo de duplicación.
Respuesta del usuario: compruebe si
el servicio de servidor está disponible

CWOBJ2002W: no hay ninguna tabla de
direccionamiento disponible para este grupo de duplicación {0}.
Descripción:: no hay ninguna tabla de
direccionamiento disponible para este grupo de duplicación.
Respuesta del usuario: compruebe si
los clientes han aportado la tabla de direccionamiento

CWOBJ2003I: no se encuentra la
antememoria de direccionamiento para la clave de antememoria {0}, se creará
una nueva antememoria de direccionamiento.
Descripción: primera vez que se
realizan cambios en el direccionamiento o en el clúster.
Respuesta del usuario: ninguna.

CWOBJ2010E: el destino de esta
petición es nulo.
Descripción: la petición no ha
llegado con la información de destino.
Respuesta del usuario: póngase en
contacto con el soporte técnico de IBM

WOBJ2060I: el cliente ha recibido una
nueva versión del clúster de grupo de duplicación {0}.
Descripción: el cliente ha recibido
una nueva versión del clúster de grupo de duplicación.
Respuesta del usuario: ninguna.

CWOBJ2068I: el control de
accesibilidad ha detectado un problema en el miembro del grupo de
duplicación {0}.
Descripción: no se puede acceder a
algún servidor, se encargará de ello el mecanismo de accesibilidad.
Respuesta del usuario: ninguna.

CWOBJ2069I: el temporizador de
control de accesibilidad libera el miembro de grupo de duplicación {0}.
Descripción: este miembro está
disponible para direccionamiento.
Respuesta del usuario: ninguna.

p **CWOBJ2086I**: el control de hebras de
direccionamiento se ha activado debido a la sobrecarga para
el grupo de duplicación {0}.
Descripción: el control de hebras se
ha activado.
Respuesta del usuario: ninguna.

CWOBJ2088I: el control de
accesibilidad se ha activado para regular la
disponibilidad del servidor para el grupo de duplicación {0}.
Descripción: se ha activado la
accesibilidad.
Respuesta del usuario: ninguna.

CWOBJ2090W: no se encuentra la tabla
de direccionamiento para el grupo de duplicación {0}.
Descripción: el clúster del grupo de
duplicación es nulo.
Respuesta del usuario: ninguna.

CWOBJ2091W: la tabla de
direccionamiento no es nula, pero no contiene servidores
para el grupo de duplicación {0}.
Descripción: el clúster del grupo de
duplicación está vacío.
Respuesta del usuario: ninguna.

CWOBJ2092I: la tabla de
direccionamiento es nula en el tiempo de ejecución para el grupo de duplicación {0}.
Descripción: se obtiene la tabla de
direccionamiento del tiempo de ejecución.
Respuesta del usuario: ninguna.

CWOBJ2093I: la tabla de
direccionamiento no es nula en el almacenamiento del clúster del grupo de duplicación
para el grupo de duplicación {0}
Descripción: se obtiene la tabla de
direccionamiento del almacenamiento del clúster.
Respuesta del usuario: ninguna.

CWOBJ2096I: la tabla de
direccionamiento se ha obtenido del almacenamiento del clúster del grupo de duplicación
para el grupo de duplicación {0}.
Descripción: se ha obtenido el
clúster del grupo de duplicación del
almacenamiento del clúster del grupo de duplicación.
Respuesta del usuario: ninguna.

CWOBJ2097I: el direccionamiento se
basa en un algoritmo por turno circular para el grupo de duplicación {0}.
Descripción: el direccionamiento se
basa en un algoritmo por turno circular.
Respuesta del usuario: ninguna.

CWOBJ2098I: el direccionamiento se
basa en la selección aleatoria para el grupo de duplicación {0}.
Descripción: el direccionamiento se
basa en la selección aleatoria.
Respuesta del usuario: ninguna.

CWOBJ2100I: la conexión ({0}) no
tiene actividad y no puede reutilizarse.
Descripción: la conexión no tiene
actividad.
Respuesta del usuario: ninguna.

CWOBJ2101W: la conexión no puede
adquirirse una vez transcurrido el tiempo de espera máximo.
Descripción: no quedan muchas
conexiones en la agrupación.
Respuesta del usuario: aumente el
número máximo de conexiones de la configuración.

CWOBJ1600I: el servicio
ManagementGateway se ha iniciado en el puerto ({0}).
Descripción: el servicio
ManagementGateway está listo para procesar peticiones.
Respuesta del usuario: el servicio
ManagementGateway está disponible.

CWOBJ1601E: error en el inicio del
servicio ManagementGateway en el puerto ({0}).
Descripción: error al iniciarse el
servicio ManagementGateway.
Respuesta del usuario: asegúrese de
que el puerto especificado no esté siendo utilizado.

CWOBJ1602E: error en la conexión del
servicio ManagementGateway con el servidor en
({0}):({1}).
Descripción: error en la conexión del
servicio ManagementGateway al servidor.
Respuesta del usuario: asegúrese de
que el servidor está ejecutándose.

CWOBJ1603E: el servicio de gestión no
ha podido responder a la petición remota ({0}).
Descripción: CMSG0001
Respuesta del usuario: CMSG0002

CWOBJ2400E:
Configuración no válida: la correlación de respaldo {0} es un
miembro de más de un conjunto de correlaciones.
Descripción: un backingMap sólo puede
pertenecer a un conjunto de correlaciones.
Respuesta del usuario: edite el
archivo XML del clúster de forma que cada correlación de respaldo
sólo pertenezca a un conjunto de correlaciones.

CWOBJ2401E: Configuración no válida:
la correlación de respaldo {0} en la
ObjectGrid distribuida {1} no está en el conjunto de correlaciones.
Descripción: cada correlación de
respaldo de una ObjectGrid distribuida debe
colocarse en un conjunto de correlaciones.
Respuesta del usuario: edite el
archivo XML del clúster de forma que cada correlación de respaldo
de una ObjectGrid distribuida sólo pertenezca a un conjunto de correlaciones.

CWOBJ2402E: Configuración no válida:
el conjunto de correlaciones hace referencia
a una correlación {0}. Esta correlación de respaldo no existe en el archivo XML de la
ObjectGrid.
Descripción: cada correlación de un
conjunto de correlaciones debe hacer referencia a
una correlación de respaldo del archivo XML de ObjectGrid.
Respuesta del usuario: edite el
archivo XML de forma que cada correlación del
conjunto de correlaciones haga referencia a una correlación de respaldo del archivo XML
de ObjectGrid.

CWOBJ2403E: el archivo XML no es
válido.
Se ha detectado un problema con
{0} en la línea {1}. El mensaje de error es {2}.
Descripción: el archivo XML no se
ajusta al esquema.
Respuesta del usuario: edite el
archivo XML de forma que se ajuste al esquema.

CWOBJ2404W: el valor especificado
para {0} es {1}.
Este es un valor no válido.
{0} no se establecerá.
Descripción: el valor de este
atributo de configuración no es válido.
Respuesta del usuario: establezca el
atributo de configuración en un valor adecuado en el
archivo XML.

CWOBJ2405E: la referencia
objectgrid-binding {0} del archivo XML del clúster
no hace referencia a una ObjectGrid válida del archivo XML de ObjectGrid.
Descripción: cada uno de los
objectgrid-bindings debe hacer referencia a
una ObjectGrid del archivo XML de ObjectGrid.

Respuesta del usuario: edite los
archivos XML de forma que el objectgrid-binding
del XML del clúster haga referencia a una ObjectGrid válida en el XML de ObjectGrid.

CWOBJ2500E: error al iniciar el
servidor ObjectGrid {0}.
Descripción: el servidor ObjectGrid
no se ha iniciado adecuadamente.
Respuesta del usuario: compruebe si
hay excepciones en el archivo de anotaciones cronológicas.

CWOBJ2501I: iniciando el servidor
ObjectGrid {0}.
Descripción: un servidor ObjectGrid
se está iniciando.
Respuesta del usuario: ninguna.

CWOBJ2502I: iniciando el servidor
ObjectGrid mediante la URL del archivo XML de ObjectGrid
"{0}" y el URL del archivo XML del clúster "{1}".
Descripción: un servidor ObjectGrid
se está iniciando utilizando un archivo XML del clúster
y un archivo XML de ObjectGrid.
Respuesta del usuario: ninguna.

CWOBJ2503I: inicio de rutina de carga
en un servidor Objectgrid similar en el sistema principal {0}
y el puerto {1}.
Descripción: este servidor ObjectGrid
iniciará la rutina de carga en un servidor similar
para recuperar la información necesaria para iniciarse.
Respuesta del usuario: ninguna.

COBJ2504I: se está intentando iniciar
la rutina de carga en un servidor ObjectGrid similar mediante
los siguientes sistemas principales y puertos "{0}".
Descripción: este servidor ObjectGrid
utilizará la lista de sistemas principales y puertos
que se proporcionan en un intento de conectar con un servidor ObjectGrid similar.
Respuesta del usuario: ninguna.

CWOBJ2505I: se está intentando
iniciar la rutina de carga en un servidor ObjectGrid similar mediante
el URL del archivo XML del clúster "{0}".
Descripción: este servidor ObjectGrid
utilizará la lista de servidores del
archivo XML del clúster en un intento de conectar con un servidor ObjectGrid similar.
Respuesta del usuario: ninguna.

CWOBJ2506I: el rastreo se está
anotando cronológicamente en {0}.
Descripción: el archivo de rastreo se
ha establecido en la línea de mandatos.
Respuesta del usuario: consulte el
archivo de rastreo especificado para el
rastreo de inicio del servidor ObjectGrid.

CWOBJ2507I: la especificación de
rastreo se establece en {0}.
Descripción: la especificación de
rastreo se ha establecido en la línea de mandatos.
Respuesta del usuario: ninguna.

CWOBJ2508I: se ha especificado un
archivo de propiedades de seguridad "{0}" y
se utilizará para iniciar el servidor.
Descripción: se ha proporcionado un
archivo de propiedades de seguridad para iniciar un
servidor seguro.
Respuesta del usuario: ninguna.

CWOBJ2509E: se ha excedido el tiempo
de espera después de esperar {0} segundos a que se iniciara el servidor.
Descripción: el servidor ObjectGrid
no se ha iniciado en el intervalo de tiempo de espera.
Respuesta del usuario: compruebe si
hay excepciones en el archivo de anotaciones cronológicas.

CWOBJ2510I: deteniendo el servidor
ObjectGrid {0}.
Descripción: deteniendo el servidor
ObjectGrid.
Respuesta del usuario:

CWOBJ2511I: esperando a que se
detenga el servidor.
Descripción: esperando a que se
detenga el servidor ObjectGrid.
Respuesta del usuario: ninguna.

CWOBJ2512I: se ha detenido el
servidor ObjectGrid {0}.
Descripción: se ha detenido el
servidor ObjectGrid.
Respuesta del usuario: ninguna.

CWOBJ2513E: se ha excedido el tiempo
de espera después de esperar {0} segundos a que se detuviera el servidor.
Descripción: el servidor ObjectGrid
no se ha detenido en el intervalo de tiempo de espera.
Respuesta del usuario: compruebe si
hay excepciones en el archivo de anotaciones cronológicas.

CWOBJ2514I: esperando a que se
complete la activación del servidor ObjectGrid.
Descripción: el servidor ObjectGrid
se ha iniciado.
Esperando a que el servidor
complete la activación.
Respuesta del usuario: ninguna.

CWOBJ2515E: los argumentos que se
proporcionan no son válidos.
Éstos son los argumentos válidos.
{0}{1}
Descripción: los argumentos
proporcionados en este script no son válidos.
Respuesta del usuario: especifique
argumentos válidos.

CWOBJ2516I: el servidor ObjectGrid ha
completado la activación.
Descripción: el servidor ObjectGrid
está activo y listo para procesar las peticiones.
Respuesta del usuario: ninguna.

CWOBJ2517I: se ha iniciado
satisfactoriamente la rutina de carga con el servidor Objectgrid similar
en el sistema principal {0} y el puerto {1}.
Descripción: este servidor ObjectGrid
ha iniciado satisfactoriamente la rutina de carga
con un servidor similar para recuperar la información necesaria para iniciar este
servidor.
Respuesta del usuario: ninguna.

CWOBJ2407W: Es posible que no se haya
establecido la propiedad {0} de la clase de plug-in {1}.
La excepción es {2}.
Descripción: no se ha podido
establecer la propiedad de este plug-in.
Respuesta del usuario: consulte la
excepción si desea más información.

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 éste 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 y de servicio

Los siguientes términos son marcas registradas de IBM Corporation en los Estados Unidos y/o 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 los Estados Unidos y/o en otros países.

LINUX es una marca registrada de Linus Torvalds en los Estados Unidos y/o en otros países.

Microsoft, Windows, Windows NT y el logotipo de Windows son marcas registradas de Microsoft Corporation en los Estados Unidos y/o en otros países.

UNIX es una marca registrada de The Open Group en los Estados Unidos y en otros países.

Otros nombres de compañías, productos y servicios pueden ser marcas registradas o de servicio de terceros.