IBM WebSphere eXtreme Scale Versión 7.1.1
Versión 7 Release 1

Guía de programación

21 de noviembre de 2011

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

Contenido

Figuras

Tablas

Acerca de la *Guía de programación*

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

Biblioteca de WebSphere eXtreme Scale

La biblioteca de WebSphere eXtreme Scale contiene las siguientes publicaciones:

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

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

También puede acceder a la misma información en esta biblioteca en el Information Center de WebSphere eXtreme Scale Versión 7.1.1.

Utilización fuera de línea de los manuales

Todos los manuales de la biblioteca de WebSphere eXtreme Scale contienen enlaces al Information Center, con el siguiente URL raíz: `http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r1m1`. Estos enlaces le llevan directamente a la información relacionada. Sin embargo, si está trabajando fuera de línea y se encuentra con uno de estos enlaces, puede buscar el título del enlace en los otros manuales de la biblioteca. La documentación de la API, el glosario y los mensajes de referencia no están disponibles en los manuales en formato PDF.

Quién debe utilizar esta publicación

Esta publicación está especialmente indicada para los desarrolladores de aplicaciones.

Cómo obtener actualizaciones de esta publicación

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

Envío de comentarios

Póngase en contacto con el equipo de documentación. ¿Ha encontrado lo que necesita? ¿Ha sido la información precisa y completa? Envíe sus comentarios sobre

esta documentación mediante correo electrónico a wasdoc@us.ibm.com.

Capítulo 1. Guías de aprendizaje

Puede utilizar guías de aprendizaje como ayuda para comprender los escenarios de uso del producto, incluido el gestor de entidades, las consultas y la seguridad.

Guía de aprendizaje: Consulta de una cuadrícula de datos local en memoria

Puede desarrollar un ObjectGrid local en memoria que puede almacenar información de pedidos para un sitio web y utilizar la API ObjectQuery para consultar la cuadrícula de datos.

Antes de empezar

Asegúrese de que el archivo `objectgrid.jar` está en la classpath.

Acerca de esta tarea

Cada paso de la guía de aprendizaje se basa en el paso anterior. Siga cada uno de los pasos para crear una aplicación sencilla de Java Platform, Standard Edition Versión 5 o posterior que utilice una cuadrícula de datos local en memoria.

Guía de aprendizaje de ObjectQuery - Paso 1

Con los siguientes pasos, podrá seguir desarrollando un ObjectGrid local en memoria que almacena la información de pedidos para una tienda al detalle en línea mediante las API ObjectMap. Defina un esquema para la correlación y ejecute una consulta en la correlación.

Procedimiento

1. Cree un ObjectGrid con un esquema de correlación.

   Cree un ObjectGrid con un esquema de correlación para la correlación y luego inserte un objeto en la memoria y más adelante recupérelo utilizando una consulta simple.

   OrderBean.java

   ```
   public class OrderBean implements Serializable {
       String orderNumber;
       java.util.Date date;
       String customerName;
       String itemName;
       int quantity;
       double price;
   }
   ```

2. Defina la clave primaria.

   El código anterior muestra un objeto OrderBean. Este objeto implementa la interfaz java.io.Serializable porque todos los objetos de la memoria caché se deben poder serializar (de forma predeterminada).

El atributo orderNumber es la clave primaria del objeto. El siguiente programa de ejemplo se puede ejecutar en una modalidad autónoma. Debe seguir esta guía de aprendizaje en un proyecto Eclipse Java que tenga el archivo `objectgrid.jar` añadido a la classpath.

Application.java

```
package querytutorial.basic.step1;

import java.util.Iterator;

import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.config.QueryConfig;
import com.ibm.websphere.objectgrid.config.QueryMapping;
import com.ibm.websphere.objectgrid.query.ObjectQuery;

public class Application
{

    static public void main(String [] args) throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");

        // Definir el esquema
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping("Order", OrderBean.class.getName(),
     "orderNumber", QueryMapping.FIELD_ACCESS));
                og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");

        s.begin();
        OrderBean o = new OrderBean();
        o.customerName = "John Smith";
        o.date = new java.util.Date(System.currentTimeMillis());
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.put(o.orderNumber, o);
        s.commit();

        s.begin();
        ObjectQuery query = s.createObjectQuery("SELECT o FROM Order o WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        o = (OrderBean) result.next();
        System.out.println("Found order for customer: " + o.customerName);
        s.commit();
    }
}
```

Esta aplicación eXtreme Scale primero inicializa un ObjectGrid local con un nombre generado automáticamente. A continuación, la aplicación crea BackingMap y QueryConfig que define qué tipo Java se asocia a la correlación, el nombre del campo que es la clave primaria de la correlación y cómo acceder a los datos del objeto. A continuación, puede obtener una sesión para obtener la instancia de ObjectMap e insertar un objeto OrderBean en la correlación en una transacción.

Después de que se confirmen los datos en la memoria caché, puede utilizar ObjectQuery para encontrar el OrderBean utilizando uno cualquiera de los campos persistentes de la clase. Los campos persistentes son aquellos que no tienen el modificador transient. Puesto que no ha definido ningún índice en BackingMap, ObjectQuery debe explorar cada objeto de la correlación utilizando el reflejo Java.

Qué hacer a continuación

"Guía de aprendizaje de ObjectQuery - Paso 2" en la página 3 demuestra cómo se puede utilizar un índice para optimizar la consulta.

Guía de aprendizaje de ObjectQuery - Paso 2

Con los siguientes pasos, puede seguir creando una ObjectGrid con una correlación y un índice, junto con un esquema para la correlación. A continuación, puede insertar un objeto en la memoria caché y, posteriormente, recuperarlo mediante una simple consulta.

Antes de empezar

Asegúrese de que ha completado "Guía de aprendizaje de ObjectQuery - Paso 1" en la página 1 antes de continuar con este paso de la guía de aprendizaje.

Procedimiento

Esquema e índice

Application.java

```
// Crear un índice
    HashIndex idx= new HashIndex();
    idx.setName("theItemName");
    idx.setAttributeName("itemName");
    idx.setRangeIndex(true);
    idx.setFieldAccessAttribute(true);
    orderBMap.addMapIndexPlugin(idx);
}
```

El índice debe ser una instancia com.ibm.websphere.objectgrid.plugins.index.HashIndex con los siguientes valores:

- El Name es arbitrario, pero debe ser exclusivo para una BackingMap dad.
- El AttributeName es el nombre del campo o propiedad de bean que utilice el motor para realizar una introspección de la clase. En este caso, es el nombre del campo para el que ha creado un índice.
- RangeIndex debe ser siempre true.
- FieldAccessAttribute debe coincidir con el valor establecido en el objeto QueryMapping cuando se creó el esquema de consulta. En este caso, se accede al objeto Java utilizando los campos directamente.

Cuando se ejecuta una consulta que aplica un filtro en el campo itemName, el motor de consulta utiliza automáticamente el índice definido. El uso del índice permite que la consulta se ejecute mucho más rápido y no es necesario una exploración de la correlación. El siguiente paso demuestra cómo se puede utilizar un índice para optimizar la consulta.
Paso siguiente

Guía de aprendizaje de ObjectQuery - Paso 3

Con el paso siguiente, puede crear un ObjectGrid con dos correlaciones y un esquema para las correlaciones con una relación y, más adelante, insertar objetos en la memoria caché y, posteriormente, recuperarlos utilizando una consulta simple.

Antes de empezar

Asegúrese de haber completado el apartado "Guía de aprendizaje de ObjectQuery - Paso 2" antes de llevar a cabo este paso.

Acerca de esta tarea

En este ejemplo, hay dos correlaciones, cada una con un único tipo Java correlacionado. La correlación Orden tiene objetos OrderBean y la correlación Customer tiene objetos CustomerBean.

Procedimiento

Defina las correlaciones con una relación.

OrderBean.java

```
public class OrderBean implements Serializable {
    String orderNumber;
    java.util.Date date;
    String customerId;
    String itemName;
    int quantity;
    double price;
}
```

El OrderBean ya no tiene customerName. En su lugar, tiene el customerId, que es la clave primaria para el objeto CustomerBean y la correlación Customer.

CustomerBean.java

```
public class CustomerBean implements Serializable{
    private static final long serialVersionUID = 1L;
    String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

La relación entre los tipos o correlaciones es la siguiente:

Application.java

```
public class Application
{

    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");
        og.defineMap("Customer");

        // Definir el esquema
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping(
            "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping(
            "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryRelationship(new QueryRelationship(
             OrderBean.class.getName(), CustomerBean.class.getName(), "customerId", null));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");
        ObjectMap custMap = s.getMap("Customer");

        s.begin();
        CustomerBean cust = new CustomerBean();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        custMap.insert(cust.id, cust);

        OrderBean o = new OrderBean();
        o.customerId = cust.id;
```

```
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.insert(o.orderNumber, o);
        s.commit();

        s.begin();
        ObjectQuery query = s.createObjectQuery(
            "SELECT c FROM Order o JOIN o.customerId as c WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        cust = (CustomerBean) result.next();
        System.out.println("Found order for customer: " + cust.firstName + " " + cust.surname);
        s.commit();
    }
}
```

El XML equivalente en el descriptor de despliegue de ObjectGrid es el siguiente:

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

      <querySchema>
         <mapSchemas>
           <mapSchema
             mapName="Order"
             valueClass="com.mycompany.OrderBean"
             primaryKeyField="orderNumber"
             accessType="FIELD"/>
           <mapSchema
             mapName="Customer"
             valueClass="com.mycompany.CustomerBean"
             primaryKeyField="id"
             accessType="FIELD"/>
         </mapSchemas>
         <relationships>
           <relationship
             source="com.mycompany.OrderBean"
             target="com.mycompany.CustomerBean"
             relationField="customerId"/>
         </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Qué hacer a continuación

"Guía de aprendizaje de ObjectQuery - Paso 4", amplía el paso actual incluyendo los objetos de acceso de propiedad y campo y las relaciones adicionales.

Guía de aprendizaje de ObjectQuery - Paso 4

El siguiente paso muestra cómo crear un ObjectGrid con cuatro correlaciones y un esquema para las correlaciones con varias relaciones unidireccionales y bidireccionales. Puede insertar objetos en la memoria caché y, posteriormente, recuperarlos mediante varias consultas.

Antes de empezar

Asegúrese de haber completado el apartado "Guía de aprendizaje de ObjectQuery - Paso 3" en la página 3 antes de continuar con el paso actual.

Procedimiento

Varias relaciones de correlaciones

OrderBean.java

```
public class OrderBean implements Serializable {
    String orderNumber;
    java.util.Date date;
    String customerId;
    String itemName;
    int quantity;
    double price;
}
```

Como en el paso anterior, OrderBean ya no tiene customerName. En su lugar, tiene el customerId, que es la clave primaria para el objeto CustomerBean y la correlación Customer.

CustomerBean.java

```
public class CustomerBean implements Serializable{
    private static final long serialVersionUID = 1L;
    String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

Después de crear las clases especificadas más arriba, puede ejecutar la aplicación siguiente.

Application.java

```
public class Application
{

    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og = ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.defineMap("Order");
        og.defineMap("Customer");

        // Definir el esquema
        QueryConfig queryCfg = new QueryConfig();
        queryCfg.addQueryMapping(new QueryMapping(
            "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryMapping(new QueryMapping(
            "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
        queryCfg.addQueryRelationship(new QueryRelationship(
             OrderBean.class.getName(), CustomerBean.class.getName(), "customerId", null));
        og.setQueryConfig(queryCfg);

        Session s = og.getSession();
        ObjectMap orderMap = s.getMap("Order");
        ObjectMap custMap = s.getMap("Customer");

        s.begin();
        CustomerBean cust = new CustomerBean();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        custMap.insert(cust.id, cust);

        OrderBean o = new OrderBean();
        o.customerId = cust.id;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;
        orderMap.insert(o.orderNumber, o);
        s.commit();
```

```
        s.begin();
        ObjectQuery query = s.createObjectQuery(
            "SELECT c FROM Order o JOIN o.customerId as c WHERE o.itemName='Widget'");
        Iterator result = query.getResultIterator();
        cust = (CustomerBean) result.next();
        System.out.println("Found order for customer: " + cust.firstName + " " + cust.surname);
        s.commit();
    }
}
```

El uso de la configuración XML siguiente (en el descriptor de despliegue de ObjectGrid) es equivalente al enfoque programático anterior.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="og1">
      <backingMap name="Order"/>
      <backingMap name="Customer"/>

      <querySchema>
         <mapSchemas>
           <mapSchema
             mapName="Order"
             valueClass="com.mycompany.OrderBean"
             primaryKeyField="orderNumber"
             accessType="FIELD"/>
           <mapSchema
             mapName="Customer"
             valueClass="com.mycompany.CustomerBean"
             primaryKeyField="id"
             accessType="FIELD"/>
         </mapSchemas>
         <relationships>
           <relationship
             source="com.mycompany.OrderBean"
             target="com.mycompany.CustomerBean"
             relationField="customerId"/>
         </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Guía de aprendizaje: Almacenamiento de información de pedidos en entidades

La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Antes de empezar

Asegúrese de que satisface los siguientes requisitos antes de empezar la guía de aprendizaje:

- Debe tener Java SE 5.
- Debe tener el archivo `objectgrid.jar` en la vía de acceso de clases.

Conceptos relacionados:

"Almacenamiento en memoria caché de objetos sin relaciones implicadas (API ObjectMap)" en la página 155
ObjectMaps son como correlaciones Java que permiten a los datos almacenarse como pares clave-valor. Los ObjectMap proporcionan un acercamiento sencillo e intuitivo para el almacenamiento de los datos de la aplicación. Un ObjectMap es ideal para almacenar en memoria caché los objetos que no tienen relaciones. Si hubiera relaciones de objeto, debería utilizar la API EntityManager.

"Ajuste del rendimiento de la interfaz EntityManager" en la página 468
La interfaz EntityManager separa las aplicaciones del estado alojado en su almacén de datos de cuadrícula de servidores.

"Almacenamiento en memoria caché de objetos y sus relaciones (API EntityManager)" en la página 166
La mayoría de los productos de memoria caché utilizan las API basadas en correlaciones para almacenar los datos como pares de clave-valor. La API ObjectMap y la memoria caché dinámica de WebSphere Application Server, entre otras, utilizan este enfoque. No obstante, las API basadas en correlaciones tienen limitaciones. La API EntityManager simplifica la interacción con la cuadrícula de datos proporcionando una forma fácil de declarar un gráfico complejo de objetos relacionados e interactuar con él.

"Gestor de entidades en un entorno distribuido" en la página 178
Puede utilizar la API EntityManager con un ObjectGrid local o en un entorno distribuido de eXtreme Scale. La diferencia principal es el modo de conectarse a este entorno remoto. Después de establecer una conexión, no hay ninguna diferencia entre el uso de un objeto Session o el uso de la API EntityManager.

"Interacción con EntityManager" en la página 183
Normalmente, las aplicaciones en primer lugar obtienen una referencia de ObjectGrid y, a continuación, una Session de dicha referencia para cada hebra. Las sesiones no pueden compartirse entre hebras. Hay disponible un método adicional en el elemento Session, el método getEntityManager. Este método devuelve una referencia a un gestor de entidades para utilizar para esta hebra. La interfaz EntityManager puede sustituir las interfaces Session y ObjectMap para todas las aplicaciones. Puede utilizar estas API EntityManager si el cliente tiene acceso a las clases de entidad definidas.

"Soporte de planes de captación de EntityManager" en la página 193
Un FetchPlan es la estrategia que el gestor de entidades utiliza para recuperar los objetos asociados si la aplicación tiene que acceder a las relaciones.

"Colas de consulta de entidades" en la página 198
Las colas de consulta permiten a las aplicaciones crear una cola calificada por una consulta en el servidor o en eXtreme Scale local para una entidad. Las entidades del resultado de la consulta se almacenan en esta cola. Actualmente, la cola de consulta sólo se admite en una correlación que utilice la estrategia de bloqueo pesimista.

Referencia relacionada:

"Agente de instrumentación de rendimiento de entidades" en la página 470
Puede mejorar el rendimiento de las entidades de acceso a campos habilitando el agente de instrumentación de WebSphere eXtreme Scale cuando se utiliza Java Development Kit (JDK) versión 1.5 o posterior.

"Definición de un esquema de entidad" en la página 169
Un ObjectGrid puede tener varios un número ilimitado de esquemas de entidades lógicas. Las entidades se definen utilizando las clases Java anotadas, XML o una combinación de XML y clases Java. Las entidades definidas se registran con un servidor eXtreme Scale y se enlazan a BackingMaps, índices y otros plug-ins.

"Métodos de devolución de llamada y escuchas de entidad" en la página 186
Se puede notificar a las aplicaciones cuando el estado de una entidad pasa de un estado a otro. Existen dos mecanismos de devolución de llamada para los sucesos de cambio de estado: los métodos de devolución de llamada de ciclo de vida que están definidos en una clase de entidad y se invocan siempre que cambia el estado de la entidad, y los escuchas de entidad, que son más generales porque el escucha de entidad se puede registrar en varias entidades.

"Ejemplos de escucha de entidad" en la página 190
Puede escribir EntityListeners según sus necesidades. A continuación se ofrecen varios scripts de ejemplo.

"Interfaz EntityTransaction" en la página 203
Puede utilizar la interfaz EntityTransaction para delimitar transacciones.

Información relacionada:

"Lección 2 de la guía de aprendizaje de iniciación: Creación de una aplicación cliente" en la página 63
Para insertar, suprimir, actualizar y recuperar datos de la cuadrícula de datos, debe escribir una aplicación de cliente. El ejemplo de iniciación incluye una aplicación cliente que puede utilizar para aprender a crear su propia aplicación cliente.

Guía de aprendizaje del gestor de entidades: creación de una clase de entidad

Cree un ObjectGrid local con una entidad creando una clase de entidad, registrando el tipo de entidad y almacenando una instancia de entidad en la memoria caché.

Procedimiento

1. Cree el objeto Order. Para identificar el objeto como una entidad ObjectGrid, añada la anotación @Entity. Cuando añade esta anotación, todos los atributos serializables del objeto persisten automáticamente en eXtreme Scale, salvo que utilice anotaciones en los atributos para alterar temporalmente los atributos. El atributo **orderNumber** se anota con `@Id` para indicar que este atributo es la clave primaria. A continuación se muestra un ejemplo de un objeto Order:

   Order.java

   ```
   @Entity
   public class Order
   {
       @Id String orderNumber;
       Date date;
       String customerName;
       String itemName;
       int quantity;
       double price;
   }
   ```

2. Ejecute la aplicación eXtreme Scale Hello World para demostrar las operaciones de entidad. El siguiente programa de ejemplo se puede emitir en modalidad autónoma para demostrar las operaciones de entidad. Utilice este programa en un proyecto Eclipse Java que tiene el archivo `objectgrid.jar` añadido a la classpath. A continuación se muestra un ejemplo de una aplicación Hello World simple que utiliza eXtreme Scale:

   Application.java

   ```
   package emtutorial.basic.step1;

   import com.ibm.websphere.objectgrid.ObjectGrid;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
   import com.ibm.websphere.objectgrid.Session;
   import com.ibm.websphere.objectgrid.em.EntityManager;
   ```

```
public class Application
{

    static public void main(String [] args)
        throws Exception
    {
        ObjectGrid og =
    ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
        og.registerEntities(new Class[] {Order.class});

        Session s = og.getSession();
        EntityManager em = s.getEntityManager();

        em.getTransaction().begin();

        Order o = new Order();
        o.customerName = "John Smith";
        o.date = new java.util.Date(System.currentTimeMillis());
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;

        em.persist(o);
        em.getTransaction().commit();

        em.getTransaction().begin();
        o = (Order)em.find(Order.class, "1");
        System.out.println("Found order for customer: " + o.customerName);
        em.getTransaction().commit();
    }
}
```

Esta aplicación de ejemplo lleva a cabo las siguientes operaciones:

a. Inicializa un eXtreme Scale local con un nombre generado automáticamente.
b. Registra las clases de entidad con la aplicación utilizando la API registerEntities, aunque no siempre necesario utilizar la API registerEntities.
c. Recupera un elemento Session y una referencia al gestor de entidades para Session.
d. Asocia cada Session de eXtreme Scale con un solo EntityManager y EntityTransaction. Ahora se utiliza EntityManager.
e. El método registerEntities crea un objeto BackingMap que se llama Order, y asocia los metadatos para el objeto Order con el objeto BackingMap. Estos metadatos incluyen los atributos de clave y no de clave, junto con los nombres y tipos de atributos.
f. Se inicia una transacción y ésta crea una instancia Order. La transacción se llena con algunos valores. Entonces la transacción se conserva utilizando el método EntityManager.persist, que identifica que la entidad está en espera de incluirse en la correlación asociada.
g. A continuación, la transacción se confirma y la entidad se incluye en la instancia ObjectMap.
h. Se ejecuta otra transacción y el objeto Order se recupera utilizando la clave 1. La difusión de tipo en el método EntityManager.find es necesaria. La prestación Java SE 5 no se utiliza para asegurar que el archivo `objectgrid.jar` funcione en una Máquina virtual Java Java SE Versión 5 o posterior.

Guía de aprendizaje del gestor de entidades: creación de relaciones de entidad

Crear una relación simple entre entidades creando dos clases de entidad con una relación, registrando las entidades con el ObjectGrid, y almacenando las instancias de entidades en la memoria caché.

Procedimiento

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

   Customer.java

   ```
   @Entity
   public class Customer
   {
       @Id String id;
       String firstName;
       String surname;
       String address;
       String phoneNumber;
   }
   ```

   Esta clase incluye información sobre el cliente, como el nombre, la dirección y el número de teléfono.

2. Cree el objeto Order, que es parecido al objeto Order del tema "Guía de aprendizaje del gestor de entidades: creación de una clase de entidad" en la página 9. A continuación se muestra un ejemplo del objeto Order:

   Order.java

   ```
   @Entity
   public class Order
   {
       @Id String orderNumber;
       Date date;
       @ManyToOne(cascade=CascadeType.PERSIST) Customer customer;
       String itemName;
       int quantity;
       double price;
   }
   ```

   En este ejemplo, una referencia a un objeto Customer sustituye el atributo customerName. La referencia tiene una anotación que indica una relación de muchos con uno. Una relación de muchos con uno indica que cada pedido tiene un cliente, aunque varios pedidos pueden hacer referencia al mismo cliente. El modificador de anotación en cascada indica que si el gestor de entidades persiste el objeto Order, también debe persistir el objeto Customer. Si decide no establecer la opción de persistencia en cascada, que es la opción predeterminada, debe persistir manualmente el objeto Customer con el objeto Order.

3. Utilizando las entidades, defina las correlaciones para la instancia de ObjectGrid. Cada correlación se define para una entidad específica, y una entidad se denomina Order y la otra Customer. En la siguiente aplicación de ejemplo se muestra cómo almacenar y recuperar un pedido de cliente:

   Application.java

   ```
   public class Application
   {
       static public void main(String [] args)
           throws Exception
       {
           ObjectGrid og =
        ObjectGridManagerFactory.getObjectGridManager().createObjectGrid();
           og.registerEntities(new Class[] {Order.class});

           Session s = og.getSession();
           EntityManager em = s.getEntityManager();

           em.getTransaction().begin();

           Customer cust = new Customer();
           cust.address = "Main Street";
   ```

```
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";

        Order o = new Order();
        o.customer = cust;
        o.date = new java.util.Date();
        o.itemName = "Widget";
        o.orderNumber = "1";
        o.price = 99.99;
        o.quantity = 1;

        em.persist(o);
        em.getTransaction().commit();

        em.getTransaction().begin();
        o = (Order)em.find(Order.class, "1");
        System.out.println("Found order for customer: "
    + o.customer.firstName + " " + o.customer.surname);
        em.getTransaction().commit();
    }
}
```

Esta aplicación es parecida a la aplicación de ejemplo del paso anterior. En el ejemplo anterior, sólo se registra un objeto Order de una sola clase. WebSphere eXtreme Scale detecta e incluye automáticamente la referencia a la entidad Customer y se crea una instancia Customer para `John Smith`, a la que se hace referencia desde el nuevo objeto Order. Como resultado, el nuevo cliente se persiste automáticamente porque la relación entre dos pedidos incluye el modificador en cascada, que requiere que cada objeto sea persistente. Cuando se encuentra el objeto Order, el gestor de entidad encuentra automáticamente el objeto Customer asociado e inserta una referencia al objeto.

Guía de aprendizaje del gestor de entidades: esquema de entidades Order

Crear cuatro clases de entidades utilizando relaciones unidireccionales y bidireccionales, listas ordenadas y relaciones de claves foráneas. Las API de EntityManager se utilizan para persistir y encontrar las entidades. Basándose en las entidades Order y Customer que se encuentran en las partes anteriores de la guía de aprendizaje, este paso de la guía de aprendizaje añade dos entidades adicionales: las entidades Item y OrderLine.

Acerca de esta tarea

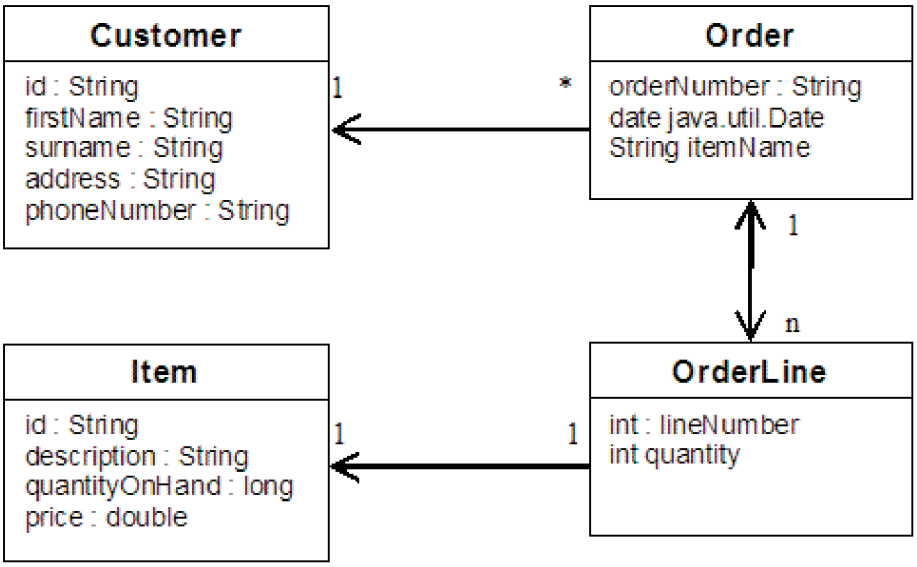


Figura 1. Esquema de entidades Order. Una entidad Order tiene una referencia a un cliente y cero o más OrderLines. Cada entidad OrderLine tiene una referencia a un sólo artículo e incluye la cantidad solicitada.

Procedimiento

1. Cree la entidad de cliente, que es parecida a los ejemplos anteriores.

   Customer.java
   ```
   @Entity
   public class Customer
   {
       @Id String id;
       String firstName;
       String surname;
       String address;
       String phoneNumber;
   }
   ```

2. Cree la entidad Item, que mantiene información sobre un producto incluido en el inventario de la tienda como, por ejemplo, la descripción del producto, la cantidad y el precio.

   Item.java
   ```
   @Entity
   public class Item
   {
       @Id String id;
       String description;
       long quantityOnHand;
       double price;
   }
   ```

3. Cree una entidad OrderLine. Cada Order tiene cero o más OrderLines, que identifican la cantidad de cada artículo en el pedido. La clave para OrderLine es una clave compuesta que consta del Order que es propietario de la OrderLine y un entero que asigna un número a el elemento de línea. Añada el modificador de persistencia en cascada a cada relación de sus entidades.

   OrderLine.java
   ```
   @Entity
   public class OrderLine
   {
   ```

```
    @Id @ManyToOne(cascade=CascadeType.PERSIST) Order order;
    @Id int lineNumber;
    @OneToOne(cascade=CascadeType.PERSIST) Item item;
    int quantity;
    double price;
}
```

4. Cree el objeto Order final, que tiene una referencia a Customer para el pedido y una colección de objetos OrderLine.

   Order.java
   ```
   @Entity
   public class Order
   {
       @Id String orderNumber;
       java.util.Date date;
       @ManyToOne(cascade=CascadeType.PERSIST) Customer customer;
       @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
      @OrderBy("lineNumber") List<OrderLine> lines;
   }
   ```

   Las cascada ALL se utiliza como modificador para líneas. Este modificador indica a EntityManager conectar en cascada la operación PERSIST y la operación REMOVE. Por ejemplo, si la entidad Order se mantiene o elimina, también se mantiene o eliminan todas las entidades OrderLine.

   Si una entidad OrderLine se elimina de la lista de líneas del objeto Order, la referencia se rompe. Sin embargo, la entidad OrderLine no se elimina de la memoria caché. Debe utilizar la API de supresión de EntityManager para eliminar entidades de la memoria caché. La operación REMOVE no se utiliza en la entidad de cliente o la entidad de artículo de OrderLine. Como resultado, la entidad de cliente permanece incluso si se elimina el pedido o el artículo cuando se elimina OrderLine.

   El modificador mappedBy indica una relación inversa con la entidad de destino. El modificador identifica qué atributo en la entidad de destino hace referencia a la entidad de origen, y el lado propietario de una relación uno con uno o muchos con muchos. En general podrá omitir el modificador. Si embargo, se visualiza un error para indicar que debe especificarse si WebSphere eXtreme Scale no puede descubrirlo automáticamente. Una entidad OrderLine que contiene dos tipos de atributos Order en una relación de muchos con uno, normalmente, genera el error.

   La anotación @OrderBy especifica el orden en el que cada entidad OrderLine debe aparece en la lista de líneas. Si la anotación no se especifica, las líneas aparecen en un orden arbitrario. Aunque las líneas se añaden a la entidad Order emitiendo ArrayList, que conserva el pedido, EntityManager no reconoce necesariamente el pedido. Cuando se emite el método find para recuperar el objeto Order de la memoria caché, el objeto de lista no es un objeto ArrayList.

5. Cree la aplicación. En el siguiente ejemplo se muestra el objeto Order final, que tiene una referencia a Customer para el pedido y una colección de objetos OrderLine.
   a. Busque los artículos a solicitar, que luego se convierten en entidades gestionadas.
   b. Cree el elemento de línea y adjúntelo a cada artículo.
   c. Cree un pedido y asócielo a cada elemento de línea y el cliente.
   d. Persiste el pedido, que persiste automáticamente cada elemento de línea.
   e. Compromete la transacción, que desconecta cada entidad y sincroniza el estado de las entidades con la memoria caché.
   f. Imprimir la información del pedido. Las entidades OrderLine se clasifican automáticamente por el ID de OrderLine.

Application.java

```
static public void main(String [] args)
        throws Exception
    {
        ...

        // Añadir algunos elementos al inventario.
        em.getTransaction().begin();
        createItems(em);
        em.getTransaction().commit();

        // Crear un nuevo cliente con los artículos en su carro.
        em.getTransaction().begin();
        Customer cust = createCustomer();
        em.persist(cust);

        // Crear nuevo pedido y añadir un elemento de línea para cada artículo.
        // Cada elemento de línea se persiste automáticamente ya que la opción
    // Cascade=ALL está establecida.
        Order order = createOrderFromItems(em, cust, "ORDER_1",
     new String[]{"1", "2"}, new int[]{1,3});
        em.persist(order);
        em.getTransaction().commit();

        // Imprimir el resumen de pedido
        em.getTransaction().begin();
        order = (Order)em.find(Order.class, "ORDER_1");
        System.out.println(printOrderSummary(order));
        em.getTransaction().commit();
    }

    public static Customer createCustomer() {
        Customer cust = new Customer();
        cust.address = "Main Street";
        cust.firstName = "John";
        cust.surname = "Smith";
        cust.id = "C001";
        cust.phoneNumber = "5555551212";
        return cust;
    }

    public static void createItems(EntityManager em) {
        Item item1 = new Item();
        item1.id = "1";
        item1.price = 9.99;
        item1.description = "Widget 1";
        item1.quantityOnHand = 4000;
        em.persist(item1);

        Item item2 = new Item();
        item2.id = "2";
        item2.price = 15.99;
        item2.description = "Widget 2";
        item2.quantityOnHand = 225;
        em.persist(item2);

    }

    public static Order createOrderFromItems(EntityManager em,
   Customer cust, String orderId, String[] itemIds, int[] qty) {

        Item[] items = getItems(em, itemIds);

        Order order = new Order();
        order.customer = cust;
        order.date = new java.util.Date();
```

```
        order.orderNumber = orderId;
        order.lines = new ArrayList<OrderLine>(items.length);
    for(int i=0;i<items.length;i++){
      OrderLine line = new OrderLine();
            line.lineNumber = i+1;
            line.item = items[i];
            line.price = line.item.price;
            line.quantity = qty[i];
            line.order = order;
            order.lines.add(line);
        }
        return order;
    }

    public static Item[] getItems(EntityManager em, String[] itemIds) {
        Item[] items = new Item[itemIds.length];
        for(int i=0;i<items.length;i++){
    items[i] = (Item) em.find(Item.class, itemIds[i]);
        }
        return items;
    }
```

El paso siguiente será suprimir una entidad. La interfaz EntityManager tiene un método remove que marca un objeto como suprimido. La aplicación debe eliminar la entidad de todas las colecciones de relaciones antes de llamar al método remove. Edite las referencias y emita el método remove, o em.remove(object), como último paso.

Guía de aprendizaje del gestor de entidades: actualización de entradas

Si desea cambiar una entidad, puede buscar la instancia, actualizar la instancia y todas las entidades referenciadas, y confirmar la transacción.

Procedimiento

Actualice entradas. En el siguiente ejemplo se muestra cómo buscar la instancia de Order, cambiar la instancia y todas las entidades referenciadas, y confirmar la transacción.

```
public static void updateCustomerOrder(EntityManager em) {
    em.getTransaction().begin();
    Order order = (Order) em.find(Order.class, "ORDER_1");
    processDiscount(order, 10);
    Customer cust = order.customer;
    cust.phoneNumber = "5075551234";
    em.getTransaction().commit();
}

public static void processDiscount(Order order, double discountPct) {
    for(OrderLine line : order.lines) {
        line.price = line.price * ((100-discountPct)/100);
    }
}
```

Al desechar la transacción se sincronizan todas las entidades gestionadas con la memoria caché. Cuando se confirma una transacción, automáticamente se produce un desecho. En este caso, el pedido pasa a ser una entidad gestionada. Todas las entidades a las que se hace referencia desde el pedido, cliente y el elemento de pedido también pasan a ser entidades gestionadas. Cuando la transacción se desecha, se comprueba cada una de las entidades para determinar si se han modificado. Aquéllas que se han modificado se actualizan en la memoria caché. Una vez que se ha completado la transacción, confirmándose o retrotrayéndose, las entidades pasan a estar desconectadas y todos los cambios que se realizan en las

entidades no se reflejan en la memoria caché.

Guía de aprendizaje del gestor de entidades: actualización y eliminación de entradas con un índice

Puede utilizar un índice para buscar, actualizar y eliminar entidades.

Procedimiento

Actualice y elimine entidades utilizando un índice. Utilice un índice para buscar, actualizar y eliminar entidades. En los siguientes ejemplos, la clase de entidad Order se actualiza para utilizar la anotación @Index. La anotación @Index señala WebSphere eXtreme Scale para crear un índice de rango para un atributo. El nombre del índice es el mismo nombre que el nombre del atributo y siempre es un tipo de índice MapRangeIndex.

Order.java

```
@Entity
public class Order
{
    @Id String orderNumber;
    @Index java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
    @OrderBy("lineNumber") List<OrderLine> lines;  }
```

En el siguiente ejemplo se muestra cómo cancelar todos los pedidos que se someten en el último minuto. Busque el pedido utilizando un índice, vuelva a añadir los artículos del pedido al inventario y elimine del sistema el pedido y los elementos de línea asociados.

```
public static void cancelOrdersUsingIndex(Session s)
  throws ObjectGridException {
    // Cancelar todos los pedidos que se sometieron hace un minuto
    java.util.Date cancelTime = new
   java.util.Date(System.currentTimeMillis() - 60000);
    EntityManager em = s.getEntityManager();
    em.getTransaction().begin();
    MapRangeIndex dateIndex = (MapRangeIndex)
   s.getMap("Order").getIndex("date");
    Iterator<Tuple> orderKeys = dateIndex.findGreaterEqual(cancelTime);
  while(orderKeys.hasNext()) {
   Tuple orderKey = orderKeys.next();
   // Buscar el pedido para eliminarlo.
   Order curOrder = (Order) em.find(Order.class, orderKey);
   // Verificar que el pedido no haya sido actualizado por otro usuario.
   if(curOrder != null && curOrder.date.getTime() >= cancelTime.getTime()) {
    for(OrderLine line : curOrder.lines) {
     // Vuelva a añadir el elemento al inventario.
     line.item.quantityOnHand += line.quantity;
     line.quantity = 0;
    }
   em.remove(curOrder);
   }
  }
 em.getTransaction().commit();
}
```

Guía de aprendizaje del gestor de entidades: actualización y eliminación de entradas utilizando una consulta

Puede actualizar y eliminar entidades utilizando una consulta.

Procedimiento

Actualice y elimine entradas utilizando una consulta.

Order.java

```
@Entity
public class Order
{
    @Id String orderNumber;
    @Index java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order")
   @OrderBy("lineNumber") List<OrderLine> lines;
}
```

La clase de entidad de pedido es la misma que en el ejemplo anterior. La clase sigue proporcionando la anotación @Index porque la serie de consulta utiliza la fecha para buscar la entidad. El motor de consultas utiliza índices cuando pueden utilizarse.

```
public static void cancelOrdersUsingQuery(Session s) {
        // Cancelar todos los pedidos que se sometieron hace un minuto
        java.util.Date cancelTime =
     new java.util.Date(System.currentTimeMillis() - 60000);
        EntityManager em = s.getEntityManager();
        em.getTransaction().begin();

        // Crear una consulta que buscará el pedido en base a la fecha. Como
        // tenemos un índice definido en la fecha del pedido, la consulta
    // lo utilizará automáticamente.
        Query query = em.createQuery("SELECT order FROM Order order
     WHERE order.date >= ?1");
        query.setParameter(1, cancelTime);
        Iterator<Order> orderIterator = query.getResultIterator();
    while(orderIterator.hasNext()) {
     Order order = orderIterator.next();
     // Verificar que otro usuario no haya actualizado el pedido.
     // Dado que la consulta utilizó un índice, no había ningún bloqueo en la fila.
     if(order != null && order.date.getTime() >= cancelTime.getTime()) {
       for(OrderLine line : order.lines) {
        // Vuelva a añadir el elemento al inventario.
        line.item.quantityOnHand += line.quantity;
        line.quantity = 0;
       }
       em.remove(order);
     }
   }
  em.getTransaction().commit();
}
```

Como en el ejemplo anterior, el método cancelOrdersUsingQuery intenta cancelar todos los pedidos que se sometieron en el último minuto. Para cancelar el pedido, busque el pedido utilizando una consulta, vuelva a añadir los elementos al inventario y elimine el pedido y los elementos de línea asociados del sistema.

Guía de aprendizaje: Ejecución de paquetes de eXtreme Scale en la infraestructura OSGi

El ejemplo de OSGi se basa en los ejemplos de serializador de Google Protocol Buffers. Cuando haya completado este conjunto de lecciones, habrá ejecutado los plug-ins de ejemplo de serializador en la infraestructura OSGi.

Objetivos del aprendizaje

Este ejemplo muestra los paquetes OSGi. El plug-in de serializador es secundario y no es necesario. El ejemplo de OSGi está disponible en la galería de ejemplos de WebSphere eXtreme Scale. Debe descargar el ejemplo y extraerlo en el directorio *inicio_wxs*/samples. El directorio raíz para el ejemplo de OSGi es wxs_home/samples/OSGiProto.

El ejemplo de serializador de Google Protocol Buffers se encuentra en el directorio *inicio_wxs*/samples/SerializerProto.

El ejemplo de serializador de Binary JSON (BSON) se encuentra en el directorio *inicio_wxs*/samples/SerializerBSON.

Los ejemplos de mandato de esta guía de aprendizaje supone que se ejecuta en el sistema operativo UNIX. Debe ajustar el ejemplo de mandato para que se ejecute en un sistema operativo Windows.

Después de completar las lecciones de este módulo, comprenderá los conceptos del ejemplo de OSGi y sabrá cómo completar los objetivos siguientes:

- Instalar el paquete de servidor WebSphere eXtreme Scale en el contenedor OSGi para iniciar el servidor eXtreme Scale.
- Configurar el entorno de desarrollo de eXtreme Scale para ejecutar el cliente de ejemplo.
- Utilizar el mandato xscmd para consultar la clasificación de servicio del paquete de ejemplo, actualizarlo a una nueva clasificación de servicio y verificar la nueva clasificación de servicio.

Tiempo necesario

Este módulo requiere aproximadamente 60 minutos para completarse.

Requisitos previos

Además de descargar y extraer los ejemplos de serializador, esta guía de aprendizaje también tiene los requisitos previos siguientes:

- Instale y extraiga el producto eXtreme Scale
- Configure el entorno Eclipse Equinox

Introducción: Inicio y configuración del servidor y contenedor de eXtreme Scale para ejecutar plug-ins en la infraestructura OSGi

En esta guía de aprendizaje inicia un servidor eXtreme Scale en la infraestructura OSGi, inicia un contenedor de eXtreme Scale y conecta los plug-ins de ejemplo al entorno de ejecución de eXtreme Scale.

Objetivos del aprendizaje

Después de completar las lecciones de este módulo, comprenderá los conceptos del ejemplo de OSGi y sabrá cómo completar los objetivos siguientes:

- Instalar el paquete de servidor WebSphere eXtreme Scale en el contenedor OSGi para iniciar el servidor eXtreme Scale.
- Configurar el entorno de desarrollo de eXtreme Scale para ejecutar el cliente de ejemplo.
- Utilizar el mandato xscmd para consultar la clasificación de servicio del paquete de ejemplo, actualizarlo a una nueva clasificación de servicio y verificar la nueva clasificación de servicio.

Tiempo necesario

Esta guía de aprendizaje requiere aproximadamente 60 minutos para completarse. Si explora otros conceptos relacionados con esta guía de aprendizaje, podría requerir más tiempo para completarse.

Nivel de conocimientos

Intermedio.

A quién va dirigida

Desarrolladores y administradores que deseen crear, instalar y ejecutar paquetes de eXtreme Scale en la infraestructura OSGi.

Requisitos del sistema

- El cliente de línea de mandatos Luminis OSGi Configuration Admin, versión 0.2.5
- Apache Felix File Install, versión 3.0.2
- Cuando se utiliza Eclipse Gemini como proveedor de contenedor Blueprint, se requiere lo siguiente:
  - Eclipse Gemini Blueprint, versión 1.0.0
  - Spring Framework, versión 3.0.5
  - SpringSource AOP Alliance API, versión 1.0.0
  - SpringSource Apache Commons Logging, versión 1.1.1
- Cuando se utiliza Apache Aries como proveedor de contenedor Blueprint, debe tener los requisitos siguientes:
  - Apache Aries, instantánea más reciente
  - Biblioteca ASM
  - Registro PAX

Requisitos previos

Para completar esta guía de aprendizaje, debe descargar el ejemplo y extraerlo en el directorio `wxs_home/samples`. El directorio raíz para el ejemplo de OSGi es `wxs_home/samples/OSGiProto`.

Resultados esperados

Cuando haya completado esta guía de aprendizaje, habrá instalado los paquetes de ejemplo y ejecutado un cliente de eXtreme Scale para insertar datos en la cuadrícula. También puede esperar consultar y actualizar estos paquetes de ejemplo utilizando las prestaciones dinámicas que proporciona el contenedor OSGi.

Conceptos relacionados:

"Visión general de la infraestructura OSGi" en la página 37
OSGi define un sistema de módulo dinámico para Java. La plataforma de servicio OSGi tiene una arquitectura por capas, y está diseñada para ejecutarse en diversos perfiles Java estándar. Puede iniciar servidores y clientes de WebSphere eXtreme Scale en un contenedor OSGi.

Tareas relacionadas:

"Instalación de la infraestructura OSGi de Eclipse Equinox con Eclipse Gemini para clientes y servidores" en la página 39
Si desea desplegar WebSphere eXtreme Scale en una infraestructura OSGi, debe configurar el entorno de Eclipse Equinox.

Referencia relacionada:

Archivo de propiedades de servidor
El archivo de propiedades de servidor contiene varias propiedades que definen distintos valores para el servidor como, por ejemplo, los valores de rastreo, el inicio de sesión y la configuración de seguridad. El archivo de propiedades del servidor lo utilizan el servicio de catálogo y los servidores de contenedor tanto en servidores autónomos como en servidores alojados en WebSphere Application Server.

Módulo 1: Preparación para instalar y configurar los paquetes del servidor de eXtreme Scale

Complete este módulo para explorar los paquetes de ejemplo de OSGi y examinar los archivos de configuración que utiliza para configurar el servidor de eXtreme Scale.

Objetivos del aprendizaje

Después de completar las lecciones de este módulo, comprenderá los conceptos y sabrá cómo completar los objetivos siguientes:

- Localizar y explorar los paquetes incluidos en el ejemplo de OSGi.
- Examinar los archivos de configuración que se utilizan para configurar el servidor y la cuadrícula de eXtreme Scale.

Lección 1.1: Comprender los paquetes de ejemplo de OSGi

Complete esta lección para localizar y explorar los paquetes que se proporcionan en el ejemplo de OSGi.

Paquetes de ejemplo de OSGi:

Además de los paquetes configurados en el archivo `config.ini`, que se muestra en el tema sobre configuración del entorno de Eclipse Equinox, se utilizan los siguientes paquetes adicionales en el ejemplo de OSGi:

`objectgrid.jar`
: Paquete de tiempo de ejecución del servidor WebSphere eXtreme Scale. Este paquete se encuentra en el directorio *`inicio_wxs`*`/lib`.

`com.google.protobuf_2.4.0a.jar`
: Paquete de Google Protocol Buffers, versión 2.4.0a. Este paquete se encuentra en el directorio *`raíz_wxs_sample_osgi`*`/lib`.

`ProtoBufSamplePlugins-1.0.0.jar`
: Versión 1.0.0 del paquete de plug-in de usuario con las implementaciones de los plug-ins ObjectGridEventListener y MapSerializerPlugin de ejemplo.

Este paquete se encuentra en el directorio *raíz_wxs_sample_osgi*/lib. Los servicios se configuran con clasificación de servicio 1.

Esta versión utiliza XML Blueprint estándar para configurar los servicios de plug-in de eXtreme Scale. La clase de servicio es una clase implementada por el usuario para la interfaz de WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory. La clase implementada por el usuario crea un bean para cada solicitud y funciona de forma similar a un bean con ámbito de prototipo.

ProtoBufSamplePlugins-2.0.0.jar
Versión 2.0.0 del paquete de plug-in de usuario con implementaciones de los plug-ins ObjectGridEventListener y MapSerializerPlugin de ejemplo. Este paquete se encuentra en el directorio *raíz_wxs_sample_osgi*/lib. Los servicios se configuran con clasificación de servicio 2.

Esta versión utiliza XML Blueprint estándar para configurar los servicios de plug-in de eXtreme Scale. La clase de servicio utiliza una clase incorporada WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, que utiliza el servicio BlueprintContainer. Utilizando la configuración XML Blueprint estándar, los beans se pueden configurar como ámbito de prototipo o ámbito de singleton. El bean no se configura como ámbito de fragmento.

ProtoBufSamplePlugins-Gemini-3.0.0.jar
Versión 3.0.0 del paquete de plug-in de usuario con las implementaciones de los plug-ins ObjectGridEventListener y MapSerializerPlugin de ejemplo. Este paquete se encuentra en el directorio *raíz_wxs_sample_osgi*/lib. Los servicios se configuran con clasificación de servicio 3.

Esta versión utiliza el XML Blueprint específico de Eclipse Gemini para configurar los servicios de plug-in de eXtreme Scale. La clase de servicio utiliza una clase incorporada WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, que utiliza el servicio BlueprintContainer. La forma de configurar un bean con ámbito de fragmento utiliza un enfoque específico de Gemini. Esta versión configura el bean myShardListener como un bean con ámbito de fragmento proporcionando {http://www.ibm.com/schema/objectgrid}shard como valor de ámbito, y configurando un atributo ficticio para que Gemini reconozca el ámbito personalizado. Esto se debe al siguiente problema de Eclipse: https://bugs.eclipse.org/bugs/show_bug.cgi?id=348776

ProtoBufSamplePlugins-Aries-4.0.0.jar
Versión 4.0.0 del paquete de plug-in de usuario con las implementaciones de los plug-ins ObjectGridEventListener y MapSerializerPlugin de ejemplo. Este paquete se encuentra en el directorio *raíz_wxs_sample_osgi*/lib. Los servicios se configuran con clasificación de servicio 4.

Esta versión utiliza XML Blueprint estándar para configurar los servicios de plug-in de eXtreme Scale. La clase de servicio utiliza una clase incorporada WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, que utiliza el servicio BlueprintContainer. Utilizando la configuración de XML Blueprint estándar, los beans se pueden configurar mediante un ámbito personalizado. Esta versión configura myShardListenerbean como un bean con ámbito de fragmento proporcionando {http://www.ibm.com/schema/objectgrid}shard como valor de ámbito.

ProtoBufSamplePlugins-Activator-5.0.0.jar
Versión 5.0.0 del paquete de plug-in de usuario con las implementaciones de los plug-ins ObjectGridEventListener y MapSerializerPlugin de ejemplo. Este paquete se encuentra en el directorio *raíz_wxs_sample_osgi*/lib. Los servicios se configuran con clasificación de servicio 5.

Esta versión no utiliza contenedor Blueprint en absoluto. En esta versión, los servicios se registran utilizando el registro de servicios OSGi. La clase de servicio es una clase implementada por el usuario para la interfaz de WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory. La clase implementada por el usuario crea un bean para cada solicitud. Funciona de forma similar a un bean con ámbito de prototipo.

Punto de comprobación de la lección:

Explorando los paquetes proporcionados con el ejemplo de OSGi, podrá comprender mejor cómo desarrollar sus propias implementaciones que se ejecutarán en el contenedor OSGi.

Ha aprendido sobre lo siguiente:

- Los paquetes incluidos con el ejemplo de OSGi
- La ubicación de estos paquetes
- La clasificación de servicio con la que se ha configurado cada uno de los paquetes

Lección 1.2: Comprender los archivos de configuración de OSGi

El ejemplo de OSGi incluye tres archivos de configuración. Utiliza estos archivos para iniciar y configurar la cuadrícula y el servidor WebSphere eXtreme Scale.

Archivos de configuración de OSGi:

En esta lección, explorará los siguientes archivos de configuración:

- collocated.server.properties
- protoBufObjectGrid.xml
- protoBufDeployment.xml

collocated.server.properties

Se requiere una configuración de servidor para iniciar un servidor. Cuando se inicia el paquete de servidor de eXtreme Scale, no inicia un servidor. Espera que se cree el PID de configuración, com.ibm.websphere.xs.server, con un archivo de propiedades del servidor. Este archivo de propiedades del servidor especifica el nombre del servidor, el número de puerto y otras propiedades del servidor.

En la mayoría de casos, creará una configuración para establecer el archivo de propiedades del servidor. En casos excepcionales, es posible que solo desee iniciar un servidor con todas las propiedades establecidas en un valor predeterminado. En ese caso, puede crear una configuración denominada com.ibm.websphere.xs.server con el valor establecido en default.

Para obtener más detalles sobre el archivo de propiedades de servidor, consulte el tema Archivo de propiedades de servidor .

El ejemplo de OSGi incluye el archivo de propiedades del servidor de ejemplo, *raíz_wxs_sample_osgi*/server/properties/collocated.server.properties. Este archivo de propiedades de ejemplo inicia un único servicio de catálogo y un servidor de contenedor en el proceso de la infraestructura OSGi. Los clientes de eXtreme Scale se conectan al puerto 2809 y los clientes JMX se conectan al puerto 1099. El contenido del archivo de propiedades del servidor de ejemplo es el siguiente:

```
serverName=collocatedServer
isCatalog=true
catalogClusterEndPoints=collocatedServer:
localhost:6601:6602traceSpec=
ObjectGridOSGi=all=enabled
traceFile=logs/trace.log
listenerPort=2809
JMXServicePort=1099
```

protoBufObjectGrid.xml

El archivo XML de descriptor de ObjectGrid protoBufObjectGrid.xml de ejemplo contiene lo siguiente, con los comentarios eliminados.

```
<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" txTimeout="15">

            <bean id="ObjectGridEventListener"
                osgiService="myShardListener"/>

            <backingMap name="Map" readOnly="false"
                lockStrategy="PESSIMISTIC" lockTimeout="5"
                copyMode="COPY_TO_BYTES"
                pluginCollectionRef="serializer"/>

        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="serializer">
            <bean id="MapSerializerPlugin"
                osgiService="myProtoBufSerializer"/>"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
 </objectGridConfig>
```

Hay dos plug-ins configurados en este archivo XML de descriptor de ObjectGrid:

ObjectGridEventListener
Plug-in de nivel de fragmento. Para cada instancia de ObjectGrid, hay una instancia de ObjectGridEventListener. Está configurada para utilizar el servicio OSCi myShardListener. Esto significa que cuando se crea la cuadrícula, el plug-in ObjectGridEventListener utiliza el servicio OSGi myShardListener con la clasificación de servicio más alta disponible.

MapSerializerPlugin
Plug-in de nivel de correlación. Para la correlación de respaldo denominada Map, hay un plug-in MapSerializerPlugin configurado. Está configurado para utilizar el servicio OSGi myProtoBufSerializer. Esto significa que cuando se crea la correlación, el plug-in MapSerializerPlugin utiliza el servicio, myProtoBufSerializer, con la clasificación de servicio con el rango más alto disponible.

protoBufDeployment.xml

El archivo XML de descriptor de despliegue describe la política de despliegue de la cuadrícula denominada `Grid`, que utiliza cinco particiones. Consulte el siguiente ejemplo de código de este archivo XML:

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

  <objectgridDeployment objectgridName="Grid">
    <mapSet name="MapSet" numberOfPartitions="5">
      <map ref="Map"/>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

blueprint.xml

Como alternativa a la utilización del archivo `collocated.server.properties` junto con el PID de configuración, `com.ibm.websphere.xs.server`, puede incluir los archivos XML del ObjectGrid y XML de despliegue en un paquete OSGi, junto con un archivo XML Blueprint, tal como se muestra en el ejemplo siguiente:

```
<blueprint>
   xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
       xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
       default-activation="lazy">

    <objectgrid:server id="server" isCatalog="true"
                name="server"
                tracespec="ObjectGridOSGi=all=enabled"
                tracefile="C:/Temp/logs/trace.log"
                workingDirectory="C:/Temp/working"
                jmxport="1099">
        <objectgrid:catalog host="localhost" port="2809"/>
    </objectgrid:server>

    <objectgrid:container id="container"
objectgridxml="/META-INF/objectgrid.xml"
       deploymentxml="/META-INF/deployment.xml"
   server="server"/>
</blueprint>
```

Punto de comprobación de la lección:

En esta lección, ha aprendido acerca de los archivos de configuración que se utilizan en el ejemplo de OSGi. Ahora, cuando inicie y configure el servidor y la cuadrícula de eXtreme Scale, comprenderá qué archivos se utilizan en estos procesos y cómo interactúan estos archivos con los plug-ins de la infraestructura OSGi.

Módulo 2: Instalación e inicio de paquetes de eXtreme Scale en la infraestructura OSGi

Utilice los módulos de estas lecciones para instalar el paquete de servidor eXtreme Scale en el contenedor OSGi e iniciar el servidor WebSphere eXtreme Scale.

El inicio del servidor en la infraestructura OSGi no significa que los paquetes OSGi estén listos para su ejecución. Debe configurar las propiedades del servidor y los contenedores para que los paquetes OSGi que instale se reconozcan y se ejecuten correctamente.

Objetivos del aprendizaje

Después de completar las lecciones de este módulo, comprenderá los conceptos y sabrá cómo completar las tareas siguientes:

- Instalar los paquetes de eXtreme Scale utilizando la consola de Equinox OSGi.
- Configurar el servidor eXtreme Scale.
- Configurar el contenedor de eXtreme Scale.
- Iniciar paquetes de ejemplo de eXtreme Scale.

Requisitos previos

Para completar este módulo, se requieren las tareas siguientes antes de empezar:

- Instale y extraiga el producto eXtreme Scale
- Configure el entorno Eclipse Equinox

También debe prepararse para acceder a los archivos siguientes para completar las lecciones de este módulo:

- El paquete `objectgrid.jar`. Instala este paquete de eXtreme Scale.
- El archivo `collocated.server.properties`. Añade las propiedades del servidor a este archivo de configuración.
- Puede esperar instalar e iniciar los paquetes siguientes:
- El paquete `protobuf-java-2.4.0a-bundle.jar`
- El paquete `ProtoBufSamplePlugins-1.0.0.jar`
- El paquete `ProtoBufSamplePlugins-2.0.0.jar`

Lección 2.1: Iniciar la consola e instalar el paquete de servidor de eXtreme Scale

En esta lección, utiliza la consola de Equinox OSGi para iniciar e instalar un WebSphere eXtreme Scale

1. Utilice el mandato siguiente para iniciar la consola de Equinox OSGi:

```
cd raíz_equinox

java -jar
plugins\org.eclipse.osgi_3.6.1.R36x_v20100806.jar
-console
```

2. Una vez que se haya iniciado la consola OSGI, emita el mandato `ss` en la consola y se iniciarán los paquetes siguientes:

Salida de Eclipse Gemini:

```
osgi> ss
Framework is launched.
id State Bundle
0 ACTIVE org.eclipse.osgi_3.6.1.R36x_v20100806
1 ACTIVE org.eclipse.osgi.services_3.2.100.v20100503
2 ACTIVE org.eclipse.osgi.util_3.2.100.v20100503
3 ACTIVE org.eclipse.equinox.cm_1.0.200.v20100520
4 ACTIVE com.springsource.org.apache.commons.logging_1.1.1
5 ACTIVE com.springsource.org.aopalliance_1.0.0
6 ACTIVE org.springframework.aop_3.0.5.RELEASE
7 ACTIVE org.springframework.asm_3.0.5.RELEASE
8 ACTIVE org.springframework.beans_3.0.5.RELEASE
9 ACTIVE org.springframework.context_3.0.5.RELEASE
10 ACTIVE org.springframework.core_3.0.5.RELEASE
11 ACTIVE org.springframework.expression_3.0.5.RELEASE
12 ACTIVE org.apache.felix.fileinstall_3.0.2
13 ACTIVE net.luminis.cmc_0.2.5
```

```
14 ACTIVE org.eclipse.gemini.blueprint.core_1.0.0.RELEASE
15 ACTIVE org.eclipse.gemini.blueprint.extender_1.0.0.RELEASE
16 ACTIVE org.eclipse.gemini.blueprint.io_1.0.0.RELEASE
```

Salida de Apache Aries:

```
osgi> ss
Framework is launched.
id State Bundle
0 ACTIVE org.eclipse.osgi_3.6.1.R36x_v20100806
1 ACTIVE org.eclipse.osgi.services_3.2.100.v20100503
2 ACTIVE org.eclipse.osgi.util_3.2.100.v20100503
3 ACTIVE org.eclipse.equinox.cm_1.0.200.v20100520
4 ACTIVE org.ops4j.pax.logging.pax-logging-api_1.6.3
5 ACTIVE org.ops4j.pax.logging.pax-logging-service_1.6.3
6 ACTIVE org.objectweb.asm.all_3.3.0
7 ACTIVE org.apache.aries.blueprint_0.3.2.SNAPSHOT
8 ACTIVE org.apache.aries.util_0.4.0.SNAPSHOT
9 ACTIVE org.apache.aries.proxy_0.4.0.SNAPSHOT
10 ACTIVE org.apache.felix.fileinstall_3.0.2
11 ACTIVE net.luminis.cmc_0.2.5
```

3. Instale el paquete `objectgrid.jar`. Para iniciar un servidor en la máquina virtual Java (JVM), necesita instalar un paquete de servidor de eXtreme Scale. Este paquete de servidor eXtreme Scale puede iniciar un servidor y crear contenedores. Utilice el mandato siguiente para instalar el archivo `objectgrid.jar`:

   ```
   osgi> install file:///inicio_wxs/lib/objectgrid.jar
   ```

   Consulte el siguiente ejemplo:

   ```
   osgi> install
   file:///opt/wxs/ObjectGrid/lib/objectgrid.jar
   ```

   Equinox visualiza su ID de paquete; por ejemplo:

   ```
   El ID de paquete es 19
   ```

   Recuerde: El ID de paquete puede ser distinto. La vía de acceso del archivo debe ser un URL absoluto a la vía de acceso del paquete. No se da soporte a vías de acceso relativas.

Punto de comprobación de la lección:

En esta lección, ha utilizado la consola de Equinox OSGi para instalar el paquete `objectgrid.jar`, que utilizará para iniciar un servidor y crear un contenedor posteriormente en esta guía de aprendizaje.

Lección 2.2: Personalizar y configurar el servidor eXtreme Scale

Utilice esta lección para personalizar y añadir las propiedades de servidor al servidor WebSphere eXtreme Scale.

1. Edite el archivo `raíz_wxs_sample_osgi/server/properties/collocated.server.properties`.
   a. Cambie la propiedad workingDirectory a `raíz_equinox`.
   b. Cambie la propiedad traceFile a `raíz_equinox/logs/trace.log`.
2. Guarde el archivo.
3. Especifique las siguientes líneas de código en la consola OSGI para crear la configuración de servidor desde el archivo:

```
osgi> cm create com.ibm.websphere.xs.server

osgi> cm put com.ibm.websphere.xs.server
objectgrid.server.props
wxs_sample_osgi_root/server/properties/collocated.serve
r.properties
```

4. Para visualizar la configuración, ejecute el mandato siguiente:

```
osgi> cm get com.ibm.websphere.xs.server
Configuration for service (pid) "com.ibm.websphere.xs.server"
(bundle location = null)
key value
------ ------
objectgrid.server.props objectgrid.server.props
```

Punto de comprobación de la lección:

En esta lección, ha editado el archivo `raíz_wxs_sample_osgi/server/properties/collocated.server.properties` para especificar valores de servidor como, por ejemplo, el directorio de trabajo y la ubicación de los archivos de registro de rastreo.

Lección 2.3: Configurar el contenedor de eXtreme Scale

Complete esta lección para configurar un contenedor, que incluye el archivo XML de descriptor de ObjectGrid de WebSphere eXtreme Scale y el archivo XML de despliegue de ObjectGrid. Estos archivos incluyen la configuración de la cuadrícula y su topología.

Para crear un contenedor, primero cree un servicio de configuración utilizando el número de identificador de proceso (PID) de la fábrica de servicios gestionados, com.ibm.websphere.xs.container. La configuración de servicio es una fábrica de servicios gestionados, así que puede crear varios PID de servicio a partir de un PID de fábrica. A continuación, para iniciar el servicio de contenedor, establezca los PID de objectgridFile y deploymentPolicyFile para cada PID de servicio.

Complete los pasos siguientes para personalizar y añadir las propiedades de servicio a la infraestructura OSGi:

1. En la consola OSGI, especifique el mandato siguiente para crear el contenedor a partir del archivo:

```
osgi> cm createf com.ibm.websphere.xs.container
PID: com.ibm.websphere.xs.container-1291179621421-0
```

2. Especifique el mandato siguiente para enlazar el PID que se acaba de crear con los archivos XML de ObjectGrid.

   Recuerde: El número de PID será distinto a lo que se incluye en este ejemplo.

```
osgi> cm put com.ibm.websphere.xs.container-1291179621421-0
objectgridFile wxs_sample_osgi_root/server/META-INF/protoBufObjectgrid.xml

osgi> cm put com.ibm.websphere.xs.container-1291179621421-0
deploymentPolicyFile wxs_sample_osgi_root/server/META-INF/protoBufDeployment.xml
```

3. Utilice el mandato siguiente para visualizar la configuración:

```
osgi> cm get com.ibm.websphere.xs.container-1291760127968-0
Configuration for service (pid) "com.ibm.websphere.xs.container-1291760127968-0"
(bundle location = null)

key value
------                  ------
deploymentPolicyFile    /opt/wxs/ObjectGrid/samples/OSGiProto/server/META-INF/protoBufDeployment.xml
objectgridFile          /opt/wxs/ObjectGrid/samples/OSGiProto/server/META-INF/protoBufObjectgrid.xml
service.factoryPid    com.ibm.websphere.xs.container
service.pid        com.ibm.websphere.xs.container-1291760127968-0
```

Punto de comprobación de la lección:

En esta lección, ha creado un servicio de configuración, que ha utilizado para crear un contenedor de eXtreme Scale. Puesto que los archivos XML de ObjectGrid contienen la configuración para la cuadrícula y su topología, debía enlazar el contenedor que había creado a estos archivos XML de ObjectGrid. Con esta configuración, el contenedor de eXtreme Scale puede reconocer los paquetes OSGi que ejecutará posteriormente en esta guía de aprendizaje.

Lección 2.4: Instalar los paquetes Google Protocol Buffers y de plug-in de ejemplo

Complete esta guía de aprendizaje para instalar el paqueteprotobuf-java-2.4.0a-bundle.jar y el paquete del plug-in ProtoBufSamplePlugins-1.0.0.jar mediante la consola de Equinox OSGi.

Complete los pasos siguientes para instalar el paquete Google Protocol Buffers.

En la consola OSGI, especifique el mandato siguiente para instalar el paquete:

```
osgi> install file:///wxs_sample_osgi_root/common/lib/com.google.protobuf_2.4.0a.jar
```

Se visualiza la salida siguiente:

```
El ID de paquete es 21
```

Visión general de los paquetes de plug-in de ejemplo:

Este ejemplo de OSGi incluye cinco paquetes de ejemplo que incluyen plug-ins eXtreme Scale, incluido un plug-in ObjectGridEventListener y un plug-in MapSerializerPlugin personalizados. El plug-in MapSerializerPlugin utiliza el ejemplo Google Protocol Buffers y los mensajes proporcionados por el ejemplo MapSerializerPlugin.

Los paquetes siguientes se encuentran en el directorio *raíz_osgi_ejemplo_wxs*/lib: ProtoBufSamplePlugins-1.0.0.jar y ProtoBufSamplePlugins-2.0.0.jar.

El archivo blueprint.xml tiene el siguiente contenido con los comentarios eliminados:

```
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
 <bean id="myShardListener" class="com.ibm.websphere.samples.xs.proto.osgi.MyShardListenerFactory"/>
 <service ref="myShardListener" interface="com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory" ranking="1">
 </service>

 <bean id="myProtoBufSerializer" class="com.ibm.websphere.samples.xs.proto.osgi.ProtoMapSerializerFactory">
  <property name="keyType" value="com.ibm.websphere.samples.xs.serializer.app.proto.DataObjects1$OrderKey" />
  <property name="valueType" value="com.ibm.websphere.samples.xs.serializer.app.proto.DataObjects1$Order" />
 </bean>

 <service ref="myProtoBufSerializer" interface="com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory"
ranking="1">
 </service>
</blueprint>
```

El archivo XML Blueprint exporta dos servicios, myShardListener y myProtoBufSerializer. Se hace referencia a estos dos servicios en el archivo protoBufObjectgrid.xml.

Instalar el paquete de plug-in de ejemplo:

Complete los pasos siguientes para instalar el paquete ProtoBufSamplePlugins-1.0.0.jar.

Ejecute el mandato siguiente en la consola de Equinox OSGi para instalar el paquete del plug-in `ProtoBufSamplePlugins-1.0.0.jar`:

```
osgi> install file:///wxs_sample_osgi_root/common/lib/ProtoBufSamplePlugins-1.0.0.jar
```

Se visualiza la salida siguiente:

```
El ID de paquete es 22
```

Punto de comprobación de la lección:

En esta sesión, ha instalado el paquete `protobuf-java-2.4.0a-bundle.jar` y el paquete de plug-in `ProtoBufSamplePlugins-1.0.0.jar`.

Lección 2.5: Iniciar los paquetes OSGi

El servidor WebSphere eXtreme Scale se empaqueta como un paquete de servidor OSGi. Complete esta lección para instalar el paquete de servidor eXtreme Scale así como otros paquetes OSGi que ha instalado.

1. Inicie el paquete del plug-in de ejemplo. Ejecute el mandato siguiente en la consola de Equinox OSGi para iniciar el paquete. En este ejemplo, el ID del paquete del plug-in de ejemplo es 22.

   ```
   osgi> start 22
   ```

2. Inicie el paquete de Google Protocol Buffers. Ejecute el mandato siguiente en la consola de Equinox OSGi para iniciar el paquete. En este ejemplo, el ID del paquete del plug-in de Google Protocol Buffers es 21.

   ```
   osgi> start 21
   ```

3. Inicie el paquete del servidor. Ejecute el mandato siguiente en la consola de OSGi para iniciar el servidor. En este ejemplo, el ID de paquete del paquete de servidor eXtreme Scale es 19.

   ```
   osgi> start 19
   ```

Después de iniciar el servidor, el escucha de sucesos de MyShardListener se ha iniciado y está preparado para insertar o actualizar registros. Puede ver la salida siguiente en la consola de OSGI para confirmar que el paquete del plug-in se ha iniciado satisfactoriamente:

```
SystemOut O MyShardListener@1253853884(version=1.0.0) order
com.ibm.websphere.samples.xs.serializer.proto.DataObjects1$Order$Builder
@1aba1aba(22) inserted
```

Punto de comprobación de la lección:

En esta lección, ha iniciado dos paquetes de plug-in y el paquete del servidor en el contenedor de eXtreme Scale que ha configurado en la infraestructura OSGi.

Módulo 3: Ejecución del cliente de ejemplo de eXtreme Scale

El servidor de WebSphere eXtreme Scale ahora se ejecuta en un entorno OSGi. Complete los pasos de este módulo para ejecutar un cliente de WebSphere eXtreme Scale que inserte datos en la cuadrícula.

Objetivos del aprendizaje

Después de completar las lecciones de este módulo, sabrá cómo completar las tareas siguientes:

- Ejecutar una aplicación cliente que se conecta a la cuadrícula e inserta y recupera datos de ella.
- Iniciar un pedido utilizando una aplicación cliente no OSGi.

Requisitos previos

Complete el Módulo 2: Instalación e inicio de paquetes de eXtreme Scale en la infraestructura OSGi.

Lección 3.1: Configurar Eclipse para ejecutar el cliente y construir los ejemplos

Complete esta lección para importar el proyecto Eclipse que utilizará para ejecutar el cliente y construir los plug-ins de ejemplo.

El ejemplo incluye un programa cliente Java SE que se conecta a la cuadrícula e inserta y recupera datos de la misma. También incluye proyectos que puede utilizar para construir y volver a desplegar los paquetes OSGi.

El proyecto proporcionado se ha probado con Eclipse 3.x y posterior y sólo necesita la perspectiva de proyecto de desarrollo Java estándar. Complete los pasos siguientes para configurar el entorno de desarrollo de WebSphere eXtreme Scale.

1. Abra Eclipse en un espacio de trabajo nuevo o existente.
2. En el menú Archivo, seleccione **Importar**.
3. Expanda la carpeta General. Seleccione **Proyectos existentes en espacio de trabajo** y pulse **Siguiente**.
4. En el campo **Seleccionar directorio raíz**, escriba o vaya al directorio *raíz_wxs_sample_osgi*. Pulse **Finalizar**. Se visualizan varios proyectos en el espacio de trabajo. Debe corregir varios errores de construcción definiendo la biblioteca de usuario de eXtreme Scale. Complete los pasos siguientes para definir la biblioteca de usuario.
5. En el menú Ventana, seleccione **Preferencias**.
6. Expanda la rama **Java** > **Vía de acceso de compilación** y seleccione **Bibliotecas de usuario**.
7. Pulse **Nueva**.
8. Especifique `eXtremeScale` en el campo **Nombre de biblioteca de usuario** y pulse **Aceptar**.
9. Seleccione la nueva biblioteca de usuario y pulse **Añadir JAR**.
   a. Vaya al archivo `objectgrid.jar` del directorio *raíz_instalación_wxs*`/lib` y selecciónelo. Pulse **Aceptar**.
   b. Para incluir la documentación de la API para las API de ObjectGrid, seleccione la ubicación de la documentación de la API para el archivo `objectgrid.jar` que ha añadido en el paso anterior. Pulse **Editar**.
   c. En el recuadro de vía de acceso de ubicación de la documentación de la API, seleccione el archivo `Javadoc.zip` incluido en el directorio siguiente: *raíz_instalación_wxs*`/docs/javadoc.zip`.

Punto de comprobación de la lección:

En esta lección, ha importado el proyecto Eclipse de ejemplo, ha definido la biblioteca de usuario de eXtreme Scale y ha incluido documentación de la API de soporte para el proyecto de ejemplo. Ahora está preparado para iniciar la aplicación cliente de ejemplo.

Lección 3.2: Iniciar un cliente e insertar datos en la cuadrícula

Complete esta lección para iniciar un cliente no OSGi y ejecutar una aplicación cliente.

La aplicación de cliente Java es
`com.ibm.websphere.samples.xs.proto.client.Client`.

Este cliente utiliza una sustitución del cliente, el archivo XML de descriptor de ObjectGrid para sustituir la configuraación de OSGi, de forma que el cliente pueda ejecutarse en un entorno no OSGi. Consulte el contenido siguiente del archivo donde se han eliminado los comentarios y las cabeceras.Algunas líneas de código se muestran en varias líneas por razones de formato.

```
<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" txTimeout="15">
            <bean id="ObjectGridEventListener" className="" osgiService=""/>
            <backingMap name="Map" readOnly="false"
                lockStrategy="PESSIMISTIC" lockTimeout="5"
                copyMode="COPY_TO_BYTES" pluginCollectionRef="serializer"/>

        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="serializer">

<bean id="MapSerializer"
className="com.ibm.websphere.samples.xs.serializer.proto.ProtoMapSerializer"
       osgiService="">
<property name="keyType" type="java.lang.String"
value="com.ibm.websphere.samples.xs.serializer.proto.DataObjects2$OrderKey" />
<property name="valueType" type="java.lang.String"
value="com.ibm.websphere.samples.xs.serializer.proto.DataObjects2$Order" />
   </bean>
  </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Complete los pasos siguientes para iniciar la aplicación cliente.

1. Utilice el siguiente ejemplo de código para modificar los atributos de la clase cliente para que reflejen su entorno.

   ```
   private String catHost = "localhost";
   private int catListenerPort = 2809;
   private String clientOGXML = "wxs_sample_osgi_root/client/META-INF/
   clientProtoBufObjectgrid.xml";
   private String gridName = "Grid";
   private String mapName = "Map";
   ```

2. Ejecute la aplicación cliente.

Al ejecutar la aplicación, se visualiza el mensaje siguiente. El mensaje indica que se ha insertado un pedido:

```
order
com.ibm.websphere.samples.xs.serializer.proto.DataObjects1$Or
der$Builder@5d165d16(5000000) inserted
```

Punto de comprobación de la lección:

En esta lección, ha iniciado la aplicación
`com.ibm.websphere.samples.xs.proto.client.Client`, que ha generado un pedido.

Módulo 4: Consulta y actualización del paquete de ejemplo

Complete las lecciones de este módulo para utilizar el mandato **xscmd** para consultar la clasificación de servicio del paquete de ejemplo, actualizarla a una nueva clasificación de servicio y verificar la nueva clasificación de servicio.

Se proporciona un proyecto eclipse como una manera cómoda de ejecutar las aplicaciones de ejemplo.

Objetivos del aprendizaje

Después de completar las lecciones de este módulo, sabrá cómo completar las tareas siguientes:

- Consultar la clasificación de servicio actual del servicio.
- Consultar la clasificación actual de todos los servicios.
- Consultar todas las clasificaciones disponibles para un servicio.
- Consultar todas las clasificaciones de servicio disponibles.
- Utilizar la herramienta xscmd para verificar si hay disponibles clasificaciones de servicio específicas.
- Actualizar las clasificaciones de servicio para servicios OSGi de ejemplo.

Requisitos previos

Complete el Módulo 3: Ejecución del cliente de ejemplo de eXtreme Scale.

Lección 4.1: Consultar clasificaciones de servicio

Complete esta lección para consultar las clasificaciones de servicio actuales así como las clasificaciones de servicio disponibles para su actualización.

- Consulte la clasificación de servicio actual del servicio. Especifique el mandato siguiente para consultar la clasificación de servicio actual que se utiliza para el servicio, myShardListener, que utiliza el ObjectGrid denominado `Grid` y el conjunto de correlaciones denominado `MapSet`.
  1. Cambie al directorio siguiente:
     ```
     cd inicio_wxs/bin
     ```
  2. Especifique el mandato siguiente para consultar la clasificación de servicio actual correspondiente al servicio, myShardListener.
     ```
     ./xscmd.sh -c osgiCurrent -g Grid -ms MapSet -sn myShardListener
     ```

     Se visualiza la salida siguiente:
     ```
     OSGi Service Name: myShardListener
     ObjectGrid Name MapSet Name Server Name        Current Ranking
     --------------- ----------- -----------        ---------------
     Grid            MapSet      collocatedServer   1

     CWXSI0040I: The command osgiCurrent has completed successfully.
     ```
- Consulte la clasificación actual de todos los servicios. Especifique el mandato siguiente para consultar la clasificación de servicio actual de todos los servicios utilizados por el ObjectGrid denominado `Grid` y el conjunto de correlaciones denominado `MapSet`.
  1. Cambie al directorio siguiente:
     ```
     cd inicio_wxs/bin
     ```
  2. Especifique el mandato siguiente para consultar la clasificación de servicio actual de todos los servicios.
     ```
     ./xscmd.sh -c osgiCurrent -g Grid -ms MapSet
     ```

     Se visualiza la salida siguiente:

```
OSGi Service Name    Current Ranking ObjectGrid Name MapSet Name Server Name
-----------------    --------------- --------------- ----------- -----------
myProtoBufSerializer 1               Grid            MapSet      collocatedServer
```

```
myShardListener      1              Grid           MapSet      collocatedServer

CWXSI0040I: The command osgiCurrent has completed successfully.
```

- Consulte todas las clasificaciones disponibles para un servicio. Especifique el mandato siguiente para consultar todas las clasificaciones de servicio disponibles para el servicio denominado myShardListener.
  1. Cambie al directorio siguiente:
     ```
     cd inicio_wxs/bin
     ```
  2. Especifique el mandato siguiente para consultar todas las clasificaciones disponibles para un servicio.
     ```
     ./xscmd.sh -c osgiAll -sn myShardListener
     ```

     Se visualiza la salida siguiente:
     ```
     Server: collocatedServer
        OSGi Service Name Available Rankings
        ---------------- -----------------
        myShardListener 1   Summary - All servers have the same service rankings.

     CWXSI0040I: The command osgiAll has completed successfully.
     ```

     La salida la agrupa el servidor. En este ejemplo, sólo existe el siguiente servidor: collocatedServer.
- Consulte todas las clasificaciones de servicio disponibles. Entre el mandato siguiente para consultar todas las clasificaciones de servicio disponible para todos los servicios.
  1. Cambie al directorio siguiente:
     ```
     cd inicio_wxs/bin
     ```
  2. Especifique el mandato siguiente para consultar todas las clasificaciones de servicio disponibles.
     ```
     ./xscmd.sh -c osgiAll
     ```

     Se visualiza la salida siguiente:
     ```
     Server: collocatedServer
        OSGi Service Name    Available Rankings
        -----------------    ------------------
        myProtoBufSerializer 1
        myShardListener      1

     Summary - All servers have the same service rankings.
     ```
- Instale e inicie la versión 2 del paquete de plug-in. En la consola OSGi del servidor, instale un paquete nuevo que contenga una nueva versión de la clase Order y el plug-in MapSerializerPlugin. Consulte Lección 2.4: Instalar los paquetes Google Protocol Buffers y de plug-in de ejemplo para obtener más información sobre cómo instalar el paquete `ProtoBufSamplePlugins-2.0.0.jar`.
  1. Después de la instalación, inicie el nuevo paquete. Los servicios para el nuevo paquete están disponibles, pero el servidor eXtreme Scale no los utiliza aún. Debe ejecutar una solicitud de actualización de servicio para utilizar un servicio con una versión específica.
- Ahora al consultar de nuevo todas las clasificaciones de servicio disponibles, la clasificación de servicio 2 se añadirá a la salida.
  1. Cambie al directorio siguiente:
     ```
     cd inicio_wxs/bin
     ```
  2. Especifique el mandato siguiente para consultar todas las clasificaciones de servicio disponibles.

```
./xscmd.sh -c osgiAll
```

Se visualiza la salida siguiente:

```
Server: collocatedServer
   OSGi Service Name    Available Rankings
   -----------------    ------------------
   myProtoBufSerializer 1, 2
   myShardListener      1, 2

Summary - All servers have the same service rankings.
```

Punto de comprobación de la lección:

En esta guía de aprendizaje, ha consultado las clasificaciones de servicio especificadas actualmente y todas las disponibles. También ha visualizado la clasificación de servicio para un nuevo paquete que ha instalado e iniciado.

Lección 4.2: Determinar si hay clasificaciones de servicio específicas disponibles

Complete esta lección para determinar si hay clasificaciones de servicio específicas disponibles para los nombres de servicio que especifique.

1. Especifique el mandato siguiente para determinar si el servicio denominado myShardListener, con la clasificación de servicio 2 y el servicio denominado myProtoBufSerializer, con la clasificación de servicio 2, están disponibles. La lista de clasificaciones de servicio se proporciona utilizando la opción `-sr`.
   a. Cambie al directorio siguiente:
      ```
      cd inicio_wxs/bin
      ```
   b. Especifique el mandato siguiente para determinar si los servicios están disponibles:
      ```
      ./xscmd.sh -c osgiCheck -g Grid -ms MapSet -sr
      "myShardListener;2,myProtoBufSerializer;2"
      ```

      Se visualiza la salida siguiente:
      ```
      CWXSI0040I: The command osgiCheck has completed successfully.
      ```
2. Especifique el mandato siguiente para determinar si el servicio denominado myShardListener, con la clasificación de servicio 2 y el servicio denominado myProtoBufSerializer, con la clasificación de servicio 3, están disponibles.
   a. Cambie al directorio siguiente:
      ```
      cd inicio_wxs/bin
      ```
   b. Especifique el mandato siguiente para determinar si los servicios están disponibles:
      ```
      ./xsadmin.sh -c osgiCheck -g Grid -ms MapSet -sr
      "myShardListener;2,myProtoBufSerializer;3"
      ```

      Se visualiza la salida siguiente:
      ```
      Server OSGi Service Unavailable Rankings
      ------ ------------ --------------------
      collocatedServer myProtoBufSerializer 3
      ```

Punto de comprobación de la lección:

En esta lección, ha especificado los servicios myShardListener y myProtoBufSerializer, junto con clasificaciones de servicio específicas para determinar si estas clasificaciones estaban disponibles.

Lección 4.3: Actualizar las clasificaciones de servicio

Complete esta lección para actualizar clasificaciones de servicio actuales que haya consultado.

1. Si se especifica el mandato siguiente, se actualizarán las clasificaciones de servicio de los servicios denominados myShardListener y myProtoBufSerializer a la clasificación de servicio 2. La lista de clasificaciones de servicio se proporciona mediante la opción `-sr`.
   a. Cambie al directorio siguiente:
      ```
      cd inicio_wxs/bin
      ```
   b. Especifique el mandato siguiente para actualizar las clasificaciones de servicio:
      ```
      ./xscmd.sh -c osgiUpdate -g Grid -ms MapSet
      -sr "myShardListener;2,myProtoBufSerializer;2"
      ```

      Se visualiza la salida siguiente:
      ```
      La actualización ha sido satisfactoria para las siguientes clasificaciones
      de servicio:
      Service Ranking
      ------- -------
      myProtoBufSerializer 2
      myShardListener 2

      CWXSI0040I: The command osgiUpdate has completed successfully.
      ```

      Se visualiza la salida siguiente en la consola OSGi:
      ```
      SystemOut O MyShardListener@326505334(version=2.0.0) order
      com.ibm.websphere.samples.xs.serializer.proto.DataObjects2$Order$Builder@
      22342234(34) updated
      ```

      Tenga en cuenta que el servicio MyShardListener es ahora la versión 2.0.0, que tiene clasificación de servicio 2.
2. Si ejecuta el mandato **xscmd** para consultar la clasificación de servicio actual que se utiliza para todos los servicios que utiliza el ObjectGrid denominado `Grid` y el conjunto de correlaciones denominado `MapSet`.
   a. Cambie al directorio siguiente:
      ```
      cd inicio_wxs/bin
      ```
   b. Especifique el mandato siguiente para consultar las clasificaciones de servicio correspondientes a todos los servicios que utilizan `Grid` y `MapSet`:
      ```
      ./xscmd.sh -c osgiCurrent -g Grid -ms MapSet
      ```

      Se visualiza la salida siguiente:
      ```
      OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
      ----------------- --------------- --------------- ----------- -----------
      myProtoBufSerializer 2 Grid MapSet collocatedServer
      myShardListener 2 Grid MapSet collocatedServer

      CWXSI0040I: The command osgiCurrent has completed successfully.
      ```

Punto de comprobación de la lección:

En esta lección, ha actualizado las clasificaciones de servicio para los servicios myShardListener y myProtoBufSerializer.

.

Capítulo 2. Escenarios

Un escenario presenta ejemplos para ayudar al usuario a comprender un concepto o realizar una tarea. El escenario utiliza información real para crear una imagen completa.

Utilización de un entorno OSGi para desarrollar y ejecutar plug-ins de eXtreme Scale

Utilice estos escenarios para completar tareas comunes en un entorno OSGi. Por ejemplo, la infraestructura OSGi es ideal para iniciar servidores y clientes en un contenedor OSGi, lo que le permite añadir y actualizar dinámicamente plug-ins de WebSphere eXtreme Scale en el entorno de ejecución.

Los siguientes escenarios son sobre la creación y ejecución de plug-ins dinámicos, lo que le permite instalar, iniciar, detener, modificar y desinstalar plug-ins. También puede completar otro escenario probable, lo que le permite utilizar la infraestructura OSGi sin posibilidades dinámicas. Puede seguir empaquetando las aplicaciones como paquetes, que se definen y comunican mediante servicios. Estos paquetes basados en servicios ofrecen muchas ventajas, que incluyen posibilidades de desarrollo y despliegue más eficaces.

Objetivos del escenario

Después de completar las lecciones de este módulo, sabrá cómo completar las tareas siguientes:

- Crear plug-ins dinámicos de eXtreme Scale para utilizar en un entorno OSGi.
- Ejecutar contenedores de eXtreme Scale en un entorno OSGi sin prestaciones dinámicas.

Requisitos previos

Lea el tema “Visión general de la infraestructura OSGi” para obtener más información sobre el soporte de OSGi y las ventajas que puede ofrecer.

Visión general de la infraestructura OSGi

OSGi define un sistema de módulo dinámico para Java. La plataforma de servicio OSGi tiene una arquitectura por capas, y está diseñada para ejecutarse en diversos perfiles Java estándar. Puede iniciar servidores y clientes de WebSphere eXtreme Scale en un contenedor OSGi.

Ventajas de la ejecución de aplicaciones en el contenedor OSGi

El soporte OSGi de WebSphere eXtreme Scale le permite desplegar el producto en la infraestructura OSGi de Eclipse Equinox. Anteriormente, si deseaba actualizar los plug-ins utilizados por eXtreme Scale, tenía que reiniciar la máquina virtual Java (JVM) para aplicar las nuevas versiones de los plug-ins. Con la prestación de actualización dinámica que proporciona la infraestructura OSGi, ahora puede actualizar las clases de plug-in sin reiniciar la JVM. Estos plug-ins los exportan los

paquetes de usuario como servicios. WebSphere eXtreme Scale accede al servicio o a los servicios buscándolos en el registro OSGi.

Los contenedores de eXtreme Scale se pueden configurar para que se inicien de forma más fácil y dinámica utilizando el servicio de administración de configuración OSGi o con OSGi Blueprint. Si desea desplegar una cuadrícula de datos nueva con la estrategia de colocación, puede hacerlo creando una configuración OSGi o desplegando un paquete con archivos XML de descriptor de eXtreme Scale. Con el soporte de OSGi, los paquetes de aplicaciones que contienen eXtreme Scale se pueden instalar, iniciar, detener, actualizar y desinstalar sin reiniciar todo el sistema. Con esta posibilidad, puede actualizar la aplicación sin interrumpir la cuadrícula de datos.

Se pueden configurar beans y servicios de plug-in con ámbitos de fragmento personalizados, lo que permite opciones de integración sofisticadas opciones con otros servicios que se ejecutan en la cuadrícula de datos. Cada plug-in puede utilizar clasificaciones OSGi Blueprint para verificar que cada instancia del plug-in está activada en la versión correcta. Se proporcionan un bean gestionado por OSGi (MBean) y el programa de utilidad **xscmd**, que permiten consultar los servicios OSGi de plug-in de eXtreme Scale y sus clasificaciones.

Esta prestación permite a los administradores reconocer rápidamente los errores potenciales de configuración y administración y actualizar las clasificaciones de servicio de plug-in utilizadas por eXtreme Scale.

Paquetes OSGi

Para interactuar con los plug-ins y desplegarlos en la infraestructura OSGi, debe utilizar *paquetes*. En la plataforma de servicio OSGi, un paquete es un archivo de archivado Java (JAR) que contiene código Java, recursos y un manifiesto que describe el paquete y sus dependencias. El paquete es la unidad de despliegue de una aplicación. El producto eXtreme Scale da soporte a los siguientes tipos de paquete:

Paquete de servidor
El paquete de servidor es el archivo `objectgrid.jar`, se instala con la instalación de servidor autónomo de eXtreme Scale, es necesario para ejecutar servidores eXtreme Scale y también se puede utilizar para ejecutar clientes de eXtreme Scale o cachés locales en memoria. El ID de paquete para el archivo `objectgrid.jar` es com.ibm.websphere.xs.server_`<versión>`, donde la versión tiene el formato: `<Versión>.<Release>.<Modificación>`. Por ejemplo, el paquete de servidor para eXtreme Scale versión 7.1.1 es com.ibm.websphere.xs.server_7.1.1.

Paquete de cliente
El paquete de cliente es el archivo `ogclient.jar`, se instala con las instalaciones autónomas y de cliente de eXtreme Scale y se utiliza para ejecutar clientes de eXtreme Scale o cachés locales en memoria. El ID de paquete para el archivo `ogclient.jar` es com.ibm.websphere.xs.client_`versión`, donde la versión tiene el formato: `<Versión>.<Release>.<Modificación>`. Por ejemplo, el paquete de cliente para eXtreme Scale versión 7.1.1 es com.ibm.websphere.xs.client_7.1.1.

Limitaciones

No puede reiniciar el paquete de eXtreme Scale porque no puede reiniciar el Intermediario para solicitudes de objetos (ORB). Para reiniciar el servidor eXtreme

Scale, debe reiniciar la infraestructura OSGi.

Tareas relacionadas:

"Instalación de la infraestructura OSGi de Eclipse Equinox con Eclipse Gemini para clientes y servidores"
Si desea desplegar WebSphere eXtreme Scale en una infraestructura OSGi, debe configurar el entorno de Eclipse Equinox.

"Gestión de ciclos de vida de plug-ins" en la página 301
Puede gestionar ciclos de vida de plug-ins con métodos especializados para cada plug-in, que están disponibles para su invocación en puntos funcionales designados. Tanto el método initialize como el método destroy definen el ciclo de vida de los plug-ins, que controlan sus objetos *owner*. Un objeto propietario es el objeto que utiliza realmente el plug-in determinado. Un propietario puede ser un cliente de cuadrícula, un servidor o una correlación de respaldo.

Referencia relacionada:

Archivo de propiedades de servidor
El archivo de propiedades de servidor contiene varias propiedades que definen distintos valores para el servidor como, por ejemplo, los valores de rastreo, el inicio de sesión y la configuración de seguridad. El archivo de propiedades del servidor lo utilizan el servicio de catálogo y los servidores de contenedor tanto en servidores autónomos como en servidores alojados en WebSphere Application Server.

Información relacionada:

"Introducción: Inicio y configuración del servidor y contenedor de eXtreme Scale para ejecutar plug-ins en la infraestructura OSGi" en la página 19
En esta guía de aprendizaje inicia un servidor eXtreme Scale en la infraestructura OSGi, inicia un contenedor de eXtreme Scale y conecta los plug-ins de ejemplo al entorno de ejecución de eXtreme Scale.

Documentación de la API

Instalación de la infraestructura OSGi de Eclipse Equinox con Eclipse Gemini para clientes y servidores

Si desea desplegar WebSphere eXtreme Scale en una infraestructura OSGi, debe configurar el entorno de Eclipse Equinox.

Acerca de esta tarea

La tarea requiere que descargue e instale la infraestructura Blueprint, lo que le permite configurar posteriormente JavaBeans y exponerlos como servicios. El uso de los servicios es importante porque puede exponer plug-ins como servicios OSGi de forma que los pueda utilizar el entorno de ejecución de eXtreme Scale. El producto da soporte a dos contenedores blueprint en la infraestructura OSGi principal de Eclipse Equinox: Eclipse Gemini y Apache Aries. Utilice este procedimiento para configurar el contenedor Eclipse Gemini.

Procedimiento

1. Descargue Eclipse Equinox SDK Versión 3.6.1 o posterior del sitio web de Eclipse. Cree un directorio para la infraestructura Equinox, por ejemplo: `/opt/equinox`. Estas instrucciones hacen referencia a este directorio como `raíz_equinox`. Extraiga el archivo comprimido en el directorio `raíz_equinox`.
2. Descargue el archivo comprimido de gemini-blueprint incubation 1.0.0 del sitio web de Eclipse. Extraiga el contenido del archivo en un directorio temporal y copie los siguientes archivos extraídos en el directorio `raíz_equinox/plugins`:

```
dist/gemini-blueprint-core-1.0.0.jar
dist/gemini-blueprint-extender-1.0.0.jar
dist/gemini-blueprint-io-1.0.0.jar
```

3. Descargue la infraestructura Spring versión 3.0.5 de la siguiente página web de SpringSource: http://www.springsource.com/download/community. Extráigala en un directorio temporal y copie los siguientes archivos extraídos en el directorio raíz_equinox/plugins:

   ```
   org.springframework.aop-3.0.5.RELEASE.jar
   org.springframework.asm-3.0.5.RELEASE.jar
   org.springframework.beans-3.0.5.RELEASE.jar
   org.springframework.context-3.0.5.RELEASE.jar
   org.springframework.core-3.0.5.RELEASE.jar
   org.springframework.expression-3.0.5.RELEASE.jar
   ```

4. Descargue el archivo de archivado Java archive (JAR) de AOP Alliance de la página web de SpringSource. Copie el archivo com.springsource.org.aopalliance-1.0.0.jar en el directorio raíz_equinox/plugins.
5. Descargue el archivo JAR de Apache Commons Logging 1.1.1 de la página web de SpringSource. Copie el archivo com.springsource.org.apache.commons.logging-1.1.1.jar en el directorio raíz_equinox/plugins.
6. Descargue el cliente de línea de mandatos de Luminis OSGi Configuration Admin. Utilice este paquete para gestionar las configuraciones administrativas de OSGi. Puede descargar el archivo JAR de la siguiente página web: https://opensource.luminis.net/wiki/display/SITE/OSGi+Configuration+Admin+command+line+client. Copie el archivo net.luminis.cmc-0.2.5.jar en el directorio raíz_equinox/plugins.
7. Descargue el paquete de instalación de archivos de la Versión 3.0.2 de Apache Felix de la siguiente página web: http://felix.apache.org/site/index.html. Copie el archivo org.apache.felix.fileinstall-3.0.2.jar en el directorio raíz_equinox/plugins.
8. Cree un directorio de configuración en el directorio equinox_root/plugins, por ejemplo:

   ```
   mkdir equinox_root/plugins/configuration
   ```

9. Cree el archivo config.ini siguiente en el directorio equinox_root/plugins/configuration, sustituyendo *equinox_root* por la vía de acceso absoluta al directorio equinox_root y eliminando todos los espacios de cola después de la barra inclinada invertida de cada línea. Debe incluir una línea en blanco al final del archivo; por ejemplo:

   ```
   osgi.noShutdown=true
   osgi.java.profile.bootdelegation=none
   org.osgi.framework.bootdelegation=none
   eclipse.ignoreApp=true
   osgi.bundles=\
   org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
   org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
   org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
   com.springsource.org.apache.commons.logging-1.1.1.jar@1:start, \
   com.springsource.org.aopalliance-1.0.0.jar@1:start, \
   org.springframework.aop-3.0.5.RELEASE.jar@1:start, \
   org.springframework.asm-3.0.5.RELEASE.jar@1:start, \
   org.springframework.beans-3.0.5.RELEASE.jar@1:start, \
   org.springframework.context-3.0.5.RELEASE.jar@1:start, \
   org.springframework.core-3.0.5.RELEASE.jar@1:start, \
   org.springframework.expression-3.0.5.RELEASE.jar@1:start, \
   org.apache.felix.fileinstall-3.0.2.jar@1:start, \
   net.luminis.cmc-0.2.5.jar@1:start, \
   gemini-blueprint-core-1.0.0.jar@1:start, \
   gemini-blueprint-extender-1.0.0.jar@1:start, \
   gemini-blueprint-io-1.0.0.jar@1:start
   ```

   Si ya ha configurado el entorno, puede limpiar el repositorio de plug-ins de Equinox eliminando el directorio siguiente: raíz_equinox\plugins\configuration\org.eclipse.osgi.

10. Ejecute los mandatos siguientes para iniciar la consola de equinox.

    Si está ejecutando una versión distinta de Equinox, el nombre de archivo JAR será distinto al del ejemplo siguiente:

    ```
    java -jar plugins\org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
    ```

Conceptos relacionados:

"Visión general de la infraestructura OSGi" en la página 37
OSGi define un sistema de módulo dinámico para Java. La plataforma de servicio OSGi tiene una arquitectura por capas, y está diseñada para ejecutarse en diversos perfiles Java estándar. Puede iniciar servidores y clientes de WebSphere eXtreme Scale en un contenedor OSGi.

Referencia relacionada:

Archivo de propiedades de servidor
El archivo de propiedades de servidor contiene varias propiedades que definen distintos valores para el servidor como, por ejemplo, los valores de rastreo, el inicio de sesión y la configuración de seguridad. El archivo de propiedades del servidor lo utilizan el servicio de catálogo y los servidores de contenedor tanto en servidores autónomos como en servidores alojados en WebSphere Application Server.

Información relacionada:

"Introducción: Inicio y configuración del servidor y contenedor de eXtreme Scale para ejecutar plug-ins en la infraestructura OSGi" en la página 19
En esta guía de aprendizaje inicia un servidor eXtreme Scale en la infraestructura OSGi, inicia un contenedor de eXtreme Scale y conecta los plug-ins de ejemplo al entorno de ejecución de eXtreme Scale.

Instalación de paquetes de eXtreme Scale

WebSphere eXtreme Scale incluye paquetes que se pueden instalar en una infraestructura OSGi de Eclipse Equinox. Estos paquetes son necesarios para iniciar los servidores eXtreme Scale o utilizar clientes de eXtreme Scale en OSGi.

Antes de empezar

En esta tarea se supone que se han instalado los productos siguientes:

- Infraestructura OSGi de Eclipse Equinox
- Cliente o servidor autónomo de eXtreme Scale

Acerca de esta tarea

eXtreme Scale incluye dos paquetes. Sólo se necesita uno de los paquetes siguientes en una infraestructura OSGi:

objectgrid.jar
: El paquete de servidor es el archivo `objectgrid.jar` que se instala con la instalación de servidor autónomo de eXtreme Scale, es necesario para ejecutar servidores eXtreme Scale y también se puede utilizar para ejecutar clientes eXtreme Scale o cachés locales en memoria. El ID de paquete para el archivo `objectgrid.jar` es com.ibm.websphere.xs.server_<versión>, donde la versión tiene el formato: <Versión>.<Release>.<Modificación>. Por ejemplo, el paquete de servidor para eXtreme Scale versión 7.1.1 es com.ibm.websphere.xs.server_7.1.1.

ogclient.jar
: El paquete `ogclient.jar` se instala con las instalaciones autónomas y de cliente de eXtreme Scale y se utiliza para ejecutar clientes de eXtreme Scale o cachés locales en memoria. El ID de paquete para el archivo

ogclient.jar es com.ibm.websphere.xs.client_<versión>, donde la versión está en el formato: <Versión>_<Release>_<Modificación. Por ejemplo, el paquete de cliente para eXtreme Scale Versión 7.1.1 es com.ibm.websphere.xs.client_7.1.1.

Para obtener más información sobre el desarrollo de plug-ins de eXtreme Scale, consulte el tema Plug-ins y API del sistema.

Procedimiento

Para instalar el paquete de cliente o servidor de eXtreme Scale en la infraestructura OSGi de Eclipse Equinox utilizando la consola de OSGi:

1. Inicie la infraestructura de Eclipse Equinox con la consola habilitada; por ejemplo:

   ```
   inicio_java/bin/java -jar <raíz_equinox>/plugins/
   org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```

2. Instale el paquete de cliente o servidor de eXtreme Scale en la consola de Equinox:

   ```
   osgi> install file:///<vía_acceso_archivo>
   ```

3. Equinox visualiza el ID de paquete para el paquete recién instalado:

   ```
   El ID de paquete es 25
   ```

4. Inicie el paquete en la consola de Equinox, donde <id> es el ID de paquete asignado al instalar el paquete:

   ```
   osgi>  start <id>
   ```

5. Recupere el estado de servicio en la consola de Equinox para verificar que el paquete se ha iniciado; por ejemplo:

   ```
   osgi> ss
   ```

   Cuando el paquete se ha iniciado satisfactoriamente, visualiza el estado ACTIVO; por ejemplo:

   ```
   25      ACTIVE      com.ibm.websphere.xs.server_7.1.1
   ```

Instale el paquete de cliente o servidor de eXtreme Scale en la infraestructura OSGi de Eclipse Equinox utilizando el archivo config.ini:

6. Copie el paquete de cliente o servidor de eXtreme Scale (objectgrid.jar o ogclient.jar) del directorio <raíz_instalación_wxs>/ObjectGrid/lib en el directorio de plug-ins de Eclipse Equinox; por ejemplo: <raíz_equinox>/plugins
7. Edite el archivo de configuración config.ini de Eclipse Equinox y añada el paquete a la propiedad osgi.bundles; por ejemplo:

   ```
   osgi.bundles=\
   org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
   org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
   org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
   objectgrid.jar@1:start
   ```

   Importante: Verifique que haya una línea en blanco después del último nombre de paquete. Cada paquete está separado por una coma.

8. Inicie la infraestructura de Eclipse Equinox con la consola habilitada; por ejemplo:

   ```
   inicio_java/bin/java -jar <raíz_equinox>/plugins/
   org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```

9. Recupere el estado de servicio en la consola de Equinox para verificar que el paquete se ha iniciado:

```
osgi> ss
```

Cuando el paquete se haya iniciado satisfactoriamente, visualizará el estado ACTIVO; por ejemplo:

```
25      ACTIVE      com.ibm.websphere.xs.server_7.1.1
```

Resultados

El paquete de servidor o cliente de eXtreme Scale se ha instalado e iniciado en la infraestructura OSGi de Eclipse Equinox.

Creación y ejecución de plug-ins dinámicos de eXtreme Scale para su uso en un entorno OSGi

Todos los plug-ins de eXtreme Scale se pueden configurar para un entorno OSGi. La principal ventaja de los plug-ins dinámicos es que le permiten actualizarlos sin concluir la cuadrícula. Esto le permite desarrollar una aplicación sin reiniciar los procesos del contenedor de cuadrícula.

Acerca de esta tarea

El soporte OSGi de WebSphere eXtreme Scale le permite desplegar el producto en una infraestructura OSGi como por ejemplo Eclipse Equinox. Anteriormente, si deseaba actualizar los plug-ins utilizados por eXtreme Scale, tenía que reiniciar la máquina virtual Java (JVM) para aplicar las nuevas versiones de los plug-ins. Con el soporte de plug-ins dinámicos proporcionado por eXtreme Scale y la capacidad de actualizar paquetes que proporciona la infraestructura OSGi, ahora puede actualizar las clases de plug-in sin reiniciar la JVM. Estos plug-ins los exportan los *paquetes* como servicios. WebSphere eXtreme Scale accede al servicio consultando el registro OSGi. En la plataforma de servicio OSGi, un paquete es un archivo de archivado Java (JAR) que contiene código Java, recursos y un manifiesto que describe el paquete y sus dependencias. El paquete es la unidad de despliegue de una aplicación.

Procedimiento

1. Crear plug-ins dinámicos de eXtreme Scale.
2. Configurar plug-ins de eXtreme Scale con OSGi Blueprint.
3. Instalar e iniciar plug-ins habilitados para OSGi.

Creación de plug-ins dinámicos de eXtreme Scale

WebSphere eXtreme Scale incluye los plug-ins ObjectGrid y BackingMap. Estos plug-ins se implementan en Java y se configuran utilizando el archivo XML de descriptor de ObjectGrid. Para crear un plug-in dinámico que se pueda actualizar dinámicamente, es necesario estar al corriente de los sucesos de ciclo de vida de ObjectGrid y BackingMap porque es posible que sea necesario completar algunas acciones durante la actualización. La ampliación de un paquete de plug-in con métodos de devolución de llamada de ciclo de vida, escuchas de sucesos, o ambos, permite al plug-in completar estas acciones en los momentos adecuados.

Antes de empezar

En este tema se supone que ha creado el plug-in apropiado. Para obtener más información sobre el desarrollo de plug-ins de eXtreme Scale, consulte el tema Plug-ins y API del sistema.

Acerca de esta tarea

Todos los plug-ins de eXtreme Scale se aplican a una instancia BackingMap u ObjectGrid. Muchos plug-ins también interactúan con otros plug-ins. Por ejemplo, un cargador y un plug-in TransactionCallback trabajan juntos para interactuar correctamente con una transacción de base de datos y las diversas llamadas JDBC de base de datos. Es posible que algunos plug-ins requieran también que se almacenen en la memoria caché datos de configuración de otros plug-ins a fin de mejorar el rendimiento.

Los plug-ins BackingMapLifecycleListener y ObjectGridLifecycleListener proporcionan operaciones de ciclo de vida para las instancias BackingMap y ObjectGrid respectivas. Este proceso permite notificar a los plug-ins cuando es posible que se cambien la BackingMap o la ObjectGrid padre y sus respectivos plug-ins. Los plug-ins BackingMap implementan la interfaz BackingMapLifecyleListener y los plug-ins ObjectGrid implementan la interfaz ObjectGridLifecycleListener. Estos plug-ins se invocan automáticamente cuando cambia el ciclo de vida de la BackingMap o ObjectGrid padre. Para obtener más información sobre los plug-ins de ciclo de vida, consulte el tema “Gestión de ciclos de vida de plug-ins” en la página 301.

Puede esperar ampliar los paquetes utilizando los métodos de ciclo de vida o escuchas de suceso en las siguientes tareas comunes:

- Inicio y detención de recursos, como por ejemplo hebras o suscriptores de mensajería.
- Si se especifica que se produzca una notificación cuando los plug-ins de igual se actualicen, lo que permite acceso directo al plug-in y la detección de los cambios.

Siempre que acceda a otro plug-in directamente, acceda a ese plug-in mediante el contenedor OSGi para asegurarse de que todas las partes del sistema hagan referencia al plug-in correcto. Si, por ejemplo, algún componente de la aplicación almacena en la memoria caché o hace referencia directamente a una instancia de un plug-in, mantendrá su referencia a esa versión del plug-in, incluso después de que el plug-in se haya actualizado dinámicamente. Este comportamiento puede causar problemas relacionados con la aplicación así como fugas de memoria. Por consiguiente, escriba código que dependa de plug-ins dinámicos que obtienen la referencia utilizando la semántica OSGi, getService(). Si la aplicación debe almacenar en memoria caché uno o varios plug-ins, escucha los sucesos de ciclo de vida utilizando las interfaces ObjectGridLifecycleListener y BackingMapLifecycleListener. La aplicación debe poder renovar también su memoria caché cuando sea necesario, en modalidad de seguridad de hebra.

Todos los plug-ins de eXtreme Scale utilizados con OSGi también deben implementar las interfaces BackingMapPlugin u ObjectGridPlugin respectivas. Los plug-ins nuevos, como la interfaz MapSerializerPlugin, imponen esta práctica. Estas interfaces proporcionan al entorno de ejecución de eXtreme Scale y a OSGi una interfaz coherente para inyectar el estado en el plug-in y controlar su ciclo de vida.

Utilice esta tarea para especificar que se produzca una notificación cuando se actualicen plug-ins de igual. Puede crear una fábrica de escuchas que genere una instancia de escucha.

Procedimiento

- Actualice la clase de plug-in ObjectGrid para implementar la interfaz ObjectGridPlugin. Esta interfaz incluye métodos que permiten a eXtreme Scale inicializar, establecer la instancia de ObjectGrid y destruir el plug-in. Consulte el siguiente código de ejemplo:

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridPlugin;
...

public class MyTranCallback implements TransactionCallback, ObjectGridPlugin {

    private ObjectGrid og = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setObjectGrid(ObjectGrid grid) {
        this.og = grid;
    }

    public ObjectGrid getObjectGrid() {
        return this.og;
    }
    void initialize() {
        // Manejar la inicialización de plug-in aquí. Lo llama
        // eXtreme Scale y no el gestor de beans OSGi.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destruir el plug-in y liberar los recursos. A éste
        // lo puede llamar el gestor de beans OSGi o eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- Actualice la clase de plug-in ObjectGrid para implementar la interfaz ObjectGridLifecycleListener. Consulte el siguiente código de ejemplo:

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener;
      import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener.LifecycleEvent;
      ...

      public class MyTranCallback implements TransactionCallback, ObjectGridPlugin, ObjectGridLifecycleListener{
          public void objectGridStateChanged(LifecycleEvent event) {
              switch(event.getState()) {
              case NEW:
              case DESTROYED:
              case DESTROYING:
              case INITIALIZING:
                  break;
              case INITIALIZED:
                  // Buscar un cargador o MapSerializerPlugin utilizando
                  // OSGi o directamente desde la instancia de ObjectGrid.
                  lookupOtherPlugins()
                  break;
              case STARTING:
              case PRELOAD:
                  break;
              case ONLINE:
                  if (event.isWritable()) {
                      startupProcessingForPrimary();
                  } else {
                      startupProcessingForReplica();
                  }
                  break;
              case QUIESCE:
                  if (event.isWritable()) {
                      quiesceProcessingForPrimary();
                  } else {
                      quiesceProcessingForReplica();
                  }
                  break;
              case OFFLINE:
                  shutdownShardComponents();
                  break;
```

```
            }
        }
        ...
    }
```

- Actualice un plug-in BackingMap. Actualice la clase de plug-in BackingMap para implementar la interfaz de plug-in BackingMap. Esta interfaz incluye métodos que permiten a eXtreme Scale inicializar, establecer la instancia de BackingMap y destruir el plug-in. Consulte el siguiente código de ejemplo:

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.BackingMapPlugin;
...

public class MyLoader implements Loader, BackingMapPlugin {

    private BackingMap bmap = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setBackingMap(BackingMap map) {
        this.bmap = map;
    }

    public BackingMap getBackingMap() {
        return this.bmap;
    }
    void initialize() {
        // Manejar la inicialización de plug-in aquí. Lo llama
        // eXtreme Scale y no el gestor de beans OSGi.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destruir el plug-in y liberar los recursos. A éste
        // lo puede llamar el gestor de beans OSGi o eXtreme Scale.
        state = State.DESTROYED;
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- Actualice la clase de plug-in BackingMap para implementar la interfaz BackingMapLifecycleListener. Consulte el siguiente código de ejemplo:

```
package com.mycompany;

    import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener;
    import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener.LifecycleEvent;
    ...

    public class MyLoader implements Loader, ObjectGridPlugin, ObjectGridLifecycleListener{
        ...
        public void backingMapStateChanged(LifecycleEvent event) {
            switch(event.getState()) {
            case NEW:
            case DESTROYED:
            case DESTROYING:
            case INITIALIZING:
                break;
            case INITIALIZED:
                // Buscar un MapSerializerPlugin utilizando
                // OSGi o directamente desde la instancia de ObjectGrid.
                lookupOtherPlugins()
                break;
            case STARTING:
            case PRELOAD:
                break;
            case ONLINE:
                if (event.isWritable()) {
                    startupProcessingForPrimary();
                } else {
                    startupProcessingForReplica();
                }
                break;
            case QUIESCE:
                if (event.isWritable()) {
                    quiesceProcessingForPrimary();
                } else {
                    quiesceProcessingForReplica();
                }
                break;
            case OFFLINE:
```

```
                shutdownShardComponents();
                break;
            }
        }
        ...
    }
```

Resultados

Al implementar la interfaz ObjectGridPlugin o BackingMapPlugin, eXtreme Scale puede controlar el ciclo de vida del plug-in en los momentos correctos.

Al implementar la interfaz ObjectGridLifecycleListener o BackingMapLifecycleListener, el plug-in se registra automáticamente como escucha de los sucesos de ciclo de vida ObjectGrid o BackingMap asociados. El suceso INITIALIZING se utiliza para señalar que todos los plug-ins ObjectGrid y BackingMap se han inicializado y están disponibles para buscarse y utilizarse. El suceso ONLINE se utiliza para señalar que el ObjectGrid está en línea y listo para iniciar el proceso de sucesos.

Configuración de plug-ins de eXtreme Scale con OSGi Blueprint

Todos los plug-ins de eXtreme Scale ObjectGrid y BackingMap se pueden definir como servicios y beans OSGi utilizando el servicio OSGi Blueprint disponible con Eclipse Gemini o Apache Aries.

Antes de empezar

Antes de configurar los plug-ins como servicios OSGi, primero debe empaquetar los plug-ins en un paquete OSGi y conocer los principios fundamentales de los plug-ins necesarios. El paquete debe importar los paquetes de servidor o cliente de WebSphere eXtreme Scale y otros paquetes dependientes necesarios para los plug-ins o crear una dependencia de paquete en los paquetes de servidor o cliente de eXtreme Scale. Este tema describe cómo configurar el XML de Blueprint para crear beans de plug-ins y exponerlos como servicios OSGi para que eXtreme Scale los utilice.

Acerca de esta tarea

Los beans y servicios están definidos en un archivo XML Blueprint y el contenedor Blueprint descubre, crea y conecta los beans entre ellos y los expone como servicios. El proceso deja los beans disponibles para otros paquetes OSGi, incluidos los paquetes de servidor y cliente de eXtreme Scale.

Al crear servicios de plug-in personalizados para utilizarlos con eXtreme Scale, el paquete que va a alojar los plug-ins, debe estar configurado para utilizar Blueprint. Además, se debe crear y almacenar un archivo XML Blueprint dentro del paquete. Para obtener una visión general de la especificación, lea la información sobre la creación de aplicaciones OSGi con la especificación de contenedor Blueprint.

Procedimiento

1. Cree un archivo XML Blueprint. Puede utilizar el nombre que desee para el archivo. No obstante, debe incluir el espacio de nombre blueprint:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0">
   ...
   </blueprint>
   ```

2. Cree definiciones de bean en el archivo XML Blueprint para cada plug-in de eXtreme Scale.

Los beans se definen utilizando el elemento <bean>, se pueden conectar a otras referencias de bean y pueden incluir parámetros de inicialización.

Importante: Al definir un bean, debe utilizar el ámbito correcto. Blueprint soporta los ámbitos de singleton y prototipo. eXtreme Scale también soporta un ámbito de fragmento personalizado.

Defina la mayoría de los plug-ins de eXtreme Scale como beans de ámbito de fragmento o prototipo, ya que todos los beans deben ser exclusivos para cada fragmento ObjectGrid o instancia de BackingMap con los que estén asociados. Los beans de ámbito de fragmento pueden ser útiles cuando se utilizan los beans en otros contextos para permitir recuperar la instancia correcta.

Para definir un bean de ámbito de prototipo, utilice el atributo scope="prototype" en el bean:

```
<bean id="myPluginBean" class="com.mycompany.MyBean" scope="prototype">
...
</bean>
```

Para definir un bean de ámbito de fragmento, debe añadir el espacio de nombres objectgrid al esquema XML y utilizar el atributo scope="objectgrid:shard" en el bean:

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"

    xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
                http://www.ibm.com/schema/objectgrid/objectgrid.xsd">

    <bean id="myPluginBean" class="com.mycompany.MyBean"
   scope="objectgrid:shard">
    ...
    </bean>

...
```

3. Cree definiciones de bean PluginServiceFactory para cada bean de plug-in. Todos los beans de eXtreme Scale deben tener un bean PluginServiceFactory definido para que se pueda aplicar el ámbito de bean correcto. eXtreme Scale incluye un BlueprintServiceFactory que se puede utilizar. Incluye dos propiedades que se deben establecer. Debe establecer la propiedad blueprintContainer en la referencia blueprintContainer y la propiedad beanId se debe establecer en el nombre de identificador de bean. Cuando eXtreme Scale busca el servicio para instanciar los beans adecuados, el servidor busca la instancia de componente de bean utilizando el contenedor Blueprint.

```
bean id="myPluginBeanFactory"
    class="com.ibm.websphere.objectgrid.plugins.osgi.BluePrintServiceFactory">
      <property name="blueprintContainer" ref="blueprintContainer"/>
<property name="beanId" value="myPluginBean" />
</bean>
```

4. Crear un administrador de servicios para cada bean PluginServiceFactory. Cada administrador de servicios expone el bean PluginServiceFactory, utilizando el elemento <service>. El elemento de servicio identifica el nombre a exponer en OSGi, la referencia al bean PluginServiceFactory, la interfaz a exponer y la clasificación del servicio. eXtreme Scale utiliza la clasificación de administrador de servicios para realizar actualizaciones de servicio cuando la cuadrícula de eXtreme Scale está activa. Si no se especifica la clasificación, la infraestructura OSGi supone una clasificación de 0. Lea la información sobre la actualización de clasificaciones de servicio para obtener más información.

Blueprint incluye varias opciones para configurar administradores de servicios. Para definir un administrador de servicios simple para un bean PluginServiceFactory, cree un elemento `<service>` para cada bean PluginServiceFactory:

```
<service ref="myPluginBeanFactory"
    interface="com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory"
    ranking="1">
</service>
```

5. Almacene el archivo XML Blueprint en el paquete de plug-ins. El archivo XML Blueprint debe almacenarse en el directorio `OSGI-INF/blueprint` para que se descubra el contenedor Blueprint.

   Para almacenar el archivo XML Blueprint en un directorio diferente, debe especificar la siguiente cabecera de manifiesto Bundle-Blueprint:

   ```
   Bundle-Blueprint: OSGI-INF/blueprint.xml
   ```

Resultados

Los plug-ins de eXtreme Scale están ahora configurados para exponerse en un contenedor OSGi Blueprint. Además, el archivo XML de descriptor ObjectGrid está configurado para hacer referencia a los plug-ins utilizando el servicio OSGi Blueprint.

Instalación e inicio de plug-ins habilitados para OSGi

En esta tarea, instalará el paquete de plug-in dinámico en la infraestructura OSGi. A continuación, iniciará el plug-in.

Antes de empezar

En este tema se supone que se han completado las tareas siguientes:

- Se ha instalado el paquete de servidor o cliente de eXtreme Scale en la infraestructura OSGi de Eclipse Equinox. Consulte “Instalación de paquetes de eXtreme Scale” en la página 41.
- Se han implementado uno o varios plug-ins dinámicos de BackingMap u ObjectGrid. Consulte “Creación de plug-ins dinámicos de eXtreme Scale” en la página 43.
- Los plug-ins dinámicos se han empaquetado como servicios OSGi en paquetes OSGi.

Acerca de esta tarea

Esta tarea describe cómo instalar el paquete utilizando la consola Eclipse Equinox. El paquete se puede instalar utilizando varios métodos diferentes, incluida la modificación del archivo de configuración `config.ini`. Los productos que incorporan Eclipse Equinox incluyen métodos alternativos para gestionar paquetes. Para obtener más información sobre cómo añadir paquetes en el archivo `config.ini` de Eclipse Equinox, consulte los opciones de ejecución de Eclipse.

OSGi permite que se inicien paquetes que tienen servicios duplicados. WebSphere eXtreme Scale utiliza la clasificación de servicios más reciente. Al iniciar varias infraestructuras OSGi en una cuadrícula de datos de eXtreme Scale, debe asegurarse de que se inician las clasificaciones de servicio correctas en cada servidor. Si no es así, la cuadrícula se inicia con una mezcla de versiones diferentes.

Para ver qué versiones están siendo utilizadas por la cuadrícula de datos, utilice el programa de utilidad xscmd para comprobar las clasificaciones actuales y disponibles. Para obtener más información sobre las clasificaciones de servicio disponibles, consulte Actualización de servicios OSGi para plug-ins de eXtreme Scale con **xscmd**.

Procedimiento

Instalar el paquete de plug-in en la infraestructura OSGi de Eclipse Equinox utilizando la consola OSGi.

1. Inicie la infraestructura de Eclipse Equinox con la consola habilitada; por ejemplo:
   ```
   <inicio_java>/bin/java -jar <raíz_equinox>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
2. Instale el paquete de plug-in en la consola de Equinox.
   ```
   osgi> install file:///<vía_acceso_archivo>
   ```

   Equinox visualiza el ID de paquete para el paquete recién instalado:
   ```
   Bundle id is 17
   ```
3. Entre la línea siguiente para iniciar el paquete en la consola de Equinox, donde <id> es el ID de paquete asignado al instalar el paquete:
   ```
   osgi>  install <id>
   ```
4. Recupere el estado de servicio en la consola de Equinox para verificar que el paquete se ha iniciado:
   ```
   osgi> ss
   ```

   Cuando el paquete se ha iniciado satisfactoriamente, visualiza el estado ACTIVO; por ejemplo:
   ```
   17      ACTIVE      com.mycompany.plugin.bundle_VRM
   ```

Instalar el paquete de plug-in en la infraestructura OSGi de Eclipse Equonix utilizando el archivo config.ini.

5. Copie el paquete de plug-in en el directorio de plug-ins de Eclipse Equinox; por ejemplo:
   ```
   <raíz_equinox>/plugins
   ```
6. Edite el archivo de configuración config.ini de Eclipse Equinox y añada el paquete a la propiedad osgi.bundles; por ejemplo:
   ```
   osgi.bundles=\
   org.eclipse.osgi.services_3.2.100.v20100503.jar@1:start, \
   org.eclipse.osgi.util_3.2.100.v20100503.jar@1:start, \
   org.eclipse.equinox.cm_1.0.200.v20100520.jar@1:start, \
   com.mycompany.plugin.bundle_VRM.jar@1:start
   ```

   Importante: Verifique que haya una línea en blanco después del último nombre de paquete. Cada paquete está separado por una coma.
7. Inicie la infraestructura de Eclipse Equinox con la consola habilitada; por ejemplo:
   ```
   <inicio_java>/bin/java -jar <raíz_equinox>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
8. Recupere el estado de servicio en la consola de Equinox para verificar que el paquete se ha iniciado; por ejemplo:
   ```
   osgi> ss
   ```

   Cuando el paquete se ha iniciado satisfactoriamente, visualiza el estado ACTIVO; por ejemplo:

```
17      ACTIVE      com.mycompany.plugin.bundle_VRM
```

Resultados

El paquete de plug-in ya está instalado e iniciado. Ahora ya se puede iniciar el contenedor o cliente de eXtreme Scale. Para obtener más información sobre el desarrollo de plug-ins de eXtreme Scale, consulte el tema Plug-ins y API del sistema.

Ejecución de contenedores de eXtreme Scale con plug-ins dinámicos en un entorno OSGi

Si la aplicación se aloja en la infraestructura OSGi de Eclipse Equinox con Eclipse Gemini o Apache Aries, puede utilizar esta tarea para ayudar a instalar y configurar la aplicación WebSphere eXtreme Scale en OSGi.

Antes de empezar

Antes de iniciar esta tarea, asegúrese de completar las tareas siguientes:

- Instalar la infraestructura OSGi de Eclipse Equinox con Eclipse Gemini
- Crear y ejecutar plug-ins dinámicos de eXtreme Scale para utilizarlos en un entorno OSGi

Acerca de esta tarea

Con los plug-ins dinámicos, puede actualizar dinámicamente el plug-in mientras la cuadrícula sigue activa. Esto le permite actualizar una aplicación sin reiniciar los procesos del contenedor de cuadrícula. Para obtener más información sobre el desarrollo de plug-ins de eXtreme Scale, consulte Plug-ins y API del sistema.

Procedimiento

1. Configure los plug-ins habilitados para OSGi utilizando el archivo XML de descriptor ObjectGrid.
2. Inicie los servidores de contenedor eXtreme Scale utilizando la infraestructura OSGi de Eclipse Equinox.
3. Administre los servicios OSGi para los plug-ins eXtreme Scale con el programa de utilidad xscmd.
4. Configure servdires con OSGi Blueprint.

Configuración de plug-ins habilitados para OSGi mediante el archivo XML de descriptor de ObjectGrid

En esta tarea, se añaden servicios OSGi existentes a un archivo XML de descriptor para que el contenedor de WebSphere eXtreme Scale pueda reconocer y cargar correctamente los plug-ins habilitados para OSGi.

Antes de empezar

Para configurar los plug-ins, asegúrese de:

- Crear el paquete y habilitar plug-ins dinámicos para despliegue OSGi.
- Tener los nombres de los servicios OSGi que representan los plug-ins disponibles.

Acerca de esta tarea

Ha creado un servicio OSGi para recortar los plug-in. Ahora, estos servicios deben definirse en el archivo `objectgrid.xml` para que los contenedores de eXtreme Scale pueden cargar y configurar el plug-in o los plug-ins correctamente.

Procedimiento

1. Cualquier plug-in específico de cuadrícula, por ejemplo TransactionCallback, debe especificarse en el elemento objectGrid. Consulte el ejemplo siguiente del archivo `objectgrid.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>

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

    <objectGrids>
        <objectGrid name="MyGrid" txTimeout="60">
            <bean id="myTranCallback" osgiService="myTranCallbackFactory"/>
            ...
        </objectGrid>
        ...
    </objectGrids>
    ...
/objectGridConfig>
```

   Importante: El valor de atributo `osgiService` debe coincidir con el valor de atributo `ref` que se especifica en el archivo XML `blueprint`, donde se ha definido el servicio para myTranCallback PluginServiceFactory.

2. Cualquier plug-in específico de correlación, por ejemplo los cargadores o serializadores, se debe especificar en el elemento backingMapPluginCollections y se debe hacer referencia a él desde el elemento backingMap. Consulte el ejemplo siguiente del archivo `objectgrid.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>

objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="MyGrid" txTimeout="60">
            <backingMap name="MyMap1" lockStrategy="PESSIMISTIC"
                copyMode="COPY_TO_BYTES" nullValuesSupported="false"
                pluginCollectionRef="myPluginCollectionRef1"/>
            <backingMap name="MyMap2" lockStrategy="PESSIMISTIC"
                copyMode="COPY_TO_BYTES" nullValuesSupported="false"
                pluginCollectionRef="myPluginCollectionRef2"/>
            ...
        </objectGrid>
        ...
    </objectGrids>
    ...
    <backingMapPluginCollections>
        <backingMapPluginCollection id="myPluginCollectionRef1">
            <bean id="MapSerializerPlugin" osgiService="mySerializerFactory"/>
        </backingMapPluginCollection>
        <backingMapPluginCollection id="myPluginCollectionRef2">
            <bean id="MapSerializerPlugin" osgiService="myOtherSerializerFactory"/>
            <bean id="Loader" osgiService="myLoader"/>
        </backingMapPluginCollection>
        ...
    </backingMapPluginCollections>
    ...
</objectGridConfig>
```

Resultados

El archivo `objectgrid.xml` de este ejemplo indica a eXtreme Scale que cree una cuadrícula denominada `MyGrid` con dos correlaciones, `MyMap1` y `MyMap2`. La correlación `MyMap1` utiliza el serializador recortado por el servicio OSGi, mySerializerFactory. La correlación `MyMap2` utiliza un serializador del servicio OSGi,

myOtherSerializerFactory, y un cargador del servicio OSGi, myLoader.

Inicio de servidores eXtreme Scale utilizando la infraestructura OSGi de Eclipse Equinox

Los servidores de contenedor de WebSphere eXtreme Scale se pueden iniciar en una infraestructura OSGi de Eclipse Equinox utilizando varios métodos.

Antes de empezar

Para poder iniciar un contenedor eXtreme Scale, debe haber completado las siguientes tareas:

1. El paquete de servidor de WebSphere eXtreme Scale debe estar instalado en Eclipse Equinox.
2. La aplicación debe estar empaquetado como un paquete OSGi.
3. Los plug-ins de WebSphere eXtreme Scale (si existen) deben estar empaquetados como un paquete OSGi. Pueden estar empaquetados en el mismo paquete que la aplicación o como paquetes independientes.

Acerca de esta tarea

Esta tarea describe cómo iniciar un servidor de contenedor eXtreme Scale en una infraestructura OSGi de Eclipse Equinox. Puede utilizar cualquiera de los métodos siguientes para iniciar los servidores de contenedor utilizando la implementación de Eclipse Equinox:

- Servicio OSGi Blueprint

  Puede incluir toda la configuración y los metadatos en un paquete OSGi. Consulte la imagen siguiente para comprender el proceso de Eclipse Equinox para este método:

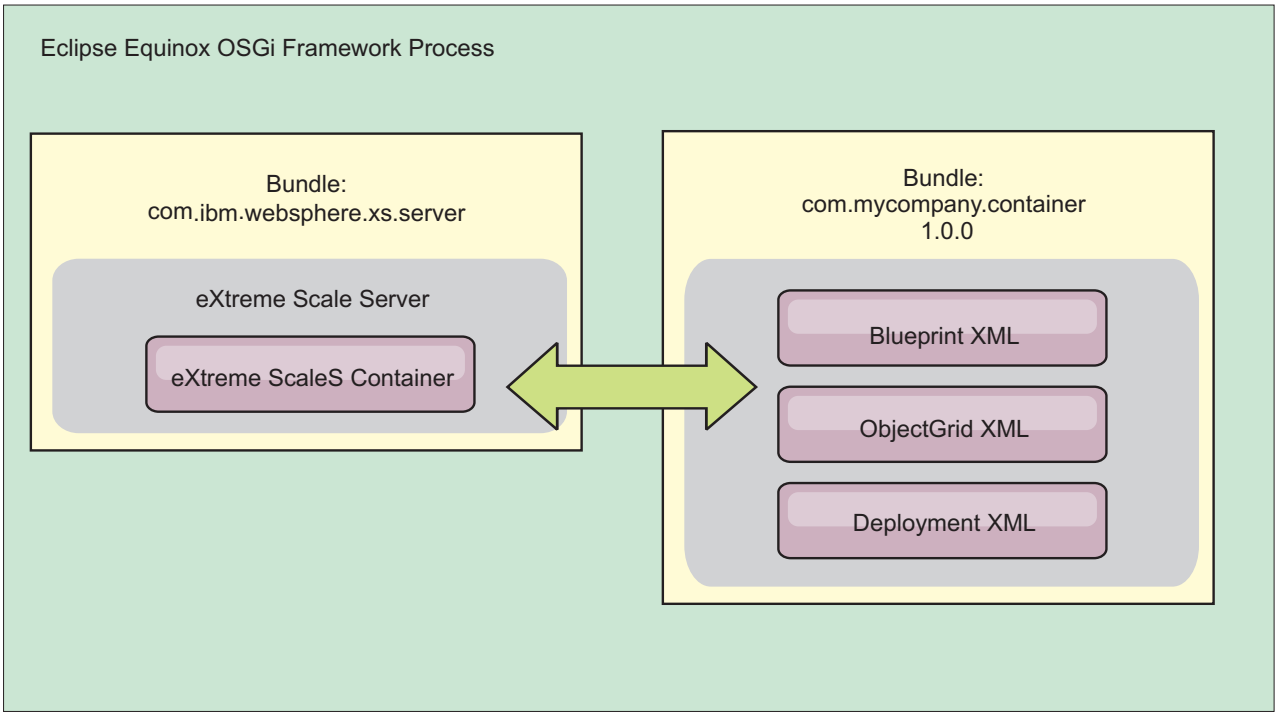


Figura 2. Proceso de Eclipse Equinox para incluir toda la configuración y los metadatos en un paquete OSGi

- Servicio de administración de configuración OSGi

  Puede especificar la configuración y los metadatos fuera de un paquete OSGi. Consulte la imagen siguiente para comprender el proceso de Eclipse Equinox para este método:

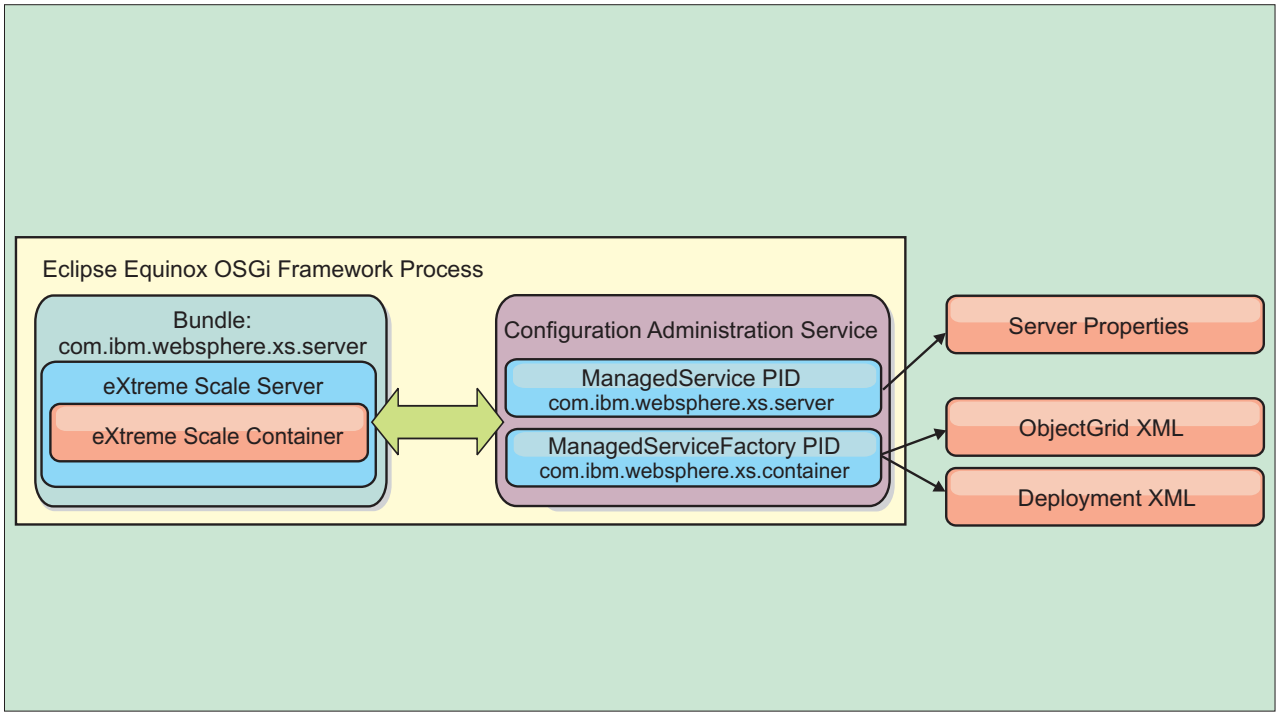


Figura 3. Proceso de Eclipse Equinox para especificar la configuración y los metadatos fuera de un paquete OSGi

- A través de programas

  Soporta soluciones de configuración personalizadas.

En cada caso, se configura un singleton de servidor eXtreme Scale y se configuran uno o varios contenedores.

El paquete de servidor eXtreme Scale, `objectgrid.jar`, incluye todas las bibliotecas necesarias para iniciar y ejecutar un contenedor de cuadrícula de eXtreme Scale en una infraestructura OSGi. El entorno de ejecución de servidor se comunica con los objetos de datos y los plug-ins proporcionados por el usuario utilizando el administrador de servicios OSGi.

Importante: Después de que un paquete de servidor eXtreme Scale se haya iniciado y el servidor eXtreme Scale se haya inicializado, no se puede reiniciar. Se debe reiniciar el proceso de Eclipse Equinox para reiniciar un servidor eXtreme Scale.

Puede utilizar el soporte de eXtreme Scale para el espacio de nombres Spring para configurar los servidores de contenedor de eXtreme Scale en un archivo XML Blueprint. Cuando se añaden los elementos XML de servidor y contenedor al archivo XML Blueprint, el manejador de espacio de nombres de eXtreme Scale inicia automáticamente un servidor de contenedor utilizando los parámetros definidos en el archivo XML Blueprint cuando se inicia el paquete. El manejador detiene el contenedor cuando se detiene el paquete.

Para configurar servidores de contenedor de eXtreme Scale con XML Blueprint, realice los pasos siguientes:

Procedimiento

- Inicie un servidor de contenedor de eXtreme Scale utilizando OSGi Blueprint.
  1. Cree un paquete de contenedor.
  2. Instale el paquete de contenedor en la infraestructura OSGi de Eclipse Equinox. Consulte "Instalación e inicio de plug-ins habilitados para OSGi" en la página 49.
  3. Inicie el paquete de contenedor.
- Inicie un servidor de contenedor de eXtreme Scale utilizando la administración de configuración de OSGi.
  1. Configure el servidor y el contenedor utilizando la administración de configuración.
  2. Cuando el paquete de servidor de eXtreme Scale se ha iniciado o los identificadores persistentes se crean con la administración de configuración, el servidor y el contenedor se inician automáticamente.
- Inicie un servidor de contenedor de eXtreme Scale utilizando la API ServerFactory. Consulte la documentación de API de servidor.
  1. Cree una clase de activador de paquete OSGi y utilice la API ServerFactory de eXtreme Scale para iniciar un servidor.

Administración de servicios habilitado para OSGi utilizando el programa de utilidad `xscmd`

Puede utilizar el programa de utilidad **`xscmd`** para completar las tareas de administrador, por ejemplo ver los servicios y sus clasificaciones que están siendo utilizados por cada contenedor y actualizar el entorno de ejecución para utilizar las nuevas versiones de los paquetes.

Acerca de esta tarea

Con la infraestructura OSGi de Eclipse Equinox, puede instalar varias versiones del mismo paquete y puede actualizar esos paquetes durante la ejecución. WebSphere eXtreme Scale es un entorno distribuido que ejecuta los servidores de contenedor en muchas instancias de infraestructura OSGi.

Los administradores son responsables de copiar, instalar e iniciar manualmente paquetes en la infraestructura OSGi. eXtreme Scale incluye un ServiceTrackerCustomizer OSGi para realizar un seguimiento de los servicios que se han identificado como plug-ins de eXtreme Scale en el archivo XML de descriptor de ObjectGrid. Utilice el programa de utilidad **`xscmd`** para validar qué versión del plug-in se utiliza, qué versiones están disponibles para utilizarse y para realizar actualizaciones de paquete.

eXtreme Scale utiliza el número de clasificación de servicio para identificar la versión de cada servicio. Cuando se cargan dos o más servicios con la misma referencia, eXtreme Scale utiliza automáticamente el servicio con la clasificación más alta.

Procedimiento

- Ejecute el mandato **`osgiCurrent`** y verifique que cada servidor de eXtreme Scale utiliza la clasificación de servicio de plug-in correcta.

Dado que eXtreme Scale elige automáticamente la referencia de servicio con la clasificación más alta, es posible que la cuadrícula de datos empiece con varias clasificaciones de un servicio de plug-in.

Si el mandato detecta una discrepancia de clasificaciones o si no puede encontrar un servicio, se establece un nivel de error distinto de cero. Si el mandato se ha completado satisfactoriamente, el nivel de error se establece en 0.

El siguiente ejemplo muestra la salida del mandato **osgiCurrent** cuando dos plug-ins se instalan en la misma cuadrícula en cuatro servidores. El plug-in loaderPlugin utiliza la clasificación 1 y txCallbackPlugin utiliza la clasificación 2.

```
OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
----------------- --------------- --------------- ----------- -----------
loaderPlugin      1               MyGrid          MapSetA     server1
loaderPlugin      1               MyGrid          MapSetA     server2
loaderPlugin      1               MyGrid          MapSetA     server3
loaderPlugin      1               MyGrid          MapSetA     server4
txCallbackPlugin  2               MyGrid          MapSetA     server1
txCallbackPlugin  2               MyGrid          MapSetA     server2
txCallbackPlugin  2               MyGrid          MapSetA     server3
txCallbackPlugin  2               MyGrid          MapSetA     server4
```

El siguiente ejemplo muestra la salida del mandato **osgiCurrent** cuando server2 se ha iniciado con una clasificación más reciente de loaderPlugin:

```
OSGi Service Name Current Ranking ObjectGrid Name MapSet Name Server Name
----------------- --------------- --------------- ----------- -----------
loaderPlugin      1               MyGrid          MapSetA     server1
loaderPlugin      2               MyGrid          MapSetA     server2
loaderPlugin      1               MyGrid          MapSetA     server3
loaderPlugin      1               MyGrid          MapSetA     server4
txCallbackPlugin  2               MyGrid          MapSetA     server1
txCallbackPlugin  2               MyGrid          MapSetA     server2
txCallbackPlugin  2               MyGrid          MapSetA     server3
txCallbackPlugin  2               MyGrid          MapSetA     server4
```

- Ejecute el mandato **osgiAll** para verificar que los servicios de plug-in se han iniciado correctamente en cada servidor de contenedor de eXtreme Scale.

  Al iniciar paquetes que contienen servicios a los que una configuración ObjectGrid hace referencia, el entorno de ejecución de eXtreme Scale realiza automáticamente un seguimiento del plug-in, pero no lo utiliza inmediatamente. El mandato **osgiAll** muestra qué plug-ins están disponibles para cada servidor.

  Cuando se ejecuta sin parámetros, se muestran todos los servicios para todas las cuadrículas y servidores. Se pueden especificar filtros adicionales, incluido el filtro **-serviceName** <nombre_servicio>, para limitar la salida a un solo servicio o a un subconjunto de la cuadrícula de datos.

  El ejemplo siguiente muestra la salida del mandato **osgiAll** cuando se inician dos plug-ins en dos servidores. loaderPlugin tiene las dos clasificaciones 1 y 2 iniciadas y txCallbackPlugin tiene la clasificación 1 iniciada. El mensaje de resumen al final de la salida confirma que ambos servidores ven las mismas clasificaciones de servicio:

```
Server: server1
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Server: server2
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Summary - All servers have the same service rankings.
```

El ejemplo siguiente muestra la salida del mandato **osgiAll** cuando el paquete que incluye loaderPlugin con la clasificación 1 se detiene en server1. El mensaje de resumen en la parte inferior de la salida confirma que en server1 falta ahora loaderPlugin con la clasificación 1:

```
Server: server1
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        2
   txCallbackPlugin    1

Server: server2
   OSGi Service Name   Available Rankings
   -----------------   ------------------
   loaderPlugin        1, 2
   txCallbackPlugin    1

Summary - The following servers are missing service rankings:
   Server  OSGi Service Name Missing Rankings
   ------  ----------------- ----------------
   server1 loaderPlugin      1
```

El siguiente ejemplo muestra la salida si el nombre de servicio se especifica con el argumento **-sn**, pero el servicio no existe:

```
Server: server2
   OSGi Service Name Available Rankings
   ----------------- ------------------
   invalidPlugin     No service found

Server: server1
   OSGi Service Name Available Rankings
   ----------------- ------------------
   invalidPlugin     No service found

Summary - All servers have the same service rankings.
```

- Ejecute el mandato **osgiCheck** para comprobar conjuntos de servicios de plug-in y clasificaciones para ver si están disponibles.

  El mandato **osgiCheck** acepta uno o más conjuntos de clasificaciones de servicio en el formato: `-serviceRankings <nombre de servicio>;<clasificación>[,<nombreServicio>;<clasificación>]`

  Cuando las clasificaciones están todas disponibles, el método vuelve con un nivel de error de 0. Si una o más clasificaciones no están disponibles, se establece un nivel de error distinto de cero y una tabla de todos los servidores que no incluyen las clasificaciones de servicio especificadas. Se pueden utilizar filtros adicionales para limitar la comprobación de servicio a un subconjunto de los servidores disponibles en el dominio de eXtreme Scale.

  Por ejemplo, si la clasificación o el servicio especificados están ausentes, se visualiza el siguiente mensaje:

  ```
  Server  OSGi Service Unavailable Rankings
  ------  ------------ --------------------
  server1 loaderPlugin 3
  server2 loaderPlugin 3
  ```

- Ejecute el mandato **osgiUpdate** para actualizar la clasificación de uno o más plug-ins para todos los servidores de una sola ObjectGrid y MapSet en una sola operación.

  El mandato acepta uno o más conjuntos de clasificaciones de servicio con el formato: `-serviceRankings <nombre de servicio>;<clasificación>[,<nombreServicio>;<clasificación>] -g <nombre de cuadrícula> -ms <nombre de conjunto de correlaciones>`

  Con este mandato, puede completar las siguientes operaciones:

- Verifique que los servicios especificados están disponibles para actualizarse en cada uno de los servidores.
- Cambie el estado de la cuadrícula a fuera de línea utilizando la interfaz StateManager. Si desea más información, consulte Gestión de la disponibilidad del ObjectGrid. Este proceso inmoviliza la cuadrícula, espera a que se hayan completado las transacciones en ejecución e impide que se inicien transacciones nuevas. Este proceso también señala a los plug-ins ObjectGridLifecycleListener y BackingMapLifecycleListener que interrumpan cualquier actividad transaccional. Consulte "Plug-ins para proporcionar escuchas de sucesos" en la página 320 para obtener información sobre los plug-ins de escucha de sucesos.
- Actualice cada contenedor de eXtreme Scale que se ejecuta en una infraestructura OSGi para utilizar las nuevas versiones de servicio.
- Cambie el estado de la cuadrícula para que esté en línea, lo que permite que continúen las transacciones.

El proceso de actualización es idempotent, de modo que si un cliente no puede completar ninguna tarea, la operación se retrotrae. Si un cliente no puede realizar la retrotracción o se interrumpe durante el proceso de actualización, se puede emitir el mismo mandato de nuevo y continúa en el paso adecuado.

Si el cliente no puede continuar y el proceso se reinicia desde otro cliente, utilice la opción `-force` para permitir que el cliente realice la actualización. El mandato **`osgiUpdate`** impide que varios clientes actualicen el mismo conjunto de correlaciones simultáneamente. Para obtener más detalles sobre el mandato **`osgiUpdate`**, consulte Actualización de servicios OSGi para plug-ins de eXtreme Scale con **`xscmd`**.

Configuración de servidores con OSGi Blueprint

Puede configurar servidores de contenedor de WebSphere eXtreme Scale utilizando un archivo XML de OSGi Blueprint, lo que permite simplificar el empaquetado y el desarrollo de paquetes de servidor autocontenidos.

Antes de empezar

En este tema se supone que se han completado las tareas siguientes:

- Se ha instalado e iniciado la infraestructura OSGi de Eclipse Equinox con el contenedor Eclipse Gemini o Apache Aries Blueprint.
- Se ha instalado e iniciado el paquete de servidor de eXtreme Scale.
- Se ha creado el paquete de plug-ins dinámicos de eXtreme Scale.
- Se han creado el archivo XML de política de despliegue y el archivo XML de descriptor de ObjectGrid de eXtreme Scale.

Acerca de esta tarea

Esta tarea describe cómo configurar un servidor de eXtreme Scale con un contenedor utilizando un archivo XML Blueprint. El resultado del procedimiento es paquete de contenedor. Cuando se inicie el paquete de contenedor, el paquete de el servidor eXtreme Scale realizará un seguimiento del paquete, analizará el XML de servidor e iniciará un servidor y contenedor.

Un paquete de contenedor se puede combinar de manera opcional con la aplicación y los plug-ins de eXtreme Scale cuando no son necesarias actualizaciones de plug-in dinámicas o los plug-ins no soportan la actualización dinámica.

Procedimiento

1. Cree un archivo XML Blueprint con el espacio de nombres `objectgrid` incluido. Puede utilizar el nombre que desee para el archivo. No obstante, debe incluir el espacio de nombres blueprint:

```
<?xml version="1.0" encoding="UTF-8"?>

<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
           xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
           xsi:schemaLocation="http://www.ibm.com/schema/objectgrid
                http://www.ibm.com/schema/objectgrid/objectgrid.xsd">
...
</blueprint>
```

2. Añada la definición XML para el servidor de eXtreme Scale con las propiedades de servidor adecuadas. Consulte el archivo XML de descriptor Spring para obtener detalles sobre todas las propiedades de configuración disponibles. Consulte el ejemplo siguiente de la definición XML:

```
objectgrid:server
    id="xsServer"
tracespec="ObjectGridOSGi=all=enabled"
   tracefile="logs/osgi/wxsserver/trace.log"
    jmxport="1199"
   listenerPort="2909">
    <objectgrid:catalog host="catserver1.mycompany.com" port="2809" />
    <objectgrid:catalog host="catserver2.mycompany.com" port="2809" />
</objectgrid:server>
```

3. Añadir la definición XML para el contenedor de eXtreme Scale con la referencia a la definición de servidor y a los archivos XML de descriptor ObjectGrid y XML de despliegue ObjectGrid incorporados en el paquete; por ejemplo:

```
<objectgrid:container id="container"
    objectgridxml="/META-INF/objectGrid.xml"
    deploymentxml="/META-INF/objectGridDeployment.xml"
    server="xsServer" />
```

4. Almacene el archivo XML Blueprint en el paquete de contenedor. El XML Blueprint se debe almacenar en el directorio `OSGI-INF/blueprint` para que se encuentre el contenedor Blueprint.

   Para almacenar el archivo XML Blueprint en un directorio diferente, debe especificar la cabecera de manifiesto Bundle-Blueprint; por ejemplo:

```
Bundle-Blueprint: OSGI-INF/blueprint.xml
```

5. Empaquete los archivos en un solo archivo JAR de paquete. Consulte el ejemplo siguiente de una jerarquía de directorios de paquete:

```
MyBundle.jar
    /META-INF/manifest.mf
    /META-INF/objectGrid.xml
    /META-INF/objectGridDeployment.xml
    /OSGI-INF/blueprint/blueprint.xml
```

Resultados

Ya se ha creado un paquete de contenedor de eXtreme Scale y ahora se puede instalar en Eclipse Equinox. Cuando se inicia el paquete de contenedor, el entorno de ejecución de servidor de eXtreme Scale del paquete de servidor de eXtreme Scale inicia automáticamente el servidor de eXtreme Scale de singleton utilizando los parámetros definidos en el paquete e inicia un servidor de contenedor. El paquete se puede detener e iniciar, lo que hace que el contenedor se detenga y se inicie. El servidor es un singleton y no se detiene cuando el paquete se inicia por primera vez.

Capítulo 3. Cómo empezar

Después de instalar el producto, puede utilizar el ejemplo de iniciación para probar la instalación y utilizar el producto por primera vez.

Guía de aprendizaje: Cómo empezar con WebSphere eXtreme Scale

Tras instalar WebSphere eXtreme Scale en un entorno autónomo, puede utilizar la aplicación de ejemplo de iniciación como una introducción sencilla a su capacidad como una cuadrícula de datos en memoria.

Objetivos del aprendizaje

- Obtener información sobre el archivo XML de descriptor de ObjectGrid y los archivos XML de descriptor de política de despliegue que utiliza para configurar el entorno
- Iniciar servidores de catálogo y contenedor mediante los archivos de configuración
- Obtener información sobre el desarrollo de una aplicación cliente
- Ejecutar la aplicación cliente para insertar datos en la cuadrícula de datos
- Supervisar las cuadrículas de datos con la consola web

Tiempo necesario

60 minutos

Lección 1 de la guía de aprendizaje de iniciación: Definición de cuadrículas de datos con archivos de configuración

Para configurar cuadrículas de datos simples, utilice los archivos `objectgrid.xml` y `deployment.xml` que se proporciona en el ejemplo de iniciación.

El ejemplo utiliza los archivos `objectgrid.xml` y `deployment.xml` que están en el directorio *raíz_intal_wxs*`/ObjectGrid/gettingstarted/xml`. Estos archivos se pasan a los mandatos de inicio para iniciar los servidores de contenedor y un servidor de catálogo. El archivo `objectgrid.xml` es el archivo XML de descriptor de ObjectGrid. El archivo `deployment.xml` es el archivo XML de política de descriptor de ObjectGrid. Estos archivos definen conjuntamente una topología distribuida.

Referencia relacionada:

Archivo XML de descriptor ObjectGrid
Para configurar WebSphere eXtreme Scale, utilice el archivo XML de descriptor de ObjectGrid y la API ObjectGrid.

Archivo XML de descriptor de política de despliegue
Para configurar una política de despliegue, utilice un archivo XML de descriptor de política de despliegue.

Archivo XML de descriptor ObjectGrid

Se utiliza un archivo XML de descriptor de ObjectGrid para definir la estructura del ObjectGrid que es utilizado por la aplicación. Incluye una lista de configuraciones de correlación de respaldo. Estas correlaciones de respaldo

almacenan los datos de memoria caché. El ejemplo siguiente es un archivo `objectgrid.xml` de ejemplo. Las primeras líneas del archivo incluyen la cabecera necesaria para cada archivo XML de ObjectGrid. Este archivo de ejemplo define el ObjectGrid `Grid` con las correlaciones de respaldo `Map1` y `Map2`.

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

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

</objectGridConfig>
```

Archivo XML de descriptor de política de despliegue

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

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

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

    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numberOfPartitions="13" minSyncReplicas="0"
     maxSyncReplicas="1" >
            <map ref="Map1"/>
            <map ref="Map2"/>
        </mapSet>
    </objectgridDeployment>

</deploymentPolicy>
```

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

El atributo minSyncReplicas de mapSet especifica el número mínimo de réplicas síncronas para cada partición del mapSet. Se trata de un atributo opcional y el valor predeterminado es 0. El primario y la réplica no se colocan hasta que el dominio pueda soportar el número mínimo de réplicas síncronas. Para dar soporte

al valor minSyncReplicas, es necesario un contenedor más que el valor de minSyncReplicas. Si el número de réplicas síncronas cae por debajo del valor de minSyncReplicas, ya no se permiten transacciones de grabación para esa partición.

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

Punto de comprobación de la lección

En esta lección, ha aprendido lo siguiente:

- Cómo definir correlaciones que almacenan datos en el archivo XML de descriptor de ObjectGrid.
- Cómo utilizar el archivo XML de descriptor de despliegue para definir el número de particiones y réplicas para la cuadrícula de datos.

Lección 2 de la guía de aprendizaje de iniciación: Creación de una aplicación cliente

Para insertar, suprimir, actualizar y recuperar datos de la cuadrícula de datos, debe escribir una aplicación de cliente. El ejemplo de iniciación incluye una aplicación cliente que puede utilizar para aprender a crear su propia aplicación cliente.

El archivo `Client.java` del directorio *raíz_intal_wxs*`/ObjectGrid/gettingstarted/client/src/` es el programa cliente que muestra cómo conectarse a un servidor de catálogo, obtener la instancia de ObjectGrid y utilizar la API ObjectMap. La API ObjectMap almacena datos como pares de clave-valor y es ideal para almacenar en memoria caché objetos que no tienen relaciones implicadas.

Si necesita almacenar en memoria caché objetos que tienen relaciones, utilice la API EntityManager.

1. Conéctese al servicio de catálogo obteniendo una instancia de ClientClusterContext.

   Para conectarse al servidor de catálogo, utilice el método connect de la API ObjectGridManager. El método connect que se utiliza sólo necesita el punto final de servidor de catálogo en el formato *nombre_de_host*`:`*puerto*. Pude indicar varios puntos finales de servidor de catálogo separando la lista de valores de *nombre_de_host*`:`*puerto* con comas. El fragmento de código siguiente muestra cómo conectar con un servidor de catálogo y obtener una instancia de ClientClusterContext:

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

   Si las conexiones con los servidores de catálogo son satisfactorias, el método connect devuelve una instancia ClientClusterContext. La instancia de ClientClusterContext es necesaria para obtener el ObjectGrid de la API ObjectGridManager.
2. Obtenga una instancia de ObjectGrid.

Para obtener una instancia de ObjectGrid, utilice el método getObjectGrid de la API ObjectGridManager. El método getObjectGrid requiere tanto la instancia de ClientClusterContext, como el nombre de la instancia de cuadrícula de datos. La instancia de ClientClusterContext se obtiene durante la conexión con el servidor de catálogo. El nombre de la instancia de ObjectGrid es `Grid` (cuadrícula) que se especifica en el archivo `objectgrid.xml`. El siguiente fragmento de código demuestra cómo obtener la cuadrícula de datos llamando al método getObjectGrid de la interfaz de programación de aplicaciones ObjectGridManager.

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

3. Obtenga una instancia de Session.

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

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

4. Obtenga una instancia de ObjectMap.

   Después de obtener una sesión, puede obtener una instancia de ObjectMap desde una sesión llamando al método getMap de la interfaz de programación de aplicaciones de la sesión en cuestión. Debe pasar el nombre de la correlación como correlación al métodogetMap para obtener la instancia de ObjectMap. El fragmento de código siguiente muestra cómo obtener ObjectMap llamando al método getMap de la API de sesión.

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

5. Utilice los métodos ObjectMap.

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

   El siguiente fragmento de código demuestra cómo utilizar la API ObjectMap con una transacción de confirmación automática.

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

   El siguiente fragmento de código demuestra cómo utilizar la API ObjectMap con la demarcación de transacción explícita.

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

Conceptos relacionados:

"Almacenamiento en memoria caché de objetos sin relaciones implicadas (API ObjectMap)" en la página 155
ObjectMaps son como correlaciones Java que permiten a los datos almacenarse como pares clave-valor. Los ObjectMap proporcionan un acercamiento sencillo e intuitivo para el almacenamiento de los datos de la aplicación. Un ObjectMap es ideal para almacenar en memoria caché los objetos que no tienen relaciones. Si hubiera relaciones de objeto, debería utilizar la API EntityManager.

Tareas relacionadas:

"Iniciación al desarrollo de aplicaciones" en la página 70
Para empezar a desarrollar aplicaciones WebSphere eXtreme Scale, configure un entorno de desarrollo en Eclipse.

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Punto de comprobación de la lección

En esta lección, ha aprendido cómo crear una única aplicación cliente para realizar operaciones de cuadrícula de datos.

Lección 3 de la guía de aprendizaje de iniciación: Ejecución de la aplicación cliente de ejemplo de iniciación

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

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

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

Para ejecutar la aplicación, en primer lugar inicie el proceso de servicio de catálogo. El servicio de catálogo es el centro de control de la cuadrícula de datos. Realiza un seguimiento de las ubicaciones de servidores de contenedor y controla la colocación de los datos para alojar servidores de contenedor. Después de que se inicie el servicio de catálogo, puede iniciar los servidores de contenedor, que almacenan los datos de la aplicación para la cuadrícula de datos. Para almacenar varias copias de los datos, puede iniciar varios servidores de contenedor. Cuando se han iniciado todos los servidores, puede ejecutar la aplicación cliente para insertar, actualizar, eliminar y obtener datos de la cuadrícula de datos.

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

   `cd` *`raíz_intal_wxs`*`/ObjectGrid/gettingstarted`

   Sustituya *raíz_intal_wxs* por la vía de acceso al directorio raíz de instalación de eXtreme Scale o por la vía de acceso de archivo raíz de *raíz_intal_wxs* de la versión de prueba extraída de eXtreme Scale.
3. Ejecute el siguiente script para iniciar un proceso de servicio de catálogo en el sistema principal local:

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

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

También puede iniciar el servicio de catálogo con el mandato **startOgServer**. Ejecute **startOgServer** en el directorio *raíz_intal_wxs*/ObjectGrid/bin:

- UNIX Linux `startOgServer.sh cs0 -catalogServiceEndPoints cs0:localhost:6600:6601 -listenerPort 2809`
- Windows `startOgServer.bat cs0 -catalogServiceEndPoints cs0:localhost:6600:6601 -listenerPort 2809`

4. Abra otra ventana de sesión de terminal o de línea de mandatos, y ejecute el siguiente mandato para iniciar una instancia de servidor de contenedor:
   - UNIX Linux `./runcontainer.sh server0`
   - Windows `runcontainer.bat server0`

   El servidor de contenedor se ejecuta en la ventana actual del terminal. Puede repetir este paso con un nombre de servidor distinto si desea iniciar más instancias de servidor de contenedor para dar soporte a la réplica.

   También puede iniciar servidores de contenedor con el mandato **startOgServer**. Ejecute **startOgServer** en el directorio *raíz_intal_wxs*/ObjectGrid/bin:

   - UNIX Linux `startOgServer.sh c0 -catalogServiceEndPoints localhost:2809 -objectgridFile gettingstarted\xml\objectgrid.xml -deploymentPolicyFile gettingstarted\xml\deployment.xml`
   - Windows `startOgServer.bat c0 -catalogServiceEndPoints localhost:2809 -objectgridFile gettingstarted\xml\objectgrid.xml -deploymentPolicyFile gettingstarted\xml\deployment.xml`
5. Abra otra ventana de sesión de terminal o de línea de mandatos para ejecutar los mandatos de cliente.

   El script `runclient.sh|bat` ejecute el cliente de CRUD sencillo e inicia la operación determinada. El script `runclient.sh|bat` se ejecuta con los parámetros siguientes:

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

   Para *mandato*, utilice una de las siguientes opciones:
   - Especifique `i` para insertar *valor2* en la cuadrícula de datos con la clave *valor1*
   - Especifique como `u` para actualizar el objeto con clave de *valor1* a *valor2*
   - Especifique como `d` para suprimir el objeto con clave por *valor1*
   - Especifique como `g` para recuperar y visualizar el objeto con clave por *valor1*

   a. Añada datos a la cuadrícula de datos:
      - UNIX Linux `./runclient.sh i key1 helloWorld`
      - Windows `runclient.bat i key1 helloWorld`

   b. Busque y visualice el valor:
      - UNIX Linux `./runclient.sh g key1`
      - Windows `runclient.bat g key1`

   c. Actualice el valor:
      - UNIX Linux `./runclient.sh u key1 goodbyeWorld`

- Windows `runclient.bat u key1 goodbyeWorld`

d. Suprima el valor:

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

Tareas relacionadas:

Inicio y detención de los servidores autónomos
Puede iniciar y detener los servidores de contenedor y de catálogo autónomos con los scripts **startOgServer** y **stopOgServer** o la API de servidor incorporado.

Referencia relacionada:

Script **startOgServer**
El script **startOgServer** inicia los servidores de contenedor y catálogo. Puede utilizar diversos parámetros al iniciar los servidores para habilitar el rastreo, especificar números de puerto, etc.

Punto de comprobación de la lección

En esta lección, ha aprendido lo siguiente:

- Cómo iniciar servidores de catálogo y servidores de contenedor
- Cómo ejecutar la aplicación cliente de ejemplo

Lección 4 de la guía de aprendizaje de iniciación: Supervisar el entorno

Puede utilizar el programa de utilidad **xscmd** y las herramientas de la consola web para supervisar el entorno de la cuadrícula de datos.

Tareas relacionadas:

Visualización de estadísticas con la consola web
Puede supervisar estadísticas y otra información de rendimiento con la consola web.

Supervisión con la consola web
Con la consola web, puede representar gráficos de las estadísticas actuales e históricas. Esta consola proporciona algunos gráficos configurados previamente para visiones generales de alto nivel y tiene una página de informes personalizados que puede utilizar para crear gráficos de las estadísticas disponibles. Puede utilizar las posibilidades de representación gráfica en la consola de supervisión de WebSphere eXtreme Scale para ver el rendimiento general de las cuadrículas de datos del entorno.

Inicio e inicio de sesión en la consola web
Inicie el servidor de la consola ejecutando el mandato **startConsoleServer** e iniciando sesión en el servidor con el ID de usuario y la contraseña predeterminados.

Conexión de la consola web a servidores de catálogo
Para empezar a visualizar estadísticas en la consola web, en primer lugar debe conectarse a los servidores de catálogo que desea supervisar. Se requieren pasos adicionales si los servidores de catálogo tienen la seguridad habilitada.

Supervisión con el programa de utilidad **xscmd**
El programa de utilidad **xscmd** sustituye al programa de utilidad **xsadmin** de ejemplo como una herramienta de administración y supervisión completamente soportada. Con el programa de utilidad **xscmd** puede visualizar información textual sobre la topología de WebSphere eXtreme Scale.

Administración con el programa de utilidad **xscmd**
Con **xscmd** puede completar tareas administrativas en el entorno como, por ejemplo: establecer enlaces de réplica multimaestro, sustituir quórum y detener grupos de servidores con el mandato teardown.

Referencia relacionada:

Estadísticas de la consola web
En función de la vista que utilice en la consola web, podrá visualizar distintas estadísticas sobre la configuración. Estas estadísticas incluyen la memoria utilizada, las cuadrículas de datos más utilizadas y el número de entradas de memoria caché.

Script stopOgServer
El script **stopOgServer** detiene los servidores de catálogo y contenedor.

Supervisión con la consola web

Con la consola web, puede representar gráficos de las estadísticas actuales e históricas. Esta consola proporciona algunos gráficos configurados previamente para visiones generales de alto nivel y tiene una página de informes personalizados que puede utilizar para crear gráficos de las estadísticas disponibles. Puede utilizar las posibilidades de representación gráfica en la consola de supervisión de WebSphere eXtreme Scale para ver el rendimiento general de las cuadrículas de datos del entorno.

Instalar la consola web como una característica opcional cuando se ejecuta el asistente de instalación.

1. Inicie el servidor de consola. El script **startConsoleServer.bat|sh** para iniciar el servidor de la consola se encuentra en el directorio *raíz_intal_wxs*/`ObjectGrid/bin` de la instalación.
2. Inicie la sesión en la consola.

a. Desde el navegador web, vaya a `https://su.host.consola:7443`, sustituyendo `su.host.consola` por el nombre de host del servidor en el que ha instalado la consola.

b. Inicie la sesión en la consola.

- **ID de usuario:** `admin`
- **Contraseña:** `admin`

Se visualiza la página de bienvenida de la consola.

3. Edite la configuración de la consola. Pulse **Valores** > **Configuración** para revisar la configuración de la consola. La configuración de la consola incluye información como:
   - Serie de rastreo del cliente WebSphere eXtreme Scale, como `*=all=disabled`
   - Nombre y contraseña del administrador
   - Dirección de correo electrónico del administrador
4. Establezca y mantenga las conexiones a los servidores de catálogo que desea supervisar. Repita los pasos siguientes para añadir cada servidor de catálogo a la configuración.
   a. Pulse **Valores** > **Servidores de catálogo eXtreme Scale**.
   b. Añada un servidor de catálogo nuevo.

   1) Pulse el icono de añadir ( ) para registrar un servidor de catálogo existente.
   2) Proporcione información, como el nombre de host y el puerto de escucha. Consulte Planificación de puertos de red para obtener más información sobre la configuración de puerto y los valores predeterminados.
   3) Pulse **Aceptar**.
   4) Verifique que el servidor de catálogo se ha añadido al árbol de navegación.
5. Consulte el estado de conexión. El campo **Dominio actual** indica el nombre del dominio de servicio de catálogo que se utiliza actualmente para visualizar información en la consola web. El estado de conexión se visualiza junto al nombre del dominio de servicio de catálogo.
6. Visualice estadísticas para las cuadrículas de datos y los servidores, o cree un informe personalizado.

Supervisión con el programa de utilidad xscmd

1. Abra una ventana de línea de mandatos.En la línea de mandatos, establezca las variables de entorno correspondientes.
   a. Establezca la variable de entorno *CLIENT_AUTH_LIB*:
      - Windows `set CLIENT_AUTH_LIB=<vía_acceso_a_JAR_o_clases_seguridad>`
      - UNIX `set CLIENT_AUTH_LIB=<vía_acceso_a_JAR_o_clases_seguridad>`
        `export CLIENT_AUTH_LIB`
2. Vaya al directorio *inicio_wxs*`/bin`.

   `cd inicio_wxs/bin`
3. Ejecute varios mandatos para visualizar información sobre el entorno.
   - Mostrar todos los servidores de contenedor en línea para la cuadrícula de datos Grid y el conjunto de correlaciones mapSet:

     `xscmd -c showPlacement -g Grid -ms mapSet`

- Visualizar la información de direccionamiento de la cuadrícula de datos.
  ```
  xscmd -c routetable -g Grid
  ```
- Visualizar el número de entradas de correlación en la cuadrícula de datos.
  ```
  xscmd -c showMapSizes -g Grid -ms mapSet
  ```

Detención de los servidores

Cuando ha terminado de utilizar la aplicación cliente y de supervisar el entorno de ejemplo de iniciación, puede detener los servidores.

- Si ha utilizado los archivos de script para iniciar los servidores, utilice <ctrl+c> para detener el proceso de servicio de catálogo y los servidores de contenedor en las ventanas respectivas.
- Si ha utilizado el mandato **startOgServer** para iniciar los servidores, utilice el mandato **stopOgServer** para detener los servidores.

  Detenga el servidor de contenedor:

  – UNIX Linux `stopOgServer.sh c0 -catalogServiceEndPoints localhost:2809`

  – Windows `stopOgServer.bat c0 -catalogServiceEndPoints localhost:2809`

  Detenga el servidor de catálogo:

  – UNIX Linux `stopOgServer.sh cs1 -catalogServiceEndPoints localhost:2809`

  – Windows `stopOgServer.bat cs1 -catalogServiceEndPoints localhost:2809`

Punto de comprobación de la lección

En esta lección, ha aprendido lo siguiente:

- Cómo iniciar la consola web y conectarla al servidor de catálogo
- Cómo supervisar las estadísticas del servidor y de la cuadrícula de datos
- Cómo detener los servidores

Iniciación al desarrollo de aplicaciones

Para empezar a desarrollar aplicaciones WebSphere eXtreme Scale, configure un entorno de desarrollo en Eclipse.

Acerca de esta tarea

Cuando esté desarrollando aplicaciones WebSphere eXtreme Scale, puede utilizar la API de servidor incorporado para crear e iniciar servidores, instancias de ObjectGrid, y para insertar datos en la cuadrícula de datos. Puede realizar la prueba unitaria de la aplicación y la configuración asociada directamente en el entorno de Eclipse.

Cuando esté listo para mover la aplicación a un entorno más amplio, puede crear archivos XML de configuración que importará para crear el despliegue.

Procedimiento

1. Configure un entorno de desarrollo en Eclipse.

Al añadir los archivos de archivado Java (JAR) de WebSphere eXtreme Scale al entorno de despliegue, puede empezar a utilizar las API para desarrollar las aplicaciones.

Más información: "Configuración de un entorno de despliegue autónomo" en la página 125

2. Cree una aplicación simple que inicie los servidores, cree una instancia de ObjectGrid e inserte datos en la cuadrícula de datos.
   a. Utilice la API ServerFactory para iniciar y detener los servidores.

      Más información: Utilización de la API de servidor incorporado para iniciar y detener servidores
   b. Utilice la API ObjectGridManager para recuperar la instancia de ObjectGrid que ha creado.

      Más información: "Interacción con un ObjectGrid utilizando la interfaz ObjectGridManager" en la página 136
   c. Utilice la API ObjectMap para insertar datos en la cuadrícula de datos.

      Más información: "Almacenamiento en memoria caché de objetos sin relaciones implicadas (API ObjectMap)" en la página 155La API ObjectMap es la forma más sencilla de acceder a datos y grabar datos en la cuadrícula de datos. Si los objetos tienen relaciones complejas, puede utilizar las API siguientes para leer, grabar y actualizar datos:

      - "Acceso a datos con índices (API Index)" en la página 144
      - "Almacenamiento en memoria caché de objetos y sus relaciones (API EntityManager)" en la página 166
      - "Recuperación de entidades y objetos (API de consulta)" en la página 204
      - "Acceso a los datos con el servicio de datos REST" en la página 271

      Para obtener más información sobre cómo elegir entre las distintas API, consulte Capítulo 5, "Desarrollo de aplicaciones", en la página 131.
3. Realice la prueba unitaria de la aplicación.

   También puede utilizar el programa de utilidad **xscmd** para visualizar información sobre los servidores en ejecución, las réplicas, etc. Si desea más información, consulte Administración con el programa de utilidad **xscmd**.
4. Cuando esté satisfecho con la aplicación dentro del entorno de desarrollo, cree archivos de configuración XML y actualice la aplicación para utilizar la configuración. La aplicación de ejemplo de iniciación proporciona ejemplos de estos archivos de configuración y una aplicación Java simple que utiliza los archivos de configuración.

   Más información: "Guía de aprendizaje: Cómo empezar con WebSphere eXtreme Scale" en la página 61
5. Ejecute la aplicación utilizando los archivos de configuración XML. La forma de iniciar los servidores depende del entorno que está utilizando.

   Puede ejecutar la aplicación en uno de los contenedores siguientes:

   - Máquina virtual Java (JVM) autónoma
   - Tomcat
   - WebSphere Application Server
   - OSGi

Conceptos relacionados:

"Almacenamiento en memoria caché de objetos sin relaciones implicadas (API ObjectMap)" en la página 155
ObjectMaps son como correlaciones Java que permiten a los datos almacenarse como pares clave-valor. Los ObjectMap proporcionan un acercamiento sencillo e intuitivo para el almacenamiento de los datos de la aplicación. Un ObjectMap es ideal para almacenar en memoria caché los objetos que no tienen relaciones. Si hubiera relaciones de objeto, debería utilizar la API EntityManager.

Información relacionada:

"Lección 2 de la guía de aprendizaje de iniciación: Creación de una aplicación cliente" en la página 63
Para insertar, suprimir, actualizar y recuperar datos de la cuadrícula de datos, debe escribir una aplicación de cliente. El ejemplo de iniciación incluye una aplicación cliente que puede utilizar para aprender a crear su propia aplicación cliente.

Capítulo 4. Planificación

Antes de instalar WebSphere eXtreme Scale y desplegar las aplicaciones de cuadrícula de datos, debe decidir sobre la topología de almacenamiento en memoria caché, completar la planificación de capacidad, revisar los requisitos de hardware y software, valores de red y ajuste, etc. ambién puede utilizar la lista de comprobación operacional para asegurarse de que el entorno está preparado para tener una aplicación desplegada.

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

Planificación de la topología

Con WebSphere eXtreme Scale, la arquitectura puede utilizar el almacenamiento en memoria caché de datos en memoria local o el almacenamiento en memoria caché de datos de cliente-servidor distribuido. La arquitectura puede tener distintas relaciones con las bases de datos. También puede configurar la topología para que abarque varios centros de datos.

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

Entornos en memoria

Cuando realiza un despliegue en un entorno local en memoria, WebSphere eXtreme Scale se ejecuta en una única Máquina virtual Java y no se replica. Para configurar un entorno local puede utilizar un archivo XML de ObjectGrid o las API de ObjectGrid.

Entornos distribuidos

Cuando realiza un despliegue en un entorno distribuido, WebSphere eXtreme Scale se ejecuta en un conjunto de Máquinas virtuales Java, aumentando el rendimiento, disponibilidad y escalabilidad. Con esta configuración, puede utilizar el particionamiento y la réplica de datos. También puede añadir servidores adicionales sin reiniciar los servidores eXtreme Scale existentes. Igual que en el entorno local, en el entorno distribuido se necesita un archivo XML ObjectGrid, o una configuración equivalente mediante programa. Debe también proporcionar un archivo XML de política de despliegue con detalles de configuración

Puede crear despliegues sencillos o grandes despliegues con terabytes en los que son necesarios miles de servidores.

Almacenamiento local de memoria caché en memoria

En el caso más sencillo, WebSphere eXtreme Scale se puede utilizar como una memoria caché de cuadrícula de datos en memoria local (no distribuida). El caso local beneficia especialmente a las aplicaciones de simultaneidad alta donde varias hebras necesitan acceder y modificar los datos transitorios. Los datos que se mantienen en una cuadrícula de datos local se pueden indexar y recuperar mediante consultas. Las consultas le ayudan a utilizar conjuntos de datos en memoria grandes. El soporte proporcionado con Máquina virtual Java (JVM), aunque está listo para su uso, tiene una estructura de datos limitada.

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

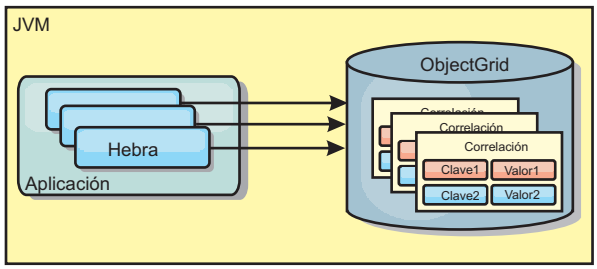


Figura 4. Escenario de memoria caché en memoria local

Ventajas

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

Desventajas

- No es tolerante a errores.
- Los datos no se replican. Las memorias caché en memoria son la mejor solución para los datos de referencia de sólo lectura.
- No es escalable. La cantidad de memoria necesaria para la base de datos podría desbordar la máquina virtual Java.
- Se producen problemas al añadir máquinas virtuales Java:
  - Los datos no se pueden particionar fácilmente.
  - Se debe replicar manualmente el estado entre las máquinas virtuales Java o cada instancia podría tener distintas versiones de los mismos datos.
  - La operación de invalidación es muy costosa.

– Cada memoria caché se debe calentar de forma independientemente. El calentamiento es el periodo de carga de un conjunto de datos, de forma que la memoria caché se rellena con datos válidos.

Cuándo se debe utilizar

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

Memoria caché local replicada de igual

Debe asegurarse de que la memoria caché esté sincronizada si existen varios procesos con instancias de memoria caché independientes. Para asegurarse de que las instancias de memoria caché están sincronizadas, habilite una memoria caché replicada por un igual con JMS (Java Message Service).

WebSphere eXtreme Scale incluye dos plug-ins que propagan automáticamente los cambios de las transacciones entre instancias de ObjectGrid de un igual. El plug-in JMSObjectGridEventListener propaga automáticamente los cambios de eXtreme Scale mediante JMS.

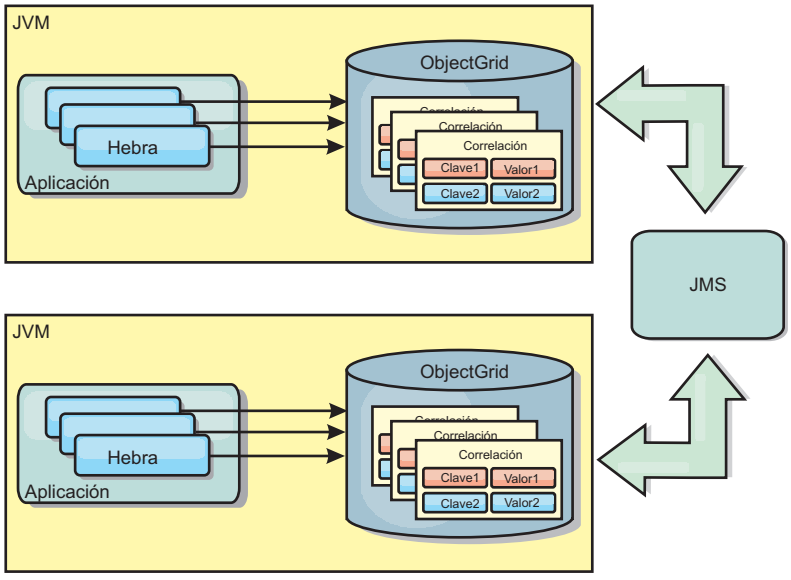


Figura 5. La memoria caché duplicada por un igual con los cambios que se propagan con JMS

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

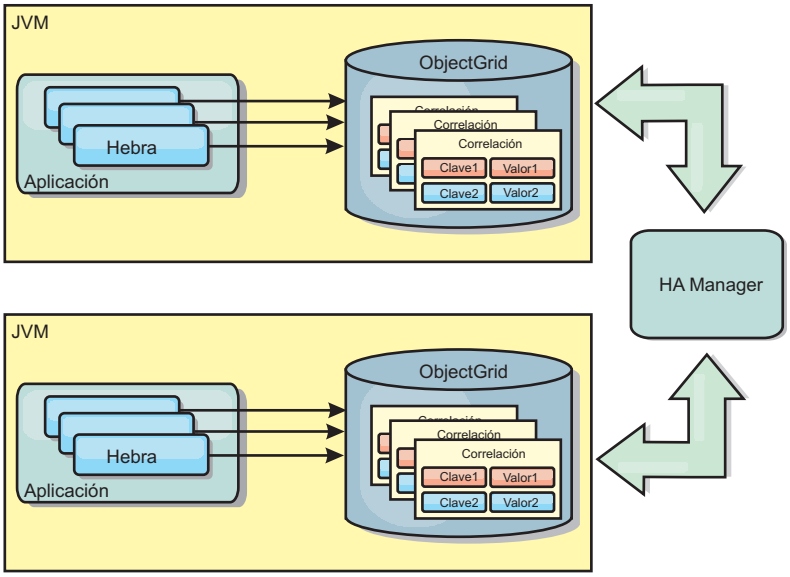


Figura 6. La memoria caché duplicada por un igual con los cambios propagados con el High Availability Manager.

Ventajas

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

Desventajas

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

Cuándo se debe utilizar

Utilice topología de despliegue solo cuando la cantidad de datos que se deben almacenar en memoria caché sea pequeña, pueda caber en una única JVM y sea relativamente estable.

Memoria caché incorporada

Las cuadrículas de WebSphere eXtreme Scale pueden ejecutarse en procesos existentes como servidores eXtreme Scale incorporados o bien puede gestionarlas como procesos externos.

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

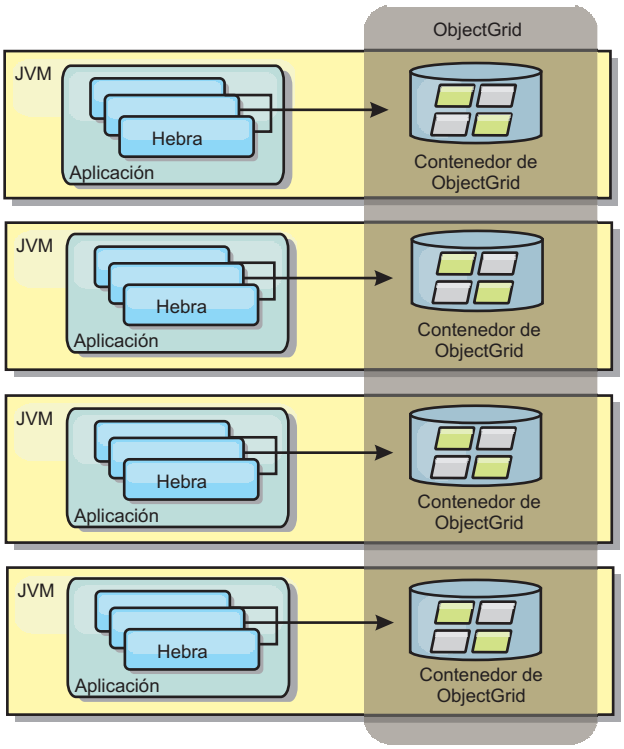


Figura 7. Memoria caché incorporada

Ventajas

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

Desventajas

- Aumenta el uso de la memoria en procesos de cliente ya que todos los datos se colocan en el proceso.
- Aumenta el uso de la CPU para dar servicio a las solicitudes de los clientes.

- Es más difícil manejar las actualizaciones de las aplicaciones ya que los clientes utilizan los mismos archivos JAR (Java Archive) de aplicación que los servidores.
- Menos flexible. Escalar clientes y servidores de cuadrícula no puede aumentar a la misma velocidad. Si los servidores se definen externamente, puede tener más flexibilidad al gestionar el número de procesos.

Cuándo se debe utilizar

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

Para obtener más información, consulte el tema sobre la habilitación del mecanismo de invalidación de cliente en la*Guía de administración*.

Memoria caché distribuida

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

Coherencia de la memoria caché

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

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

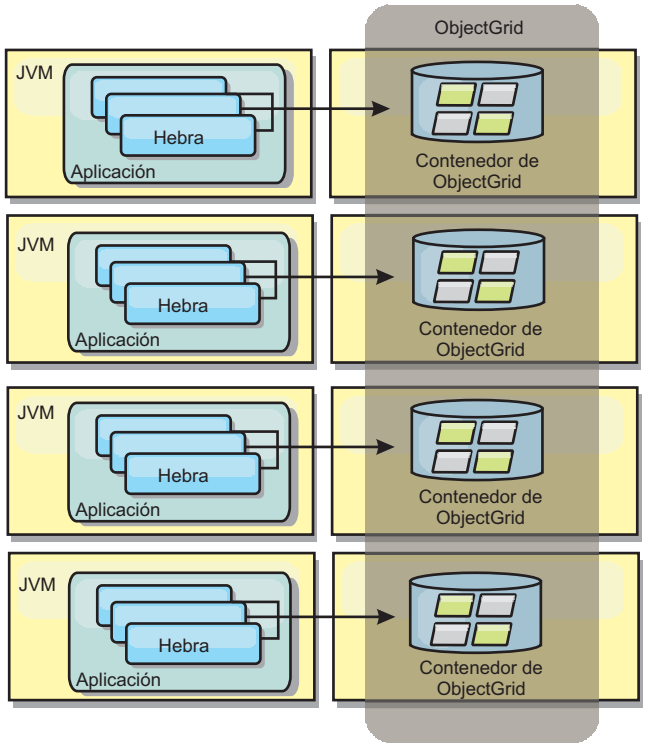


Figura 8. Memoria caché distribuida

Memoria caché cercana

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

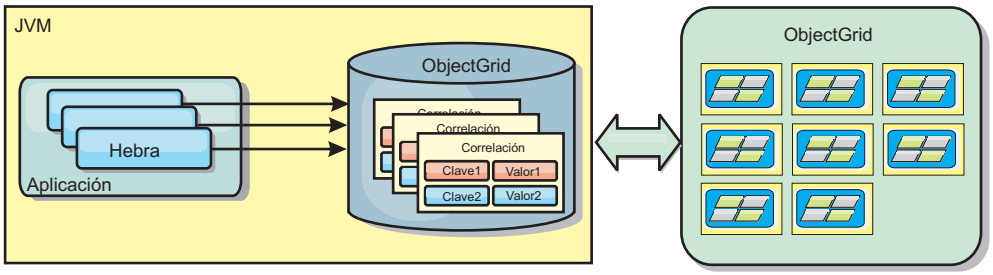


Figura 9. Memoria caché cercana

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

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

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

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

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

Ventaja

- Un tiempo de respuesta rápido porque todos los accesos a los datos son locales. Buscando los datos en la memoria caché cercana primero se guarda un recorrido a la cuadrícula de los servidores, por lo hace que incluso los datos remotos se puedan acceder de forma local.

Desventajas

- Aumenta la duración de los datos obsoletos debido a que la memoria caché cercana en cada nivel puede no estar sincronizada con los datos actuales de la cuadrícula de datos.
- Se basa en un desalojador para invalidar los datos a fin de evitar quedarse sin memoria.

Cuándo se debe utilizar

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

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

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

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

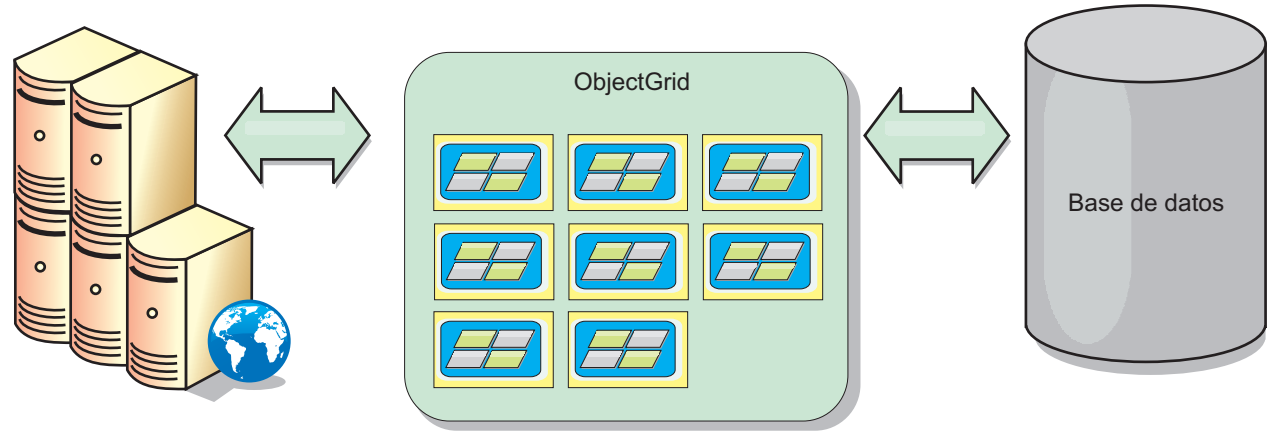


Figura 10. ObjectGrid como un almacenamiento intermedio de base de datos

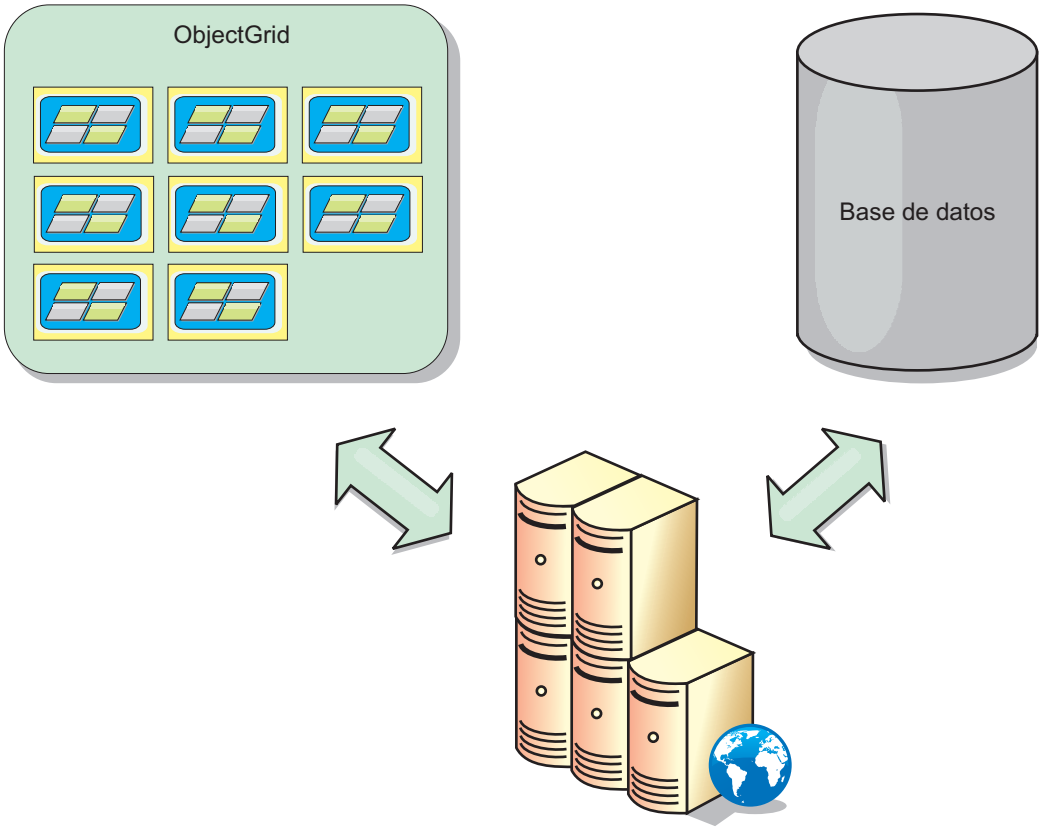


Figura 11. ObjectGrid como una memoria caché secundaria

Memoria caché escasa y completa

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

memoria caché escasa

Cuando una clave no está presente en una memoria caché escasa, o los datos no están disponibles y se produce una falta de coincidencia de memoria caché, se invoca el siguiente nivel. Los datos se captan, desde una base de datos, por ejemplo, y se insertan en el nivel de la memoria caché de cuadrícula de datos. Si utiliza una consulta o un índice, sólo se accede a los valores cargados actualmente y las solicitudes no se remiten a los demás niveles.

Memoria caché completa

Una memoria caché completa contiene todos los datos necesarios y se puede acceder a la misma utilizando atributos que no son de clave con índices o consultas. Una memoria caché completa se precarga con datos de la base de atos antes de que la aplicación intente acceder a los datos. Una memoria caché completa puede funcionar como una sustitución de base de datos después de que se carguen los datos. Puesto que están disponibles todos los datos, las consultas y los índices se pueden utilizar para encontrar y agregar datos.

Memoria caché complementaria

Cuando se utiliza WebSphere eXtreme Scale como memoria caché complementaria, se utiliza el programa de fondo con la cuadrícula de datos.

Memoria caché complementaria

Puede configurar el producto como una memoria caché complementaria para la capa de acceso a datos de una aplicación. En este escenario, WebSphere eXtreme Scale se utiliza para almacenar temporalmente objetos que normalmente se recuperarían de una base de datos de programa de fondo. Las aplicaciones comprueban si la cuadrícula de datos contiene los datos. Si los datos están en la cuadrícula de datos, los datos se devuelven al emisor. Si los datos no existen, los datos se recuperan de la base de datos de fondo. A continuación, los datos se insertan en la cuadrícula de datos de forma que la siguiente solicitud pueda utilizar la copia almacenada en memoria caché. El diagrama siguiente muestra cómo se puede utilizar WebSphere eXtreme Scale como una memoria caché complementaria con una capa de acceso a datos arbitrarios como por ejemplo OpenJPA o Hibernate.

Plug-ins de memoria caché para Hibernate y OpenJPA

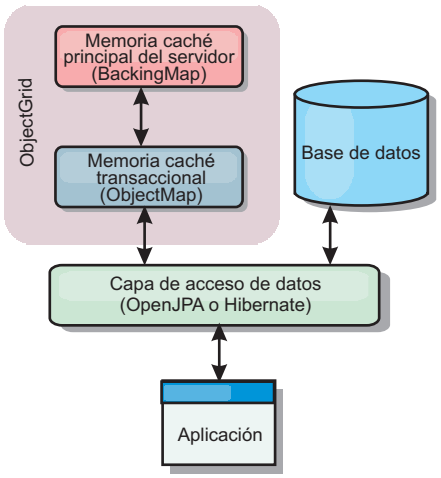


Figura 12. Memoria caché complementaria

Los plug-ins de memoria caché para OpenJPA e Hibernate se incluyen en WebSphere eXtreme Scale, de forma que puede utilizar el producto como una memoria caché complementaria automática. El uso de WebSphere eXtreme Scale como un proveedor de memoria caché aumenta el rendimiento cuando se leen y consultan datos y reduce la carga de la base de datos. WebSphere eXtreme Scale presenta algunas ventajas sobre las implementaciones de memoria caché incorporada ya que la memoria caché se replica automáticamente entre procesos. Cuando un cliente almacena en memoria caché un valor, todos los demás clientes pueden utilizar el valor almacenado en la memoria.

Memoria caché en línea

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

Memoria caché en línea

Cuando se utiliza como una memoria caché en línea, WebSphere eXtreme Scale interactúa con el programa de fondo utilizando un plug-in Loader. Este escenario puede simplificar el acceso a datos porque las aplicaciones pueden acceder a las API eXtreme Scale directamente. Se da soporte a distintos escenarios de almacenamiento en memoria caché en eXtreme Scale para garantizar que los datos de la memoria caché y los datos del programa de fondo estarán sincronizados. El diagrama siguiente ilustra cómo una memoria caché en línea interactúa con la aplicación y el programa de fondo.

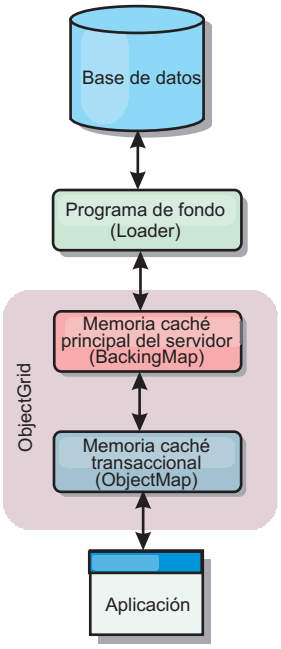


Figura 13. Memoria caché en línea

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

- Lectura directa
- Grabación directa
- Grabación diferida

Caso de ejemplo de almacenamiento en memoria caché de lectura directa

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

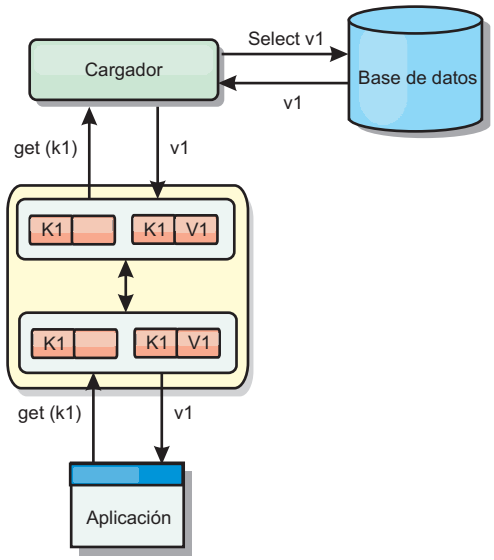


Figura 14. Almacenamiento en memoria caché de lectura directa

Caso de ejemplo de almacenamiento en memoria caché de grabación directa

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

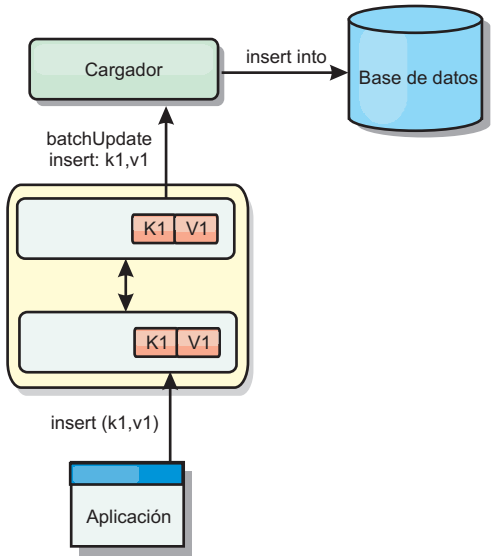


Figura 15. Almacenamiento en memoria caché de grabación directa

Caso de ejemplo de almacenamiento en memoria caché de grabación anticipada

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

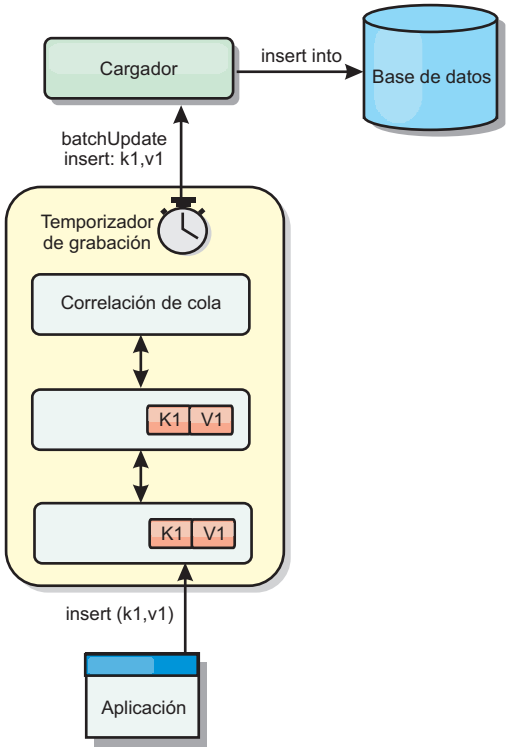


Figura 16. Almacenamiento en memoria caché de grabación diferida

Almacenamiento en memoria caché de grabación diferida

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

Visión general del almacenamiento en memoria caché con grabación diferida

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

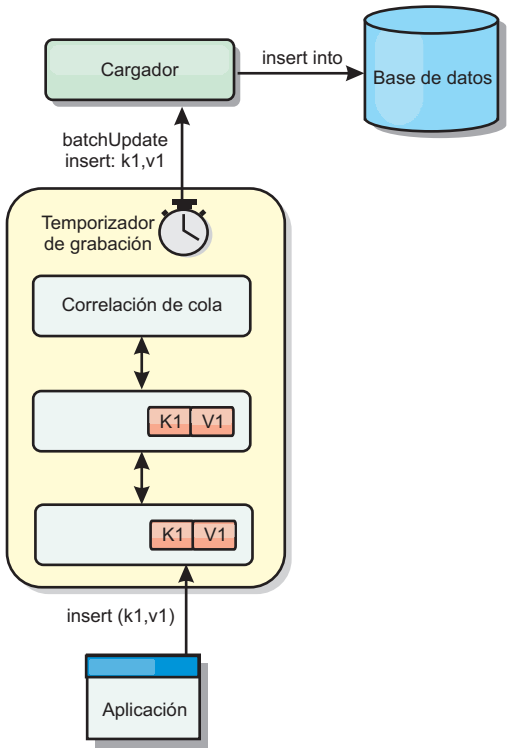


Figura 17. Almacenamiento en memoria caché de grabación diferida

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

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

El soporte de grabación diferida *es* una ampliación del plug-in Loader, que puede utilizar para integrar eXtreme Scale con la base de datos. Por ejemplo, consulte la información del apartado Configuración de cargadores JPA sobre cómo configurar un cargador JPA.

Ventajas

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

- **Aislamiento de anomalía de programa de fondo:** el almacenamiento de grabación diferida proporciona una capa de aislamiento de las anomalías de programa de fondo. Cuando la base de datos de programa de fondo falla, las actualizaciones se ponen en cola en la correlación de cola. Las aplicaciones

pueden continuar con las transacciones a eXtreme Scale. Cuando se recupera el programa de fondo, los datos de la correlación de cola se envían al programa de fondo.

- **Carga reducida de programa de fondo** el cargador de grabación diferida fusiona las actualizaciones según una clave, de forma que sólo existe una actualización fusionada por clave en la correlación de cola. Este procedimiento reduce el número de actualizaciones en la base de datos de programa de fondo.
- **Rendimiento mejorado de transacciones:** los tiempos individuales de las transacciones de eXtreme Scale se reducen porque la transacción no necesita esperar a que los datos se sincronicen con el programa de fondo.

Consideraciones sobre el diseño de aplicaciones

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

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

Restricciones de la integridad referencial

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

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

Comportamiento de bloqueo de correlaciones de cola

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

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

Debido a este comportamiento de correlación de colas adicional, puede ver algunas diferencias en el comportamiento del bloqueo.

- Si la correlación de usuarios está configurada como estrategia de bloqueo pesimista (PESSIMISTIC), no hay mucha diferencia de comportamiento en el bloqueo. Cada vez que se llama a una operación de desecho o confirmación, se coloca un bloqueo S en la misma clave de la misma correlación de colas. Durante la confirmación, no sólo se adquiere un bloqueo X para la clave en la correlación de usuarios, sino que además de adquiere para la clave en la correlación de colas.
- Si la correlación de usuarios está configurada como estrategia de bloqueo optimista (OPTIMISTIC) o ninguna (NONE), la transacción de usuario seguirá el patrón de estrategia de bloqueo pesimista (PESSIMISTIC). Cada vez que se llama a una operación de desecho o confirmación, se adquiere un bloqueo S en la misma clave de la misma correlación de colas. Durante la confirmación se adquiere un bloqueo X para la clave en la correlación de colas utilizando la misma transacción.

Reintentos de transacción de cargador

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

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

Anomalías del cargador

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

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

- Si el cargador de grabación diferida obtiene una excepción LoaderException, el cargador de grabación diferida considerará la anomalía como un error de los datos, como por ejemplo un error de clave duplicada. El cargador de grabación diferida anulará el proceso por lotes de la actualización, e intentará actualizar un registro cada vez para aislar la anomalía de los datos. Si se vuelve a obtener una excepción {{LoaderException}} durante la actualización de un registro, se crea un registro de actualización con errores y se anota en la correlación de actualizaciones con errores.
- Si el cargador de grabación diferida obtiene una excepción LoaderNotAvailableException, el cargador de grabación diferida la considerará como un error porque no puede conectase a la base de datos, por ejemplo, el programa de fondo de la base de datos está inactivo, una conexión de base de datos no está disponible, o la red no está activa. El cargador de grabación diferida esperará 15 segundos y después volverá a intentar realizar la actualización por lotes en la base de datos.

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

Consideraciones sobre el rendimiento

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

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

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

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

Referencia relacionada:

"Ejemplo: Escribir una clase de volcador de grabación diferida" en la página 366
Este código fuente de ejemplo muestra cómo escribir un observador (volcador) para manejar actualizaciones de grabación diferida anómalas.

Cargadores

Con un plug-in Loader plug-in, una correlación de cuadrícula de datos puede actuar como una memoria caché de datos para los datos que se mantienen normalmente en un almacén persistente en el mismo sistema o en otro sistema. Generalmente, se utiliza una base de datos o un sistema de archivos como almacenamiento persistente. Una máquina virtual Java (JVM) remota también se

puede utilizar como el origen de datos, lo que permite crear memorias caché basadas en hub utilizando eXtreme Scale. Un cargador tiene la lógica para leer y escribir datos en un almacén persistente.

Visión general

Los cargadores son plug-ins de correlaciones de respaldo que se invocan cuando se realizan cambios en la correlación de respaldo o ésta no puede satisfacer una solicitud de datos (una falta de memoria caché). Se invoca el cargador cuando la memoria caché no puede satisfacer la solicitud de una clave, proporcionando la capacidad de lectura a través y el relleno poco activo de la memoria caché. Un cargador también permite actualizar la base de datos cuando los valores de la memoria caché cambian. Todos los cambios de una transacción se agrupan para minimizar el número de interacciones de la base de datos. Se utiliza un plug-in TransactionCallback junto con el cargador para desencadenar la demarcación de la transacción de fondo. Utilizar este plug-in es importante cuando se incluyen varias correlaciones en una única transacción, o cuando se desechan los datos de una transacción en la memoria caché sin confirmar.

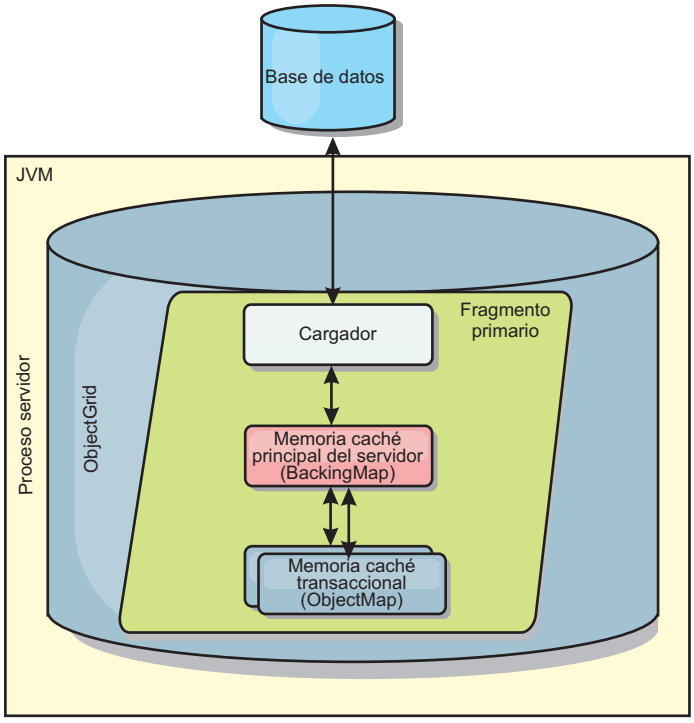


Figura 18. Cargador

El cargador también puede utilizar las actualizaciones sobrecualificadas para evitar mantener los bloqueos de base de datos. Al almacenar un atributo de versión en el valor de memoria caché, el cargador puede ver la imagen antes y después del valor tal como se actualiza en la memoria caché. Este valor se puede utilizar cuando se actualiza la base de datos o cuando se realiza un programa de fondo para verificar que los datos no se han actualizado. Un cargador también se puede configurar para precargar la cuadrícula de datos cuando se inicia. Cuando se realizan particiones, se asocia una instancia de cargador con cada partición. Si la correlación "Company" tiene diez particiones, hay diez instancias de cargador, una por partición primaria. Cuando se activa el fragmento primario de la correlación,

se invoca el método preloadMap para el cargador de forma síncrona o asíncrona, que permite cargar automáticamente la partición de la correlación con los datos procedentes del programa de fondo. Cuando se invocan de forma síncrona, todas las transacciones de cliente se bloquean, lo que impide el acceso incoherente a la cuadrícula de datos. De forma alternativa, se puede utilizar un precargador de cliente para cargar toda la cuadrícula de datos.

Dos cargadores incorporados pueden simplificar en gran medida la integración con los programas de fondo de la base de datos relacional. Los cargadores JPA utilizan las funciones de correlación de objetos relacionales (ORM) de ambas implementaciones, OpenJPA e Hibernate, de la especificación de JPA (Java Persistence API). Si desea más información, consulte "Cargadores JPA" en la página 400.

Si utiliza cargadores en una configuración de varios centros de datos, debe considerar cómo se mantiene la coherencia de los datos y la memoria caché entre las cuadrículas de datos. Para obtener más información, consulte "Consideraciones sobre el cargador en una topología multimaestro" en la página 106.

Configuración de cargador

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

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

Referencia relacionada:

"Consideraciones de programación del cargador JPA" en la página 369
Un cargador Java Persistence API (JPA) es una implementación de plug-in de cargador que utiliza JPA para interactuar con la base de datos. Utilice las siguientes consideraciones cuando desarrolle una aplicación que utiliza un cargador JPA.

Precarga de datos y calentamiento

En muchos escenarios que incorporan el uso de un cargador, puede preparar la cuadrícula de datos precargándola con datos.

Cuando se utiliza como una memoria caché completa, la cuadrícula de datos debe alojar todos los datos y se debe cargar antes de que los clientes se puedan conectar a ella. Cuando se utiliza una memoria caché escasa, puede preparar la memoria caché con datos de forma que los clientes tengan acceso inmediato a los datos cuando estos se conecten.

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

Plug-in Loader

El plug-in Loader está asociado con cada correlación y es responsable de sincronizar un fragmento de partición primaria con la base de datos. El método preloadMap del plug-in Loader se invoca automáticamente cuando se activa un fragmento. Por ejemplo, si tiene 100 particiones, existen 100 instancias de cargador,

y cada una carga los datos para su partición. Si se ejecuta de forma síncrona, todos los clientes se bloquean hasta que se complete la precarga.

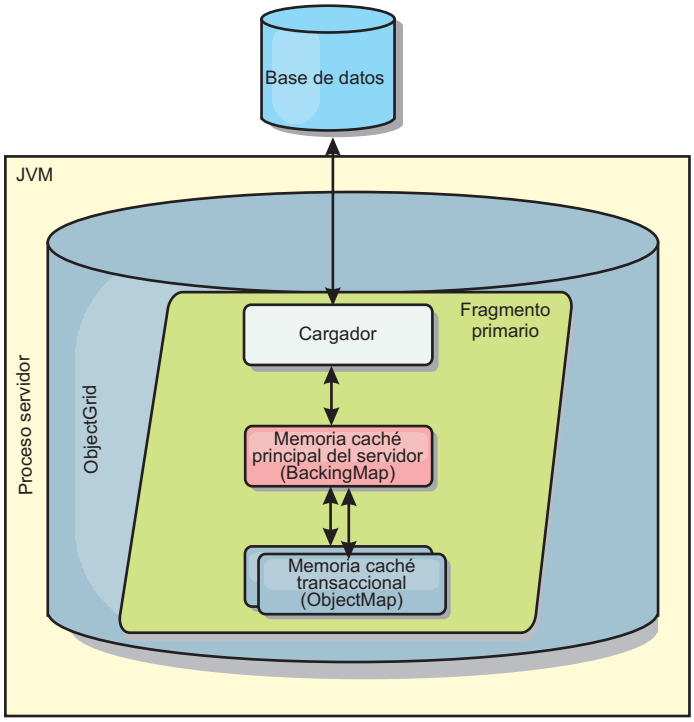


Figura 19. Plug-in Loader

Cargador de clientes

Un cargador de clientes es un patrón para utilizar uno o más clientes para carga la cuadrícula con datos. El uso de varios clientes para cargar los datos de cuadrícula puede ser eficaz cuando el esquema de partición no se almacena en la base de datos. Puede invocar los cargadores de clientes manual o automáticamente cuando se inicia la cuadrícula de datos. De forma opcional, los cargadores de clientes pueden utilizar StateManager para establecer el estado de la cuadrícula de datos en la modalidad de precarga, de forma que los clientes no pueden acceder a la cuadrícula mientras está precargando los datos. WebSphere eXtreme Scale incluye un cargador basado en JPA (Java Persistence API) que puede utilizar para cargar automáticamente la cuadrícula de datos con los proveedores OpenJPA o Hibernate JPA. Para obtener más información sobre los proveedores de memoria caché, consulte Plug-in de memoria caché de nivel 2 (L2) JPA.

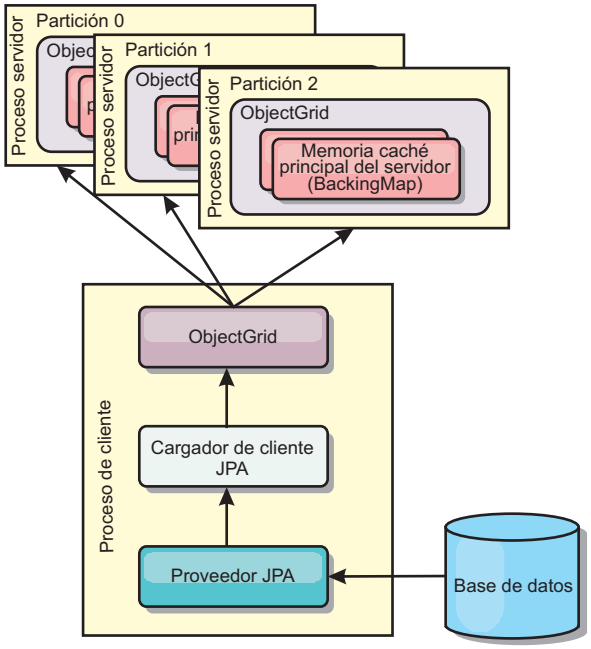


Figura 20. Cargador de clientes

Técnicas de sincronización de base de datos

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

Técnicas de sincronización de base de datos

Renovación periódica

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

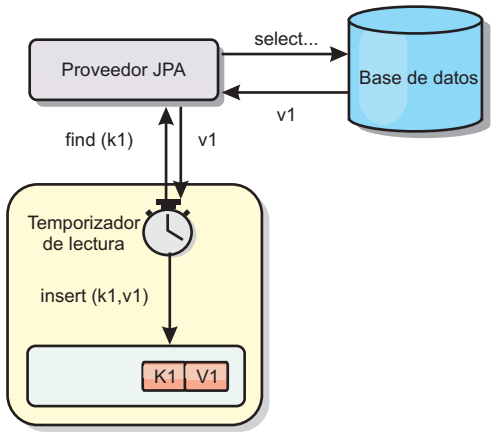


Figura 21. Renovación periódica

Desalojo

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

Anulación basada en sucesos

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

Anulación programática

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

Invalidación de datos

Para eliminar los datos de memoria caché de escala, puede utilizar un mecanismo de invalidación basado en suceso o mediante programa.

Invalidación basada en sucesos

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

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

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

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

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

Anulación programática

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

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

Índices

Utilice el plug-in MapIndexPlugin para crear un índice o varios índice en una BackingMap para dar soporte al acceso a datos no de clave.

Tipos de índices y configuración

La característica de indexación la representa el plug-in MapIndexPlugin, o Index de forma abreviada. Index es un plug-in BackingMap. Una BackingMap puede tener varios plug-ins Index configurados, siempre que cada uno de ellos siga las normas de configuración de Index.

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

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

Con el índice dinámico, puede crear un índice en una correlación BackingMap antes o después de la inicialización de la instancia de ObjectGrid que contiene. Las aplicaciones tienen un control del ciclo de vida sobre el proceso de indexación dinámica, de forma que pueda eliminar un índice dinámico, cuando ya no sea necesario. Cuando una aplicación crea un índice dinámico, éste podría no estar listo para su uso inmediato debido al tiempo que tarda en completarse el proceso de creación del índice. Puesto que la cantidad de tiempo depende de la cantidad de datos indexados, se proporciona la interfaz DynamicIndexCallback para aplicaciones que desean recibir notificaciones cuando se produzcan determinados sucesos de indexación. Estos sucesos pueden incluir sucesos de error, destrucción y preparado. Las aplicaciones pueden implementar esta interfaz de devolución de llamada y registrarla con el proceso de índices dinámicos.

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

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

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

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

Índice predeterminado

Si desea iterar a través de las claves en una correlación local, puede utilizar el índice predeterminado. Este índice no requiere ninguna configuración, pero se debe utilizar en el fragmento, utilizando una instancia de ObjectGrid o agente recuperada del método ShardEvents.shardActivated(ObjectGrid shard).

Consideraciones sobre la calidad de los datos

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

Consideraciones sobre el rendimiento de los índices

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

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

- **El número de índices creados sobre la misma BackingMap:** cada índice consume recursos del sistema. A medida que el número de índices creados sobre la BackingMap aumenta, disminuye el rendimiento.

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

Tareas relacionadas:

"Configuración del plug-in HashIndex" en la página 330
Puede configurar el HashIndex incorporado, la clase com.ibm.websphere.objectgrid.plugins.index.HashIndex, con un archivo XML, programáticamente, o con una anotación de entidad en una correlación de entidad.

"Acceso a datos con índices (API Index)" en la página 144
Utilice la indexación para acceder más eficazmente a los datos.

Referencia relacionada:

"Atributos del plug-in HashIndex" en la página 333
Puede utilizar los atributos siguientes para configurar el plug-in HashIndex. Estos atributos definen propiedades como por ejemplo si utiliza un HashIndex compuesto o de atributos, o si la indexación de rango está habilitada.

Planificación de topologías de varios centros de datos

Mediante la utilización de la réplica asíncrona multimaestro, dos o más cuadrículas de datos pueden convertirse en copias exactas entre ellas. Cada cuadrícula de datos está alojada en un dominio de servicio de catálogo independiente, con su propio servicio de catálogo, servidores de contenedor y un nombre exclusivo. Con la réplica asíncrona multimaestro, puede utilizar enlaces para conectar una colección de dominios de servicio de catálogo. A continuación, los dominios de servicio de catálogo se sincronizan utilizando la réplica mediante los enlaces. Puede construir casi cada topología mediante la definición de enlaces entre los dominios de servicio de catálogo.

Tareas relacionadas:

Configuración de topologías de varios centros de datos
Con la réplica asíncrona multimaestro, enlaza un conjunto de dominios de servicio de catálogo. A continuación, los dominios de servicio de catálogo conectados se sincronizan mediante réplica a través de los enlaces. Puede definir los enlaces utilizando archivos de propiedades, en tiempo de ejecución con programas JMX (Java Management Extensions) o con programas de utilidad de línea de mandatos. El conjunto de enlaces actuales de un dominio se almacena en el servicio de catálogo. Puede añadir y eliminar enlaces sin reiniciar el dominio de servicio de catálogo que aloja la cuadrícula de datos.

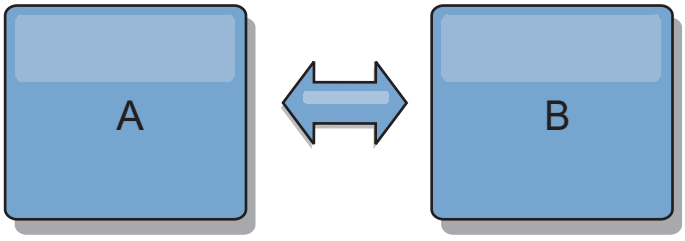
"Desarrollo de árbitros personalizados para la réplica con varios maestros" en la página 306
Se podrían producir colisiones de cambio si se pueden cambiar en dos lugares a la vez los mismos registros. En una topología de réplica multimaestro, los dominios de servicio de catálogo detectan automáticamente las colisiones. Cuando el dominio de servicio de catálogo detecta una colisión, invoca un árbitro. Normalmente, las colisiones se resuelven con el árbitro de colisión predeterminado. No obstante, una aplicación puede proporcionar un árbitro de colisión personalizado.

Topologías para réplica multimaestro

Tiene diversas opciones cuando elige una topología para el despliegue que incorpora réplica multimaestro.

Enlaces que conectan dominios de servicio de catálogo

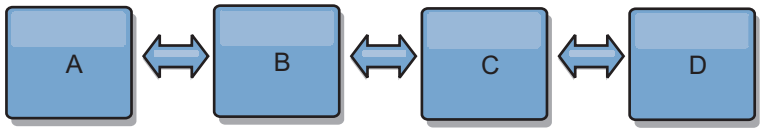
Una infraestructura de cuadrícula de datos de réplica es un gráfico conectado de dominios de servicio de catálogo con enlaces bidireccionales entre ellos. Con un enlace, dos dominios de servicio de catálogo pueden comunicar los cambios en los datos. Por ejemplo, la topología más sencilla es un par de dominios de servicio de catálogo con un único enlace entre ellos. Los dominios de servicio de catálogo se nombran alfabéticamente: A, B, C y así sucesivamente, desde la izquierda. Un enlace puede cruzar una red de área amplia (WAN), abarcando distancias grandes. Incluso si se interrumpe el enlace, aún puede cambiar los datos en cualquiera de los dos dominios de servicio de catálogo. La topología reconcilia los cambios cuando el enlace reconecta los dominios de servicio de catálogo. Los enlaces intentan volverse a conectar automáticamente si se interrumpe la conexión de red.



Después de haber configurado los enlaces, eXtreme Scale en primer lugar intenta hacer que cada dominio de servicio de catálogo sea idéntico. A continuación, eXtreme Scale intenta mantener las condiciones idénticas a los cambios producidos en cualquier dominio de servicio de catálogo. El objetivo es que cada dominio de servicio de catálogo sea un reflejo exacto de cada uno de los otros dominios de servicio de catálogo conectados mediante los enlaces. Los enlaces de réplica entre los dominios de servicio de catálogo ayudan a garantizar que los cambios realizados en un dominio se copian en los otros dominios.

Topologías de línea

Aunque es un despliegue muy simple, una topología de línea muestra algunas cualidades de los enlaces. En primer lugar, no es necesario que un dominio de servicio de catálogo esté conectado directamente a cada uno de los otros dominios de servicio de catálogo para que reciba los cambios. El Dominio B obtiene los cambios del Dominio A. El Dominio C recibe los cambios del Dominio A a través del Dominio B, que se conecta a los Dominios A y C. De forma similar, el Dominio D recibe los cambios de los otros dominios mediante el Dominio C. Esta capacidad esparce la carga de distribuir los cambios lejos del origen de los cambios.



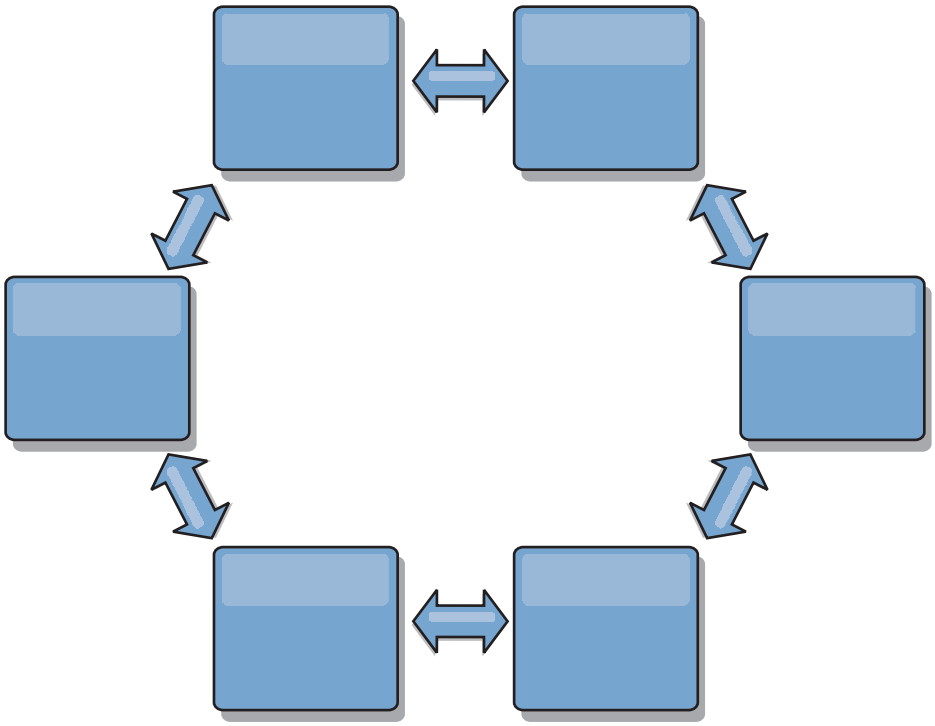
Tenga en cuenta que si el Dominio C falla, se producirán las acciones siguientes:

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

En última instancia, los Dominios A, B, C y D serán todos ellos idénticos entre ellos de nuevo.

Topologías de anillo

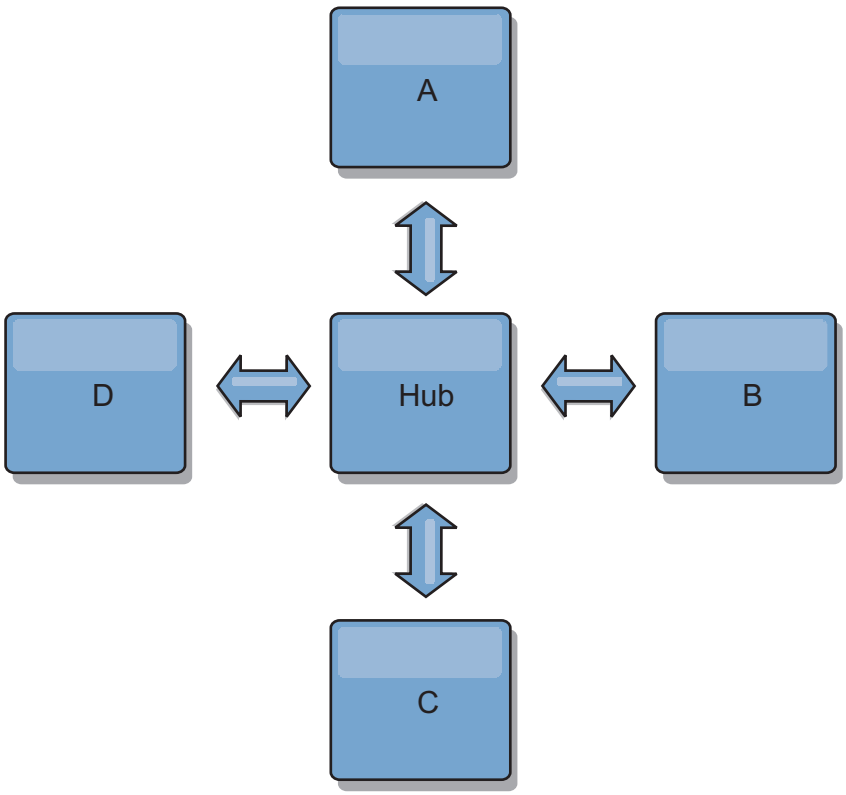
Las topologías de anillo son un ejemplo de una topología más flexible. Cuando un dominio de servicio de catálogo o un único enlace falla, los dominios de servicio de catálogo supervivientes todavía pueden obtener los cambios. Los dominios de servicio de catálogo viajan alrededor del anillo, alejándose de la anomalía. Cada dominio de servicio de catálogo tiene como máximo dos enlaces a otros dominios de servicio de catálogo, independientemente del tamaño de la topología de anillo. La latencia para propagar los cambios puede ser grande. Podría ser necesario que los cambios de un dominio de servicio de catálogo determinado viajaran a través de varios enlaces antes de que todos los dominios de servicio de catálogo tengan los cambios. Una topología de línea tiene la misma característica.



También puede desplegar una topología de anillo más sofisticada, con un dominio de servicio de catálogo raíz en el centro del anillo. El dominio de servicio de catálogo raíz funciona como el punto central de reconciliación. Los otros dominios de servicio de catálogo actúan como puntos remotos de reconciliación para los cambios que se producen en el dominio de servicio de catálogo raíz. El dominio de servicio de catálogo raíz puede arbitrar los cambios entre los dominios de servicio de catálogo. Si una topología de anillo contiene más de un anillo alrededor de un dominio de servicio de catálogo raíz, el dominio sólo puede arbitrar los cambios entre el anillo más interno. Sin embargo, los resultados del arbitraje se distribuyen por los dominios de servicio de catálogo de los otros anillos.

Topologías de hub y radio

Con una topología de hub y radio, los cambios viajan a través de un dominio de servicio de catálogo de hub. Debido a que el hub es el único dominio de servicio de catálogo intermedio especificado, las topologías de hub y radio tienen una latencia menor. El dominio de hub está conectado a cada dominio de radio mediante un enlace. El hub distribuye los cambios entre los dominios de servicio de catálogo. El hub actúa como un punto de reconciliación para las colisiones. En un entorno con una tasa de actualización alta, es posible que el hub necesite ejecutarse en más hardware que los radios para permanecer sincronizado. WebSphere eXtreme Scale está diseñado para escalar de forma lineal, lo que significa que puede ampliar el hub, según sea necesario, sin dificultad. Sin embargo, si el hub falla, los cambios no se distribuyen hasta que se reinicia el hub. Los cambios en los dominios de servicio de catálogo de radio se distribuirán una vez que se vuelva a conectar el hub.



También puede utilizar una estrategia con clientes completamente replicados, una variación de la topología que utiliza un par de servidores eXtreme Scale en ejecución como hub. Cada cliente crea una cuadrícula de datos de un solo contenedor autocontenida con un catálogo en la JVM de cliente. Un cliente utiliza su cuadrícula de datos para conectarse al catálogo de hub. Esta conexión hace que el cliente se sincronice con el hub tan pronto como el cliente obtenga una conexión del hub.

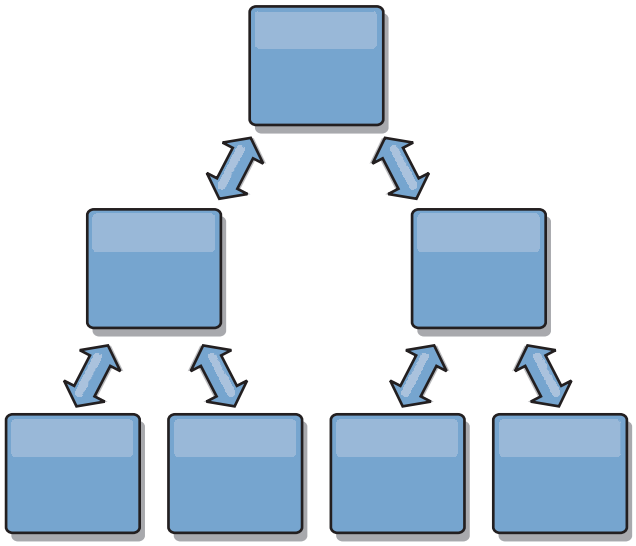
Los cambios realizados por el cliente son locales al cliente y se hace una réplica de ellos asíncrona en el hub. El hub actúa como un dominio de arbitraje, que distribuye los cambios a todos los clientes conectados. La topología de clientes completamente replicados proporciona una memoria caché L2 fiable para una correlacionador relacional de objetos como, por ejemplo, OpenJPA. Los cambios se distribuyen rápidamente entre las JVM de cliente a través del hub. Si el tamaño de memoria caché puede estar contenido en el espacio de almacenamiento intermedio disponible, la topología es una arquitectura fiable para este estilo de L2.

Utilice varias particiones para escalar el dominio del hub en varias JVM, si es necesario. Debido a que todos los datos aún deben caber en una única JVM de cliente, varias plataformas aumentan la capacidad del concentrador para distribuir y arbitrar cambios. Sin embargo, tener varias particiones no cambia la capacidad de un único dominio.

Topologías de árbol

También puede utilizar un árbol dirigido acíclico. Un árbol acíclico no tiene ciclos ni bucles, y una configuración dirigida limita los enlaces a los existentes solo entre padres e hijos. Esta configuración puede ser útil para topologías que tengan muchos dominios de servicio de catálogo y no es práctico tener un hub central que

esté conectado a todos los radios posibles. Este tipo de topología también puede resultar útil cuando debe añadir dominios de servicio de catálogo actualizando el dominio de servicio de catálogo raíz.



Una topología de árbol aún puede tener un punto central de reconciliación en el dominio de servicio de catálogo raíz. El segundo nivel aún puede funcionar como un punto remoto de reconciliación para los cambios que se producen en el dominio de servicio de catálogo por debajo de ellos. El dominio de servicio de catálogo raíz puede arbitrar los cambios entre los dominios de servicio de catálogo en el segundo nivel solamente. También puede utilizar árboles "n-ários", cada uno de los cuales tiene N hijos en cada nivel. Cada dominio de servicio de catálogo se conecta a *n* enlaces.

Clientes totalmente replicados

En esta variación de topología interviene un par de servidores eXtreme Scale que se ejecutan como un hub. Cada cliente crea una cuadrícula de datos de un solo contenedor autocontenida con un catálogo en la JVM de cliente. Un cliente utiliza su cuadrícula de datos para conectarse al catálogo de hub, lo que hace que el cliente se sincronice con el hub tan pronto como el cliente obtiene una conexión del hub.

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

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

Tareas relacionadas:

Configuración de topologías de varios centros de datos
Con la réplica asíncrona multimaestro, enlaza un conjunto de dominios de servicio de catálogo. A continuación, los dominios de servicio de catálogo conectados se sincronizan mediante réplica a través de los enlaces. Puede definir los enlaces utilizando archivos de propiedades, en tiempo de ejecución con programas JMX (Java Management Extensions) o con programas de utilidad de línea de mandatos. El conjunto de enlaces actuales de un dominio se almacena en el servicio de catálogo. Puede añadir y eliminar enlaces sin reiniciar el dominio de servicio de catálogo que aloja la cuadrícula de datos.

"Desarrollo de árbitros personalizados para la réplica con varios maestros" en la página 306
Se podrían producir colisiones de cambio si se pueden cambiar en dos lugares a la vez los mismos registros. En una topología de réplica multimaestro, los dominios de servicio de catálogo detectan automáticamente las colisiones. Cuando el dominio de servicio de catálogo detecta una colisión, invoca un árbitro. Normalmente, las colisiones se resuelven con el árbitro de colisión predeterminado. No obstante, una aplicación puede proporcionar un árbitro de colisión personalizado.

Consideraciones sobre la configuración para topologías multimaestro

Considere los puntos siguientes cuando decida si desea utilizar topologías de réplica multimaestro y cómo utilizarlas.

- **Requisitos de conjunto de correlaciones**

  Los conjuntos de correlaciones deben tener las características siguientes para replicar cambios en todos los enlaces del dominio de servicio de catálogo:

  – El nombre de ObjectGrid y el nombre de conjunto de correlaciones de un dominio de servicio de catálogo deben coincidir con el nombre de ObjectGrid y el nombre de conjunto de correlaciones de otros dominios de servicio de catálogo. Por ejemplo, el ObjectGrid "og1" y el conjunto de correlaciones "ms1" deben estar configurados en el dominio de servicio de catálogo A y el dominio de servicio de catálogo B para replicar los datos del conjunto de correlaciones entre los dominios de servicio de catálogo.

  – Es una cuadrícula de datos `FIXED_PARTITION`. Las cuadrículas de datos `PER_CONTAINER` no se pueden replicar.

  – Tiene el mismo número de particiones en cada dominio de servicio de catálogo. El conjunto de correlaciones podría tener o no el mismo número y los mismos tipos de réplicas.

  – Tiene los mismos tipos de datos que se están replicando en cada dominio de servicio de catálogo.

  – Contiene las mismas correlaciones y plantillas de correlación dinámica en cada uno de los dominios de servicio de catálogo.

  – No utiliza el gestor de entidades. Un conjunto de correlaciones que contiene una correlación de entidades no se replica en todos los dominios de servicio de catálogo.

  – No utiliza el soporte de almacenamiento en memoria caché de grabación diferida. Un conjunto de correlaciones que contiene una correlación que está configurada con soporte de grabación diferida no se replica en todos los dominios de servicio de catálogo.

  Los conjuntos de correlaciones con las características anteriores empiezan la réplica una vez que se han iniciado los dominios de servicio de catálogo en la topología.

- **Cargadores de clases con varios dominios de servicio de catálogo**

  Los dominios de servicio de catálogo deben tener acceso a todas las clases utilizadas como claves y valores. Todas las dependencias se deben reflejar en todas las vías de acceso de clases para máquinas virtuales Java (JVM) de contenedor de cuadrícula de datos para todos los dominios. Si un plug-in CollisionArbiter recupera el valor para una entrada de memoria caché, las clases para los valores deben estar presentes para el dominio que inicia el árbitro.

Tareas relacionadas:

Configuración de topologías de varios centros de datos
Con la réplica asíncrona multimaestro, enlaza un conjunto de dominios de servicio de catálogo. A continuación, los dominios de servicio de catálogo conectados se sincronizan mediante réplica a través de los enlaces. Puede definir los enlaces utilizando archivos de propiedades, en tiempo de ejecución con programas JMX (Java Management Extensions) o con programas de utilidad de línea de mandatos. El conjunto de enlaces actuales de un dominio se almacena en el servicio de catálogo. Puede añadir y eliminar enlaces sin reiniciar el dominio de servicio de catálogo que aloja la cuadrícula de datos.

"Desarrollo de árbitros personalizados para la réplica con varios maestros" en la página 306
Se podrían producir colisiones de cambio si se pueden cambiar en dos lugares a la vez los mismos registros. En una topología de réplica multimaestro, los dominios de servicio de catálogo detectan automáticamente las colisiones. Cuando el dominio de servicio de catálogo detecta una colisión, invoca un árbitro. Normalmente, las colisiones se resuelven con el árbitro de colisión predeterminado. No obstante, una aplicación puede proporcionar un árbitro de colisión personalizado.

Consideraciones sobre el cargador en una topología multimaestro

Cuando se utilizan cargadores en una topología multimaestro, debe considerar los posibles retos de mantenimiento de la información de revisión y colisión. La cuadrícula de atos mantiene información de revisión sobre los elementos de la cuadrícula de datos de forma que se pueden detectar las colisiones cuando otros fragmentos primarios de la configuración graban entradas en la cuadrícula de datos. Cuando se añaden entradas desde un cargador, esta información de revisión no se incluye y la entrada asume una revisión nueva. Debido a que la revisión de la entrada parece una inserción nueva, se produciría una falta colisión si otro fragmento primario también cambia este estado u obtiene la misma información de un cargador.

Los cambios de la réplica invocan el método get en el cargador con una lista de las claves que no están aún en la cuadrícula de datos pero que se han a cambiar durante la transacción de réplica. Cuando se produce la réplica, estas entradas son entradas de colisión. Cuando se arbitran las colisiones y se aplica la revisión, se llama a una actualización por lotes en el cargador para aplicar los cambios en la base de datos. Todas las correlaciones modificadas en la ventana de revisión se actualizan en la misma transacción.

Interrogante de la precarga

Considere una topología de dos centros de datos con el centro de datos A y el centro de datos B. Ambos centros de datos tienen bases de datos independientes, pero solo el centro de datos A tiene una cuadrícula de datos en ejecución. Al establecer un enlace entre los cnetros de datos para una configuración multimaestro, las cuadrículas de datos del centro de datos A inician el envío de

datos a las nuevas cuadrículas de datos del centro de datos B, lo que causa una colisión con cada entrada. Otro problema importante que se produce es con los datos que se encuentra en la base de datos del centro de datos B pero no en la base de atos del centro de datos A. Estas filas no se llenan ni arbitran, lo que genera incoherencias que no se resuelven.

Solución al interrogante de la precarga

Debido a que los datos que se encuentran solo en la base de datos no puede tener revisiones, debe precargar siempre completamente la cuadrícula de datos desde la base de datos local antes de establecer un enlace multimaestro. A continuación, ambas cuadrículas de datos pueden revisar y arbitrar los datos, y finalmente llegar a un estado coherente.

Interrogante de la memoria caché escasa

Con una memoria caché escasa, en primer lugar la aplicación intenta encontrar datos en la cuadrícula de datos. Si los datos no se encuentran en la cuadrícula de datos, se buscan los dato s en la base de atos utilizando el cargador. Se desalojan periodicamente entradas de la cuadrícula de datos para mantener un tamaño pequeño de la memoria caché.

Este tipo de memoria caché puede ser problemático en un escenario de configuración multimaestro porque las entradas de la cuadrícula de datos tienen metadatos de revisión que le ayudarán a detectar qué colisiones se producen y qué lado ha realizado cambios. Cuando los enlaces entre los centros de datos no funcionan, un centro de datos puede actualizar una entrada y a continuación en última instancia actualizar la base de datos e invalidar la entrada en la cuadrícula de datos. Cuando se recupera el enlace, los centros de datos intentan sincronizar las revisiones entre ellos. Sin embargo, debido a que la base de datos se ha actualizado y la entrada de la cuadrícula de datos se ha invalidado, el cambio se pierde desde la perspectiva del centro de datos que se ha interrumpido. Como resultado, los dos lados de la cuadrícula de datos están desincronizados y no son coherentes.

Solución a los interrogantes de memoria caché escasa

Topología y hub y radio:

Puede ejecutar el cargador solo en el hub de una topología de hub y radio, lo que mantiene la coherencia de los datos al mismo tiempo que se escala la cuadrícula de datos. Sin embargo, si está considerando el despliegue, tenga en cuenta que los cargadores pueden permitir que la cuadrícula de datos se cargue parcialmente, lo que significa que se ha configurado el desalojador. Si los radios de la configuración son memorias caché escasas pero no tienen cargadores, las faltas de coincidencia de memoria caché no tienen ninguna manera de recuperar los datos de la base de datos. Debido a esta restricción, debe utilizar una topología de memoria caché llenada completamente con una configuración de hub y radio.

Invalidaciones y desalojo

La invalidación crea coherencia entre la cuadrícula de datos y la base de datos. Los datos se pueden eliminar de la cuadrícula de datos mediante programación o con desalojo. Al desarrollar la aplicación, debe tener en cuenta que el manejo de revisiones no replica los cambios que se han invalidado, lo que genera incoherencias entre fragmentos primarios.

Los sucesos de invalidación no son cambios de estado de memoria caché y no generan réplica. Los desalojadores configurados se ejecutan independientemente de otros desalojadores de la configuración. Por ejemplo, podría tener un desalojador configurado para un umbral de memoria en un dominio de servicio de catálogo, pero un tipo distinto de desalojador menos agresivo en el servicio de catálogo enlazado. Cuando se eliminan entradas de cuadrícula de datos debido a la política de umbral de memoria, las entradas del otro dominio de servicio de catálogo no resultan afectadas.

Actualizaciones de base de datos e invalidación de cuadrícula de datos

Se producen problemas al actualizar la base de datos directamente en segundo plano al llamar a la invalidación en la cuadrícula de datos para las entradas actualizadas en una configuración multimaestro. Este problema se produce porque la cuadrícula de datos no puede replicar el cambio en otros fragmentos primarios hasta que algún tipo de acceso de memoria caché mueve la entrada a la cuadrícula de datos.

Varios grabadores en una única base de datos lógica

Cuando utiliza una única base de datos con varios fragmentos primarios que se conectan mediante un cargador, se producen conflictos de transacciones. La implementación de cargador debe manejar especialmente estos tipos de escenarios.

Duplicación de datos utilizando réplica multimaestro

Puede configurar bases de datos independientes conectadas a dominios de servicio de catálogo independiente. En esta configuración, el cargador puede enviar cambios de un centro de datos al otro centro de datos.

Tareas relacionadas:

Configuración de topologías de varios centros de datos
Con la réplica asíncrona multimaestro, enlaza un conjunto de dominios de servicio de catálogo. A continuación, los dominios de servicio de catálogo conectados se sincronizan mediante réplica a través de los enlaces. Puede definir los enlaces utilizando archivos de propiedades, en tiempo de ejecución con programas JMX (Java Management Extensions) o con programas de utilidad de línea de mandatos. El conjunto de enlaces actuales de un dominio se almacena en el servicio de catálogo. Puede añadir y eliminar enlaces sin reiniciar el dominio de servicio de catálogo que aloja la cuadrícula de datos.

"Desarrollo de árbitros personalizados para la réplica con varios maestros" en la página 306
Se podrían producir colisiones de cambio si se pueden cambiar en dos lugares a la vez los mismos registros. En una topología de réplica multimaestro, los dominios de servicio de catálogo detectan automáticamente las colisiones. Cuando el dominio de servicio de catálogo detecta una colisión, invoca un árbitro. Normalmente, las colisiones se resuelven con el árbitro de colisión predeterminado. No obstante, una aplicación puede proporcionar un árbitro de colisión personalizado.

Consideraciones sobre el diseño para la réplica multimaestro

Al implementar la réplica multimaestro, debe tener en cuenta aspectos del diseño como los siguientes: arbitraje, enlace y rendimiento.

Consideraciones sobre arbitraje en el diseño de topología

Se podrían producir colisiones de cambio si se pueden cambiar en dos lugares a la vez los mismos registros. Configure cada uno de los dominios de servicio de catálogo para que tenga aproximadamente la misma cantidad de recursos de procesador, memoria y red. Podría observar que los dominios de servicio de catálogo que realicen el manejo de colisiones de cambio (arbitraje) utilicen más recursos que otros dominios de servicio de catálogo. Las colisiones se detectan automáticamente. Se manejan con uno de dos mecanismos:

- **Árbitro de colisión predeterminado**: el protocolo predeterminado utilizará los cambios del dominio de servicio de catálogo con el nombre léxicamente inferior. Por ejemplo, si los dominios de servicio de catálogo A y B generan un conflicto para un registro, el cambio del dominio de servicio de catálogo B se ignorará. El dominio de servicio de catálogo A mantiene su versión y el registro en el dominio de servicio de catálogo B se cambia para que coincida con el registro del dominio de servicio de catálogo A. Este comportamiento se aplica también a las aplicaciones en las que los usuarios o sesiones normalmente se enlazan o tienen una afinidad con una de las cuadrículas siguientes.
- **Árbitro de colisiones personalizado**: las aplicaciones pueden proporcionar un árbitro personalizado. Cuando un dominio de servicio de catálogo detecta una colisión, se inicia un árbitro. Para obtener información sobre cómo desarrollar un árbitro personalizado útil, consulte "Desarrollo de árbitros personalizados para la réplica con varios maestros" en la página 306.

Para topologías en las que las colisiones son posibles, considere implementar una topología de hub y radio o una topología de árbol. Estas dos topologías son propicias para evitar colisiones constantes, lo que puede suceder en los escenarios siguientes:

1. Varios dominios de servicio de catálogo sufren una colisión
2. Cada dominio de servicio de catálogo maneja la colisión localmente, lo que genera revisiones
3. Las revisiones colisionan, con lo que se producen revisiones de revisiones

Para evitar colisiones, elija un dominio de servicio de catálogo específico, denominado un *dominio de servicio de catálogo de arbitraje* como el árbitro de colisión para un subconjunto de dominios de servicio de catálogo. Por ejemplo, una topología de hub y radio podría utiliza el hub como el manejador de colisiones. El manejador de colisiones de radio ignora las colisiones detectadas por los dominios de servicio de catálogo de radio. El dominio de servicio de catálogo de hub crea revisiones, lo que evita revisiones de colisiones inesperadas. El dominio de servicio de catálogo que se asigna para manejar colisiones debe enlazar a todos los dominios para los que es responsable para manejar colisiones. En una topología de árbol, los dominios padre internos manejan colisiones para sus hijos inmediatos. Por el contrario, si utiliza una topología en anillo, no puede designar un dominio de servicio de catálogo en el anillo como el árbitro.

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

Tabla 1. Enfoques de arbitraje. En esta tabla se indica si el arbitraje de la aplicación es compatible con distintas tecnologías.

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

Consideraciones sobre enlaces en el diseño de topología

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

- **Latencia de cambios**

  La latencia de cambios la determina el número de dominios de servicio de catálogo intermedio por los que debe pasar un cambio antes de llegar a un dominio de servicio de catálogo específico.

  Una topología tiene la mejor latencia de cambios cuando elimina dominios de servicio de catálogo intermedios enlazando cada dominio de servicio de catálogo a cada uno de los otros dominios de servicio de catálogo. Sin embargo, un dominio de servicio de catálogo debe realizar trabajo de réplica en proporción a su número de enlaces. Para topologías grandes, el gran número de enlaces que se definirán puede causar carga administrativa.

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

  – Procesador y ancho de banda de red en el dominio de servicio de catálogo de origen
  – Número de dominios de servicio de catálogo intermedios y enlaces entre los dominios de servicio de catálogo de origen y de destino
  – Recursos de procesador y de red disponibles a los dominios de servicio de catálogo de origen, de destino e intermedio

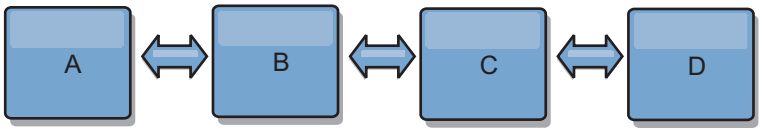
- **Tolerancia al error**

  La tolerancia a errores la determina el número de vías de acceso existentes entre los dos dominios de servicio de catálogo para la réplica de cambios.

  Si solo tiene un enlace entre un par determinado de dominios de servicio de catálogo, una anomalía de enlace no permite la propagación de cambios. De

forma similar, los cambios no se propagan entre los dominios de servicio de catálogo si alguno de los dominios intermedios experimenta anomalía de enlace. La topología podría tener un único enlace desde un dominio de servicio de catálogo a otro de tal forma que el enlace pase por dominios intermedios. Si es así, los cambios no se propagarán si alguno de los dominios de servicio de catálogo intermedios está inactivo.

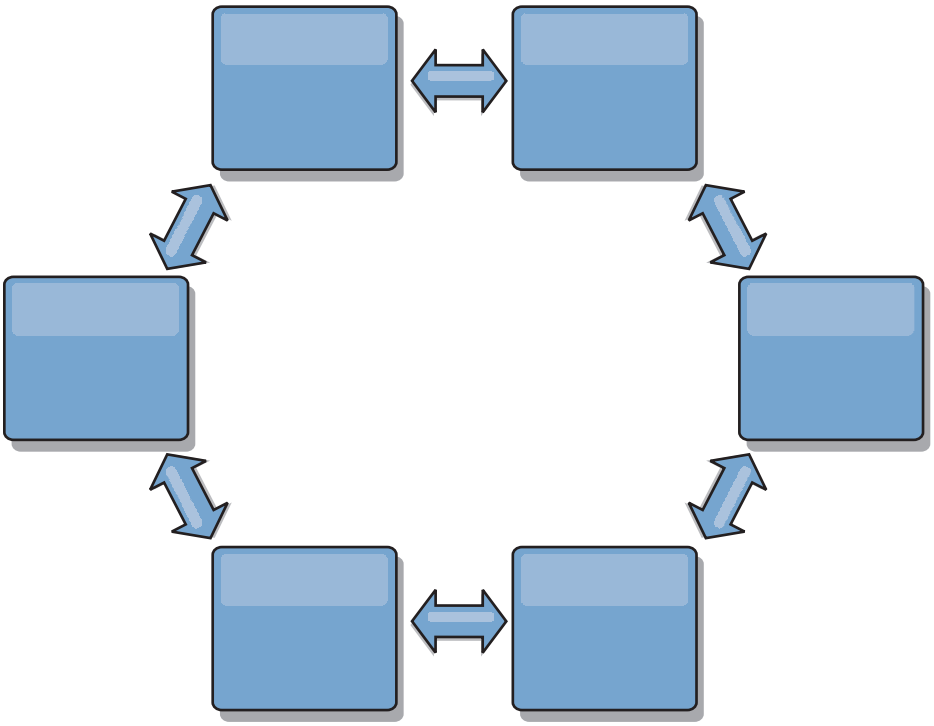
Considere la topología de línea con cuatro dominios de servicio de catálogo A, B, C, y D:



Si se mantiene alguna de estas condiciones, el Dominio D no verá los cambios de A:

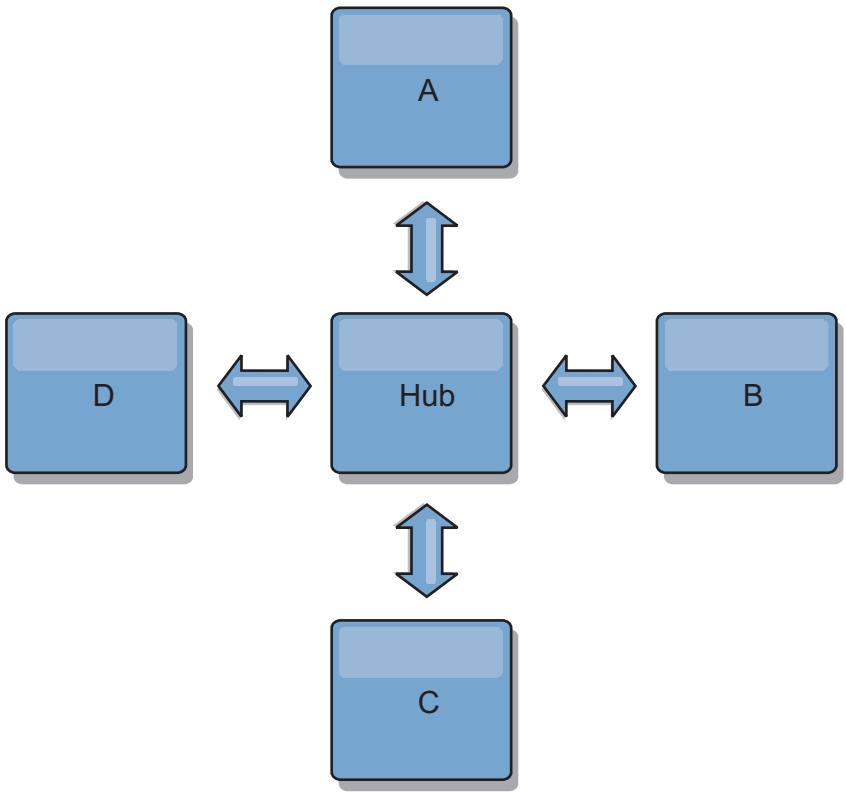
- El dominio A está activo y el dominio B está inactivo
- Los dominios A y B están activos y el dominio C está inactivo
- El enlace entre A y B está inactivo
- El enlace entre B y C está inactivo
- El enlace entre C y D está inactivo

En cambio, con una topología de anillo, cada uno de los dominios de servicio de catálogo puede recibir cambios desde cualquier dirección.



Por ejemplo, si un servicio de catálogo determinado de la topología de anillo está inactivo, los dos dominios adyacentes aún pueden obtener cambios directamente uno del otro.

Todos los cambios se propagan mediante el hub. Por lo tanto, a diferencia de las topologías de línea y de anillo, el diseño de hub y radio puede desglosarse, si el hub falla.



Un único dominio de servicio de topología es resistente a una determinada cantidad de pérdida de servicio. Sin embargo, anomalías mayores como interrupciones de la red amplia o la pérdida de enlaces entre centros de datos físicos puede interrumpir cualquiera de los dominios de servicio de catálogo.

- **Enlace y rendimiento**

  El número de enlaces definidos en un dominio de servicio de catálogo afecta al rendimiento. Más enlaces utilizan más recursos y como resultado el rendimiento de la réplica puede disminuir. La posibilidad de recuperar cambios para un dominio A mediante otros dominios libera de forma efectiva al dominio A de tener que replicar las transacciones en todas partes. La carga de distribución de cambios de un dominio está limitada por el número de enlaces que utiliza, no por cuántos dominios haya en la topología. Esta propiedad de carga proporciona escalabilidad, de forma que los dominios de la topología pueden compartir la carga de la distribución de cambios.

  Un dominio de servicio de catálogo puede recuperar los cambios indirectamente mediante otros dominios de servicio de catálogo. Considere una topología de línea con cinco dominios de servicio de catálogo.

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

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

  La carga de distribución de los dominios de servicio de catálogo A y E es la menor, ya que cada uno de ellos tiene un enlace a un único dominio de servicio de catálogo. Cada uno de los dominios B, C y D tiene un enlace a dos dominios. Por lo tanto, la carga de distribución de los dominios B, C y D es el doble de la

carga de los dominios A y E. La carga de trabajo depende del número de enlaces de cada dominio, no del número global de dominios de la topología. Por lo tanto, la distribución de cargas descrita permanecería constante, incluso si la línea contuviera 1000 dominios.

Consideraciones sobre el rendimiento de réplica multimaestros

Tenga en cuenta las limitaciones siguientes cuando utilice topologías de réplica multimaestro:

- **Cambiar ajuste de distribución**, se trata en la sección anterior.
- **Rendimiento de enlace de réplica** WebSphere eXtreme Scale crea un único socket TCP/IP entre cualquier par de JVM. Todos el tráfico entre las JVM se produce entre el único socket, incluido tráfico de la réplica multimaestro. Los dominios de servicio de catálogo se alojan en como mínimo *n* JVM de contenedor, lo que proporciona como mínimo *n* enlaces TCP a dominios de servicio de catálogo de igual. Por lo tanto, los dominios de servicio de catálogo con una gran cantidad de contenedores tienen niveles más altos de rendimiento de la réplica. Más contenedores requieren más recursos de procesador y red.
- **Ajuste de la ventana deslizante TCP y RFC 1323** El soporte de RFC 1323 en ambos extremos de un enlace proporciona más datos para un viaje de ida y vuelta. Este soporte produce un mejor rendimiento, ampliando la capacidad de la ventana en un factor de aproximadamente 16.000.

  Recuerde que los sockets TCP utilizan un mecanismo de ventana deslizando para controlar el flujo de datos masivo. Este mecanismo normalmente limita el socket a 64 KB para un intervalo de viaje de ida y vuelta. Si el intervalo de viaje de ida y vuelta es 100 ms, el ancho de banda se limita a 640 KB/segundo sin ajuste adicional. El uso de todo el ancho de banda disponible en un enlace podría requerir un ajuste específico de un sistema operativo. La mayoría de sistemas operativos incluyen parámetros de ajuste, incluidas las opciones de RFC 1323, para ampliar el rendimiento sobre los enlaces de latencia alta.

  Varios factores pueden afectar al rendimiento de la réplica:

  – La velocidad a la que eXtreme Scale recupera cambios.
  – La velocidad a la que eXtreme Scale puede dar servicio a solicitudes de recuperación de réplica.
  – La capacidad de la ventana deslizante.
  – Con el ajuste de almacenamiento intermedio de red en ambos lados de un enlace, eXtreme Scale recupera cambios sobre el socket de forma eficiente.
- **Serialización de objetos** Todos los datos deben ser serializables. Si un dominio de servicio de catálogo no utiliza `COPY_TO_BYTES`, el dominio de servicio de catálogo debe utilizar Java o ObjectTransformers para optimizar el rendimiento de serialización.
- **Compresión** WebSphere eXtreme Scale comprime todos los datos enviados entre dominios de servicio de catálogo de forma predeterminada. La inhabilitación de la compresión no está disponible actualmente.
- **Ajuste de la memoria** El uso de memoria para una topología de réplica multimaestro es considerablemente independiente del número de dominios de servicio de catálogo de la topología.

  La réplica multimaestro añade una cantidad fija de proceso por entrada Map para manejar el mantenimiento de versiones. Cada contenedor también realiza un seguimiento de una cantidad fija de datos para cada dominio de servicio de catálogo de la topología. Una topología con dos dominios de servicio de catálogo utiliza aproximadamente la misma memoria que una topología con 50 dominios de servicio de catálogo. WebSphere eXtreme Scale no utiliza registros

de reproducción o colas similares en su implementación. Por lo tanto, no hay ninguna estructura de recuperación lista en el caso de que un enlace de réplica no esté disponible durante el periodo de tiempo considerable y se reinicie posteriormente.

Tareas relacionadas:

Configuración de topologías de varios centros de datos
Con la réplica asíncrona multimaestro, enlaza un conjunto de dominios de servicio de catálogo. A continuación, los dominios de servicio de catálogo conectados se sincronizan mediante réplica a través de los enlaces. Puede definir los enlaces utilizando archivos de propiedades, en tiempo de ejecución con programas JMX (Java Management Extensions) o con programas de utilidad de línea de mandatos. El conjunto de enlaces actuales de un dominio se almacena en el servicio de catálogo. Puede añadir y eliminar enlaces sin reiniciar el dominio de servicio de catálogo que aloja la cuadrícula de datos.

"Desarrollo de árbitros personalizados para la réplica con varios maestros" en la página 306
Se podrían producir colisiones de cambio si se pueden cambiar en dos lugares a la vez los mismos registros. En una topología de réplica multimaestro, los dominios de servicio de catálogo detectan automáticamente las colisiones. Cuando el dominio de servicio de catálogo detecta una colisión, invoca un árbitro. Normalmente, las colisiones se resuelven con el árbitro de colisión predeterminado. No obstante, una aplicación puede proporcionar un árbitro de colisión personalizado.

Planificación para desarrollar aplicaciones WebSphere eXtreme Scale

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

Visión general de la API

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

API de WebSphere eXtreme Scale

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

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

se pueden configurar con las API de configuración. Los plug-ins pueden estar incorporados, pero en algunas situaciones es posible que tenga que desarrollar sus propios plug-ins.

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

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

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

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

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

Visión general de los plug-ins

Un plug-in de WebSphere eXtreme Scale es un componente que proporciona un determinado tipo de función a los componentes conectables que incluyen ObjectGrid y BackingMap. WebSphere eXtreme Scale proporciona varios puntos de conexión para permitir a las aplicaciones y a los proveedores de memoria caché integrarse con distintos almacenes de datos, las API de cliente alternativas y para mejorar el rendimiento general de la memoria caché. El producto se entrega con varios plug-ins predeterminados y preincorporados, pero también puede crear plug-ins personalizados con la aplicación.

Todos los plug-ins son clases concretas que implementan una o más interfaces de plug-in de eXtreme Scale. ObjectGrid crea instancias de estas clases y las invoca cuando conviene. ObjectGrid y BackingMaps permiten que se registren plug-ins personalizados.

Plug-ins ObjectGrid

Están disponibles los plug-ins siguientes para una instancia de ObjectGrid. Si el plug-in es solo del lado del servidor, los plug-ins se eliminan en las instancias de ObjectGrid y BackingMap del cliente. Las instancias de ObjectGrid y BackingMap solo están en el servidor.

- **TransactionCallback**: un plug-in TransactionCallback proporciona sucesos de ciclo de vida de transacción. Si el plug-in TransactionCallback plug-in es la clase de implementación incorporada JPATxCallback (com.ibm.websphere.objectgrid.jpa.JPATxCallback), el plug-in es solo el lado del servidor. Sin embargo, las subclases de la clase JPATxCallback no son solo del lado del servidor.
- **ObjectGridEventListener**: un plug-in ObjectGridEventListener proporciona sucesos de ciclo de vida de ObjectGrid para ObjectGrid, los fragmentos y las transacciones.
- **ObjectGridLifecycleListener**: un plug-in ObjectGridLifecycleListener proporciona sucesos de ciclo de vida de ObjectGrid para la instancia de ObjectGrid. El plug-in ObjectGridLifecycleListener se puede utilizar como una interfaz combinada opcional para todos los demás plug-ins de ObjectGrid.
- **ObjectGridPlugin**: un ObjectGridPlugin es una interfaz mixin opcional que proporciona sucesos de gestión de ciclo de vida ampliados para todos los demás plug-ins de ObjectGrid.
- **SubjectSource, ObjectGridAuthorization, SubjectValidation**: eXtreme Scale proporciona varios puntos finales de seguridad para permitir que se integren mecanismos de autenticación personalizados con eXtreme Scale.(Sólo en el servidor)
- **MapAuthorization**: (solo lado del servidor)

Requisitos comunes de los plug-ins de ObjectGrid

ObjectGrid crea instancia de plug-in y las inicializa mediante los convenios de JavaBeans. Todas las implementaciones de plug-in anteriores tienen estos requisitos:

- La clase de plug-in debe ser una clase pública de nivel superior.
- La clase de plug-in debe proporcionar un constructor público sin argumentos.
- La clase de plug-in debe estar disponible en la vía de acceso de clase para servidores y clientes (como convenga).
- Los atributos se deben establecer utilizando los métodos de propiedad del estilo JavaBeans.
- Los plug-ins, a menos que se especifique lo contrario, se registran antes de que se inicialice ObjectGrid y no se pueden modificar una vez inicializado ObjectGrid.

Plug-ins de BackingMap

Los plug-ins siguientes están disponibles para un objeto BackingMap:

- **Evictor**: un plug-in de desalojador es un mecanismo predeterminado proporcionado para desalojar entradas de memoria caché y un plug-in para crear desalojadores personalizados.
- **ObjectTransformer**: un plug-in ObjectTransformer le permite serializar, deserializar y copiar objetos en la memoria caché.

- **OptimisticCallback**: un plug-in OptimisticCallback le permite personalizar las operaciones de mantenimiento de versiones y comparación de objetos de memoria caché al utilizar la estrategia de bloqueo optimista.
- **MapEventListener**: un plug-in MapEventListener proporciona notificaciones de devolución de llamada y cambios de estado de memoria caché significativos que se producen para BackingMap.
- **BackingMapLifecycleListener**: un plug-in BackingMapLifecycleListener proporciona sucesos de ciclo de vida de BackingMap para la instancia de BackingMap. El plug-in BackingMapLifecycleListener se puede utilizar como una interfaz mixin opcional para todos los demás plug-ins BackingMap.
- **BackingMapPlugin**: un BackingMapPlugin es una interfaz mixin opcional que proporciona sucesos de gestión de ciclo de vida ampliados para todos los demás plug-ins BackingMap.
- **Indexing**: utilice la característica de indexación, representada por el plug-in MapIndexplug-in, para generar un índice o varios índices en una correlación de BackingMap para soportar el acceso de datos que no son clave.
- **Loader**: un plug-in Loader en una correlación de ObjectGrid actúa como una memoria caché para los datos que normalmente se conservan en un almacén persistente o en el mismo sistema o en algún otro sistema. (Sólo en el servidor)

Visión general de los servicios de datos REST

El servicio de datos REST de WebSphere eXtreme Scale es un servicio HTTP Java compatible con Microsoft WCF Data Services (anteriormente ADO.NET Data Services) e implementa el Protocolo de datos abierto (OData). Microsoft WCF Data Services es compatible con esta especificación al utilizar Visual Studio 2008 SP1 y .NET Framework 3.5 SP1.

Requisitos de compatibilidad

El servicio de datos REST permite a cualquier cliente HTTP acceder a una cuadrícula de datos. El servicio de datos REST es compatible con el soporte de WCF Data Services que se proporciona con Microsoft .NET Framework 3.5 SP1. Se pueden desarrollar aplicaciones RESTful con las útiles herramientas proporcionadas por Microsoft Visual Studio 2008 SP1. En la figura se proporciona una visión general de cómo interactúa WCF Data Services con clientes y bases de datos.

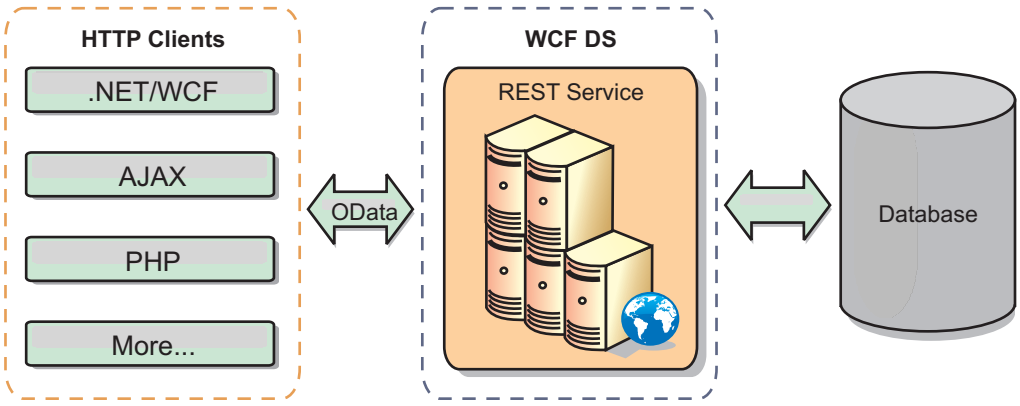


Figura 22. Microsoft WCF Data Services

WebSphere eXtreme Scale incluye una API con muchas funciones para clientes Java. Como se muestra en la figura siguiente, el servicio de datos REST es una

pasarela entre clientes HTTP y la cuadrícula de datos de WebSphere eXtreme Scale, comunicando con la cuadrícula mediante un cliente de WebSphere eXtreme Scale. El servicio de datos REST es un servlet Java, que permite despliegues flexibles para plataformas Java Platform, Enterprise Edition (JEE) comunes, como WebSphere Application Server. El servicio de datos REST se comunica con la cuadrícula de datos de WebSphere eXtreme Scale utilizando las API Java de WebSphere eXtreme Scale. Permite los clientes de WCF Data Services o cualquier otro cliente que pueda comunicarse con HTTP y XML.

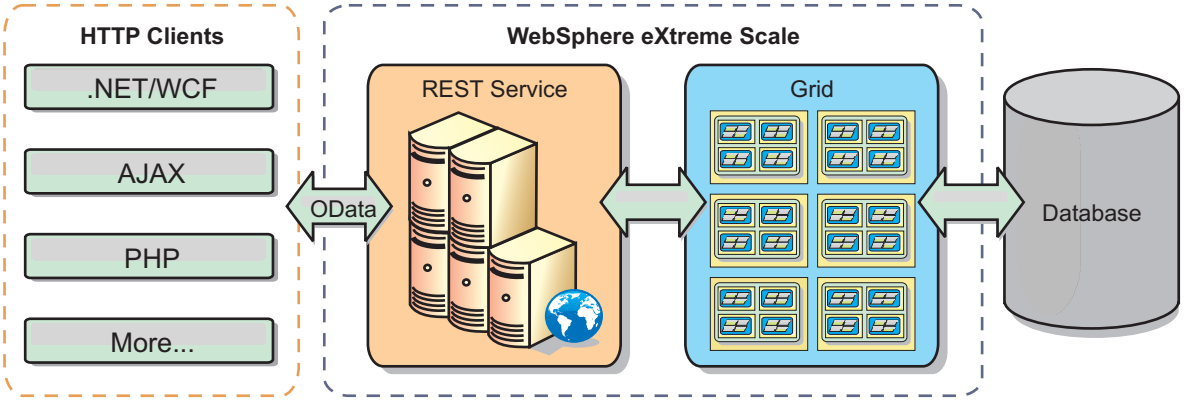


Figura 23. Servicio de datos REST de WebSphere eXtreme Scale

Consulte Configuración de servicios de datos REST, o utilice los enlaces siguientes para obtener más información sobre WCF Data Services.

- Microsoft WCF Data Services Developer Center
- Visión general de ADO.NET Data Services en MSDN
- Libro blanco: Uso de ADO.NET Data Services
- Protocolo de publicación Atom: URI de servicios de datos y ampliaciones de la carga útil
- Formato de archivo de definición de esquema conceptual
- Formato de empaquetado del modelo de datos de entidad para servicios de datos
- Protocolo de datos abierto
- Preguntas frecuentes sobre el protocolo de datos abierto

Características

Esta versión del servicio de datos REST de eXtreme Scale soporta las características siguientes:

- Modelado automático de entidades de API EntityManager de eXtreme Scale como entidades de WCF Data Services que incluye el soporte siguiente:
  - Conversión del tipo de datos Java al tipo de modelo de datos de entidad
  - Soporte para la asociación de entidades
  - Soporte para la asociación de raíces de esquema y claves, necesario para cuadrículas de datos particionadas

  Si desea más información, consulte Modelo de entidad.
- Atom Publish Protocol (AtomPub o APP) XML y formato de carga útil de datos JavaScript Object Notation (JSON).

- Operaciones de creación, lectura, actualización y supresión (CRUD) utilizando los respectivos métodos de solicitud HTTP: POST, GET, PUT y DELETE. Además, la ampliación de Microsoft: MERGE está soportada.
- Consultas simples, con filtros
- Solicitudes de recuperación por lotes y conjuntos de cambios
- Soporte de cuadrícula de datos particionada para alta disponibilidad
- Interoperatividad con clientes de la API EntityManager de eXtreme Scale
- Soporte para servidores web JEE estándar
- Simultaneidad optimista
- Autorización y autenticación de usuarios entre el servicio de datos REST y la cuadrícula de datos de eXtreme Scale

Problemas y limitaciones conocidos

- Las solicitudes a través de túnel no están soportadas.

Tareas relacionadas:

Configuración de servicios de datos REST
Puede utilizar el servicio de datos REST de WebSphere eXtreme Scale con WebSphere Application Server versión 7.0, WebSphere Application Server Community Edition y Apache Tomcat.

"Acceso a los datos con el servicio de datos REST" en la página 271
Desarrolle las aplicaciones que realizan operaciones con los protocolos del servicio de datos REST.

Referencia relacionada:

"Simultaneidad optimista en el servicio de datos REST" en la página 276
El servicio de datos REST de eXtreme Scale sigue un modelo de bloqueo optimista utilizando cabeceras HTTP nativas: If-Match, If-None-Match y ETag. Estas cabeceras se envían en mensajes de solicitud y respuesta para transmitir la información de versión de una entidad del servidor al cliente y del cliente al servidor.

"Protocolos de solicitud para el servicio de datos REST" en la página 277
En general, los protocolos para interactuar con los servicios REST son los mismos que se describen en el protocolo WCF Data Services AtomPub. No obstante, eXtreme Scale proporciona detalles adicionales, de la perspectiva de modelo de entidad de eXtreme Scale. Se espera que los usuarios estén familiarizados con los protocolos de WCF Data Services antes de leer esta sección. Como alternativa, los usuarios pueden leer esta sección con la sección del protocolo WCF Data Services.

"Solicitudes de recuperación con el servicio de datos REST" en la página 278
Un cliente utiliza una solicitud RetrieveEntity para recuperar una entidad de eXtreme Scale. La carga útil de respuesta contiene los datos de la entidad en formato AtomPub o JSON. Además, se puede utilizar el operador del sistema $expand para expandir las relaciones. Las relaciones se representan en línea en la respuesta de servicio de datos como un documento de canal de información Feed, que es una relación a muchos, o un documento de entrada Atom, que es una relación a uno.

"Recuperación de elementos que no sean entidades con los servicios de datos REST" en la página 285
El servicio de datos REST permite recuperar no sólo entidades, sino también elementos como colecciones de entidades y propiedades.

"Solicitudes de inserción con los servicios de datos REST" en la página 291
Se puede utilizar una solicitud InsertEntity para insertar una nueva instancia de entidad de eXtreme Scale, potencialmente con entidades relacionadas nuevas, en el servicio de datos REST de eXtreme Scale.

"Solicitudes de actualización con los servicios de datos REST" en la página 295
El servicio de datos REST de WebSphere eXtreme Scale soporta solicitudes de actualización de entidades, propiedades primitivas de entidades, etc.

"Solicitudes de supresión con los servicios de datos REST" en la página 299
El servicio de datos REST de WebSphere eXtreme Scale suprime entidades, valores de propiedad y enlaces.

Visión general de la infraestructura Spring

Spring es una infraestructura de desarrollo de aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para permitir a Spring gestionar transacciones y configurar los clientes y servidores que conforman una cuadrícula de datos en memoria desplegada.

Transacciones nativas gestionadas de Spring

Spring proporciona transacciones gestionadas por contenedor que son similares al servidor de aplicaciones Java Platform, Enterprise Edition. Sin embargo, el mecanismo Spring puede utilizar distintas implementaciones. WebSphere eXtreme Scale proporciona una integración del gestor de transacciones que permite a Spring gestionar los ciclos de vida de transacción de ObjectGrid. Consulte la información sobre transacciones nativas en la *Guía de programación* para obtener más información.

Beans de ampliación gestionados de Spring y soporte de espacio de nombres

Además, eXtreme Scale se integra con Spring para habilitar a los beans de estilo Spring definidos para los puntos o plug-ins de ampliación. Esta característica proporciona configuraciones más sofisticadas y más flexibilidad para configurar los puntos de ampliación.

Además de los beans de ampliación gestionados de Spring, eXtreme Scale proporciona un espacio de nombres Spring denominado "objectgrid". Los beans y las implementaciones incorporadas están definidos previamente en este espacio de nombres, que hace que sea más fácil para los usuarios configurar eXtreme Scale.

Soporte de ámbito de fragmento

Con la configuración de Spring de estilo tradicional, un bean ObjectGrid puede ser un tipo singleton o un tipo de prototipo. Además, ObjectGrid soporta un nuevo ámbito denominado el ámbito de "fragmento". Si un bean está definido como ámbito de fragmento, sólo se crea un bean por fragmento. Todas las solicitudes para los beans con un ID o varios ID que coincidan con dicha definición de bean en el mismo fragmento producirán que una instancia de bean específica sea devuelta por el contenedor Spring.

El siguiente ejemplo muestra que un bean com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl está definido con el ámbito establecido en shard (fragmento). Por lo tanto, sólo se crea una instancia de la clase JPAPropFactoryImpl por fragmento.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Flujo web de Spring

El flujo web de Spring almacena su estado de sesión en una sesión HTTP de forma predeterminada. Si una aplicación web utiliza eXtreme Scale para la gestión de sesiones, Spring almacena automáticamente el estado con eXtreme Scale. Además, la tolerancia a errores está habilitada de la misma forma que la sesión.

Consulte la información de gestión de sesiones HTTP en *Visión general del producto* para obtener detalles adicionales.

Empaquetado

Las extensiones Spring de eXtreme Scale están en el archivo `ogspring.jar`. Este archivo Java (JAR) debe estar en la classpath para trabajar con el soporte de Spring. Si una aplicación Java EE en ejecución en un WebSphere Extended Deployment ha aumentado WebSphere Application Server Network Deployment, coloque el archivo `spring.jar` y sus archivos asociados en los módulos de

archivadores empresariales (EAR). También debe colocar el archivo `ogspring.jar` en la misma ubicación.

Tareas relacionadas:

"Desarrollo de aplicaciones con la infraestructura Spring" en la página 417
Obtenga información sobre cómo integrar las aplicaciones de eXtreme Scale con la conocida infraestructura Spring.

"Inicio de un servidor de contenedor con Spring" en la página 428
Puede iniciar un servidor de contenedor utilizando beans de ampliación gestionados Spring y soporte de espacio de nombres.

"Gestión de transacciones con Spring" en la página 420
Spring es una infraestructura popular para desarrollar las aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para que Spring pueda gestionar transacciones de eXtreme Scale y configurar clientes y servidores de eXtreme Scale.

Referencia relacionada:

"Beans de ampliación gestionados de Spring" en la página 423
Puede declarar POJO (Plain Old Java Objects) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml`. Si denomina los beans y luego especifica el nombre de clase, eXtreme Scale suele crear instancias de la clase especificada y utiliza esas instancias como plug-in. Ahora, WebSphere eXtreme Scale ObjectGrid puede delegar en Spring para actuar como la fábrica de beans para obtener instancias de estos objetos de plug-in.

Archivo XML de descriptor Spring
Utilice un archivo XML de descriptor Spring para configurar e integrar eXtreme Scale con Spring.

Archivo Spring `objectgrid.xsd`
Utilice el archivo Spring `objectgrid.xsd` para integrar eXtreme Scale con Spring para gestionar las transacciones eXtreme Scale y configurar clientes y servidores.

Consideraciones del cargador de clases y la classpath

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

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

Consulte el tema Carga de clases del Information Center de WebSphere Application Server para obtener más información.

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

Gestión de las relaciones

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

Cuando se utilizan relaciones en una cuadrícula de datos, los datos se deben organizar en un árbol restringido. Debe existir un tipo raíz en el árbol y todos los

hijos deben estar asociados sólo a una raíz. Por ejemplo: El Departamento puede tener muchos Empleados y un Empleado puede tener muchos Proyectos. Pero un Proyecto no puede tener muchos Empleados que pertenezcan a distintos departamentos. Una vez definida una raíz, todos los accesos a dicho objeto raíz y a sus descendientes se gestionan a través de la raíz. WebSphere eXtreme Scale utiliza el código hash de la clave del objeto raíz para elegir una partición. Por ejemplo: `partition = (hashCode MOD numPartitions).`

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

Datos de referencia

Algunas relaciones incluyen datos de búsqueda o de referencia como, por ejemplo: `CountryName`. Para datos de búsqueda o referencia, los datos deben existir en cada partición. Cualquier clave raíz puede acceder a los datos y se devuelve el mismo resultado. Los datos de referencia como los siguientes solo deben utilizarse en casos en los que los datos sean bastante estáticos. La actualización de estos datos puede resultar costosa ya que los datos deben actualizarse en cada partición. La API DataGrid es una técnica común para mantener los datos actualizados.

Costes y ventajas de la normalización

La normalización de los datos que utilizan relaciones puede ayudar a reducir la cantidad de memoria utilizada por la cuadrícula de datos porque la duplicación de datos disminuye. Sin embargo, en general, cuantos más datos relacionales se añaden, menos se ampliarán. Si los datos se agrupan de forma conjunta, será más caro conservar las relaciones y mantener los tamaños gestionables. Puesto que los datos de particiones de cuadrícula se basan en la clave de la raíz del árbol, el tamaño del árbol no se toma en consideración. Por lo tanto, si tiene muchas relaciones para una instancia de árbol, la cuadrícula de datos se desequilibra, lo que provoca que una partición tenga más datos que las demás.

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

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

Gestión de relaciones utilizando las API de acceso de datos

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

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

A continuación se muestra un ejemplo.

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

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

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

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

Consideraciones de claves de la memoria caché

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

Directrices

Al elegir una clave, considere los siguientes requisitos:

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

Datos para distintos husos horarios

Al insertar datos con los atributos calendar, java.util.Date y timestamp en un ObjectGrid, debe asegurarse de que estos atributos de fecha y hora se creen basándose en el mismo huso horario, sobre todo cuando se realiza el despliegue en diversos servidores en varios husos horarios. La utilización de los mismos objetos de hecha y hora basados en huso horario puede garantizar que la aplicación tenga

seguridad de huso horario y que se puedan consultar los datos mediante los predicados calendar, java.util.Date y timestamp.

Sin especificar explícitamente un huso horario al crear objetos de fecha y hora, Java utiliza el huso horario local y puede causar valores de fecha y hora incoherentes en clientes y servidores.

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

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

Configuración de un entorno de despliegue autónomo

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

Antes de empezar

Instale el producto WebSphere eXtreme Scale en un directorio nuevo o vacío y aplique el fixpack acumulativo más reciente de WebSphere eXtreme Scale. También puede utilizar la versión de prueba de WebSphere eXtreme Scale descomprimiendo el archivo zip. Para obtener más información sobre la instalación, consulte la información sobre la instalación del WebSphere eXtreme Scale Client o WebSphere eXtreme Scale autónomo en la *Guía de administración*.

Procedimiento

- Configure Eclipse para crear y ejecutar una aplicación Java SE con WebSphere eXtreme Scale.
  1. Defina una biblioteca de usuario para permitir que la aplicación haga referencia a las interfaces de programación de aplicaciones de WebSphere eXtreme Scale.
     a. En el entorno Eclipse o IBM® Rational Application Developer, pulse **Ventana** > **Preferencias**.
     b. Expanda la ramificación **Java** > **Vía de acceso de compilación** y seleccione **Bibliotecas de usuario**. Pulse **Nueva**.
     c. Seleccione la biblioteca de usuario de eXtreme Scale. Pulse **Añadir JAR**.
        1) Vaya al archivo `objectgrid.jar` o `ogclient.jar` del directorio *raíz_wxs*`/lib` y selecciónelo. Pulse **Aceptar**. Seleccione el archivo `ogclient.jar` si está desarrollando aplicaciones cliente o memorias caché en memoria locales. Si está desarrollando y probando servidores eXtreme Scale, utilice el archivo `objectgrid.jar`.
        2) Para incluir Javadoc para las API de ObjectGrid, seleccione la ubicación del Javadoc para el archivo `objectgrid.jar` o `ogclient.jar`

que ha añadido en el paso anterior. Pulse **Editar**. En el recuadro de vía de acceso de ubicación del Javadoc, especifique la dirección web siguiente:

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

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

Las bibliotecas de eXtreme Scale ahora se encuentran en la vía de acceso de compilación del proyecto.

2. Añada la biblioteca de usuario a su proyecto Java.
   a. En el explorador de paquetes, pulse el botón derecho del ratón en el proyecto y seleccione **Propiedades**.
   b. Seleccione el separador **Bibliotecas**.
   c. Pulse **Añadir biblioteca**.
   d. Seleccione **Biblioteca de usuario**. Pulse **Siguiente**.
   e. Seleccione la biblioteca de usuario eXtreme Scale que ha configurado anteriormente.
   f. Pulse **Aceptar** para aplicar los cambios y cerrar la ventana Propiedades.

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

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

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

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

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

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

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

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

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

Antes de empezar

Instale el producto WebSphere eXtreme Scale de prueba o completo.

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

Procedimiento

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

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

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

j. Seleccione **Biblioteca de usuario**. Pulse **Siguiente**.

k. Seleccione la biblioteca **eXtremeScaleClient** y pulse **Finalizar**.

l. Pulse **Aceptar** para cerrar el diálogo **Propiedades del proyecto**.

Aplicación de servidor En este escenario, desea configurar y ejecutar una aplicación web para ejecutar un servidor WebSphere eXtreme Scale incorporado en Rational Application Developer.

a. Pulse **Ventana** > **Preferencias** > **Java** > **Vía de acceso de compilación** > **Bibliotecas de usuario**. Pulse **Nueva**.

b. Introduzca un **Nombre de biblioteca de usuario** de `eXtremeScale` y pulse **Aceptar**.

c. Pulse **Añadir Jars...** y seleccione *inicio_wxs*`/lib/objectgrid.jar`. Pulse Abrir.

d. (Opcional) Para añadir un Javadoc, seleccione la ubicación del Javadoc y pulse **Editar.....** En la vía de acceso de ubicación del Javadoc, introduzca `http://www.ibm.com/developerworks/wikis/extremescale/docs/api/`.

e. Pulse **Aceptar**.

f. Pulse **Aceptar** para cerrar el diálogo Bibliotecas de usuario.

g. Pulse **Proyecto** > **Propiedades**.

h. Pulse **Vía de acceso de compilación Java**.

i. Pulse **Añadir biblioteca**.

j. Seleccione **Biblioteca de usuario**. Pulse **Siguiente**.

k. Seleccione la biblioteca **eXtremeScaleClient** y pulse **Finalizar**.

l. Pulse **Aceptar** para cerrar el diálogo **Propiedades del proyecto**.

2. Defina Tomcat Server para el proyecto.

a. Asegúrese de que se encuentra en la perspectiva J2EE y pulse el separador **Servidores** en el panel inferior. Puede pulsar también **Ventana** > **Mostrar vista** > **Servidores**.

b. Pulse el botón derecho del ratón en el panel Servidores y seleccione **Nuevo** > **Servidor**.

c. Seleccione **Apache, Tomcat v6.0 Server**. Pulse **Siguiente**.

d. Pulse **Examinar...** Seleccione *raíz_tomcat*. Pulse **Aceptar**.

e. Pulse **Siguiente**.

f. Seleccione la aplicación Java EE en el panel izquierdo Disponibles y pulse **Añadir >** para moverla al panel derecho Configuradas en el servidor, y pulse **Finalizar**.

3. Resuelva los errores restantes del proyecto. Utilice los pasos siguientes para eliminar errores en el panel Problemas:

a. Pulse **Proyecto** > **Limpiar** > ***nombre_proyecto***. Pulse **Aceptar**. Compile el proyecto.

b. Pulse con el botón derecho del ratón en el proyecto Java EE y elija **Vía de acceso de compilación** > **Configurar vía de acceso de compilación**.

c. Pulse el separador **Bibliotecas**. Asegúrese de que la vía de acceso está configurada correctamente:

- **Para aplicaciones cliente:** asegúrese de que Apache Tomcat, eXtremeScaleClient y Java 1.5 JRE estén en la vía de acceso.
- **Para aplicaciones de servidor:** asegúrese de que Apache Tomcat, eXtremeScale y Java 1.5 JRE estén en la vía de acceso.

4. Cree una configuración de ejecución para ejecutar la aplicación.

a. En el menú **Ejecutar**, seleccione **Ejecutar configuraciones**.

b. Pulse con el botón derecho del ratón en la categoría Aplicación Java y seleccione **Nueva**.

c. Seleccione la nueva configuración de ejecución, denominada *Nueva_configuración*.

d. Configure el perfil.

- **Proyecto** (en la página tabulada principal): *su_nombre_proyecto*
- **Clase principal** (en la página tabulada principal): *su_clase_principal*
- **Argumentos de VM** (en la página tabulada de argumentos): `-Djava.endorsed.dirs=raíz_wxs/lib/endorsed`

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

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

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

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

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

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

Qué hacer a continuación

Después de configurar y ejecutar una aplicación web con el cliente de WebSphere eXtreme Scale en Rational Application Developer, puede desarrollar un servlet. Este servlet utiliza las API de WebSphere eXtreme Scale para almacenar y recuperar datos de una cuadrícula de datos remota.

Después de habilitar una aplicación Java EE en la interfaz de Rational Application Developer con una instalación autónoma de WebSphere eXtreme Scale, puede desarrollar un servlet que utilice las API del sistema de WebSphere eXtreme Scale para iniciar y detener servicios de catálogo.

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

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

Antes de empezar

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

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

Instale el cliente o servidor WebSphere eXtreme Scale en el entorno de prueba de WebSphere Application Server, Versión 7.0 en el directorio *inicio_rad*\runtimes\base_v7. Si desea más información, consulte Instalación de WebSphere eXtreme Scale o WebSphere eXtreme Scale Client con WebSphere Application Server.

Procedimiento

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

Capítulo 5. Desarrollo de aplicaciones

Desarrolle aplicaciones que utilicen la cuadrícula de datos. Las tareas para el desarrollo de aplicaciones incluyen:

- Acceso a datos
- Plug-ins y API del sistema
- Integración de JPA
- Integración de Spring

Acceso a los datos con aplicaciones cliente

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

Acerca de esta tarea

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

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

Conexión a instancias distribuidas de ObjectGrid mediante programación

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

Antes de empezar

- Para conectarse a una cuadrícula de datos distribuida, debe haber configurado el entorno del lado del servidor con un servicio de catálogo y servidores de contenedor.
- Debe tener el puerto de escucha para cada servicio de catálogo. Si desea más información, consulte Planificación de puertos de red .

Acerca de esta tarea

El método getObjectGrid(ClientClusterContext ccc, String objectGridName) conecta el dominio de servicio de catálogo especificado y devuelve una instancia de ObjectGrid de cliente correspondiente a una instancia de ObjectGrid del lado del servidor. Los pasos varían dependiendo de si está utilizando una configuración autónoma o de WebSphere Application Server.

Procedimiento

- Conéctese a una cuadrícula de datos distribuida autónoma mediante puntos finales de servicio de catálogo explícitos.

```
    // Crear una instancia de ObjectGridManager.
    ObjectGridManager ogm = ObjectGridManagerFactory.getObjectGridManager();

// Obtener un ClientClusterContext al conectar un
    // ObjectGrid distribuido basado en servidor.  Debe proporcionar
    // un punto final de conexión para el servidor de catálogo con el formato
    // de nombrehost:puertopuntofinal.  El nombrehost es la máquina donde
    // reside el servidor de catálogo y el puertopuntofinal es el puerto
    // de escucha del servidor de catálogo, cuyo valor predeterminado es 2809.
    // Los puntos finales de servidor de catálogo para un dominio determinado
    // deben estar en el formato de una lista delimitada por comas.

        String catalogServiceEndpoints = "host1:2809,host2:2809";
        ClientClusterContext ccc = ogm.connect(catalogServiceEndpoints, null, null);

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

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

- Conéctese a un dominio de servicio de catálogo desde una aplicación cliente alojada en WebSphere Application Server, donde el dominio de servicio de catálogo se ha configurado utilizando la consola administrativa o la tarea administrativa, los puntos finales de catálogo se pueden recuperar utilizando la API de servidor incorporado:

```
    ...

    // Recuperar los puntos finales de servicio de catálogo del singleton
    // ServerPropeties, que se configura en la consola administrativa o
    // la tarea administrativa de WebSphere.

    String catalogServiceEndpoints = ServerFactory.getServerProperties()
      .getCatalogServiceBootstrap();
    ClientClusterContext ccc = ogm.connect(catalogServiceEndpoints,
      null, null);

    ...
```

  Si el dominio de servicio de catálogo en WebSphere Application Server lo aloja el gestor de despliegue, los clientes externos a la célula, incluidos los clientes Java Platform, Standard Edition, se deben conectar al servicio de catálogo utilizando el nombre de host de gestor de despliegue y el puerto de arranque de IIOP. Cuando el servicio de catálogo se ejecuta en células de WebSphere Application Server mientras que los clientes se ejecutan fuera de las células, busque en las páginas de configuración del dominio de eXtreme Scale en la consola administrativa de WebSphere Application Server para obtener la información necesaria para hacer que un cliente señale al servicio de catálogo.

Seguimiento de las actualizaciones de correlación de una aplicación

Cuando una aplicación está realizando cambios en una Correlación durante una transacción, un objeto LogSequence rastrea estos cambios. Si la aplicación cambia una entrada en la correlación, un objeto LogElement correspondiente proporciona los detalles del cambio.

Se proporciona a los cargadores un objeto LogSequence para una correlación particular siempre que una aplicación llama a un método para desechar o

confirmar la transacción. El cargador se repite en los objetos LogElement dentro del objeto LogSequence y aplica cada objeto LogElement al programa de fondo.

Los receptores de ObjectGridEventListener que se han registrado con un ObjectGrid también utilizan objetos LogSequence. Se proporciona a estos receptores un objeto LogSequence para cada correlación en una transacción confirmada. Las aplicaciones pueden utilizar estos receptores para esperar a que cambien determinadas entradas, como un desencadenante en una base de datos convencional.

Las siguientes interfaces o clases relacionadas con el registro son proporcionadas por la infraestructura de eXtreme Scale:

- com.ibm.websphere.objectgrid.plugins.LogElement
- com.ibm.websphere.objectgrid.plugins.LogSequence
- com.ibm.websphere.objectgrid.plugins.LogSequenceFilter
- com.ibm.websphere.objectgrid.plugins.LogSequenceTransformer

Interfaz LogElement

Un LogElement representa una operación o una entrada durante una transacción. Un objeto LogElement tiene varios métodos para obtener sus distintos atributos. Los atributos utilizados de forma más habitual son los atributos de valor de tipo y actual capturados por getType() y getCurrentValue().

El tipo está representado por una de las constantes definidas en la interfaz LogElement: INSERT, UPDATE, DELETE, EVICT, FETCH o TOUCH.

El valor actual representa el nuevo valor para la operación si es INSERT, UPDATE o FETCH. Si la operación es TOUCH, DELETE o EVICT, el valor actual es nulo. Este valor se puede convertir a ValueProxyInfo cuando se utiliza ValueInterface.

Consulte la documentación de la API si desea más detalles sobre la interfaz LogElement.

Interfaz LogSequence

En la mayoría de las transacciones, las operaciones que se producen en más de una entrada de la correlación, así pues se crean varios objetos LogElement. Debe crear un objeto que se comporte como un compuesto de varios objetos LogElement. La interfaz LogSequence debe servir a este propósito incluyendo una lista de objetos LogElement.

Consulte la documentación de la API si desea más detalles sobre la interfaz LogSequence.

Utilización de LogElement y LogSequence

LogElement y LogSequence se utilizan ampliamente en eXtreme Scale y por los plug-ins de ObjectGrid que se han escrito por usuarios cuando se propagan las operaciones de un componente o servidor a otro componente o servidor. Por ejemplo, un objeto LogSequence puede ser utilizado por la función de propagación de transacción de ObjectGrid distribuido para propagar los cambios en otros servidores, o el cargador lo puede aplicar en el almacén de persistencia. LogSequence es utilizado principalmente por las siguientes interfaces.

- com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener

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

Ejemplo de cargador

Esta sección demuestra cómo se utilizan los objetos LogSequence y LogElement en un cargador. Un cargador se utiliza para cargar datos y persistirlos en un almacén persistente. El método batchUpdate de la interfaz Loader utiliza el objeto LogSequence:

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

Se llama al método batchUpdate cuando un ObjectGrid debe aplicar todos los cambios actuales a Loader. Se proporciona a Loader una lista de objetos LogElement para la correlación, encapsulados en un objeto LogSequence. La implementación del método batchUpdate debe repetir los cambios y aplicarlos en el programa de fondo. El siguiente fragmento de código demuestra cómo Loader utiliza un objeto LogSequence. El fragmento de código se repite en el conjunto de cambios y genera tres sentencias de proceso por lotes JDBC (Java DataBase Connectivity): inserts, updates y deletes:

```
public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException {
    // Obtener una conexión SQL que vaya a utilizarse.
    Connection conn = getConnection(tx);
    try
    {
    // Procesar la lista de cambios y crear un conjunto de
    // sentencias preparadas para ejecutar una
    // operación SQL. Las sentencias se almacenan en la memoria caché
    // en stmtCache.
    Iterator iter = sequence.getPendingChanges();
    while(iter.hasNext())
    {
        LogElement logElement = (LogElement) iter.next();
        Object key = logElement.getCacheEntry().getKey();
        Object value = logElement.getCurrentValue();
        switch ( logElement.getType().getCode() )
        {
            case LogElement.CODE_INSERT:
                buildBatchSQLInsert( key, value, conn );
                break;
            case LogElement.CODE_UPDATE:
                buildBatchSQLUpdate( key, value, conn );
                break;
            case LogElement.CODE_DELETE:
                buildBatchSQLDelete( key, conn );
                break;
        }
    }
    // Ejecute las sentencias de proceso por lotes que se crearon mediante
    // el bucle anterior.
    Collection statements = getPreparedStatementCollection(tx, conn);
    iter = statements.iterator();
    while(iter.hasNext())
    {
        PreparedStatement pstmt = (PreparedStatement) iter.next();
        pstmt.executeBatch();
    }
} catch (SQLException e) {
```

```
    LoaderException ex = new LoaderException(e);
    throw ex;
}
}
```

El ejemplo anterior ilustra la lógica de alto nivel de proceso del argumento LogSequence. Sin embargo, el ejemplo no ilustra los detalles sobre cómo se crean una sentencia SQL insert, update o delete. Se llama al método getPendingChanges en el argumento LogSequence para obtener un iterador de objetos LogElement que un Loader debe procesar y se utiliza el método LogElement.getType().getCode() para determinar si un LogElement es para una operación SQL insert, update o delete.

Ejemplo de desalojador

También puede utilizar los objetos LogSequence y LogElement con un Evictor. Un Evictor se utiliza para desalojar las entradas de correlación de la correlación de respaldo basándose en determinados criterios. El método apply de la interfaz Evictor utiliza LogSequence.

```
/**
* Se llama a éste durante la confirmación de la memoria caché para permitir
* al desalojador rastrear el uso de objetos en una correlación de respaldo.
* También se informará sobre las entradas que se hayan desalojado
* correctamente.
*
* @param secuencia LogSequence de cambios en la correlación
*/
void apply(LogSequence sequence);
```

Interfaces LogSequenceFilter y LogSequenceTransformer

A veces, es necesario filtrar los objetos LogElement de forma que sólo se aceptan los objetos LogElement con determinados criterios y se rechazan los otros objetos. Por ejemplo, es posible que desee serializar un LogElement determinado basándose en algún criterio.

LogSequenceFilter resuelve este problema con el siguiente método.

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

Este método devuelve el valor true si el LogElement determinado se debe utilizar en la operación, y devuelve el valor false si no se debe utilizar el LogElement proporcionado.

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

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

Este método permite al interlocutor proporcionar un filtro para determinar qué LogElements incluir en el proceso de serialización. El parámetro DistributionMode permite al interlocutor controlar el proceso de serialización. Por ejemplo, si la modalidad de distribución sólo es la invalidación, no es necesario serializar el valor. El segundo método de esta clase es el método inflate, del modo siguiente.

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

El método inflate lee el formulario serializado de la secuencia de registro, que se ha sido creado por el método serialize, desde la corriente de datos de entrada de objeto proporcionada.

Interacción con un ObjectGrid utilizando la interfaz ObjectGridManager

La clase ObjectGridManagerFactory y la interfaz ObjectGridManager proporcionan un mecanismo para crear, acceder a, y añadir datos a interfaces de ObjectGrid. La clase ObjectGridManagerFactory es una clase ayudante estática que sirve para acceder a la interfaz ObjectGridManager, un singleton. La interfaz ObjectGridManager incluye varios métodos de simplificación para crear instancias de un objeto ObjectGrid. La interfaz ObjectGridManager facilita la creación de instancias de ObjectGrid y su almacenamiento en memoria caché, y varios usuarios pueden acceder a estas instancias.

Creación de instancia de ObjectGrid con la interfaz ObjectGridManager

Cada uno de estos métodos crea una instancia local de un ObjectGrid.

Instancia local en memoria

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

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

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

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

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

        // inicializar ObjectGrid
        ivObjectGrid.initialize();

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

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

Configuración compartida predeterminada

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

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

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

Si la instancia de ObjectGrid se utiliza en compartimiento de igual a igual, el almacenamiento en memoria caché debe establecerse en true. Si desea más información sobre el compartimiento de igual a igual, consulte el tema Distribución de cambios entre máquinas virtuales Java de igual.

Configuración XML

WebSphere eXtreme Scale es altamente configurable. El ejemplo anterior demuestra cómo crear un objeto ObjectGrid sencillo, sin ninguna configuración. Este ejemplo muestra cómo crear una instancia ObjectdGrid previamente configurada basada en un archivo de configuración XML. Puede configurar una instancia de ObjectGrid mediante programación o mediante un archivo de configuración XML. También puede configurar ObjectGrid mediante una combinación de estos dos procedimientos. La interfaz ObjectGridManager permite la creación de una instancia de ObjectGrid basada en la configuración XML. La interfaz ObjectGridManager tiene varios métodos que toman una dirección URL como argumento. Cada archivo XML que se pasa a ObjectGridManager debe estar validado en al esquema. La validación de XML puede inhabilitarse sólo cuando el archivo se ha validado previamente y no se han realizado cambios en el archivo desde su última validación. Si se inhabilita la validación, se evita una pequeña sobrecarga, pero podría utilizarse un archivo XML no válido. IBM Java Developer Kit (JDK) Versión 5 tiene soporte para la validación de XML. Si se utiliza un JDK que no ofrezca este soporte, podría necesitarse Apache Xerces para validar el XML.

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

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

```
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees =
oGridManager.createObjectGrid(objectGridName, allObjectGrids,
 bvalidateXML, cacheInstance);
```

El archivo XML puede contener información de configuración para varios objetos ObjectGrid. El fragmento de código anterior devuelve específicamente ObjectGrid Employees, y presupone que la configuración de Employees se ha definido en el archivo.

Métodos createObjectGrid

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

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

/**
 * Crear una instancia de ObjectGrid con el nombre de ObjectGrid especificado. La
 * instancia de ObjectGrid creada se almacenará en memoria caché.
 * @param objectGridName El nombre de la instancia de ObjectGrid que se debe crear.
 * @return una instancia de ObjectGrid
 * @throws ObjectGridException si una ObjectGrid con este nombre ya se
 * ha almacenado en memoria caché, o si ha encontrado un error durante la
 * creación de ObjectGrid
 */
public ObjectGrid createObjectGrid(String objectGridName)
  throws ObjectGridException;

/**
 * Crear una instancia de ObjectGrid basada en el nombre ObjectGrid especificado y el
 * archivo XML. La instancia de ObjectGrid definida en el archivo XML con el nombre de
 * ObjectGrid especificado se creará y se devolverá. Si dicho ObjectGrid
 * no se encuentra en el archivo XML, se emitirá una excepción.
 *
 * Esta instancia de ObjectGrid se puede almacenar en memoria caché.
 *
 * Si la dirección URL es nula, se pasará por alto. Es este caso, este método
 * se comporta de la misma manera que {@link #createObjectGrid(String, boolean)}.
 *
 * @param objectGridName El nombre de la instancia de ObjectGrid que se debe devolver.
```

```
No debe ser nulo.
 * @param xmlFile una dirección URL para un archivo XML de formato correcto
basado en el esquema ObjectGrid.
 * @param enableXmlValidation si true, se valida el XML
 * @param cacheInstance Un valor booleano que indica si las
 * instancias de ObjectGrid
 * definidas en el XML se almacenarán o no en memoria caché. Si es true
 * (verdadero), la instancia almacenará en memoria caché.
 *
 * @throws ObjectGridException si existe un ObjectGrid con el mismo nombre
 * se ha almacenado en memoria caché previamente, no se puede encontrar ningún
 * nombre de ObjectGrid en el archivo XML, ni ningún otro error durante la
 * creación de ObjectGrid.
 * @return una instancia de ObjectGrid
 * @see ObjectGrid
 */
public ObjectGrid createObjectGrid(String objectGridName, final URL xmlFile,
final boolean enableXmlValidation, boolean cacheInstance)
 throws ObjectGridException;

/**
 * Procesar un archivo XML y crear una lista de objetos ObjectGrid basados
 * en el archivo.
 * Estas instancias de ObjectGrid pueden almacenarse en memoria caché.
 * Se emitirá una excepción ObjectGridException al intentar almacenar en
 * memoria caché un ObjectGrid recién creado
 * que tenga el mismo nombre que un ObjectGrid que ya se haya almacenado en
 * memoria caché.
 *
 * @param xmlFile El archivo que define un ObjectGrid o varios
 * ObjectGrids
 * @param enableXmlValidation Si se establece en true, se validará el archivo XML
 * en el esquema
 * @param cacheInstances Se establece en true para almacenar en caché todas las
instancias de ObjectGrid
 * creadas basadas en el archivo
 * @return una instancia de ObjectGrid
 * @throws ObjectGridException si se intenta crear y almacenar en caché
 * un ObjectGrid con el mismo nombre que
 * un ObjectGrid que ya se haya almacenado en memoria caché, o si se produce
 * cualquier otro error durante la
 * creación de ObjectGrid
 */
public List createObjectGrids(final URL xmlFile, final boolean enableXmlValidation,
boolean cacheInstances) throws ObjectGridException;

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

/**
 * Procesar el archivo XML y crear una única instancia de ObjectGrid con
 * el nombre de ObjectGrid especificado sólo si se encuentra un ObjectGrid con
 * ese nombre en el archivo. Si no se ha definido ningún ObjectGrid con ese
 * nombre en el archivo XML, se producirá una excepción ObjectGridException.
 * La instancia de ObjectGrid creada se almacenará en memoria caché.
 * @param objectGridName Nombre del ObjectGrid a crear. Este ObjectGrid debe
```

```
definirse en el archivo XML.
 * @param xmlFile El archivo XML a procesar
 * @return Un ObjectGrid recién creado
 * @throws ObjectGridException si un ObjectGrid con el mismo nombre se ha
 * almacenado en memoria caché previamente, no se puede encontrar ningún nombre
 * de ObjectGrid en el archivo XML, ni ningún otro error durante la creación de
 * ObjectGrid.
 */
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
  throws ObjectGridException;
```

Tareas relacionadas:

"Resolución de problemas de la conectividad de cliente" en la página 519
Existen varios problemas comunes específicos de los clientes y de la conectividad de cliente que puede resolver tal como se describe en las secciones siguientes.

Recuperación de datos almacenados en memoria caché con la interfaz ObjectGridManager

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

Recuperación de una instancia almacenada en memoria caché

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

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

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

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

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

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

Nota: También puede utilizar el método getObjectGrid para conectarse a una cuadrícula distribuida. Si desea más información, consulte "Conexión a instancias distribuidas de ObjectGrid mediante programación" en la página 131.

Eliminación de instancias de ObjectGrid con la interfaz de ObjectGridManager

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

Eliminar una instancia de ObjectGrid

Para eliminar de la memoria caché instancias de ObjectGrid, utilice uno de los métodos removeObjectGrid. La interfaz de ObjectGridManager no mantiene una referencia de las instancias que se eliminan. Existen dos métodos de eliminación. Un método toma un parámetro booleano. Si el parámetro booleano está establecido

en true, se llama al método destroy en el ObjectGrid. La llamada al método destroy en el ObjectGrid lo concluye y libera cualquier recurso que utilice el ObjectGrid. A continuación, aparece una descripción de cómo utilizar los dos métodos removeObjectGrid:

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

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

Control del ciclo de vida de un ObjectGrid con la interfaz ObjectGridManager

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

Gestión del ciclo de vida con un bean de arranque

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

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

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

    public boolean start(){
        // Inicio del bean de arranque
        // Se llama a este método cuando se inicia la aplicación
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // crear 2 ObjectGrids y colocar en memoria caché estas instancias
            ObjectGrid bookstoreGrid =
```

```
      objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
            ObjectGrid videostoreGrid =
      objectGridManager.createObjectGrid("videostore", true);
            // dentro de la JVM,
            // estas ObjectGrids pueden recuperarse ahora del
            //ObjectGridManager mediante el método getObjectGrid(String)
        } catch (ObjectGridException e) {
            e.printStackTrace();
            return false;
        }

        return true;
    }

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

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

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

Gestión de un ciclo de vida con un servlet

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

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

    public MyObjectGridServlet() {
        super();
    }

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

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

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

Acceso al fragmento de ObjectGrid

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

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

Referencia local a datos del fragmento

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

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

Optimización de cliente-servidor de ubicación compartida

Si una aplicación utiliza las API de cliente para acceder a una partición que tiene que colocarse con el JVM que contiene el cliente, se evita la red, pero se sigue produciendo alguna ordenación debido a los problemas actuales de implementación. Si se utiliza una cuadrícula, no tiene ningún impacto en el rendimiento de la aplicación por que el número de llamadas (N-1)/N direcciona a

una JVM distinta. Si siempre es necesario el acceso local con un fragmento, utilice las API del cargador o de ObjectGrid para invocar esa lógica.

Acceso a datos con índices (API Index)

Utilice la indexación para acceder más eficazmente a los datos.

Acerca de esta tarea

La clase HashIndex es una implementación de plug-in de índice incorporado que puede dar soporte a las dos interfaces de índice de la aplicación incorporadas: MapIndex y MapRangeIndex. También puede crear sus propios índices. Puede añadir HashIndex como un índice estático o dinámico a la correlación de respaldo, obtener el objeto de proxy de índice MapIndex o MapRangeIndex y utilizar el objeto de proxy de índice para buscar objetos almacenados en la memoria caché.

Si desea iterar a través de las claves en una correlación local, puede utilizar el índice predeterminado. Este índice no requiere ninguna configuración, pero se debe utilizar en el fragmento, utilizando una instancia de ObjectGrid o agente recuperada del método ShardEvents.shardActivated(ObjectGrid shard).

Nota: En un entorno distribuido, si el objeto de índice se obtiene del ObjectGrid de cliente, el índice tiene un objeto de índice de tipo cliente y todas las operaciones de índice se ejecutan en un ObjectGrid de servidor remoto. Si la correlación está particionada, las operaciones de índice se ejecutan en cada partición de forma remota. Los resultados de cada partición se fusionan antes de devolver los resultados a la aplicación. El rendimiento lo determina el número de particiones y el tamaño del resultado devuelto por cada partición. Es posible que se produzca un rendimiento bajo si ambos factores son altos.

Procedimiento

1. Si desea utilizar índices que no sean el índice local predeterminado, añada plug-ins de índice a la correlación de respaldo.
   - **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="index name" />
          <property name="RangeIndex" type="boolean" value="true"
      description="true for MapRangeIndex" />
         <property name="AttributeName" type="java.lang.String" value="employeeCode"
     description="attribute name" />
     </bean>
</backingMapPluginCollection>
```

     En este ejemplo de configuración XML, se utiliza la clase HashIndex incorporada como el plug-in de índice. La clase HashIndex da soporte a propiedades que los usuarios pueden configurar como, por ejemplo, Name, RangeIndex y AttributeName en el ejemplo anterior.
     - La propiedad **Name** se configura como `CODE`, una serie que identifica este plug-in de índice. El valor de la propiedad Name debe ser exclusivo dentro del ámbito de la BackingMap, y se puede utilizar para recuperar el objeto de índice por el nombre de la instancia de ObjectMap para la BackingMap.
     - La propiedad **RangeIndex** se configura como `true`, lo que significa que la aplicación puede difundir el objeto de índice recuperado a la interfaz MapRangeIndex. Si la propiedad RangeIndex se configura como `false`, la aplicación solo puede difundir el objeto de índice recuperado a la interfaz MapIndex. Un MapRangeIndex soporta las funciones para encontrar los datos utilizando las funciones de rango como, por ejemplo, mayor que,

menor que, o ambos, mientras que un MapIndex sólo soporta las funciones de igual. Si el índice se utiliza por consulta, la propiedad **RangeIndex** se debe configurar en `true` en índices de un solo atributo. Para un índice de relación y un índice compuesto, la propiedad RangeIndex se debe configurar en `false`.

– La propiedad **AttributeName** se configura como `employeeCode`, lo que significa que el atributo **employeeCode** del objeto almacenado en memoria caché se utiliza para crear un índice de un solo atributo. Si una aplicación necesita buscar objetos almacenados en memoria caché con varios atributos, la propiedad **AttributeName** se puede establecer en una lista delimitada por comas de atributos, lo que proporciona un índice compuesto.

- **Configuración programática:**

  La interfaz BackingMap tiene dos métodos que puede utilizar para añadir plug-ins de índice estático: addMapIndexplugin y setMapIndexplugins. Si desea más información, consulte la documentación de API. El ejemplo siguiente crea la misma configuración que el ejemplo de configuración XML:

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

    ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
    ObjectGrid ivObjectGrid = ogManager.createObjectGrid( "grid" );
    BackingMap personBackingMap = ivObjectGrid.getMap("person");

    // utilizar la clase HashIndex incorporada como clase de plug-ins de índices.
    HashIndex mapIndexplugin = new HashIndex();
    mapIndexplugin.setName("CODE");
    mapIndexplugin.setAttributeName("EmployeeCode");
    mapIndexplugin.setRangeIndex(true);
    personBackingMap.addMapIndexplugin(mapIndexplugin);
```

2. Acceda a valores y claves de correlación con índices.

   - **Índice local:**

     Para iterar por las claves y valores de una correlación local, puede utilizar el índice predeterminado. El índice predeterminado solo funciona en el fragmento, utilizando un fragmento o utilizando la instancia de ObjectGrid recuperada del método ShardEvents.shardActivated(ObjectGrid shard). Consulte el siguiente ejemplo:

```
MapIndex keyIndex = (MapIndex)
objMap.getIndex(MapIndexPlugin.SYSTEM_KEY_INDEX_NAME);
Iterator keyIterator = keyIndex.findAll();
```

   - **Índices estáticos:**

     Después de que se añada un plug-in de índice estático a una configuración de BackingMap y la instancia de ObjectGrid que lo contiene se inicialice, las aplicaciones pueden recuperar el objeto de índice por nombre de la instancia de ObjectMap para la BackingMap. Difunda el objeto de índice a la interfaz de índices de aplicación. Ahora, las operaciones que soporta la interfaz de índice de aplicación se pueden ejecutar.

```
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);
    }
```

   - **Índices dinámicos:**

     Puede crear y eliminar, mediante programación, índices dinámicos de una instancia de BackingMap en cualquier momento. Un índice dinámico se distingue de un índice estático en que el índice dinámico puede crearse

después de que se inicialice la instancia de ObjectGrid que lo contiene. A diferencia de la indexación estática, la indexación dinámica es un proceso asíncrono y necesita estar en un estado preparado antes de poder utilizarlo. Este método utiliza el mismo acercamiento para recuperar y usar los índices dinámicos que los índices estáticos. Puede eliminar un índice dinámico si éste deja de utilizarse. La interfaz BackingMap tiene métodos para crear y eliminar índices dinámicos.

Consulte la documentación de la API BackingMap para obtener más información sobre los métodos createDynamicIndex y removeDynamicIndex.

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

        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
        ObjectGrid og = ogManager.createObjectGrid("grid");
        BackingMap bm = og.getMap("person");
        og.initialize();

        // crear índice después de la inicialización de ObjectGrid sin DynamicIndexCallback.
        bm.createDynamicIndex("CODE", true, "employeeCode", null);

        try {
            // Si no se utiliza DynamicIndexCallback, debe esperar a que el índice esté preparado.
            // El tiempo de espera depende del tamaño actual de la correlación
            Thread.sleep(3000);
        } catch (Throwable t) {
            // ...
        }

        // Cuando el índice esté preparado, las aplicaciones pueden intentar obtener
        //  la instancia de la interfaz de índices de aplicación.
        // Las aplicaciones deben encontrar un modo de asegurarse de que el índice está preparado para el uso,
        // si no se utiliza la interfaz DynamicIndexCallback.
        // El ejemplo siguiente muestra el modo de esperar a que el índice esté preparado.
        // Tenga en cuenta el tamaño de la correlación en el tiempo total de espera.

        Session session = og.getSession();
        ObjectMap m = session.getMap("person");
        MapRangeIndex codeIndex = null;

        int counter = 0;
        int maxCounter = 10;
        boolean ready = false;
        while (!ready && counter < maxCounter) {
            try {
                counter++;
                codeIndex = (MapRangeIndex) m.getIndex("CODE");
                ready = true;
            } catch (IndexNotReadyException e) {
                // implica que el índice no está preparado,...
                System.out.println("Index is not ready. continue to wait.");
                try {
                    Thread.sleep(3000);
                } catch (Throwable tt) {
                    // ...
                }
            } catch (Throwable t) {
                // excepción inesperada
                t.printStackTrace();
            }
        }

        if (!ready) {
            System.out.println("Index is not ready.  Need to handle this situation.");
        }

        // Usar el índice para realizar consultas
        // Consulte la interfaz MapIndex o MapRangeIndex para obtener información sobre las operaciones admitidas.
        // El atributo de objeto en el que se crea el índice es EmployeeCode.
        // Presuponga que el atributo EmployeeCode es de tipo Integer: el
        // parámetro que se pasa a operaciones de índices tiene este tipo de datos.

        Iterator iter = codeIndex.findLessEqual(new Integer(15));

        // eliminar el índice dinámico cuando ya no se necesite

        bm.removeDynamicIndex("CODE");
```

Qué hacer a continuación

Puede utilizar la interfaz DynamicIndexCallback para obtener notificaciones en los sucesos de índice. Si desea más información, consulte “Interfaz DynamicIndexCallback” en la página 147.

Conceptos relacionados:

"Plug-ins para la indexación de datos" en la página 330
El HashIndex incorporado, la clase com.ibm.websphere.objectgrid.plugins.index.HashIndex, es un plug-in MapIndexPlugin que puede añadir a BackingMap para crear índices estáticos o dinámicos. Esta clase da soporte a las interfaces MapIndex y MapRangeIndex. La definición y la implementación de índices puede mejorar significativamente el rendimiento de la consulta.

"Plug-ins para la indexación personalizada de los objetos de memoria caché" en la página 336
Con un plug-in o índice MapIndexPlugin, puede escribir estrategias personalizadas de indexación que van más allá de los índices incorporados que proporciona eXtreme Scale.

"Utilización de un índice compuesto" en la página 339
El índice compuesto HashIndex mejora el rendimiento de la consulta y evita la costosa exploración de correlaciones. La característica también proporciona un método práctico para que la API HashIndex encuentre los objetos almacenados en memoria caché cuando los criterios de búsqueda implican muchos atributos.

"Índices" en la página 97
Utilice el plug-in MapIndexPlugin para crear un índice o varios índice en una BackingMap para dar soporte al acceso a datos no de clave.

Referencia relacionada:

"Atributos del plug-in HashIndex" en la página 333
Puede utilizar los atributos siguientes para configurar el plug-in HashIndex. Estos atributos definen propiedades como por ejemplo si utiliza un HashIndex compuesto o de atributos, o si la indexación de rango está habilitada.

Interfaz DynamicIndexCallback

La interfaz DynamicIndexCallback se ha diseñado para aplicaciones que desean recibir notificaciones de sucesos de indexación de tipo preparado, error o destruir. DynamicIndexCallback es un parámetro opcional para el método createDynamicIndex de BackingMap. Con una instancia registrada de DynamicIndexCallback, las aplicaciones pueden ejecutar lógica empresarial al recibir una notificación de un suceso de indexación.

Sucesos de indexación

Por ejemplo, el suceso preparado significa que el índice está preparado para el uso. Cuando se recibe una notificación de este suceso, una aplicación puede intentar recuperar y utilizar la instancia de la interfaz de índices de aplicación. Consulte la API DynamicIndexCallback en la documentación de la API para obtener más información.

Ejemplo: Utilización de la interfaz DynamicIndexCallback

```
BackingMap personBackingMap = ivObjectGrid.getMap("person");
    DynamicIndexCallback callback = new DynamicIndexCallbackImpl();
    personBackingMap.createDynamicIndex("CODE", true, "employeeCode", callback);


    class DynamicIndexCallbackImpl implements DynamicIndexCallback {
        public DynamicIndexCallbackImpl() {
        }

        public void ready(String indexName) {
            System.out.println("DynamicIndexCallbackImpl.ready() -> indexName = " + indexName);

            // Simular qué debe hacer la aplicación al recibir la notificación de que el índice está preparado.
            // Normalmente, la aplicación debería esperar a que se alcance el estado preparado y después proceder
            // con una lógica de uso de índices.
            if("CODE".equals(indexName)) {
                ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
                ObjectGrid og = ogManager.createObjectGrid( "grid" );
                Session session = og.getSession();
                ObjectMap map = session.getMap("person");
```

```
            MapIndex codeIndex = (MapIndex) map.getIndex("CODE");
            Iterator iter = codeIndex.findAll(codeValue);
        }
    }

    public void error(String indexName, Throwable t) {
        System.out.println("DynamicIndexCallbackImpl.error() -> indexName = " + indexName);
        t.printStackTrace();
    }

    public void destroy(String indexName) {
        System.out.println("DynamicIndexCallbackImpl.destroy() -> indexName = " + indexName);
    }
}
```

Utilización de sesiones para acceder a los datos de la cuadrícula

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

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

Métodos

Método get

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

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

Después de obtener una Session, la hebra mantiene una referencia a la sesión para uso propio. Si se llama al método getSession varias veces se devolverá una nuevo objeto Session cada vez.

Transacciones y métodos Session

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

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

```
Session session = objectGrid.getSession();
ObjectMap objectMap = session.getMap("someMap");
session.begin();
objectMap.insert("key1", "value1");
```

```
objectMap.insert("key2", "value2");
session.commit();
objectMap.insert("key3", "value3"); // auto-commit
```

Método Session.flush

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

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

Método NoWriteThrough

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

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

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

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

Obtención del método del objeto TxID

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

- Para comparar cuando busque una determinada transacción.

- Para almacenar datos compartidos entre los objetos TransactionCallback y Loader.

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

Método de supervisión del rendimiento

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

Proceso de un método LogSequence completo

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

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

Método markRollbackOnly

Este método se utiliza para marcar la transacción actual como "sólo de retrotracción". Marcar una transacción como "sólo de retrotracción" garantiza la retrotracción de la transacción aunque la aplicación invoque el método commit. Este método lo utiliza normalmente ObjectGrid o la aplicación cuando sabe que se pueden dañar los datos si se permite confirmar la transacción. Una vez invocado el método, el objeto Throwable que se pasa a este método se encadena a la excepción com.ibm.websphere.objectgrid.TransactionException que produce el método commit si se invoca en una sesión que se ha marcado previamente como "sólo retrotracción". Las siguientes llamadas a este método para una transacción marcada previamente como "sólo retrotracción" se ignora. Es decir, sólo se utiliza la primera llamada que pasa una referencia a Throwable no nula. Una vez finalizada la

transacción marcada, se elimina la marca "sólo retrotracción" para que se pueda confirmar la siguiente transacción iniciada por la sesión.

Método isMarkedRollbackOnly

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

Método setTransactionTimeout

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

Método getTransactionTimeout

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

transactionTimedOut

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

Método isFlushing

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

Método isCommitting

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

Método setRequestRetryTimeout

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

Método getRequestRetryTimeout

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

SessionHandle para el direccionamiento

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

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

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

Puesto que sólo es aplicable en un escenario de colocación por contenedor, al obtener el objeto SessionHandle se emite una IllegalStateException si una cuadrícula de datos determinada tiene varios conjuntos de correlaciones por contenedor o no tiene ninguno. Si no invoca el método setSessionHandle antes de llamar al método getSessionHandle, se seleccionará el objeto SessionHandle adecuado en función de la configuración de las propiedades del cliente.

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

Si desea ver un ejemplo sobre cómo poder utilizar un objeto SessionHandle, consulte el tema de direccionamiento preferido por zona.

Integración de SessionHandle

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

Objeto SessionHandle para direccionamiento de solicitudes

Puede enlazar un objeto SessionHandle a una sesión de las formas siguientes:

Consejo: En cada una de los siguientes llamadas a método, una vez que se enlaza un objeto SessionHandle a una sesión, se puede obtener el objeto SessionHandle del método Session.getSessionHandle.

- **Llamar al método Session.getSessionHandle:** cuando se llama al método, si no hay ningún objeto SessionHandle enlazado a la sesión, se selecciona aleatoriamente un objeto SessionHandle y se enlaza a la sesión.
- **Llamar a las operaciones de transacción de crear, leer, actualizar y suprimir:** cuando se llama a estos métodos o durante la confirmación, si no hay ningún objeto SessionHandle enlazado a la sesión, se selecciona aleatoriamente un objeto SessionHandle y se enlaza a la sesión.

- **Llamar al método ObjectMap.getNextKey:** cuando se llama a este método, si no hay ningún objeto SessionHandle enlazado a la sesión, se direcciona aleatoriamente la solicitud de operación a particiones individuales hasta que se obtiene una clave. Si se devuelve una clave de una partición, un objeto SessionHandle correspondiente a esa partición se enlaza con la sesión. Si no se encuentra ninguna clave, no se enlaza ningún SessionHandle a la sesión.
- **Llamar a los métodos QueryQueue.getNextEntity o QueryQueue.getNextEntities:** en el momento de llamar a este método, si no hay ningún objeto SessionHandle enlazado a la sesión, la solicitud de operación se direcciona aleatoriamente a particiones individuales hasta que se obtiene un objeto. Si se devuelve un objeto de una partición, un objeto SessionHandle correspondiente a esa partición se enlaza a la sesión. Si no se encuentra ningún objeto, no se enlaza ningún SessionHandle con la sesión.
- **Establecer un SessionHandle con el método Session.setSessionHandle(SessionHandle sh):** si se obtiene un SessionHandle del método Session.getSessionHandle, el SessionHandle se puede enlazar a una sesión. El establecimiento de un SessionHandle afecta al direccionamiento de solicitudes en el ámbito de la sesión a la que se enlaza.

El método Session.getSessionHandle siempre devuelve un objeto SessionHandle. El método enlaza automáticamente un SessionHandle en la sesión si no hay ningún objeto SessionHandle enlazado a la sesión. Si desea verificar si una sesión tiene solo un objeto SessionHandle, llame al método Session.isSessionHandleSet. Si este método devuelve un valor de `false`, no hay ningún objeto SessionHandle enlazado actualmente a la sesión.

Tipos de operación principales en el escenario de colocación por contenedor

A continuación se muestra un resumen del comportamiento del direccionamiento de los principales tipos de operación en el escenario de colocación por contenedor respecto a los objetos SessionHandle.

- **Objeto de sesión con objeto SessionHandle enlazado**
  - Index - API MapIndex y MapRangeIndex: SessionHandle
  - Query y ObjectQuery: SessionHandle
  - Agent - API MapGridAgent y ReduceGridAgent: SessionHandle
  - ObjectMap.Clear: SessionHandle
  - ObjectMap.getNextKey: SessionHandle
  - QueryQueue.getNextEntity, QueryQueue.getNextEntities: SessionHandle
  - Operaciones de transacción de crear, recuperar, actualizar y suprimir (API ObjectMap y API EntityManager): SessionHandle
- **Objeto de sesión sin objeto SessionHandle enlazado**
  - Index - API MapIndex y MapRangeIndex: Todas las particiones activas actuales
  - Query y ObjectQuery: partición especificada con método setPartition de Query y ObjectQuery
  - Agent - MapGridAgent y ReduceGridAgent
    - No soportados: Método ReduceGridAgent.reduce(Session s, ObjectMap map, Collection keys) y MapGridAgent.process(Session s, ObjectMap map, Object key).

- Todas las particiones activas actuales: Método ReduceGridAgent.reduce(Session s, ObjectMap map) y MapGridAgent.processAllEntries(Session s, ObjectMap map).

– ObjectMap.clear: Todas las particiones activas actuales.
– ObjectMap.getNextKey: enlaza un SessionHandle a la sesión si se devuelve una clave de una de las particiones seleccionadas aleatoriamente. Si no se devuelve ninguna clave, la sesión no se enlaza a un SessionHandle.
– QueryQueue: Especifica una partición con el método QueryQueue.setPartition. Si no se establece ninguna partición, el método selecciona aleatoriamente un partición para devolverla. Si se devuelve un objeto, la sesión actual se enlaza al SessionHandle que se enlaza a la partición que devuelve el objeto. Si no se devuelve ningún objeto, la sesión no se enlaza a un SessionHandle.
– Operaciones de transacción de crear, recuperar, actualizar y suprimir (API ObjectMap y API EntityManager): Seleccionar aleatoriamente una partición.

En la mayoría de los casos, utilice SessionHandle para controlar el direccionamiento a una partición determinada. Puede recuperar y almacenar en memoria caché el SessionHandle de la sesión que inserta datos. Después de almacenar en memoria caché el SessionHandle, puede establecerlo en otra sesión, de forma que puede direccionar las solicitudes a la partición especificada por el SessionHandle almacenado en memoria caché. Para realizar operaciones como por ejemplo ObjectMap.clear sin SessionHandle, puede establecer temporalmente el SessionHandle en nulo llamando a Session.setSessionHandle(null). Sin un SessionHandle especificado, las operaciones se ejecutan en todas las particiones activas actuales.

- **Comportamiento de direccionamiento de QueryQueue**

  En el escenario de colocación de particiones por contenedor, puede utilizar SessionHandle para controlar el direccionamiento de los métodos getNextEntity y getNextEntities de la API QueryQueue. Si la sesión está enlazada a un SessionHandle, las solicitudes se direccionan a la partición a la que está enlazado el SessionHandle. Si la sesión no se enlaza a SessionHandle, las solicitudes se direccionan a la partición establecida con el método QueryQueue.setPartition si la partición se ha establecido de esta forma. Si la sesión no tiene ningún SessionHandle o partición enlazados, se devuelve una partición seleccionada aleatoriamente. Si no se encuentra este tipo de partición, el proceso se detiene y no se enlaza ningún and no SessionHandle a la sesión actual.

El siguiente fragmento de código muestra cómo utilizar el objeto SessionHandle.

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

// enlazando SessionHandle
SessionHandle sessionHandle = ogSession.getSessionHandle();

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

// la transacción se direcciona a la partición especificada por SessionHandle
ogSession.commit();
// almacenar en memoria caché el SessionHandle que inserta datos
SessionHandle cachedSessionHandle = ogSession.getSessionHandle();

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

// desenlazar temporalmente el SessionHandle de la sesión
```

```
if(isSessionHandleSet) {
    ogSession.setSessionHandle(null);
}

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

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

Consideraciones sobre el diseño de aplicaciones

En el escenario de la estrategia de colocación por contenedor, utilice el objeto SessionHandle para la mayoría de operaciones. El objeto SessionHandle controla el direccionamiento a las particiones. Para recuperar datos, el objeto SessionHandle que se enlaza a la sesión debe ser el mismo objeto SessionHandle de cualquier transacción de inserción de datos.

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

Cuando se enlaza una sesión a un SessionHandle, todas las solicitudes de la operación se direccionan a la partición especificada por el objeto SessionHandle. Sin el objeto SessionHandle establecido, las operaciones se direccionan a todas las operaciones o a una partición seleccionada aleatoriamente.

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

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

Si desea más información sobre la API EntityManager, consulte “Almacenamiento en memoria caché de objetos y sus relaciones (API EntityManager)” en la página 166.

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

Tareas relacionadas:

"Iniciación al desarrollo de aplicaciones" en la página 70
Para empezar a desarrollar aplicaciones WebSphere eXtreme Scale, configure un entorno de desarrollo en Eclipse.

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Información relacionada:

"Lección 2 de la guía de aprendizaje de iniciación: Creación de una aplicación cliente" en la página 63
Para insertar, suprimir, actualizar y recuperar datos de la cuadrícula de datos, debe escribir una aplicación de cliente. El ejemplo de iniciación incluye una aplicación cliente que puede utilizar para aprender a crear su propia aplicación cliente.

Introducción a ObjectMap

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

Finalidad

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

Obtener una instancia de ObjectMap

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

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

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

Confirmar automáticamente transacciones

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

Semántica de los métodos

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

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

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

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

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

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

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

La anulación local se especifica cuando se pasa el valor false como parámetro **isGlobal** del método de anulación. La anulación local descarta todos los cambios realizados en la entrada en la memoria caché de transacción. Si la aplicación emite un método get, la entrada se capta del último valor confirmado en la BackingMap. Si no existe ninguna entrada en la BackingMap, la entrada se capta del último valor confirmado o desechado del Loader. Cuando se confirma una transacción, todas las

entradas que se han marcado como anuladas localmente no tienen ningún impacto en la BackingMap. Todos los cambios que se hayan desechado al Loader siguen estando comprometidos incluso si se ha desechado la entrada.

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

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

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

Método put

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

Método remove

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

Método setCopyMode

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

Método touch

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

Método update

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

Método getIndex

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

Método clear

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

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

Correlaciones dinámicas

Con las correlaciones dinámicas, puede crear correlaciones una vez que la cuadrícula de datos ya se haya in icializado.

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

Ventajas de las correlaciones dinámicas

La introducción de correlaciones dinámicas reduce la restricción de tener que definir todas las cuadrículas antes de la inicialización. Mediante la utilización de correlaciones de plantilla, puede crear correlaciones una vez que se haya inicializado el ObjectGrid.

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

Creación de correlaciones dinámicas

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

Si se pasa una serie que coincide con la expresión regular de una correlación de plantilla produce la creación de una ObjectMap y una BackingMap asociada.

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

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

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

Nota: Cuando defina correlaciones de plantilla, compruebe que los nombres de correlación sean los suficientemente exclusivos de forma que pueda correlacionar solo una de las correlaciones de plantilla con el método Session.getMap(String mapName). Si el método getMap() coincide con más de un patrón de correlación de plantilla, se generará una excepción `IllegalArgumentException`.

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

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

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

El archivo XML anterior define una correlación de plantilla y una correlación sin plantilla. El nombre de la correlación de plantilla es una expresión regular:

templateMap.*. Cuando se llama al método Session.getMap(String) con un nombre de correlación que coincide con esta expresión regular, la aplicación crea una correlación.

Ejemplo

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

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

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

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

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

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

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

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

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

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

Ejemplos adicionales

objectGrid.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
   <objectGrid name="session">
<backingMap name="objectgrid.session.metadata.dynamicmap.*" template="true"
        lockStrategy="PESSIMISTIC" ttlEvictorType="LAST_ACCESS_TIME">
    <backingMap name="objectgrid.session.attribute.dynamicmap.*"
        template="true" lockStrategy="OPTIMISTIC"/>
    <backingMap name="datagrid.session.global.ids.dynamicmap.*"
        lockStrategy="PESSIMISTIC"/>
 </objectGrid>
</objectGrids>
</objectGridConfig>
```

objectGridDeployment.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy
   ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="session">
    <mapSet name="mapSet2" numberOfPartitions="5" minSyncReplicas="0"
      maxSyncReplicas="0" maxAsyncReplicas="1" developmentMode="false"
      placementStrategy="PER_CONTAINER">
     <map ref="logical.name"/>
     <map ref="objectgrid.session.metadata.dynamicmap.*"/>
     <map ref="objectgrid.session.attribute.dynamicmap.*"/>
     <map ref="datagrid.session.global.ids"/>
    </mapSet>
  </objectgridDeployment>

</deploymentPolicy>
```

Limitaciones y consideraciones

Limitaciones:

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

Consideraciones:

- Muchos plug-ins no tienen ningún modo para determinar la correlación con la que se asocia cada plug-in.
- Otros plug-ins se diferencian entre sí utilizando un nombre de correlación o un nombre de BackingMap como argumento.
- Al definir correlaciones de plantilla, compruebe que los nombres de correlación sean lo suficientemente exclusivos para que la aplicación se pueda correlacionar con solo una de las correlaciones de plantilla mediante el método Session.getMap(String mapName). Si el método getMap() coincide con más de un patrón de correlación de plantilla, se generará una excepción `IllegalArgumentException`.

ObjectMap y JavaMap

Una instancia de JavaMap se obtiene de un objeto ObjectMap. La interfaz JavaMap tiene las mismas firmas de método que ObjectMap, pero con un manejo de excepciones distinto. JavaMap amplía la interfaz java.util.Map, por lo que todas las

excepciones son instancias de la clase java.lang.RuntimeException. Como JavaMap amplía la interfaz java.util.Map, es fácil utilizar rápidamente WebSphere eXtreme Scale con una aplicación existente que utiliza una interfaz java.util.Map para almacenar los objetos en la memoria caché.

Obtener una instancia de JavaMap

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

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

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

Métodos

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

Método containsKey(java.lang.Object)

Método get(java.lang.Object)

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

Método putAll(java.util.Map)

Método remove(java.lang.Object)

clear()

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

Correlaciones como colas FIFO

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

Ejemplo FIFO

El siguiente fragmento de código muestra un cliente entrando un bucle para procesar entradas en la correlación hasta que la correlación se agota. El bucle inicia una transacción y luego llama al método ObjectMap.getNextKey(5000). Este método devuelve la clave de la siguiente entrada desbloqueada disponible y la bloquea. Si la transacción está bloqueada durante más de 5000 milisegundos, el método devuelve un valor nulo.

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

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

Modalidad local frente a modalidad de cliente

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

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

Ejemplo

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

Job.java

```
package tutorial.fifo;

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

@Entity
public class Job
```

```
{
 @Id String jobId;

 int jobState;
}
```

JobEvent.java

```
package tutorial.fifo;

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

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

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

JobEventListener.java

```
package tutorial.fifo;

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

public class JobEventListener
{
 boolean stopListening;

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

 synchronized boolean isStopped()
 {
  return stopListening;
 }

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

   Object jobEventKey = jobEvents.getNextKey(5000);
   if(jobEventKey == null)
   {
    em.getTransaction().rollback();
    continue;
   }
   JobEvent event = (JobEvent)em.find(JobEvent.class, jobEventKey);
   // procesar el suceso, aquí sólo se incrementa el
   // estado de trabajo
   event.job.jobState++;
```

```
    em.getTransaction().commit();
   }
  }
}
```

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

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

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

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

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

Cargadores y correlaciones FIFO

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

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

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

Limitaciones de las API basadas en correlaciones

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

- Los índices y consultas deben utilizar el reflejo para consultar los campos y las propiedades de los objetos de memoria caché.

- Se requiere serialización de datos personalizados para conseguir rendimiento para objetos complejos.
- Trabajar con gráficos de objetos es difícil. Debe desarrollar la aplicación para almacenar referencias artificiales entre los objetos y unir manualmente los objetos.

Ventajas de la API EntityManager

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

Este conjunto de API sigue el estilo POJO (Plain Old Java Object (POJO) de programación que adoptan la mayoría de infraestructuras.

Tareas relacionadas:

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Referencia relacionada:

"Agente de instrumentación de rendimiento de entidades" en la página 470
Puede mejorar el rendimiento de las entidades de acceso a campos habilitando el agente de instrumentación de WebSphere eXtreme Scale cuando se utiliza Java Development Kit (JDK) versión 1.5 o posterior.

"Definición de un esquema de entidad" en la página 169
Un ObjectGrid puede tener varios un número ilimitado de esquemas de entidades lógicas. Las entidades se definen utilizando las clases Java anotadas, XML o una combinación de XML y clases Java. Las entidades definidas se registran con un servidor eXtreme Scale y se enlazan a BackingMaps, índices y otros plug-ins.

"Métodos de devolución de llamada y escuchas de entidad" en la página 186
Se puede notificar a las aplicaciones cuando el estado de una entidad pasa de un estado a otro. Existen dos mecanismos de devolución de llamada para los sucesos de cambio de estado: los métodos de devolución de llamada de ciclo de vida que están definidos en una clase de entidad y se invocan siempre que cambia el estado de la entidad, y los escuchas de entidad, que son más generales porque el escucha de entidad se puede registrar en varias entidades.

"Ejemplos de escucha de entidad" en la página 190
Puede escribir EntityListeners según sus necesidades. A continuación se ofrecen varios scripts de ejemplo.

"Interfaz EntityTransaction" en la página 203
Puede utilizar la interfaz EntityTransaction para delimitar transacciones.

Gestión de las relaciones

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

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

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

Datos de referencia

Algunas relaciones incluyen datos de búsqueda o de referencia como, por ejemplo: `CountryName`. Para datos de búsqueda o referencia, los datos deben existir en cada partición. Cualquier clave raíz puede acceder a los datos y se devuelve el mismo resultado. Los datos de referencia como los siguientes solo deben utilizarse en casos en los que los datos sean bastante estáticos. La actualización de estos datos puede resultar costosa ya que los datos deben actualizarse en cada partición. La API DataGrid es una técnica común para mantener los datos actualizados.

Costes y ventajas de la normalización

La normalización de los datos que utilizan relaciones puede ayudar a reducir la cantidad de memoria utilizada por la cuadrícula de datos porque la duplicación de datos disminuye. Sin embargo, en general, cuantos más datos relacionales se añaden, menos se ampliarán. Si los datos se agrupan de forma conjunta, será más caro conservar las relaciones y mantener los tamaños gestionables. Puesto que los datos de particiones de cuadrícula se basan en la clave de la raíz del árbol, el tamaño del árbol no se toma en consideración. Por lo tanto, si tiene muchas relaciones para una instancia de árbol, la cuadrícula de datos se desequilibra, lo que provoca que una partición tenga más datos que las demás.

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

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

Gestión de relaciones utilizando las API de acceso de datos

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

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

A continuación se muestra un ejemplo.

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

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

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

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

Definición de un esquema de entidad

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

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

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

Configuración de esquema de entidad

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

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

Las entidades se registran con una instancia de ObjectGrid antes de que se inicialice. Cada entidad definida debe tener un nombre exclusivo y se enlaza

automáticamente a una BackingMap de ObjectGrid con el mismo nombre. El método de inicialización varía en función de la configuración que se utilice:

Configuración de eXtreme Scale local

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

Configuración de eXtreme Scale distribuido

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

Para obtener más detalles, consulte “Gestor de entidades en un entorno distribuido” en la página 178.

Requisitos de la entidad

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

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

Entidades sin clase

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

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

- El contenedor de eXtreme Scale se aloja en un servidor que no permite clases de aplicación en la vía de acceso de clases. En este caso, los clientes pueden seguir accediendo a la cuadrícula utilizando la API EntityManager desde un cliente donde las clases estén permitidas.

- El cliente de eXtreme Scale no necesita acceso a las clases de entidad porque el cliente utiliza o un cliente que no es Java, como el servicio de datos REST de eXtreme Scale, o el cliente accede a los datos de tuple de la cuadrícula utilizando la API ObjectMap.

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

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

Clase de entidad mediante programa

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

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

Campos, claves y versiones sin clase

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

Metadatos de entidad sin clase

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

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

<entity class-name="@Department" name="Department" >
    <attributes>
        <id name="number" type="int"/>
        <basic name="name" type="java.lang.String"/>
        <version name="ver" type="int"/>
         <one-to-many name="employees"
            target-entity="@Employee"
```

```
            fetch="LAZY"
            mapped-by="department">
             <cascade><cascade-all/></cascade>
          </one-to-many>
    </attributes>
</entity>
```

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

Requisitos de clases de entidades

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

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

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

Atributos persistentes

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

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

Propiedades persistentes
Las entidades de acceso a propiedades se deben adherir a los convenios de firma de JavaBeans para las propiedades de lectura y grabación. Se ignoran

los métodos que no siguen los convenios de JavaBeans o que tienen la anotación Transient en el método getter. Para una propiedad de tipo T, debe haber un método de obtención getProperty que devuelva un valor de tipo T y un método de establecimiento vacío setProperty(T). Para los tipos booleanos, el método de obtención puede expresarse como isProperty devolviendo true o false. Las propiedades persistentes no pueden tener el modificador static.

Tipos de atributos soportados

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

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

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

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

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

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

Asociaciones de entidad

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

La cuadrícula de eXtreme Scale es la memoria caché de datos y no fuerza la integridad referencial como una base de datos. Aunque las relaciones permiten las

operaciones de persistencia y eliminación en cascada para entidades hijas, no detecta ni impone enlaces rotos con los objetos. Cuando se elimina un objeto hijo, la consulta a ese objeto debe eliminarse del padre.

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

Asociaciones con un solo valor

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

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

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

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

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

Asociaciones valoradas por colecciones

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

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

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

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


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

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

Asociaciones sin clase

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

A continuación se muestra un ejemplo:

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

Claves primarias

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

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

Las claves primarias compuestas pueden, de forma opcional, definir una clase de clave primaria. Una entidad se asocia a una clase de clave primaria utilizando la anotación `@IdClass` o el archivo de descriptor XML de entidad. Una anotación `@IdClass` resulta útil conjuntamente con el método EntityManager.find.

Las clases de claves primarias tienen los siguientes requisitos:

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

A continuación se muestra un ejemplo:

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

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

public class CustomerKey {
    Zone zone;
    int custId;

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

Claves primarias sin clase

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

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

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

Proxies de entidad e intercepción de campos

Las clases de entidad y los tipos de atributos soportados mutables se amplían mediante clases de proxy para entidades de acceso a propiedades y se han mejorado mediante códigos de bytes para entidades de acceso a campos de Java Development Kit (JDK) 5. Cualquier acceso a la entidad, incluso por los métodos de negocios internos y los métodos equals, deben utilizar los métodos de acceso a propiedades o campos adecuados.

Los interceptores de proxies y campos se utilizan para permitir al gestor de entidades realizar un seguimiento del estado de la entidad, determinar si la

entidad ha cambiado y mejorar el rendimiento. Los interceptores de campos sólo están disponibles en plataformas de Java SE 5 cuando se configura el agente de instrumentación de entidad.

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

Conceptos relacionados:

"Ajuste del rendimiento de la interfaz EntityManager" en la página 468
La interfaz EntityManager separa las aplicaciones del estado alojado en su almacén de datos de cuadrícula de servidores.

"Almacenamiento en memoria caché de objetos y sus relaciones (API EntityManager)" en la página 166
La mayoría de los productos de memoria caché utilizan las API basadas en correlaciones para almacenar los datos como pares de clave-valor. La API ObjectMap y la memoria caché dinámica de WebSphere Application Server, entre otras, utilizan este enfoque. No obstante, las API basadas en correlaciones tienen limitaciones. La API EntityManager simplifica la interacción con la cuadrícula de datos proporcionando una forma fácil de declarar un gráfico complejo de objetos relacionados e interactuar con él.

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

"Interacción con EntityManager" en la página 183
Normalmente, las aplicaciones en primer lugar obtienen una referencia de ObjectGrid y, a continuación, una Session de dicha referencia para cada hebra. Las sesiones no pueden compartirse entre hebras. Hay disponible un método adicional en el elemento Session, el método getEntityManager. Este método devuelve una referencia a un gestor de entidades para utilizar para esta hebra. La interfaz EntityManager puede sustituir las interfaces Session y ObjectMap para todas las aplicaciones. Puede utilizar estas API EntityManager si el cliente tiene acceso a las clases de entidad definidas.

"Soporte de planes de captación de EntityManager" en la página 193
Un FetchPlan es la estrategia que el gestor de entidades utiliza para recuperar los objetos asociados si la aplicación tiene que acceder a las relaciones.

"Colas de consulta de entidades" en la página 198
Las colas de consulta permiten a las aplicaciones crear una cola calificada por una consulta en el servidor o en eXtreme Scale local para una entidad. Las entidades del resultado de la consulta se almacenan en esta cola. Actualmente, la cola de consulta sólo se admite en una correlación que utilice la estrategia de bloqueo pesimista.

Tareas relacionadas:

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Gestor de entidades en un entorno distribuido

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

Archivos de configuración obligatorios

Son necesarios los siguientes archivos de configuración XML:

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

Estos archivos especifican las entidades o BackingMaps que aloja un servidor.

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

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

objectgrid.xml

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

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

</objectGridConfig>
```

Este archivo `objectgrid.xml` especifica el archivo `entity.xml` con el atributo **`entityMetadataXMLFile`**. El valor puede ser un directorio relativo o una vía de acceso absoluta.

- **Para un directorio relativo:** especifique la ubicación relativa a la ubicación del archivo `objectgrid.xml`.
- **Para una vía de acceso absoluta:** especifique la ubicación con un formato de URL como, por ejemplo, `file://` o `http://`.

A continuación se muestra un ejemplo del archivo `entity.xml`:

entity.xml

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

Este ejemplo suponge que la clase Order tendría los campos **`orderNumber`** y **`desc`** anotados de forma similar.

Un archivo `entity.xml` equivalente sin clase sería:

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

Para obtener información sobre cómo iniciar los servidores, consulte la *Guía de administración*. Utiliza los archivos `deployment.xml` y `objectgrid.xml` para iniciar el servidor de catálogo.

Conexión con un servidor eXtreme Scale distribuido

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

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

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

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

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

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

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

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

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

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

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

Cliente y esquema del lado del servidor

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

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

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

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

Este cliente toma una entidad del lado del servidor y proyecta el subconjunto de la entidad en el objeto del cliente. Esta proyección produce ahorros de ancho de banda y memoria en el cliente ya que el cliente tiene solo la información que necesita en lugar de tener toda la información que se encuentra en la entidad del lado del servidor. Aplicaciones diferentes pueden usar sus propios objetos en lugar de forzar a todas las aplicaciones a compartir un conjunto de clases para el acceso a los datos.

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

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

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

Referencia al esquema

Si la aplicación se ejecuta en Java SE 5, la aplicación puede añadirse a los objetos utilizando anotaciones. EntityManager puede leer el esquema de las anotaciones en

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

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

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

Archivo de descriptor de metadatos de entidad

Consulte Archivo `emd.xsd` si desea más información sobre el archivo descriptor de metadatos.

Tareas relacionadas:

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Referencia relacionada:

"Agente de instrumentación de rendimiento de entidades" en la página 470
Puede mejorar el rendimiento de las entidades de acceso a campos habilitando el agente de instrumentación de WebSphere eXtreme Scale cuando se utiliza Java Development Kit (JDK) versión 1.5 o posterior.

"Definición de un esquema de entidad" en la página 169
Un ObjectGrid puede tener varios un número ilimitado de esquemas de entidades lógicas. Las entidades se definen utilizando las clases Java anotadas, XML o una combinación de XML y clases Java. Las entidades definidas se registran con un servidor eXtreme Scale y se enlazan a BackingMaps, índices y otros plug-ins.

"Métodos de devolución de llamada y escuchas de entidad" en la página 186
Se puede notificar a las aplicaciones cuando el estado de una entidad pasa de un estado a otro. Existen dos mecanismos de devolución de llamada para los sucesos de cambio de estado: los métodos de devolución de llamada de ciclo de vida que están definidos en una clase de entidad y se invocan siempre que cambia el estado de la entidad, y los escuchas de entidad, que son más generales porque el escucha de entidad se puede registrar en varias entidades.

"Ejemplos de escucha de entidad" en la página 190
Puede escribir EntityListeners según sus necesidades. A continuación se ofrecen varios scripts de ejemplo.

"Interfaz EntityTransaction" en la página 203
Puede utilizar la interfaz EntityTransaction para delimitar transacciones.

Interacción con EntityManager

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

Cómo obtener una instancia de EntityManager desde una sesión

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

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

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

Persistencia de una entidad

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

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

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

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

Importante: El ejemplo anterior no incluye ningún límite transaccional necesario como, por ejemplo, begin y commit. Consulte"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7 la guía de aprendizaje del gestor de entidades en la *Visión general del producto* para obtener más información.

Búsqueda de una entidad

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

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

Eliminación de una entidad

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

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

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

Invalidación de una entidad

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

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

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

Actualización de una entidad

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

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

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

Consultas y colas de consulta

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

Una entidad QueryQueue es una estructura de datos en forma de cola asociada con una consulta de entidad. Selecciona todas las entidades que coinciden con la condición WHERE en el filtro de la consulta y coloca las entidades resultantes en una cola. Los clientes puede recuperar de manera iterativa las entidades de esta cola. Si desea más información, consulte “Colas de consulta de entidades” en la página 198.

Tareas relacionadas:

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Referencia relacionada:

"Agente de instrumentación de rendimiento de entidades" en la página 470
Puede mejorar el rendimiento de las entidades de acceso a campos habilitando el agente de instrumentación de WebSphere eXtreme Scale cuando se utiliza Java Development Kit (JDK) versión 1.5 o posterior.

"Definición de un esquema de entidad" en la página 169
Un ObjectGrid puede tener varios un número ilimitado de esquemas de entidades lógicas. Las entidades se definen utilizando las clases Java anotadas, XML o una combinación de XML y clases Java. Las entidades definidas se registran con un servidor eXtreme Scale y se enlazan a BackingMaps, índices y otros plug-ins.

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

"Ejemplos de escucha de entidad" en la página 190
Puede escribir EntityListeners según sus necesidades. A continuación se ofrecen varios scripts de ejemplo.

"Interfaz EntityTransaction" en la página 203
Puede utilizar la interfaz EntityTransaction para delimitar transacciones.

Métodos de devolución de llamada y escuchas de entidad:

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

Ciclo de vida de una instancia de entidad

Una instancia de entidad tiene los siguientes estados:

- **New** (nuevo): instancia de entidad creada recientemente que no existe en la memoria caché de eXtreme Scale.
- **Managed** (gestionado): instancia de entidad que existe en la memoria caché de eXtreme Scale y que se recupera y persiste mediante el gestor de entidades. Para que una entidad esté en estado gestionado, ésta debe asociarse a una transacción activa.
- **Detached** (desconectado): la instancia de entidad existe en la memoria caché de eXtreme Scale, pero ya no está asociada a una transacción activa.

- **Removed** (eliminado): instancia de entidad que se elimina, o se programa para que se elimine, de la memoria caché de eXtreme Scale cuando la transacción se vacía o se confirma.
- **Invalidated** (invalidado): instancia de entidad que se invalida, o se programa para que se invalide, en la memoria caché de eXtreme Scale cuando la transacción se vacía o se confirma.

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

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

Métodos de devolución de llamada del ciclo de vida de entidad

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

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

Escuchas de entidad

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

- **Anotación**: la anotación EntityListeners puede utilizarse para indicar una o más clases de escuchas de entidad en una clase de entidad. Si se definen varios escuchas de entidad, el orden en el que se invocan lo determina el orden en el que se especifican en la anotación EntityListeners.
- **Descriptor XML de entidad**: el descriptor XML puede utilizarse como alternativa para especificar el orden de invocación de los escuchas de entidad o para alterar temporalmente el orden que se especifica en las anotaciones de metadatos.

Requisitos del método de devolución de llamada

Cualquier subconjunto o combinación de anotaciones se puede especificar en una clase de entidad o una clase de escucha. Una sola clase no puede tener más de un método de devolución de llamada de ciclo de vida para el mismo suceso de ciclo de vida. Sin embargo, se puede utilizar el mismo método para varios sucesos de devolución de llamada. La clase de escucha de entidad debe tener un constructor

sin argumentos público. Los escuchas de entidad no tienen estado. El ciclo de vida de un escucha de entidad no se especifica. eXtreme Scale no da soporte a la herencia de entidades, de modo que los métodos de devolución de llamada sólo se pueden definir en la clase de entidad, pero no en la superclase.

Firma de método de devolución de llamada

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

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

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

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

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

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

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

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

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

Semánticas del método de devolución de llamada de ciclo de vida

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

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

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

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

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

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

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

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

Métodos de devolución de llamada de ciclo de vida duplicados

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

1. **Métodos de devolución de llamada del ciclo de vida definidos en los escuchas de entidad**: los métodos de devolución de llamada de ciclo de vida que están definidos en las clases de escucha de entidad para una clase de entidad se invocan en el mismo orden que la especificación de las clases de escucha de entidad en la anotación EntityListeners o el descriptor XML.
2. **Superclase de escucha**: los métodos de devolución de llamada definidos en la superclase del escucha de entidad se invocan antes que los hijos.
3. **Métodos de ciclo de vida de entidad:**: WebSphere eXtreme Scale no soporta la herencia de entidad, así que los métodos de ciclo de vida de entidad sólo se pueden definir en la clase de entidad.

Excepciones

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

de llamada de ciclo de vida adicional después de que se genere una excepción de tiempo de ejecución.

Conceptos relacionados:

"Ajuste del rendimiento de la interfaz EntityManager" en la página 468
La interfaz EntityManager separa las aplicaciones del estado alojado en su almacén de datos de cuadrícula de servidores.

"Almacenamiento en memoria caché de objetos y sus relaciones (API EntityManager)" en la página 166
La mayoría de los productos de memoria caché utilizan las API basadas en correlaciones para almacenar los datos como pares de clave-valor. La API ObjectMap y la memoria caché dinámica de WebSphere Application Server, entre otras, utilizan este enfoque. No obstante, las API basadas en correlaciones tienen limitaciones. La API EntityManager simplifica la interacción con la cuadrícula de datos proporcionando una forma fácil de declarar un gráfico complejo de objetos relacionados e interactuar con él.

"Gestor de entidades en un entorno distribuido" en la página 178
Puede utilizar la API EntityManager con un ObjectGrid local o en un entorno distribuido de eXtreme Scale. La diferencia principal es el modo de conectarse a este entorno remoto. Después de establecer una conexión, no hay ninguna diferencia entre el uso de un objeto Session o el uso de la API EntityManager.

"Interacción con EntityManager" en la página 183
Normalmente, las aplicaciones en primer lugar obtienen una referencia de ObjectGrid y, a continuación, una Session de dicha referencia para cada hebra. Las sesiones no pueden compartirse entre hebras. Hay disponible un método adicional en el elemento Session, el método getEntityManager. Este método devuelve una referencia a un gestor de entidades para utilizar para esta hebra. La interfaz EntityManager puede sustituir las interfaces Session y ObjectMap para todas las aplicaciones. Puede utilizar estas API EntityManager si el cliente tiene acceso a las clases de entidad definidas.

"Soporte de planes de captación de EntityManager" en la página 193
Un FetchPlan es la estrategia que el gestor de entidades utiliza para recuperar los objetos asociados si la aplicación tiene que acceder a las relaciones.

"Colas de consulta de entidades" en la página 198
Las colas de consulta permiten a las aplicaciones crear una cola calificada por una consulta en el servidor o en eXtreme Scale local para una entidad. Las entidades del resultado de la consulta se almacenan en esta cola. Actualmente, la cola de consulta sólo se admite en una correlación que utilice la estrategia de bloqueo pesimista.

Tareas relacionadas:

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Ejemplos de escucha de entidad:

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

Ejemplo de EntityListeners utilizando anotaciones

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

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

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

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

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

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

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

Ejemplos de escuchas de entidad utilizando XML

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

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

```
        </entity-listener>
    </entity-listeners>
    <pre-persist method-name="checkEmployeeID" />
</entity>
```

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

Conceptos relacionados:

"Ajuste del rendimiento de la interfaz EntityManager" en la página 468
La interfaz EntityManager separa las aplicaciones del estado alojado en su almacén de datos de cuadrícula de servidores.

"Almacenamiento en memoria caché de objetos y sus relaciones (API EntityManager)" en la página 166
La mayoría de los productos de memoria caché utilizan las API basadas en correlaciones para almacenar los datos como pares de clave-valor. La API ObjectMap y la memoria caché dinámica de WebSphere Application Server, entre otras, utilizan este enfoque. No obstante, las API basadas en correlaciones tienen limitaciones. La API EntityManager simplifica la interacción con la cuadrícula de datos proporcionando una forma fácil de declarar un gráfico complejo de objetos relacionados e interactuar con él.

"Gestor de entidades en un entorno distribuido" en la página 178
Puede utilizar la API EntityManager con un ObjectGrid local o en un entorno distribuido de eXtreme Scale. La diferencia principal es el modo de conectarse a este entorno remoto. Después de establecer una conexión, no hay ninguna diferencia entre el uso de un objeto Session o el uso de la API EntityManager.

"Interacción con EntityManager" en la página 183
Normalmente, las aplicaciones en primer lugar obtienen una referencia de ObjectGrid y, a continuación, una Session de dicha referencia para cada hebra. Las sesiones no pueden compartirse entre hebras. Hay disponible un método adicional en el elemento Session, el método getEntityManager. Este método devuelve una referencia a un gestor de entidades para utilizar para esta hebra. La interfaz EntityManager puede sustituir las interfaces Session y ObjectMap para todas las aplicaciones. Puede utilizar estas API EntityManager si el cliente tiene acceso a las clases de entidad definidas.

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

"Colas de consulta de entidades" en la página 198
Las colas de consulta permiten a las aplicaciones crear una cola calificada por una consulta en el servidor o en eXtreme Scale local para una entidad. Las entidades del resultado de la consulta se almacenan en esta cola. Actualmente, la cola de consulta sólo se admite en una correlación que utilice la estrategia de bloqueo pesimista.

Tareas relacionadas:

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Soporte de planes de captación de EntityManager

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

Ejemplo

Supongamos por ejemplo que la aplicación tiene dos entidades: Department (Departamento) y Employee (Empleado). La relación entre la entidad Department y la entidad Employee es una relación bidireccional de uno a muchos: Un

departamento tiene muchos empleados y un empleado pertenece a un solo departamento. Ya que la mayor parte de las veces en que se capte la entidad Department es probable que se capten sus empleados, el tipo de captación de esta relación de uno a muchos se define como EAGER.

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

```
 @Entity
public class Department {

    @Id
    private String deptId;

    @Basic
    String deptName;

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

}
```

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

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

Plan de captación

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

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

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

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

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

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

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

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

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

Planes de captación diferentes

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

Impacto sobre una cuadrícula distribuida

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

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

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

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

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

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

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

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

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

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

- *Plan de captación con profundidad k*:

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

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

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

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

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

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

- *Plan de captación utilizado en la consulta:*

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

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

profundidad de captación máxima de 1 recuperará la entidad Department y sus entidades Employee en un solo trayecto cliente-servidor.

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

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

Consideraciones sobre el rendimiento en un entorno distribuido

De forma predeterminada, todas las relaciones navegables mediante EAGER desde la entidad raíz se recuperarán en un solo trayecto cliente-servidor. Esto puede mejorar el rendimiento si se van a utilizar todas las relaciones. Sin embargo, en ciertos escenarios de uso, no se utilizan todas las relaciones navegables mediante EAGER desde la entidad raíz, de modo que incurren tanto en una sobrecarga de tiempo de ejecución como de ancho de banda al recuperar las entidades no utilizadas.

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

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

Tareas relacionadas:

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Referencia relacionada:

"Agente de instrumentación de rendimiento de entidades" en la página 470
Puede mejorar el rendimiento de las entidades de acceso a campos habilitando el agente de instrumentación de WebSphere eXtreme Scale cuando se utiliza Java Development Kit (JDK) versión 1.5 o posterior.

"Definición de un esquema de entidad" en la página 169
Un ObjectGrid puede tener varios un número ilimitado de esquemas de entidades lógicas. Las entidades se definen utilizando las clases Java anotadas, XML o una combinación de XML y clases Java. Las entidades definidas se registran con un servidor eXtreme Scale y se enlazan a BackingMaps, índices y otros plug-ins.

"Métodos de devolución de llamada y escuchas de entidad" en la página 186
Se puede notificar a las aplicaciones cuando el estado de una entidad pasa de un estado a otro. Existen dos mecanismos de devolución de llamada para los sucesos de cambio de estado: los métodos de devolución de llamada de ciclo de vida que están definidos en una clase de entidad y se invocan siempre que cambia el estado de la entidad, y los escuchas de entidad, que son más generales porque el escucha de entidad se puede registrar en varias entidades.

"Ejemplos de escucha de entidad" en la página 190
Puede escribir EntityListeners según sus necesidades. A continuación se ofrecen varios scripts de ejemplo.

"Interfaz EntityTransaction" en la página 203
Puede utilizar la interfaz EntityTransaction para delimitar transacciones.

Colas de consulta de entidades

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

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

Ejemplo de cola de consulta

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

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

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

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

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

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

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

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

Recuperación de entidades en un bucle

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

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

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

    Task nextTask = null;

    do {
        tran.begin();
```

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

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

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

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

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

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

Colas de consulta desplegadas en todas las particiones

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

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

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

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

1. Cuando se llama un método QueryQueue.getNextEntity o QueryQueue.getNextEntities, el cliente elige un número de partición aleatorio de 0 a n-1.
2. El cliente envía una solicitud PEEK a la partición aleatoria.

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

Entidad de subconjunto y soporte de no entidad

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

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

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

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

Colisión de claves de cliente

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

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

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

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

Anomalías de cliente

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

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

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

Consulta con la cláusula ORDER BY

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

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

Si desea más información consulte “API EntityManager Query” en la página 215.

Una llamada por transacción

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

Tareas relacionadas:

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Referencia relacionada:

"Agente de instrumentación de rendimiento de entidades" en la página 470
Puede mejorar el rendimiento de las entidades de acceso a campos habilitando el agente de instrumentación de WebSphere eXtreme Scale cuando se utiliza Java Development Kit (JDK) versión 1.5 o posterior.

"Definición de un esquema de entidad" en la página 169
Un ObjectGrid puede tener varios un número ilimitado de esquemas de entidades lógicas. Las entidades se definen utilizando las clases Java anotadas, XML o una combinación de XML y clases Java. Las entidades definidas se registran con un servidor eXtreme Scale y se enlazan a BackingMaps, índices y otros plug-ins.

"Métodos de devolución de llamada y escuchas de entidad" en la página 186
Se puede notificar a las aplicaciones cuando el estado de una entidad pasa de un estado a otro. Existen dos mecanismos de devolución de llamada para los sucesos de cambio de estado: los métodos de devolución de llamada de ciclo de vida que están definidos en una clase de entidad y se invocan siempre que cambia el estado de la entidad, y los escuchas de entidad, que son más generales porque el escucha de entidad se puede registrar en varias entidades.

"Ejemplos de escucha de entidad" en la página 190
Puede escribir EntityListeners según sus necesidades. A continuación se ofrecen varios scripts de ejemplo.

"Interfaz EntityTransaction"
Puede utilizar la interfaz EntityTransaction para delimitar transacciones.

Interfaz EntityTransaction

Puede utilizar la interfaz EntityTransaction para delimitar transacciones.

Finalidad

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

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

Conceptos relacionados:

"Ajuste del rendimiento de la interfaz EntityManager" en la página 468
La interfaz EntityManager separa las aplicaciones del estado alojado en su almacén de datos de cuadrícula de servidores.

"Almacenamiento en memoria caché de objetos y sus relaciones (API EntityManager)" en la página 166
La mayoría de los productos de memoria caché utilizan las API basadas en correlaciones para almacenar los datos como pares de clave-valor. La API ObjectMap y la memoria caché dinámica de WebSphere Application Server, entre otras, utilizan este enfoque. No obstante, las API basadas en correlaciones tienen limitaciones. La API EntityManager simplifica la interacción con la cuadrícula de datos proporcionando una forma fácil de declarar un gráfico complejo de objetos relacionados e interactuar con él.

"Gestor de entidades en un entorno distribuido" en la página 178
Puede utilizar la API EntityManager con un ObjectGrid local o en un entorno distribuido de eXtreme Scale. La diferencia principal es el modo de conectarse a este entorno remoto. Después de establecer una conexión, no hay ninguna diferencia entre el uso de un objeto Session o el uso de la API EntityManager.

"Interacción con EntityManager" en la página 183
Normalmente, las aplicaciones en primer lugar obtienen una referencia de ObjectGrid y, a continuación, una Session de dicha referencia para cada hebra. Las sesiones no pueden compartirse entre hebras. Hay disponible un método adicional en el elemento Session, el método getEntityManager. Este método devuelve una referencia a un gestor de entidades para utilizar para esta hebra. La interfaz EntityManager puede sustituir las interfaces Session y ObjectMap para todas las aplicaciones. Puede utilizar estas API EntityManager si el cliente tiene acceso a las clases de entidad definidas.

"Soporte de planes de captación de EntityManager" en la página 193
Un FetchPlan es la estrategia que el gestor de entidades utiliza para recuperar los objetos asociados si la aplicación tiene que acceder a las relaciones.

"Colas de consulta de entidades" en la página 198
Las colas de consulta permiten a las aplicaciones crear una cola calificada por una consulta en el servidor o en eXtreme Scale local para una entidad. Las entidades del resultado de la consulta se almacenan en esta cola. Actualmente, la cola de consulta sólo se admite en una correlación que utilice la estrategia de bloqueo pesimista.

Tareas relacionadas:

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Recuperación de entidades y objetos (API de consulta)

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

Funciones de consulta de WebSphere eXtreme Scale

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

Este lenguaje de consulta ofrece las funciones siguientes:

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

Interfaz de consultas

Utilice la interfaz de consultas para controlar la ejecución de consultas de entidad.

Utilice el método EntityManager.createQuery(String) para crear una consulta. Puede utilizar cada una de las instancias de consulta varias veces con la instancia de EntityManager en la que se recuperó.

Cada resultado de consulta genera una entidad, donde la clave de la entidad es el ID de la fila (de tipo long) y el valor de la entidad contiene los resultados del campo de la cláusula SELECT. Puede utilizar cada resultado de consulta en posteriores consultas.

Los métodos siguientes están disponibles en la interfaz com.ibm.websphere.objectgrid.em.Query.

public ObjectMap getResultMap()

El método getResultMap ejecuta una consulta SELECT y devuelve los resultados en un objeto ObjectMap con los resultados en el orden especificado por la consulta. El objeto ObjectMap resultante es sólo válido para la transacción actual.

La clave de la correlación es el número de resultado, de tipo long, que empieza por 1. El valor de la correlación es de tipo com.ibm.websphere.projector.Tuple donde cada atributo y asociación se denomina en función de su posición ordinal dentro de la cláusula SELECT de la consulta. Utilice este método para recuperar el objeto EntityMetadata para el objeto Tuple almacenado dentro de la correlación.

El método getResultMap es el método más rápido para recuperar los datos del resultado de la consulta donde pueden existir varios resultados. Puede recuperar el nombre de la entidad resultante mediante los métodos ObjectMap.getEntityMetadata() y EntityMetadata.getName().

Ejemplo: la consulta siguiente devuelve dos filas.

```
String ql = SELECT e.name, e.id, d from Employee e join e.dept d WHERE d.number=5
 Query q = em.createQuery(ql);
 ObjectMap resultMap = q.getResultMap();
 long rowID = 1; // empieza con el índice 1
```

```
Tuple tResult = (Tuple) resultMap.get(new Long(rowID));
while(tResult != null) {
    // El primer atributo es name y tiene un nombre de atributo de 1
    // Pero tiene una posición ordinal de 0.
    String name = (String)tResult.getAttribute(0);
    Integer id = (String)tResult.getAttribute(1);

    // Dept es una asociación con un nombre 3, pero
    // una posición ordinal de 0 ya que es la primera asociación.
    // La asociación es siempre una relación de uno a uno,
    // por lo que sólo hay una clave.
    Tuple deptKey = tResult.getAssociation(0,0);
    ...
    ++rowID;
    tResult = (Tuple) resultMap.get(new Long(rowID));

}
```

public Iterator getResultIterator

El método getResultIterator ejecuta una consulta SELECT y devuelve los resultados de la consulta utilizando un Iterator donde cada resultado es un Object para una consulta de valor único, o una matriz de Object para una consulta de varios valores. Los valores del resultado Object[] se almacenan en el orden de la consulta. El objeto Iterator resultante es sólo válido para la transacción actual.

Este método es preferido para recuperar los resultados de la consulta dentro del contexto EntityManager. Puede utilizar de forma opcional el método setResultEntityName(String) para nombrar la entidad resultante, de modo que pueda utilizarse en otras consultas.

Ejemplo: la consulta siguiente devuelve dos filas.

```
String ql = SELECT e.name, e.id, e.dept from Employee e WHERE e.dept.number=5
 Query q = em.createQuery(ql);
 Iterator results = q.getResultIterator();
 while(results.hasNext()) {
     Object[] curEmp = (Object[]) results.next();
     String name = (String) curEmp[0];
     Integer id = (Integer) curEmp[1];
     Dept d = (Dept) curEmp[2];
     ...
 }
```

public Iterator getResultIterator(Class resultType)

El método getResultIterator(Class resultType) ejecuta una consulta SELECT y devuelve los resultados de la consulta utilizando una instancia de Iterator. El tipo de entidad está determinado por el parámetro resultType. El objeto Iterator resultante es sólo válido para la transacción actual.

Utilice este método cuando desee utilizar las API EntityManager para acceder a las entidades resultantes.

Ejemplo: las consulta siguiente devuelve todos los empleados de una división y el departamento al que pertenecen, ordenados por sueldo. Para imprimir los cinco empleados con el sueldo más alto y seleccionar trabajar con empleados de sólo un departamento en el mismo conjunto de trabajo, utilice este código:

```
String string_ql = "SELECT e.name, e.id, e.dept from Employee e WHERE
  e.dept.division='Manufacturing' ORDER BY e.salary DESC";
Query query1 = em.createQuery(string_ql);
query1.setResultEntityName("AllEmployees");
Iterator results1 = query1.getResultIterator(EmployeeResult.class);
int curEmployee = 0;
System.out.println("Highest paid employees");
while (results1.hasNext() && curEmployee++ < 5) {
 EmployeeResult curEmp = (EmployeeResult) results1.next();
 System.out.println(curEmp);
```

```
 // Eliminar empleado del conjunto de resultados.
 em.remove(curEmp);
}

// Vaciar los cambios en la correlación de resultados.
em.flush();

// Ejecutar una consulta en el conjunto de trabajo local sin los empleados
// eliminados
String string_q2 = "SELECT e.name, e.id, e.dept from AllEmployees e
  WHERE e.dept.name='Hardware'";
Query query2 = em.createQuery(string_q2);
Iterator results2 = query2.getResultIterator(EmployeeResult.class);
System.out.println("Subset list of Employees");
while (results2.hasNext()) {
 EmployeeResult curEmp = (EmployeeResult) results2.next();
 System.out.println(curEmp);
}
```

public Object getSingleResult

El método getSingleResult ejecuta una consulta SELECT que devuelve un único resultado.

Si la cláusula SELECT tiene más de un campo definido, el resultado es una matriz de objetos, donde cada elemento de la matriz se basa en su posición ordinal dentro de la cláusula SELECT de la consulta.

```
String ql = SELECT e from Employee e WHERE e.id=100"
 Employee e = em.createQuery(ql).getSingleResult();

 String ql = SELECT e.name, e.dept from Employee e WHERE e.id=100"
 Object[] empData = em.createQuery(ql).getSingleResult();
 String empName= (String) empData[0];
 Department empDept = (Department) empData[1];
```

public Query setResultEntityName(String entityName)

El método setResultEntityName(String entityName) especifica el nombre de la entidad del resultado de la consulta.

Cada vez que se invocan los métodos getResultIterator o getResultMap, se crea dinámicamente una entidad con ObjectMap para que contenga los resultados de la consulta. Si la entidad no se especifica, o es nula, el nombre de ObjectMap y la entidad se generan automáticamente.

Como todos los resultados de la consulta están disponibles durante una transacción, no puede volver a usarse un nombre de consulta en una única transacción.

public Query setPartition(int partitionId)

Establezca la partición a la que se direcciona la consulta.

Este método es necesario si las correlaciones de la consulta se particionan y si el gestor de entidades no tiene afinidad con una partición única de entidad raíz de esquema.

Utilice PartitionManager Interface para determinar el número de particiones para la correlación de respaldo de una entidad dada.

La siguiente tabla proporciona descripciones de otros métodos que están disponibles a través de la interfaz de la consulta.

Tabla 2. Otros métodos

Método	Resultado
public Query setMaxResults(int maxResult)	Establece el número máximo de resultados que se va a recuperar.
public Query setFirstResult(int startPosition)	Establece la posición del primer resultado que se va a recuperar.
public Query setParameter(String name, Object value)	Enlaza un argumento con un parámetro con nombre.
public Query setParameter(int position, Object value)	Enlaza un argumento con un parámetro posicional.
public Query setFlushMode(FlushModeType flushMode)	Establece el tipo de modalidad de vaciado que se va a utilizar cuando se ejecuta la consulta, que alterará temporalmente el tipo de modalidad de vaciado establecido en EntityManager.

Elementos de las consultas de eXtreme Scale

Con el motor de consultas de eXtreme Scale, puede utilizar un lenguaje de consultas para realizar búsquedas en la memoria caché de eXtreme Scale. Este lenguaje de consulta puede consultar los objetos Java que están almacenados en los objetos ObjectMap y los objetos Entity. Use la sintaxis siguiente para crear una serie de consulta.

Una consulta de eXtreme Scale es una serie que contiene los elementos siguientes:

- Una cláusula SELECT que especifica los objetos y valores que se van a devolver.
- Un cláusula FROM que nombra las colecciones de objetos.
- Una cláusula WHERE opcional que contiene predicados de búsqueda en las colecciones.
- Una cláusula GROUP BY y HAVING opcional (consulte las funciones de agregación de consultas de eXtreme Scale).
- Una cláusula ORDER BY opcional que especifica el orden de la colección de resultados.

Las colecciones de objetos Java se identifican en las consultas a través del uso del nombre en la cláusula FROM de la consulta.

Los elementos del lenguaje de consultas se describen con más detalle en los siguientes temas relacionados:

- "BNF (Backus-Naur Form) de consulta de ObjectGrid" en la página 228
- "Referencia para consultas de eXtreme Scale" en la página 219

Los temas siguientes describen los métodos para utilizar la API Query:

- "API EntityManager Query" en la página 215
- "Utilización de la API ObjectQuery" en la página 210

Consulta de datos en varios husos horarios

En un escenario distribuido, las consultas se ejecutan en los servidores. Al consultar datos con predicados de tipo calendar, java.util.Date y timestamp, el valor de fecha y hora especificado en una consulta se basa en el huso horario local del servidor.

En un sistema de un solo huso horario donde todos los clientes y servidores se ejecutan en un mismo huso horario, no es necesario tener en cuenta cuestiones relacionadas con los tipos de predicado con calendar, java.util.Date y timestamp. No obstante, cuando los clientes y los servidores están en husos horarios diferentes, el valor de fecha y hora especificado en las consultas se basa en el huso horario del servidor y puede devolver datos no deseados al cliente. Si se desconoce el huso horario del servidor, el valor de fecha y hora especificado no tiene sentido. Por lo tanto, el valor de fecha y hora especificado debe tener en cuenta la diferencia de desplazamiento de huso horario entre el huso horario de destino y el huso horario del servidor.

Desplazamiento de huso horario

Por ejemplo, supongamos que un cliente está en el huso horario [GMT-0] y el servidor está en el huso horario [GMT-6]. El huso horario del servidor está 6 horas por detrás del cliente. El cliente querría ejecutar la consulta siguiente:

```
SELECT e FROM Employee e WHERE e.birthDate='1999-12-31 06:00:00'
```

Suponiendo que la entidad Employee (empleado) tiene un atributo birthDate (fecha de nacimiento) que es del tipo java.util.Date, el cliente está en el huso horario [GMT-0] y desea recuperar empleados con un valor de birthDate como, por ejemplo '1999-12-31 06:00:00 [GMT-0]' de acuerdo con su huso horario.

La consulta se ejecutará en el servidor y el valor de birthDate utilizado por el motor de consulta será '1999-12-31 06:00:00 [GMT-6]', que equivale a '1999-12-31 12:00:00 [GMT-0]'. Los empleados con un valor de birthDate igual a '1999-12-31 12:00:00 [la GMT-0]' se devolverán al cliente. De este modo, el cliente dejará de obtener empleados deseados con un valor de birthDate de '1999-12-31 06:00:00 [GMT-0]'.

El problema descrito ocurre debido a la diferencia de huso horario entre cliente y servidor. Para solucionar este problema, un método consiste en calcular el desplazamiento de huso horario entre el cliente y el servidor y aplicar el desplazamiento de huso horario al valor de fecha y hora de destino en la consulta. En el ejemplo de consulta anterior, el desplazamiento de huso horario es de -6 horas y el predicado birthDate ajustado debe ser “birthDate='1999-12-31 00:00:00'” si el cliente tiene la intención de recuperar a empleados con el valor de birthDate '12-31 06:00:00 [GMT-0]'. Con el valor de birthDate ajustado, el servidor utilizará '1999-12-31 00:00:00 [GMT-6]', que equivale al valor de destino '12-31 06:00:00 [GMT-0]', y se devolverán al cliente los empleados necesarios.

Despliegue distribuido en varios husos horarios

Si la cuadrícula de eXtreme Scale distribuida se despliega en varios servidores ObjectGrid en varios husos horarios, el método de ajuste de desplazamiento del huso horario no funcionará porque el cliente no sabrá qué servidor ejecutará la consulta y, por lo tanto, no podrá determinar el desplazamiento de huso horario que debe utilizar. La única solución consiste en utilizar el sufijo ‘Z' (no sensible a mayúsculas y minúsculas) en la fecha JDBC y el formato de escape de hora para indicar que se utiliza el valor de fecha y hora basado en el huso horario GMT. El

sufijo ‘Z' (no sensible a mayúsculas y minúsculas) indica que se debe utilizar el valor de fecha y hora basado en el huso horario GMT. Sin el sufijo ‘Z', se utilizará el valor de fecha y hora basado en el huso horario local en el proceso que ejecuta la consulta.

La consulta siguiente equivale al ejemplo anterior, pero en su lugar utiliza el sufijo ‘Z':

```
SELECT e FROM Employee e WHERE e.birthDate='1999-12-31 06:00:00Z'
```

La consulta debe encontrar empleados con un valor de birthDate ‘1999-12-31 06:00:00'. El sufijo ‘Z' indica que el valor de birthDate especificado está basado en el huso horario GMT, de modo que el motor de consulta utilizará el valor de birthDate ‘1999-12-31 06:00:00 GMT-0]' para encontrar los valores de criterio. Los empleados con un valor de atributo birthDate igual a este valor de birthDate ‘1999-12-31 06:00:00 [GMT-0]' basado en GMT se incluirán en el resultado de la consulta. Utilizar el sufijo ‘Z' en el formato de escape de fecha y hora JDBC en cualquier consulta resulta crucial para conseguir que el huso horario de las aplicaciones resulte seguro. Sin este método, el valor de fecha y hora se basa en el huso horario del servidor y no tiene sentido desde la perspectiva del cliente cuando los clientes y los servidores están en husos horarios diferentes.

Si desea más información, consulte el tema sobre la inserción de datos para husos horarios distintos en la *Visión general del producto*

Datos para distintos husos horarios

Al insertar datos con los atributos calendar, java.util.Date y timestamp en un ObjectGrid, debe asegurarse de que estos atributos de fecha y hora se creen basándose en el mismo huso horario, sobre todo cuando se realiza el despliegue en diversos servidores en varios husos horarios. La utilización de los mismos objetos de hecha y hora basados en huso horario puede garantizar que la aplicación tenga seguridad de huso horario y que se puedan consultar los datos mediante los predicados calendar, java.util.Date y timestamp.

Sin especificar explícitamente un huso horario al crear objetos de fecha y hora, Java utiliza el huso horario local y puede causar valores de fecha y hora incoherentes en clientes y servidores.

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

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

Utilización de la API ObjectQuery

La API ObjectQuery proporciona métodos para consultar datos en el ObjectGrid que se almacenan utilizando la API ObjectMap. Cuando se define un esquema en

la instancia de ObjectGrid, la API ObjectQuery se puede utilizar para crear y ejecutar consultas sobre los objetos heterogéneos almacenados en las correlaciones de objeto.

Consultas y correlaciones de objeto

Puede utilizar una capacidad de consulta ampliada para los objetos que se han almacenado utilizando la API ObjectMap. Mediante estas consultas, puede recuperar objetos mediante atributos que no son de clave y realizar agregaciones simples, como sum, avg, min y max en todos los datos que coinciden con una consulta. Las aplicaciones pueden construir una consulta utilizando el método Session.createObjectQuery. Este método devuelve un objeto ObjectQuery que se puede interrogar para obtener los resultados de la consulta. Con el objeto Query también puede personalizar la consulta antes de ejecutarla. La consulta se ejecuta automáticamente cuando se llama a cualquier método que devuelva el resultado.

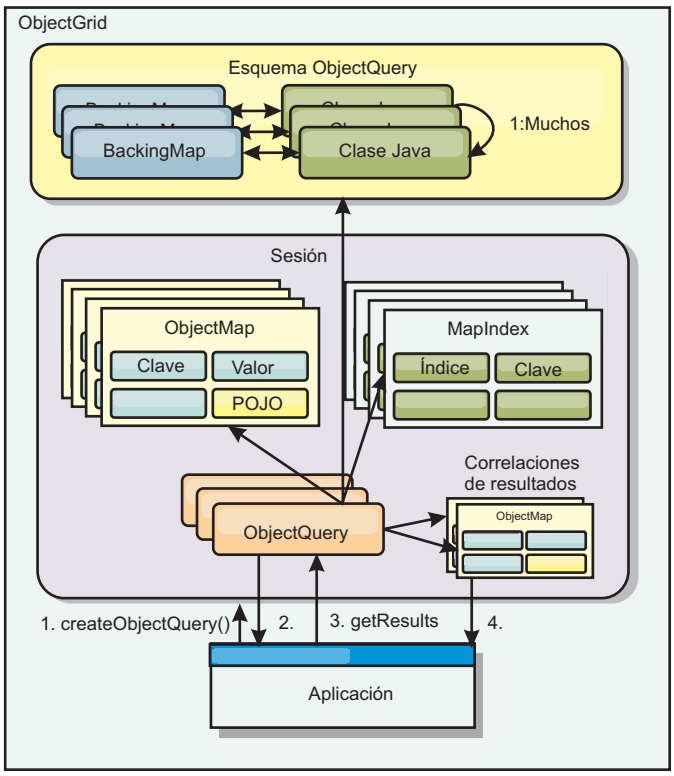


Figura 24. La interacción de la consulta con las correlaciones de objeto ObjectGrid y cómo se define un esquema para las clases y se asocia a una correlación de ObjectGrid

Definición de un esquema ObjectMap

Las correlaciones de objeto se utilizan para almacenar objetos en distintos formatos, de los que no son conscientes. Un esquema debe definirse en el objeto ObjectGrid que define el formato de los datos. Un esquema está formado por las siguientes partes:

- El tipo de objeto almacenado en ObjectMap.
- Las relaciones entre ObjectMaps.
- El método con el que cada consulta accederá a los atributos de los datos de los objetos (métodos de campos o propiedades).
- El nombre del atributo de la clave primaria del objeto.

Consulte el apartado Configuración de un esquema ObjectQuery para obtener más información.

Si desea ver un ejemplo de la creación de un esquema mediante programación o mediante el archivo XML de descriptor de ObjectGrid, consulte "Guía de aprendizaje de ObjectQuery - Paso 3" en la página 3la guía de aprendizaje del ObjectQuery en *Visión general del producto*.

Consulta de objetos con la API ObjectQuery

La interfaz ObjectQuery permite consultar objetos que no son de entidad, que son objetos heterogéneos almacenados directamente en las ObjectMaps de ObjectGrid. La API ObjectQuery proporciona una forma fácil de encontrar objetos ObjectMap sin utilizar directamente los mecanismos de palabra clave e índice.

Existen dos métodos de recuperar resultados de un objeto ObjectQuery: getResultIterator y getResultMap.

Recuperación de resultados de la consulta mediante getResultIterator

Los resultados de la consulta son básicamente una lista de atributos. Imagine que la consulta era seleccionar a,b,c de X donde y=z. Esta consulta devuelve una lista de filas que contiene a, b y c. Esta lista se almacena en una correlación con ámbito de transacciones, que significa que debe asociar una clave artificial con cada fila y utilizar un entero que aumente con cada fila. Esta correlación se obtiene mediante el método ObjectQuery.getResultMap(). Puede acceder a los elementos de cada fila con un código similar al siguiente:

```
ObjectQuery q = session.createQuery(
    "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");

  q.setParameter(1, "Claus");

  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id "
      + row[objectgrid: 0 ] + ", firstName: "
      + row[objectgrid: 1 ] + ", surname: "
      + row[objectgrid: 2 ]);
  }
```

Recuperación de resultados de la consulta mediante getResultMap

Los resultados de la consulta también se pueden recuperar mediante la correlación de resultados directamente. En el ejemplo siguiente se muestra una consulta que recupera partes específicas de clientes (Customers) coincidentes y muestra cómo acceder a las filas de resultados. Si utiliza el objeto ObjectQuery para acceder a los datos, el identificador de fila long generado no se muestra. La fila de tipo long sólo se muestra al utilizar ObjectMap para acceder al resultado.

Cuando finaliza la transacción, esta correlación desaparece. La correlación sólo está visible para la sesión utilizada, es decir, normalmente sólo para la hebra que la ha creado. La correlación utiliza una clave de tipo Long que representa el ID de la fila. Los valores almacenados en la correlación son de tipo Object u Object[], donde cada elemento coincide con el tipo de elemento de la cláusula select de la consulta.

```
ObjectQuery q = em.createQuery(
      "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
  q.setParameter(1, "Claus");
  ObjectMap qmap = q.getResultMap();
  for(long rowId = 0; true; ++rowId)
  {
    Object[] row = (Object[]) qmap.get(new Long(rowId));
    if(row == null) break;
    System.out.println(" I Found a Claus with id " + row[0]
      + ", firstName: " + row[1]
      + ", surname: " + row[2]);
  }
```

Para ver ejemplos sobre cómo utilizar el ObjectQuery, consulte "Guía de aprendizaje: Consulta de una cuadrícula de datos local en memoria" en la página 1la guía de aprendizaje de la API ObjectQuery en la *Visión general del producto*.

Configuración de un esquema ObjectQuery:

ObjectQuery se basa en información de esquema o de forma para realizar la comprobación semántica y evaluar expresiones path. En este apartado se describe cómo definir el esquema en el archivo XML o mediante programación.

Definición del esquema

El esquema ObjectMap se define en el archivo XML de descriptor de despliegue ObjectGrid o mediante programación con las técnicas normales de configuración de eXtreme Scale. Para obtener un ejemplo de cómo crear un esquema, consulte el apartado "Configuración de un esquema ObjectQuery"

La información del esquema describe los objetos POJO (plain old Java object): los atributos de los que se compone y los tipos de atributos, si los atributos son campos de clave primaria, relaciones de un valor o de varios valores o relaciones bidireccionales. La información de esquema indica a ObjectQuery que use el acceso de campos o el acceso de propiedades.

Atributos consultables

Cuando se define un esquema en ObjectGrid, se realiza una introspección en los objetos del esquema mediante el uso de un reflejo para determinar qué atributos están disponibles para realizar la consulta. Puede consultar los siguientes tipos de atributo:

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

- J2SE enum

Los tipos serializables incorporados distintos de los mencionados anteriormente también pueden incluirse en un resultado de la consulta, pero no pueden incluirse en la cláusula WHERE o FROM de la consulta. Los atributos serializables no son navegables.

Los tipos de atributo pueden excluirse del esquema si el tipo no es serializable, el campo o propiedad es estático o el campo es transitorio. Puesto que todos los objetos de correlación se deben serializar, el ObjectGrid sólo incluye atributos que se pueden persistir en el objeto. Los otros objetos se pasan por alto.

Atributos de campos

Cuando el esquema se configura para acceder al objeto mediante campos, todos los campos serializables, no transitorios se incorporan automáticamente al esquema. Para seleccionar un atributo de campo en una consulta, utilice el nombre del identificador de campo tal y como existe en la definición de clase.

Todos los campos protegidos, protegidos por paquetes, públicos y privados se incluyen en el esquema.

Atributos de propiedades

Cuando el esquema se configura para acceder al objeto mediante propiedades, todos los métodos serializables que siguen los convenios de denominación de la propiedad JavaBeans se incorporarán automáticamente en el esquema. Para seleccionar un atributo de propiedad para la consulta, utilice los convenios de denominación de propiedad del estilo JavaBeans.

Todas las propiedades protegidas, protegidas por paquetes, públicas y privadas se incluyen en el esquema.

En la clase siguiente, se han añadido al esquema estos atributos: name, birthday, valid.

```
public class Person {
  public String getName(){}
  private java.util.Date getBirthday(){}
  boolean isValid(){}
  public NonSerializableObject getData(){}
}
```

Si se utiliza CopyMode de COPY_ON_WRITE, el esquema de la consulta siempre debe utilizar el acceso basado en la propiedad. COPY_ON_WRITE crea objetos proxy siempre que los objetos se recuperen de la correlación y sólo puede acceder a dichos objetos mediante los métodos de propiedad. Si no se hace de esa manera, cada resultado de la consulta se establecerá en el valor nulo.

Relaciones

Cada relación se debe definir explícitamente en la configuración del esquema. El tipo de atributo determina automáticamente la cardinalidad de la relación. Si el atributo implementa la interfaz java.util.Collection, la relación es una relación de uno a muchos o de muchos a muchos.

A diferencia de las consultas de entidad, los atributos que se refieren a otros objetos almacenados en memoria caché no deben almacenar referencias directas al

objeto. Las referencias a otros objetos se serializan como parte de los datos del objeto que contienen. Almacene la clave para el objeto relacionado.

Por ejemplo, si hay una relación de muchos a uno entre Customer y Order:

Incorrecto.
Almacenar una referencia de objeto.

```
public class Customer {
  String customerId;
  Collection<Order> orders;
}

public class Order {
  String orderId;
  Customer customer;
}
```

Correcto. Clave para el objeto relacionado.

```
public class Customer {
  String customerId;
  Collection<String> orders;
}

public class Order {
  String orderId;
  String customer;
}
```

Cuando se ejecuta una consulta que une dos objetos de correlación, la clave se infla automáticamente. Por ejemplo, la consulta siguiente devuelve objetos Customer:

```
SELECT c FROM Order o JOIN Customer c WHERE orderId=5
```

Uso de índices

ObjectGrid utiliza plug-ins de índice para añadir índices a correlaciones. El motor de consultas incorpora automáticamente los índices definidos en un elemento de correlación de esquemas del tipo: com.ibm.websphere.objectgrid.plugins.index.HashIndex y la propiedad rangeIndex se establece en true. Si el tipo de índice no es HashIndex y la propiedad rangeIndex no se establece en true, la consulta pasa por alto el índice. Consulte “Guía de aprendizaje de ObjectQuery - Paso 2” en la página 3 la guía de aprendizaje de ObjectQuery en la *Visión general del producto* para ver un ejemplo sobre cómo añadir un índice al esquema.

API EntityManager Query

La API EntityManager proporciona métodos para consultar datos en ObjectGrid almacenados mediante la API EntityManager. La API EntityManager Query se utiliza para crear y ejecutar consultas sobre una o más entidades definidas en eXtreme Scale.

Consulta y ObjectMaps de entidades

WebSphere Extended Deployment v6.1 ha presentado y ampliado la capacidad de consulta para las entidades almacenadas en eXtreme Scale. Estas consultas permiten recuperar objetos utilizando atributos no de clave y realizar agregaciones sencillas como, por ejemplo, sum, average, minimum y maximum en todos los datos que coinciden con una consulta. Las aplicaciones construyen una consulta mediante la API EntityManager.createQuery. Ésta devuelve un objeto Query, que

puede después interrogarse para obtener los resultados de la consulta. Con el objeto Query también puede personalizar la consulta antes de ejecutarla. La consulta se ejecuta automáticamente cuando se llama a cualquier método que devuelva el resultado.

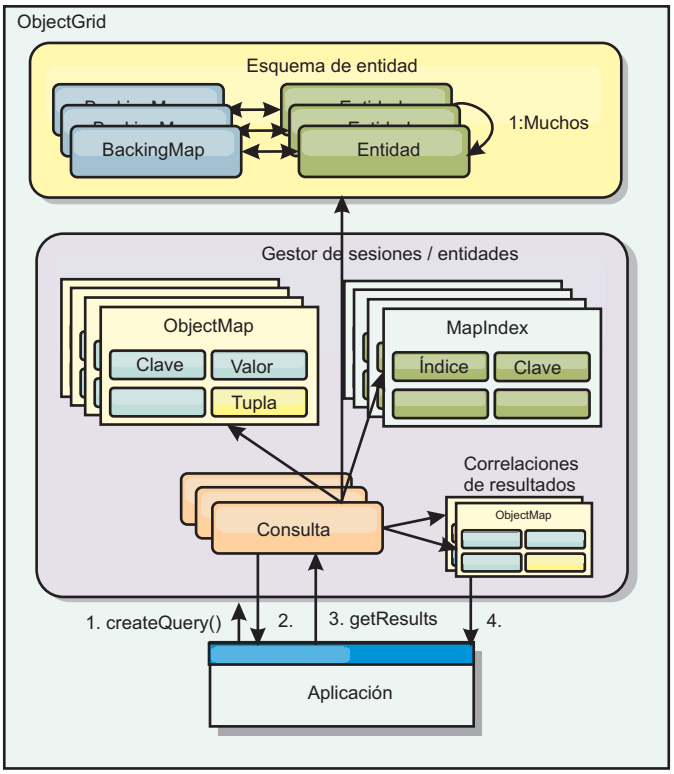


Figura 25. La interacción de la consulta con las correlaciones de objeto ObjectGrid y cómo se define y asocia el esquema de entidad con una correlación de ObjectGrid.

Recuperaciones de resultados de consulta utilizando el método getResultIterator

Los resultados de la consulta son una lista de atributos. Si la consulta era seleccionar a,b,c de X donde y=z, se devolverá una lista de filas que contengan a, b y c. Esta lista se almacena en una correlación con ámbito de transacciones, que significa que debe asociar una clave artificial con cada fila y utilizar un entero que aumente con cada fila. Esta correlación se obtiene a través del método Query.getResultMap. La correlación tiene EntityMetaData, que describe cada fila de la correlación asociada con ella. Puede acceder a los elementos de cada fila con un código similar al siguiente:

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");

  q.setParameter(1, "Claus");

  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id "  + row[objectgrid: 0 ]
      + ", firstName: " + row[objectgrid: 1 ]
      + ", surname: " + row[objectgrid: 2 ]);
  }
```

Recuperación de resultados de la consulta mediante getResultMap

El siguiente código muestra la recuperación de partes específicas de clientes (Customers) coincidentes y muestra cómo acceder a las filas de resultados. Si utiliza el objeto Query para acceder a los datos, el identificador de fila long generado no se muestra. El tipo long sólo se muestra al utilizar ObjectMap para acceder al resultado. Cuando la transacción se completa, esta Map desaparece. La correlación sólo está visible para el objeto Session utilizado, es decir, normalmente sólo para la hebra que la ha creado. La correlación utiliza el tuple para la clave con un atributo único, un tipo long con el ID de fila. El valor es otro tuple con un atributo para cada columna del conjunto de resultados.

Observe el siguiente código de ejemplo:

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from
Customer c where c.surname=?1");
q.setParameter(1, "Claus");
ObjectMap qmap = q.getResultMap();
Tuple keyTuple = qmap.getEntityMetadata().getKeyMetadata().createTuple();
for(long i = 0; true; ++i)
{
  keyTuple.setAttribute(0, new Long(i));
  Tuple row = (Tuple)qmap.get(keyTuple);
  if(row == null) break;
  System.out.println(" I Found a Claus with id "  + row.getAttribute(0)
    + ", firstName: " + row.getAttribute(1)
    + ", surname: " + row.getAttribute(2));
}
```

Recuperación de resultados de la consulta mediante un iterador de resultados de entidad

El código siguiente muestra la consulta y el bucle que recupera cada fila de resultado mediante el uso de las API de correlación normales. La clave de la correlación es un tuple. Por lo tanto, si se construye uno de los tipos correctos mediante el método createTuple, resulta en keyTuple. Intente recuperar todas las filas con los ID de fila de 0 en adelante. Cuando get devuelva null (que indica que no se ha encontrado la clave), el bucle termina. Establezca el primer atributo de keyTuple para que tenga la longitud que desea encontrar. El valor devuelto por get también es un tuple con un atributo para cada columna en el resultado de la consulta. Después, extraiga cada atributo del valor Tuple mediante getAttribute.

A continuación se muestra un fragmento de código que ilustra lo descrito:

```
Query q2 = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
q2.setResultEntityName("CustomerQueryResult");
q2.setParameter(1, "Claus");

Iterator iter2 = q2.getResultIterator(CustomerQueryResult.class);
while(iter2.hasNext())
{
  CustomerQueryResult row = (CustomerQueryResult)iter2.next();
  // firstName es el ID no el valor de firstName.
  System.out.println("Found a Claus with id " + row.id
    + ", firstName: " + row.firstName
    + ", surname: " + row.surname);
}

em.getTransaction().commit();
```

Se especificada un valor de ResultEntityName en la consulta. Este valor indica al motor de consultas que desea proyectar cada fila en un objeto específico, CustomerQueryResult en este caso. La clase es la siguiente:

```
@Entity
public class CustomerQueryResult {
 @Id long rowId;
```

```
 String id;
 String firstName;
 String surname;
};
```

En el primer fragmento de código, observe que cada fila de la consulta se devuelve como objeto CustomerQueryResult en lugar de como Object[]. Las columnas de resultado de la consulta se proyectan al objeto CustomerQueryResult. Proyectar el resultado es ligeramente más lento en el tiempo de ejecución, pero más legible. Las entidades del resultado de la consulta no se deben registrar con eXtreme Scale en el arranque. Si las entidades se registran, se crea una correlación global con el mismo nombre y la consulta falla con un error que indica un nombre de correlación duplicado.

Consultas sencillas con EntityManager:

WebSphere eXtreme Scale incluye la API de consulta EntityManager.

La API de consulta EntityManager es muy similar a otros motores de consulta SQL que realizan consultas en objetos. Se define una consulta, y el resultado se recupera de la consulta mediante diversos métodos getResult.

Los siguientes ejemplos hacen referencia a las entidades utilizadas en la guía de aprendizaje EntityManager en la visión general del producto.

Ejecución de una consulta sencilla

En este ejemplo, se realiza una consulta en los clientes con el apellido Claus:

```
em.getTransaction().begin();

  Query q = em.createQuery("select c from Customer c where c.surname='Claus'");

  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Customer c = (Customer)iter.next();
    System.out.println("Found a claus with id " + c.id);
  }

  em.getTransaction().commit();
```

Uso de parámetros

Puesto que desea buscar todos los clientes que se apelliden Claus, se utiliza un parámetro para especificar el apellido ya que puede que deba realizar esta consulta más de una vez.

Ejemplo de parámetro posicional

```
Query q = em.createQuery("select c from Customer c where c.surname=?1");
  q.setParameter(1, "Claus");
```

El uso de parámetros es muy importante si la consulta se va a utilizar más de una vez. EntityManager debe analizar la serie de consulta y crear un plan para la consulta, lo cual resulta costoso. Mediante el uso de un parámetro, EntityManager almacena en memoria caché el plan de la consulta, por lo que se reduce el tiempo que se tarda en ejecutar una consulta.

Se utilizan los parámetros posicionales y los parámetros con nombre:

Ejemplo de parámetro con nombre

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
  q.setParameter("name", "Claus");
```

Uso de un índice para mejorar el rendimiento

Si hubiera millones de clientes, la consulta anterior tendría que explorar todas las filas de la correlación de clientes. Esto no es muy eficiente. Pero eXtreme Scale proporciona un mecanismo para definir índices en atributos individuales en una entidad. La consulta utiliza de forma automática este índice cuando corresponda, lo cual acelera las consultas significativamente.

Puede especificar qué atributos indizar; el procedimiento es sencillo, basta con utilizar la anotación @Index en el atributo de la entidad:

```
@Entity
public class Customer
{
  @Id String id;
  String firstName;
  @Index String surname;
  String address;
  String phoneNumber;
}
```

EntityManager crea un índice ObjectGrid apropiado para el atributo de apellido en la entidad Customer, que el motor de consultas utiliza automáticamente. De esta manera, se reduce drásticamente el tiempo de la consulta.

Uso de paginación para mejorar el rendimiento

Si hubiera un millón de clientes con el apellido Claus, no sería útil mostrar una página con el millón de clientes. Lo más conveniente sería mostrar 10 o 25 clientes cada vez.

Con los métodos Query setFirstResult y setMaxResults se puede especificar que sólo se devuelva un subconjunto de los resultados.

Ejemplo de paginación

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
  q.setParameter("name", "Claus");
  // Mostrar la primera página
  q.setFirstResult=1;
  q.setMaxResults=25;
  displayPage(q.getResultIterator());

  // Mostrar la segunda página
  q.setFirstResult=26;
  displayPage(q.getResultIterator());
```

Referencia para consultas de eXtreme Scale

WebSphere eXtreme Scale tiene su propio lenguaje mediante el cual el usuario puede consultar datos.

Cláusula FROM de consulta de ObjectGrid

La cláusula FROM especifica las colecciones de objetos a los que se aplica la consulta. Cada colección se identifica por un nombre de esquema abstracto y una

variable de identificación, llamada variable de rango, o por una declaración de miembro de colección que identifica una relación de un solo valor o de varios valores y una variable de identificación.

Conceptualmente, la semántica de la consulta es primero formar una colección temporal de tuples, denominados R,. Los tuples están compuestos de elemento de las colecciones que se identifican en la cláusula FROM. Cada tuple contiene un elemento de cada una de las colecciones en la cláusula FROM. Se forman todas las combinaciones posibles sujetas a las limitaciones que se imponen por las declaraciones de miembros de colección. Si algún nombre de esquema identifica una colección para la que no hay ningún registro en el almacén persistente, la colección temporal R está vacía.

Ejemplos del uso de FROM

El objeto DeptBean contiene los registros 10, 20 y 30. El objeto EmpBean contiene los registros 1, 2 y 3 que están relacionados con el departamento 10 y los registros 4 y 5 que están relacionados con el departamento 20. El departamento 30 no tiene empleados asociados.

```
FROM DeptBean d, EmpBean e
```

Esta cláusula forma una colección temporal R que contiene 15 tuples.

```
FROM  DeptBean d,  DeptBean d1
```

Esta cláusula forma una colección temporal R que contiene 9 tuples.

```
FROM  DeptBean d,  IN (d.emps) AS e
```

Esta cláusula forma una colección temporal R que contiene 5 tuples. El departamento 30 no está en la colección temporal R porque no contiene ningún empleado. El departamento 10 está contenido tres veces en la colección temporal R y el departamento está contenido dos veces en R.

En lugar de usar IN(d.emps) as e, puede usar un predicado JOIN:

```
FROM DeptBean d JOIN d.emps as e
```

Después de formar la colección temporal, las condiciones de búsqueda de cláusula WHERE se aplican a la colección temporal R, lo que da una nueva colección temporal R1. Las cláusulas ORDER BY y SELECT se aplican a R1 para producir el conjunto de resultados final.

Una variable de identificación es una variable que se declara en la cláusula FROM utilizando el operador IN o el operador AS opcional.

```
FROM DeptBean AS d,  IN (d.emps) AS e
```

es equivalente a:

```
FROM DeptBean d, IN (d.emps) e
```

Una variable de identificación que se declara de modo que sea un nombre de esquema abstracto se llama variable de rango. En la consulta anterior, "d" es una variable de rango. Una variable de identificación que se declara de modo que sea

una expresión path de varios valores se llama declaración de miembro de colección. Los valores "d" y "e" en el ejemplo anterior son declaraciones de miembro de colección.

A continuación se muestra un ejemplo del uso de una expresión path de un solo valor en la cláusula FROM:

```
FROM EmpBean e, IN(e.dept.mgr) as m
```

Cláusula SELECT de consulta de ObjectGrid

La sintaxis de la cláusula SELECT se ilustra en el siguiente ejemplo:

```
SELECT { ALL | DISTINCT }  [ selection , ]*  selection
selection  ::= {single_valued_path_expression |
                identification_variable |
                OBJECT ( identification_variable) |
        aggregate_functions } [[ AS ] id ]
```

La cláusula SELECT consta de uno o más de los siguientes elementos: una sola variable de identificación que se define en la cláusula FROM, una expresión path de un solo valor que se evalúa en valores o referencias de objetos, y una función agregada. Puede usar la palabra clave DISTINCT para eliminar las referencias duplicadas.

Una subselección de escala es una subselección que devuelve un valor individual:

Ejemplos del uso de SELECT

Buscar todos los empleados que ganan más que el empleado John:

```
SELECT OBJECT(e) FROM EmpBean ej, EmpBean eWHERE  ej.name = 'John'  and
e.salary > ej.salary
```

Buscar todos los departamentos que tienen uno o más empleados que ganan menos que 20000:

```
SELECT DISTINCT e.dept  FROM EmpBean e where e.salary < 20000
```

Una consulta puede tener una expresión path que se evalúa en un valor arbitrario:

```
SELECT e.dept.name  FROM EmpBean e where e.salary < 20000
```

La consulta anterior devuelve una colección de valores de nombre para los departamentos que tienen empleados que ganan menos de 20000.

Una consulta puede devolver un valor agregado:

```
SELECT  avg(e.salary)  FROM EmpBean e
```

A continuación se muestra una consulta que recupera los nombres y referencias de objetos para los empleados con sueldo bajo:

```
SELECT e.name as name , object(e) as emp from EmpBean e where e.salary <
50000
```

Cláusula WHERE de consulta de ObjectGrid

La cláusula WHERE contiene condiciones de búsqueda que están compuestas de los elementos indicados a continuación. Cuando una condición de búsqueda se evalúa en TRUE, el tuple se añade al conjunto de resultados.

Literales de consulta de ObjectGrid

Un literal de serie se especifica en comillas simples. Una comilla simple que se encuentra dentro de un literal de serie se representa mediante dos comillas simples, por ejemplo: 'Tom''s'.

Un literal numérico puede ser cualquiera de los siguientes valores:

- Un valor exacto, como 57, -957 o +66
- Cualquier valor soportado por el tipo long de Java
- Un literal decimal como 57,5 o -47,02
- Un valor numérico aproximado como 7E3 o -57,4E-2
- Los tipos Float deben incluir el cualificador "F"; por ejemplo 1.0F
- Los tipos Long deben incluir el cualificador "L"; por ejemplo 123L

Los literales booleanos son TRUE y FALSE.

Los literales temporales siguen la sintaxis de escape JDBC en base al tipo de atributo:

- java.util.Date: aaaa-mm-ss
- java.sql.Date: aaaa-mm-ss
- java.sql.Time: hh-mm-ss
- java.sql.Timestamp: aaaa-mm-dd hh:mm:ss.f...
- java.util.Calendar: aaaa-mm-dd hh:mm:ss.f...

Los literales enum se expresan utilizando la sintaxis de literales enum de Java utilizando el nombre de clase enum plenamente calificado.

Parámetros de entrada de consulta de ObjectGrid

Puede especificar parámetros de entrada utilizando una posición ordinal o utilizando un nombre de variable. Se recomienda grabar consultas que utilicen parámetros de entrada, porque si se usan parámetros de entrada se aumentará el rendimiento permitiendo a ObjectGrid captar el plan de consulta entre acciones en ejecución.

Un parámetro de entrada puede adoptar cualquiera de los tipos siguientes: Byte, Short, Integer, Long, Float, Double, BigDecimal, BigInteger, String, Boolean, Char, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar, un Java SE 5 enum, una objeto Entity o POJO, o una serie de datos binarios con el formato de Java byte[].

Un parámetro de entrada no debe tener un valor nulo. Para buscar la aparición de un valor nulo NULL), utilice el predicado NULL.

Parámetros posicionales

Los parámetros de entrada posicionales se definen utilizando el signo de interrogación seguido de un número positivo:

```
?[entero positivo].
```

Los parámetros de entrada posicionales se enumeran empezando por 1 y corresponden a los argumentos de la consulta; por lo tanto, una consulta no puede contener un parámetro de entrada que supera el número de argumentos de entrada.

Ejemplo: `SELECT e FROM Employee e WHERE e.city = ?1 and e.salary >= ?2`

Parámetros con nombre

Los parámetros de entrada con nombre se definen utilizando un nombre de variable en el formato: :[nombre de parámetro].

Ejemplo: `SELECT e FROM Employee e WHERE e.city = :city and e.salary >= :salary`

Predicado BETWEEN de consulta de ObjectGrid

El predicado BETWEEN determina si un valor dado está comprendido entre dos otros valores dados.

```
expression [NOT]  BETWEEN expression-2  AND expression-3
```

Ejemplo 1

```
e.salary BETWEEN 50000 AND 60000
```

es equivalente a:

```
e.salary >= 50000  AND e.salary <= 60000
```

Ejemplo 2

```
e.name NOT BETWEEN 'A' AND 'B'
```

es equivalente a:

```
e.name < 'A'  OR e.name > 'B'
```

Predicado IN de consulta de ObjectGrid

El predicado IN compara un valor con un conjunto de valores. Puede utilizar el predicado IN de dos formas distintas:

```
expression [NOT] IN ( subselect )expression [NOT] IN  ( value1, value2,
.... )
```

El valor ValueN puede ser un valor literal o un parámetro de entrada. La expresión no se puede evaluar en un tipo de referencia.

Ejemplo 1

```
e.salary IN ( 10000, 15000 )
```

es equivalente a

```
( e.salary = 10000 OR  e.salary = 15000 )
```

Ejemplo 2

```
e.salary IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

es equivalente a

```
e.salary = ANY   ( select  e1.salary from EmpBean e1 where e1.dept.deptno =
10)
```

Ejemplo 3

```
e.salary NOT IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno =
10)
```

es equivalente a

```
e.salary <> ALL    ( select  e1.salary from EmpBean e1 where e1.dept.deptno
= 10)
```

Predicado LIKE de consulta de ObjectGrid

El predicado LIKE busca un valor de serie para un patrón determinado.

```
string-expression   [NOT] LIKE  pattern    [ ESCAPE escape-character ]
```

El valor del patrón es un literal de serie o un marcador de parámetro de tipo serie en el que el subrayado (_) representa un carácter individual y un signo de porcentaje (%) representa cualquier secuencia de caracteres, incluida una secuencia vacía. Cualquier otro carácter se representa a sí mismo. El carácter de escape se puede utilizar para buscar el carácter _ y %. El carácter de escape se puede especificar como literal de serie o como parámetro de entrada.

Si la expresión de serie es nula, entonces el resultado se desconoce.

Si ambas expresiones de serie y de patrón están vacías, entonces el resultado es true.

Ejemplo

```
'' LIKE '' is true
'' LIKE '%' is true
e.name LIKE '12%3' is true for '123' '12993' and false for '1234'
e.name LIKE 's_me' is true for 'some' and 'same', false for 'soome'
e.name LIKE '/_foo' escape '/' is true for '_foo', false for 'afoo'
e.name LIKE '//_foo' escape '/' is true for '/afoo' and for '/bfoo'
e.name LIKE '///_foo' escape '/' is true for '/_foo' but false for '/afoo'
```

Predicado NULL de consulta de ObjectGrid

El predicado NULL comprueba los valores nulos.

```
{single-valued-path-expression | input_parameter} IS [NOT] NULL
```

Ejemplo

```
e.name IS NULL
e.dept.name IS NOT NULL
e.dept IS NOT NULL
```

Predicado de colección EMPTY de consulta de ObjectGrid

Utilice el predicado de colección EMPTY para comprobar una colección vacía.

Para comprobar si una relación de varios valores está vacía, utilice la siguiente sintaxis:

```
collection-valued-path-expression IS [NOT]  EMPTY
```

Ejemplo

Predicado de colección vacía. Para buscar todos los departamentos que no tienen empleados:

```
SELECT OBJECT(d) FROM DeptBean d  WHERE  d.emps IS EMPTY
```

Predicado MEMBER OF de consulta de ObjectGrid

La siguiente expresión comprueba si la consulta de objeto especificada por el parámetro de entrada o la expresión path de un solo valor es miembro de la colección indicada. Si la expresión path con valor de colección designa una colección vacía, el valor de la expresión MEMBER OF es FALSE.

```
{ single-valued-path-expression | input_parameter } [ NOT ] MEMBER [ OF ]
collection-valued-path-expression
```

Ejemplo

Buscar empleados que no sean miembros de un número de departamento dado:

```
SELECT OBJECT(e) FROM EmpBean e , DeptBean d
WHERE e NOT MEMBER OF d.emps AND d.deptno = ?1
```

Buscar empleados cuyo gestor es un miembro de un número de departamento dado:

```
SELECT OBJECT(e) FROM EmpBean e, DeptBean d
WHERE e.dept.mgr MEMBER  OF d.emps  and d.deptno=?1
```

Predicado EXISTS de consulta de ObjectGrid

El predicado EXISTS comprueba si existe o no una condición especificada por una subselección.

```
EXISTS ( subselect )
```

El resultado de EXISTS es true si la subselección devuelve como mínimo un valor, de lo contrario el resultado es false.

Para negar un predicado EXISTS, debe precederlo con el operador lógico NOT.

Ejemplo

Devuelve los departamentos que tienen como mínimo un empleado que gana más de 1000000:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE EXISTS ( SELECT  e  FROM IN (d.emps) e WHERE  e.salary > 1000000 )
```

Devuelve los departamentos que no tienen empleados:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE NOT EXISTS  ( SELECT e FROM IN (d.emps) e)
```

También puede volver a escribir la consulta anterior como en el siguiente ejemplo:

```
SELECT OBJECT(d) FROM DeptBean d WHERE SIZE(d.emps)=0
```

Cláusula ORDER BY de consulta de ObjectGrid

La cláusula ORDER BY especifica una ordenación de los objetos en la colección de resultados. A continuación se muestra un ejemplo:

ORDER BY [order_element ,]* order_element order_element ::={ path-expression }[ASC | DESC]

La expresión path debe especificar un campo de valor individual que sea de un tipo primitivo de byte, short, int, long, float, double, char, o de un tipo de derivador de Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp y java.util.Calendar. El elemento de orden ASC especifica que los resultados se visualizan en orden ascendente, que es el valor predeterminado. Un elemento de orden DESC especifica que los resultados se visualicen en orden descendente.

Ejemplo

Devuelve objetos de departamento. Muestra los números de departamento en orden descendente:

```
SELECT OBJECT(d) FROM DeptBean d ORDER BY d.deptno DESC
```

Devuelve los objetos de empleado, ordenados por nombre y número de departamento:

```
SELECT OBJECT(e) FROM EmpBean e ORDER BY e.dept.deptno ASC, e.name DESC
```

Funciones de agregación de consulta de ObjectGrid

Las funciones de agregación operan en un conjunto de valores para devolver un solo valor escalar. Puede utilizar estas funciones en los métodos select y subselect. En el siguiente ejemplo se muestra una agregación:

```
SELECT SUM (e.salary) FROM EmpBean e WHERE e.dept.deptno =20
```

Esta agregación calcula el sueldo total del departamento 20.

Las funciones de agregación son: AVG, COUNT, MAX, MIN y SUM. La sintaxis de una función de agregación se muestra en el siguiente ejemplo:

```
aggregation-function ( [ ALL | DISTINCT ] expression )
```

o:

```
COUNT( [ ALL | DISTINCT ] identification-variable )
```

La opción DISTINCT elimina valores duplicados antes de aplicar la función. La opción ALL es la opción predeterminada, y no elimina valores duplicados. Los valores nulos se ignoran durante el cálculo de la función de agregación excepto cuando se utiliza la función COUNT(identification-variable), que devuelven un recuento de todos los elementos del conjunto.

Definición del tipo de retorno

Las funciones MAX y MIN pueden aplicarse a cualquier tipo de datos numérico, serie o fecha-hora y devolver el correspondiente tipo de datos. Las funciones SUM y AVG aceptan un tipo numérico como entrada. La función AVG devuelve un tipo double. La función SUM devuelve un tipo long si el tipo de entrada es un tipo de entero, excepto si la entrada es un tipo BigInteger de Java, en ese caso la función devuelve un tipo BigInteger de Java. La función SUM devuelve un tipo doble, si el tipo de entrada no es un tipo de entero, excepto si la entrada de un tipo BigDecimal de Java, en ese caso, la función devuelve un tipo BigDecimal de Java. La función COUNT puede aceptar cualquier tipo de datos excepto colecciones, y devuelve un tipo long.

Cuando se aplican a un conjunto vacío, las funciones SUM, AVG, MAX y MIN pueden devolver un valor nulo. La función COUNT devuelve cero (0) cuando se aplica a un conjunto vacío.

Utilización de cláusulas GROUP BY y HAVING

El conjunto de valores que se utiliza para la función agregada lo determina la colección que resulta de la cláusula FROM y WHERE de la consulta. Puede dividir el conjunto en grupos y aplicar la función de agregación a cada grupo. Para realizar esta acción, utilice una cláusula GROUP BY en la consulta. La cláusula GROUP BY define la agrupación de miembros, que incluyen una lista de las expresiones path. Cada expresión de vía de acceso especifica un campo que es un tipo primitivo de byte, short, int, long, float, double, boolean, char, o un tipo de derivador de Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar o un Java SE 5 enum.

El siguiente ejemplo muestra el uso de la cláusula GROUP BY en una consulta que calcula el sueldo promedio de cada departamento:

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e GROUP BY e.dept.deptno
```

Al dividir un conjunto en grupos, se considera un valor NULL igual a otro valor NULL.

Los grupos se pueden filtrar utilizando una cláusula HAVING que comprueba las propiedades del grupo antes de requerir funciones de agregación o agrupación de miembros. Este filtro es parecido a cómo la cláusula WHERE filtra tuples (es decir, los registros de los valores de colección devueltos) de la cláusula FROM. A continuación se muestra un ejemplo de la cláusula HAVING:

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e
GROUP BY e.dept.deptno
HAVING COUNT(e) > 3 AND e.dept.deptno > 5
```

Esta consulta devuelve el sueldo promedio de los departamentos que tiene más de tres empleados y el número de departamento es mayor que cinco.

Puede utilizar una cláusula HAVING sin una cláusula GROUP BY. En este caso, todo el conjunto se considera un grupo individual, al que se aplica la cláusula HAVING.

BNF (Backus-Naur Form) de consulta de ObjectGrid:

A continuación se muestra un resumen de la notación BNF (Backus-Naur Form) de consulta de ObjectGrid.

Tabla 3. Clave para el resumen de BNF

Representación	Descripción
{...}	Agrupación
[...]	Construcciones opcionales
negrita	Palabras clave
*	Cero o más
|	Alternativos

```
ObjectGrid QL ::=select_clause from_clause [where_clause]
[group_by_clause] [having_clause] [order_by_clause]

from_clause
::=FROM identification_variable_declaration [,identification_variable_declaration]*

identification_variable_declaration
::=collection_member_declaration | range_variable_declaration

collection_member_declaration
::=IN ( collection_valued_path_expression | single_valued_navigation)
[AS] identifier | [LEFT [OUTER] | INNER] JOIN collection_valued_path_expression
| single_valued_navigation [AS] identifier

range_variable_declaration
::=abstract_schema_name [AS] identifier

single_valued_path_expression
::={single_valued_navigation | identification_variable}. { state_field
| state_field.value_object_attribute } | single_valued_navigation

single_valued_navigation
::=identification_variable.[ single_valued_association_field. ]*
  single_valued_association_field

collection_valued_path_expression ::=identification_variable.[
  single_valued_association_field. ]* collection_valued_association_field

select_clause
::= SELECT [DISTINCT] [ selection , ]* selection

selection
::= {single_valued_path_expression |identification_variable | OBJECT (
identification_variable) |aggregate_functions } [[ AS ] id
]

order_by_clause ::= ORDER BY [ {identification_variable.[
single_valued_association_field.   ]*state_field} [ASC|DESC],]* {identification_variable.[
  single_valued_association_field. ]*state_field}[ASC|DESC]

where_clause
::= WHERE conditional_expression

conditional_expression
::= conditional_term | conditional_expression OR conditional_term

conditional_term
::= conditional_factor | conditional_term AND conditional_factor

conditional_factor
::= [NOT] conditional_primary

conditional_primary ::=
simple_cond_expression | (conditional_expression)
```

```
simple_cond_expression
::= comparison_expression | between_expression | like_expression |
in_expression | null_comparison_expression | empty_collection_comparison_expression
| exists_expression | collection_member_expression

between_expression ::= numeric_expression [NOT] BETWEEN numeric_expression
  AND numeric_expression | string_expression [NOT] BETWEEN
  string_expression AND string_expression | datetime_expression [NOT]
  BETWEEN datetime_expression AND datetime_expression

in_expression
::= identification_variable.[ single_valued_association_field. ]state_field
[*NOT] IN { (subselect) | ( atom ,]* atom) }

atom
::= { string_literal | numeric_literal | input_parameter }

like_expression
::=string_expression [NOT] LIKE {string_literal | input_parameter}
[ESCAPE {string_literal | input_parameter}]

null_comparison_expression
::= {single_valued_path_expression | input_parameter} IS [ NOT ] NULL

empty_collection_comparison_expression
::= collection_valued_path_expression IS [NOT] EMPTY

collection_member_expression
::={ ssingle_valued_path_expression | input_parameter }[ NOT ] MEMBER [
OF ]collection_valued_path_expression

exists_expression ::= EXISTS {(subselect)}

subselect
::= SELECT [{ ALL | DISTINCT }] subselection
from_clause [where_clause] [group_by_clause] [having_clause]

subselection
::= {single_valued_path_expression |identification_variable | aggregate_functions
}

group_by_clause ::= GROUP BY[single_valued_path_expression,]*
single_valued_path_expression

having_clause ::= HAVING conditional_expression

comparison_expression
::= numeric_ expression comparison_operator { numeric_expression |
{SOME | ANY | ALL}(subselect) } | string_expression
comparison_operator {

string_expression | {SOME | ANY | ALL}(subselect)
} |

datetime_expression comparison_operator {

datetime_expression
{SOME | ANY | ALL}(subselect) } |

boolean_expression
{=|<>} {

boolean_expression {SOME | ANY | ALL}(subselect)
} |

entity_expression {=|<>} {

entity_expression {SOME| ANY | ALL}(subselect)
}

comparison_operator ::= = | > | >= | < | <= | <>

string_expression
::= string_primary | (subselect)

string_primary ::=state_field_path_expression
|string_literal | input_parameter | functions_returning_strings

datetime_expression
::= datetime_primary |(subselect)

datetime_primary ::=state_field_path_expression
| string_literal | long_literal | input_parameter | functions_returning_datetime

boolean_expression
::= boolean_primary |(subselect)

boolean_primary ::=state_field_path_expression
| boolean_literal | input_parameter

entity_expression ::=single_valued_association_path_expression |
  identification_variable | input_parameter
```

```
numeric_expression
::= simple_numeric_expression |(subselect)

simple_numeric_expression
::= numeric_term | numeric_expression {+|-} numeric_term

numeric_term
::= numeric_factor | numeric_term {*|/} numeric_factor

numeric_factor
::= {+|-} numeric_primary

numeric_primary ::= single_valued_path_expression
| numeric_literal | ( numeric_expression ) | input_parameter | functions

aggregate_functions
:=

AVG([ALL|DISTINCT] identification_variable.[
single_valued_association_field. ]*state_field) |

COUNT([ALL|DISTINCT]
{single_valued_path_expression | identification_variable}) |

MAX([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field) |

MIN([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field) |

SUM([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field)

functions
::=

ABS (simple_numeric_expression) |

CONCAT (string_primary
, string_primary) |

LOWER (string_primary) |

LENGTH(string_primary)
|

LOCATE(string_primary, string_primary [, simple_numeric_expression])
|

MOD (simple_numeric_expression, simple_numeric_expression)
|

SIZE (collection_valued_path_expression) |

SQRT (simple_numeric_expression)
|

SUBSTRING (string_primary, simple_numeric_expression[,
simple_numeric_expression]) |

UPPER (string_primary)
|

TRIM ([[LEADING | TRAILING | BOTH]
[trim_character] FROM] string_primary)
```

Utilización de objetos que no sean claves para encontrar las particiones (interfaz PartitionableKey)

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

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

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

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

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

Programación de transacciones

Aplicaciones que requieren que las transacciones introduzcan tales consideraciones como gestión de bloqueos, gestión de colisiones y aislamiento de transacciones.

Visión general del proceso de transacciones

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

Para interactuar con los datos, la hebra de la aplicación requiere su propia sesión. Si la aplicación desea utilizar el ObjectGrid en una hebra, llame a uno de los métodos ObjectGrid.getSession para obtener una hebra. Con la sesión, la aplicación puede trabajar con los datos almacenados en las correlaciones de ObjectGrid.

Cuando una aplicación utiliza un objeto Session, la sesión debe estar en el contexto de una transacción. Una transacción empieza o se confirma y retrotrae mediante los métodos begin, commit y rollback en el objeto Session. Las aplicaciones también pueden funcionar en la modalidad de confirmación automática, en la que Session empieza automáticamente y confirma una transacción, siempre que se realiza una operación en la correlación. Una modalidad de confirmación automática no puede agrupar varias operaciones en una única transacción, de forma que es la opción más lenta si crea un proceso por lotes de varias operaciones en una única transacción. Sin embargo, para las transacciones que sólo contienen una operación, la confirmación automática es la opción más rápida.

Acceso a datos y transacciones:

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

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

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

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

La lógica del uso de transacciones

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

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

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

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

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

Por ejemplo, si la transacción ha llamado al método ObjectMap.get para el objeto en T1, obtuvo el valor en ese momento puntual. Si vuelve a llamar al método get dentro de dicha transacción en otro momento posterior T2, otra hebra podría haber cambiado el valor. Puesto que el valor ha sido modificado por otras hebra, la aplicación ve un valor distinto. Si la aplicación modifica un objeto recuperado utilizando un valor NONE de CopyMode, está cambiando la copia confirmada de dicho objeto directamente. Retrotraer la transacción no tiene sentido en esta

modalidad. Modifica la única copia de ObjectGrid. Aunque utilizar NONE CopyMode es rápido, debe ser consciente de sus consecuencias. Una aplicación que utiliza NONE CopyMode nunca debe retrotraer la transacción. Si la aplicación retrotrae la transacción, los índices no se actualizan con los cambios *y* los cambios no se duplican, si la réplica está activa. Los valores predeterminados son fáciles de utilizar y menos propensos a errores. Si inicia el rendimiento en favor de unos datos menos fiables, la aplicación necesita saber qué está haciendo para evitar problemas no deseados.

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

Interactuar con datos almacenados

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

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

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

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

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

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

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

A continuación, aparece un ejemplo para actualizar un objeto:

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

Normalmente, la aplicación utiliza el método getForUpdate en lugar de un sencillo método get para bloquear el registro. El método update debe llamarse para

proporcionar el valor actualizado a la correlación. Si no se llama este método, la correlación no se modificará. A continuación, aparece el mismo fragmento utilizando la API EntityManager:

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

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

Transacciones y particiones

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

Consultas y particiones

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

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

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

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

.

Transacciones:

Las transacciones tienen muchas ventajas para el almacenamiento de datos y la manipulación. Puede utilizar las transacciones para proteger la cuadrícula de datos de los cambios simultáneos, para aplicar varios cambios como una unidad simultánea, para replicar datos y para implementar un ciclo de vida para los bloqueos en los cambios.

Cuando se inicia una transacción, WebSphere eXtreme Scale asigna una correlación de diferencias especial para mantener los cambios o copias actuales de pares de clave y valor que la transacción utiliza. Normalmente, cuando se accede a un par de clave y valor, el valor se copia antes de que la aplicación reciba el valor. La correlación de diferencias rastrea todos los cambios para las operaciones como, por ejemplo, insert, update, get, remove, etc. Las claves no se copian porque se da por supuesto que son inmutables. Si se especifica un objeto ObjectTransformer, este objeto se utiliza para copiar el valor. Si la transacción utiliza el bloqueo optimista, también se realiza un seguimiento de las imágenes anteriores de los valores para su comparación cuando se confirma la transacción.

Si se retrotrae una transacción, se descarta la información de correlación de diferencias y se liberan los bloqueos de las entradas. Cuando se confirma una transacción, los cambios se aplican a las correlaciones y se liberan los bloqueos. Si se utiliza el bloqueo optimista, eXtreme Scale compara las versiones de imágenes anteriores de los valores con los valores incluidos en la correlación. Estos valores deben coincidir para que la transacción se confirme. Esta comparación permite un esquema de bloqueo de varias versiones, pero a costa de que se realicen dos copias cuando la transacción accede a la entrada. Se vuelven a copiar todos los valores y se almacena la nueva copia en la correlación. WebSphere eXtreme Scale realiza esta copia para evitar que la aplicación cambie la referencia de la aplicación por el valor después de una confirmación.

Puede evitar utilizar varias copias de la información. La aplicación puede guardar una copia utilizando el bloqueo pesimista en lugar del bloqueo optimista como coste de limitar la concurrencia. También se puede evitar la copia del valor durante la confirmación si la aplicación acepta no cambiar un valor después de la confirmación.

Ventajas de las transacciones

Utilice transacciones por las siguientes razones:

Mediante el uso de transacciones, puede:

- Retrotraer cambios si se produce una excepción o si la lógica empresarial necesita deshacer los cambios de estado.
- Para aplicar varios cambios como una unidad atómica durante la confirmación.
- Mantener y liberar bloqueos en los datos para aplicar varios cambios como una unidad atómica durante la confirmación.
- Proteger una hebra de los cambios simultáneos.
- Implementar un ciclo de vida para los bloqueos en cambios.
- Producir una unidad atómica de duplicación.

Tamaño de transacción

Las transacciones de mayor tamaño son más eficaces, especialmente para la réplica. Sin embargo, las transacciones de mayor tamaño pueden afectar de forma adversa a la concurrencia porque se mantienen durante más tiempo los bloqueos sobre entradas. Si utiliza transacciones de mayor tamaño, puede aumentar el rendimiento de la réplica. El aumento de este rendimiento es importante cuando se precarga una correlación. Pruebe con distintos tamaños de lotes para determinar lo que funciona mejor en cada caso.

Las transacciones de mayor tamaño también son útiles con los cargadores. Si se está utilizando un cargador que puede realizar el proceso por lotes de SQL, son posibles aumentos significativos de rendimiento en función de la transacción y las reducciones significativas de la carga en el lado de la base de datos. Esta ganancia en el rendimiento dependerá de la implementación del cargador.

Modalidad de confirmación automática

Si no se ha iniciado de forma activa ninguna transacción, cuando una aplicación interactúa con un objeto ObjectMap, empieza una operación automática de inicio y confirmación en nombre de la aplicación. Esta operación automática de inicio y confirmación funciona, pero impide que la retrotracción y el bloqueo funcionen de forma eficaz. La velocidad de réplica síncrona se ve afectado debido al tamaño de transacción muy pequeño. Si utiliza una aplicación de gestor de entidades, no utilice la modalidad de confirmación automática porque los objetos que busca el método EntityManager.find se convierten inmediatamente en no gestionados en la devolución del método y dejan de poderse utilizar.

Coordinadores de transacciones externos

Normalmente, las transacciones se inician con el método session.begin y finalizan con el método session.commit. Sin embargo, cuando se incorpora eXtreme Scale, las transacciones podrían iniciarse y terminarse a través de un coordinador de transacciones externo. Si utiliza un coordinador de transacciones externas, no tendrá que llamar al método session.begin y finalizar el método session.commit. Si utiliza WebSphere Application Server, puede utilizar el plug-in WebSphereTranscationCallback.

Atributo CopyMode:

Puede ajustar el número de copias definiendo el atributo CopyMode de los objetos BackingMap u ObjectMap en el archivo XML de descriptor de ObjectGrid.

Puede ajustar el número de copias definiendo el atributo CopyMode de los objetos BackingMap u ObjectMap. La modalidad de copia tiene los siguientes valores:

- COPY_ON_READ_AND_COMMIT
- COPY_ON_READ
- NO_COPY
- COPY_ON_WRITE
- COPY_TO_BYTES
- COPY_TO_BYTES_RAW

El valor COPY_ON_READ_AND_COMMIT es el valor predeterminado. El valor COPY_ON_READ copia los datos iniciales recuperados, pero no copia durante la confirmación. Esta modalidad es segura si la aplicación no modifica un valor después de confirmar una transacción. El valor NO_COPY no copia datos, que sólo es seguro para los datos de sólo lectura. Si los datos nunca cambian, no tendrá que copiarlos por razones de aislamiento.

Tenga cuidado cuando utilice el valor del atributo NO_COPY con las correlaciones que se pueden actualizar. WebSphere eXtreme Scale utiliza la copia en el primer toque para permitir la retrotracción de la transacción. La aplicación sólo ha cambiado la copia y, como resultado, eXtreme Scale descarta la copia. Si se utiliza el valor de atributo NO_COPY, y la aplicación modifica el valor confirmado, no es posible completar una retrotracción. Si se modifica el valor confirmado comportará

problemas con índices, réplica, etc, porque los índices y las réplicas se actualizan cuando se confirma la transacción. Si modifica los datos confirmados y, a continuación, retrotrae la transacción, que en realidad no se retrotrae, los índices no se actualizan y la réplica no tiene lugar. Otras hebras pueden ver los cambios no confirmados inmediatamente, incluso si tienen bloqueos. Utilice el valor de atributo NO_COPY para las correlaciones de sólo lectura o para aplicaciones que completan la copia apropiada antes de modificar el valor. Si utiliza el valor de atributo NO_COPY y llama al soporte de IBM con un problema de integridad de datos, se le solicitará que reproduzca el problema con la modalidad de copia establecida en COPY_ON_READ_AND_COMMIT.

El valor COPY_TO_BYTES almacena valores en la correlación de un formato serializado. En el momento de lectura, eXtreme Scale infla el valor a partir de un formato serializado y en el momento de confirmación almacena el valor en un formato serializado. Con este método, se produce una copia durante la lectura y la confirmación.

La modalidad de copia predeterminada para una correlación se puede configurar en el objeto BackingMap. También puede cambiar la modalidad de copia en las correlaciones antes de iniciar una transacción mediante el uso del método ObjectMap.setCopyMode.

A continuación, aparece un ejemplo de un fragmento de código de la correlación de respaldo de un archivo `objectgrid.xml` que muestra cómo establecer la modalidad de copia para una correlación de respaldo dada. Este ejemplo da por supuesto que utiliza `cc` como espacio de nombres de `objectgrid/config`.

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

Referencia relacionada:

Archivo XML de descriptor ObjectGrid
Para configurar WebSphere eXtreme Scale, utilice el archivo XML de descriptor de ObjectGrid y la API ObjectGrid.

Gestor de bloqueo:

Al configurar una estrategia de bloqueo, se crea un gestor de bloqueo que la correlación de respaldo mantenga la coherencia de entradas de la memoria caché.

Configuración del gestor de bloqueos

Cuando se utiliza una estrategia de bloqueo PESSIMISTIC u OPTIMISTIC, se crea un gestor de bloqueos para BackingMap. El gestor de bloqueos utiliza una correlación hash para realizar un seguimiento de las entradas bloqueadas por una o más transacciones. Cuantas más entradas de correlación existan en la correlación hash, mayor será el grupo de bloqueos con un buen rendimiento. El riesgo de las colisiones de sincronización de Java es menor a medida que crece el número de grupos. Un número mayor de grupos también implica mayor simultaneidad. Los ejemplos anteriores muestran cómo una aplicación puede establecer el número de grupos de bloqueos que se deben utilizar en una instancia determinada de BackingMap.

Para evitar una excepción java.lang.IllegalStateException, debe llamarse al método setNumberOfLockBuckets antes que a los métodos initialize o getSession en la instancia de ObjectGrid. El parámetro del método setNumberOfLockBuckets es un entero primitivo de Java que especifica el número de grupos de bloqueo para utilizar. El uso de un número primo puede permitir una distribución uniforme de

entradas de correlación en los grupos de bloqueos. Un buen punto de partida para obtener un mejor rendimiento es establecer el número de grupos de bloqueos en un 10 por ciento del número esperado de entradas de BackingMap.

Estrategias de bloqueo:

Las estrategias de bloqueo pueden ser de tipo pesimista, optimista o ninguno. Para elegir la estrategia de bloqueo, debe tener en cuenta cuestiones como el porcentaje de cada tipo de operaciones que realizará, si utilizará un cargador o no, etc.

Los bloqueos son enlazados por transacciones. Puede especificar los siguientes valores de bloqueo:

- **Sin bloqueo**: la ejecución sin el valor de bloqueo es la más rápida. Si utiliza datos de sólo lectura, es posible que no necesite el bloqueo.
- **Bloqueo pesimista**: adquiere bloqueos sobre entradas y luego mantiene los bloqueos hasta que se realiza la confirmación. Esta estrategia de bloqueo proporciona una mayor coherencia a costa del rendimiento.
- **Bloqueo optimista**: toma una imagen anterior de cada registro que toca la transacción y compara la imagen con los valores de entrada actuales cuando se confirma la transacción. Si los valores de entrada cambian, la transacción se retrotrae. No se mantiene ningún bloqueo hasta el momento de la confirmación. Esta estrategia de bloqueo proporciona una mejor concurrencia que la estrategia pesimista, con el riesgo de que la transacción se retrotraiga y el coste de memoria de realizar una copia adicional de la entrada.

Establezca la estrategia de bloqueo en la BackingMap. No puede cambiar la estrategia de bloqueo para cada transacción. A continuación, aparece un fragmento de código XML de ejemplo que muestra cómo establecer la modalidad de bloqueo en una correlación utilizando el archivo XML, que da por supuesto que `cc` es el espacio de nombres para el espacio de nombres de `objectgrid/config`:

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Bloqueo pesimista

Utilice la estrategia de bloqueo pesimista en operaciones de correlación de lectura y grabación cuando no es posible utilizar otra estrategia de bloqueo. Cuando se configura una correlación ObjectGrid para utilizar la estrategia de bloqueo pesimista, se obtiene un bloqueo de transacción pesimista para una entrada de correlación cuando una transacción obtiene por primera vez la entrada de BackingMap. El bloqueo pesimista se mantiene hasta que la aplicación completa la transacción. Por lo general, la estrategia de bloqueo pesimista se utiliza en las situaciones siguientes:

- Cuando BackingMap se configura con o sin un cargador y la información de creación de versiones no está disponible.
- Cuando BackingMap se utiliza directamente en una aplicación que necesita ayuda de eXtreme Scale para el control de simultaneidad.
- Cuando la información de creación de versiones está disponible, pero las transacciones de actualización colisionan con frecuencia en las entradas de respaldo, lo cual produce anomalías optimistas de actualización.

Como la estrategia de bloqueo pesimista tiene el mayor impacto sobre el rendimiento y la escalabilidad, esta estrategia sólo debe utilizarse para correlaciones de lectura y grabación, cuando no es viable ninguna otra estrategia de bloqueo. Por ejemplo, estas situaciones podrían incluir cuando se producen con frecuencia anomalías optimistas de actualización, o cuando es difícil para una

aplicación gestionar la recuperación de una anomalía optimista.

Bloqueo optimista

En la estrategia de bloqueo optimista, se presupone que dos transacciones no pueden intentar actualizar la misma entrada de correlación mientras se ejecutan simultáneamente. Por este motivo, la modalidad de bloqueo no necesita mantenerse para el ciclo de vida de la transacción, porque ya que es improbable que más de una transacción actualice la entrada de correlación simultáneamente. Por lo general, la estrategia de bloqueo optimista se utiliza en las situaciones siguientes:

- Cuando BackingMap se configura con o sin un cargador y la información de creación de versiones está disponible.
- Cuando BackingMap tiene mayoritariamente transacciones que realizan operaciones de lectura. En BackingMap, las operaciones insert, update o remove no se producen con frecuencia en las entradas de correlaciones.
- Cuando una correlación BackingMap se inserta, actualiza o elimina con más frecuencia de lo que se lee, pero las transacciones rara vez colisionan en la misma entrada de correlación.

Al igual que en la estrategia de bloqueo pesimista, los métodos de la interfaz ObjectMap determinan cómo eXtreme Scale intenta automáticamente adquirir una modalidad de bloqueo para una entrada de correlación a la que se accede. No obstante, las diferencias entre las estrategias optimistas y pesimistas son:

- Al igual que la estrategia de bloqueo pesimista, los métodos get y getAll adquieren una modalidad de bloqueo S cuando se invoca el método. Sin embargo, con el bloqueo optimista, la modalidad de bloqueo S no se mantiene hasta que finaliza la transacción, sino que se libera antes de que el método vuelva a la aplicación. El propósito de adquirir la modalidad de bloqueo es que eXtreme Scale pueda garantizar que sólo los datos confirmados de otras transacciones sean visibles a la transacción actual. Después de que eXtreme Scale haya comprobado que los datos se han confirmado, la modalidad de bloqueo S se libera. Durante el ciclo de confirmación, se realiza una comprobación de la creación de versiones optimista para garantizar que ninguna otra transacción haya modificado la entrada de correlación después de que la transacción actual haya liberado su modalidad de bloqueo S. Si no se capta una entrada de la correlación antes de que se actualice, invalide o suprima, el tiempo de ejecución de eXtreme Scale capta de forma implícita la entrada de la correlación. Esta operación get implícita se realiza para obtener el valor actual en el momento en que la entrada se solicitó para modificarse.
- A diferencia de la estrategia de bloqueo pesimista, los métodos getForUpdate y getAllForUpdate se manejan exactamente igual que los métodos get y getAll cuando se utiliza la estrategia de bloqueo optimista. Es decir, se adquiere una modalidad de bloqueo S al inicio del método y se libera la modalidad de bloqueo S antes de que se devuelva a la aplicación.

Todos los otros métodos ObjectMap se manejan exactamente igual que si se manejaran para la estrategia de bloqueo pesimista. Es decir, cuando se invoca el método commit, se obtiene una modalidad de bloqueo X para cualquier entrada de correlación que se inserte, actualice, elimine, manipule o invalide, y la modalidad de bloqueo X se mantiene hasta que la transacción complete el proceso de confirmación.

En la estrategia de bloqueo optimista, se presupone que ninguna transacción que se ejecute simultáneamente con otra intentará actualizar la misma entrada de

correlación. Por este motivo, la modalidad de bloqueo no necesita mantenerse durante toda la vida de la transacción ya que es improbable que más de una transacción actualice la entrada de correlación simultáneamente. Sin embargo, como no se ha mantenido la modalidad de bloqueo, otra transacción simultánea podría actualizar de forma potencial la entrada de la correlación, después de que la transacción actual haya liberado su modalidad de bloqueo S.

Para manejar esta posibilidad, eXtreme Scale obtiene un bloqueo X durante el ciclo de confirmación y realiza una comprobación de la creación de versiones optimista para verificar que ninguna otra transacción haya modificado la entrada de correlación después de que la transacción actual haya leído la entrada de correlación en BackingMap. Si otra transacción ha modificado la entrada de correlación, se produce una anomalía en la comprobación de versiones y se muestra una excepción OptimisticCollisionException. Esta excepción obliga a la transacción actual retrotraerse y la aplicación debe volver a intentar toda la transacción. La estrategia de bloqueo optimista es muy útil cuando se producen sobre todo lecturas de una correlación y rara vez se producen actualizaciones de la misma entrada de correlación.

Sin bloqueo

Cuando un objeto BackingMap se configura para que no use ninguna estrategia de bloqueo, no se obtiene ningún bloqueo de transacción para una entrada de correlación.

No usar ninguna estrategia de bloqueo es útil cuando una aplicación es un gestor de persistencia como, por ejemplo, un contenedor EJB (Enterprise JavaBeans) o cuando una aplicación utiliza Hibernate para obtener los datos persistentes. En este escenario, BackingMap se configura sin cargador y el gestor de persistencia utiliza BackingMap como memoria caché de datos. En este escenario, el gestor de persistencia proporciona control de simultaneidad entre las transacciones que están accediendo a las mismas entradas de correlación.

WebSphere eXtreme Scale no necesita obtener ningún bloqueo de transacción para el control de simultaneidad. Esta situación presupone que el gestor de persistencia no libera sus bloqueos de transacción antes de actualizar la correlación de ObjectGrid con los cambios confirmados. Si el gestor de persistencia libera sus bloqueos, debe utilizarse una estrategia de bloqueo optimista o pesimista. Por ejemplo, suponga que el gestor de persistencia de un contenedor EJB está actualizando una correlación de ObjectGrid con los datos que se confirmaron en la transacción gestionada por el contenedor EJB. Si la actualización de la correlación de ObjectGrid se produce antes de que se liberen los bloqueos de transacción del gestor de persistencia, podrá utilizar la estrategia sin bloqueos. Si la actualización de la correlación de ObjectGrid se produce después de que se liberen los bloqueos de transacción del gestor de persistencia, debe utilizar la estrategia de bloqueo optimista o pesimista.

Otro escenario en el que se puede utilizar una estrategia sin bloqueos es cuando la aplicación utiliza BackingMap directamente y se ha configurado un cargador para la correlación. En este escenario, el cargador utiliza el soporte de control de simultaneidad proporcionado por un sistema de gestión de bases de datos relacionales (RDBMS) mediante el uso de Java Database Connectivity (JDBC) o Hibernate para acceder a los datos de la base de datos relacional. La implementación de cargador puede utilizar un acercamiento optimista o pesimista. Un cargador que utiliza un bloqueo optimista o un procedimiento de creación de versiones favorece un alto nivel de simultaneidad y rendimiento. Para obtener más

información sobre cómo implementar un enfoque de bloqueo optimista, consulte la sección OptimisticCallback en la información sobre las consideraciones del cargador en la *Guía de administración*. Si utiliza un cargador que utiliza el soporte de bloqueo pesimista de una programa de fondo subyacente, es posible que desee utilizar el parámetro forUpdate que se pasa en el método get de la interfaz Loader. Establezca este parámetro en true si el método getForUpdate de la interfaz ObjectMap ha sido utilizado por la aplicación para obtener los datos. El cargador puede utilizar este parámetro para determinar si debe solicitar un bloqueo actualizable en la fila que se está leyendo. Por ejemplo, DB2 obtiene un bloqueo que se puede actualizar si una sentencia SQL select contiene una cláusula `FOR UPDATE`. Este acercamiento ofrece la misma prevención de situaciones de punto muerto que la descrita en el apartado "Bloqueo pesimista" en la página 238.

Distribución de transacciones:

Utilice JMS (Java Message Service) para los cambios de transacciones distribuidas entre las distintas capas o en entornos en plataformas combinadas.

JMS es un protocolo ideal para distribuir cambios entre distintos niveles o en entornos con diferentes plataformas. Por ejemplo, algunas aplicaciones que utilizan eXtreme Scale se podrían desplegar en IBM WebSphere Application Server Community Edition, Apache Geronimo o Apache Tomcat, mientras que otras aplicaciones se podrían ejecutar en WebSphere Application Server versión 6.x. JMS es ideal para los cambios distribuidos entre los iguales de eXtreme Scale en estos distintos entornos. El transporte de mensajes de High Availability Manager es muy rápido, pero sólo puede distribuir cambios a Máquinas virtuales Java que estén en un único grupo principal. JMS es más lento, pero permite que conjuntos más grandes y más diversos de clientes de aplicación puedan compartir un ObjectGrid. JMS es ideal si se comparten datos en un ObjectGrid entre un cliente grueso de Swing y una aplicación desplegada en WebSphere Extended Deployment.

El mecanismo de invalidación de clientes incorporado y la réplica de igual a igual son ejemplos de distribución de cambios transaccionales basados en JMS. Consulte la información sobre cómo configurar la réplica de igual a igual con JMS en la *Guía de administración* para obtener más información.

Implementación de JMS

JMS se implementa para distribuir los cambios de transacciones utilizando un objeto Java que se comporta como un ObjectGridEventListener. Este objeto puede propagar el estado de las cuatro formas siguientes:

1. Invalidación: las entradas desalojadas, actualizadas o suprimidas se eliminan de todas las Mäquinas virtuales Java de iguales al recibir el mensaje.
2. Invalidación condicional: la entrada sólo se desaloja si la versión local es la misma o más antigua que la versión del editor.
3. Envío: las entradas desalojadas, actualizadas, suprimidas o insertadas se añaden o se sobrescriben en todas las Máquinas virtuales Java de iguales al recibir el mensaje JMS.
4. Envío condicional: la entrada sólo se actualiza o se añade en el lado del receptor si la entrada local es menos reciente que la versión que se va a publicar.

Escuchar cambios de publicación

El plug-in implementa la interfaz ObjectGridEventListener para interceptar el suceso transactionEnd. Cuando eXtreme Scale invoca este método, el plug-in intenta convertir la lista LogSequence de cada correlación manipulada por la transacción a un mensaje JMS, que intentará publicar. El plug-in puede haberse configurado para publicar cambios de todas las correlaciones o de un subconjunto. Los objetos LogSequence se procesan para las correlaciones que tienen habilitada la publicación. La clase LogSequenceTransformer ObjectGrid serializa un objeto filtrado LogSequence de cada correlación en una corriente. Después de que todos los objetos LogSequences se serialicen en una corriente, se crea un objeto JMS ObjectMessage y se publica para un tema conocido.

Escuchar mensajes JMS y aplicarlos al objeto ObjectGrid local

El mismo plug-in también inicia una hebra que forma un bucle y recibe todos los mensajes publicados para un tema conocido. Cuando llega un mensaje, se pasa el contenido del mensaje a la clase LogSequenceTransformer, donde se convierte a un conjunto de objetos LogSequence. A continuación, se inicia una transacción de no escritura a través. Cada objeto LogSequence se proporciona al método Session.processLogSequence, que actualiza las correlaciones locales con los cambios. El método processLogSequence entiende la modalidad de distribución. La transacción se confirma y la memoria caché local refleja los cambios. Para obtener más información sobre cómo utilizar JMS para distribuir cambios de transacción, consulte la información sobre cómo distribuir cambios entre máquinas virtuales Java iguales en la *Guía de administración*.

Transacciones de partición única y transacciones entre cuadrículas de datos:

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

En general, las interacciones con la memoria caché se pueden categorizar como transacciones de una partición único o transacciones entre cuadrículas de datos, tal como se describe en la sección siguiente.

Transacciones de partición única

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

Además de permitir a la memoria caché realizar escaladas de forma significativa, las transacciones de partición única también maximizan la disponibilidad de la memoria caché. Cada transacción sólo depende de un sistema. Cualquiera de los

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

Transacciones entre cuadrículas de datos

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

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

Las transacciones de partición única son preferibles a las transacciones entre cuadrículas de datos para el escalado con una memoria caché de objetos distribuida con alta disponibilidad, como WebSphere eXtreme Scale. La maximización del rendimiento de estas clases de sistemas requiere el uso de técnicas distintas a las metodologías relacionales tradicionales, pero puede convertir las transacciones entre cuadrículas de datos en transacciones escalables de una partición única.

Procedimientos recomendados para crear modelos de datos escalables

Los procedimientos recomendados para crear aplicaciones escalables con productos como WebSphere eXtreme Scale incluyen dos categorías: los principios fundacionales y las sugerencias de implementación. Los principios fundacionales

son ideas principales que se deben capturar en el diseño de los propios datos. Una aplicación que no observa estos principios probablemente no realizará bien las escaladas, incluso para sus transacciones principales. Se aplican las sugerencias de implementación para las transacciones problemáticas en una aplicación bien diseñada de otra forma que observa los principios generales para los modelos de datos escalables.

Principios fundacionales

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

Duplicar en lugar de normalizar

El concepto clave para recordar sobre los productos como WebSphere eXtreme Scale es que se han diseñado para distribuir los datos entre un gran número de sistemas. Si el objetivo es completar la mayoría o todas las transacciones en una única partición, el diseño del modelo de datos debe garantizar que todos los datos que podría necesitar la transacción se encuentran en la partición. La mayoría del tiempo, la única forma de conseguir esto es duplicando los datos.

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

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

Escalabilidad en lugar de recursos

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

Medir los recursos en términos de coste, en lugar de en términos de sistema como, por ejemplo, megabytes y procesadores. Generalmente, los almacenes de datos que funcionan con datos relacionales normalizados deben estar situados en el mismo sistema. Esta ubicación necesaria significa que se debe adquirir un único gran sistema empresarial, en lugar

de varios sistemas pequeños. Con el hardware de empresa, no es raro que un sistema capaz de completar un millón de transacciones por segundo cueste muchos más que el coste combinado de 10 sistemas capaces de realizar 100.000 transacciones por segundos cada uno.

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

Con WebSphere eXtreme Scale, no es necesario concluir la aplicación para añadir capacidad. Si la empresa proyecta que se requiere un 10 por ciento más de capacidad para el próxima año, aumente el número de sistemas de la cuadrícula de datos en un 10 por ciento. Puede aumentar este porcentaje sin tiempo de inactividad de la aplicación y sin adquirir excesiva capacidad.

Evitar transformaciones de datos

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

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

Eliminar consultas no enlazadas

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

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

Definir esquema

La principal ventaja de normalizar los datos es que el sistema de la base de datos puede ocuparse de la coherencia de los datos de forma interna.

Cuando se elimina la normalización de los datos para la escalabilidad, deja de existir esta gestión automática de la coherencia de los datos. Debe implementar un modelo de datos que puede funcionar en la capa de la aplicación o como plug-in en la cuadrícula de datos distribuida para garantizar la coherencia de los datos.

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

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

Impedir la competencia

Se impiden escenarios como tener un contador global. La cuadrícula de datos no se escalará si un único registro se está utilizando un número desproporcionado de veces en comparación con los demás registros. El rendimiento de la cuadrícula de datos se limitará al rendimiento del sistema que aloja un registro determinado.

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

La lectura de un contador pasará a ser una operación entre cuadrículas de datos, pero para la mayoría, ya era ineficaz como operación entre cuadrículas de datos porque su rendimiento estaba ligado al rendimiento del sistema que alojaba el registro.

Sugerencias de implementación

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

Utilizar los índices de búsqueda inversa

Considere un modelo de datos que ha eliminado la normalización correctamente donde los registros de clientes se particionan basándose en el número del ID de cliente. Este método de particionamiento es la opción lógica porque casi todas las operaciones empresariales realizadas con el registro de clientes utilizan el número del ID del cliente. Sin embargo, una

transacción importante que no utiliza el número del ID del cliente es la transacción de inicio de sesión. Es más común tener nombres de usuario o direcciones de correo electrónico para el inicio de sesión, en lugar de números de ID de cliente.

El enfoque sencillo al escenario de inicio de sesión es utilizar una transacción entre cuadrículas de datos para buscar el registro de cliente. Tal como se ha explicado previamente, este enfoque no realiza escaladas.

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

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

Calcular en el momento de la escritura

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

Campos opcionales

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

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

Colocación de relaciones de muchos a muchos

Considere una aplicación que rastrea los productos y las tiendas en las que se venden los productos. Un único producto se vende en muchas tiendas y

una sola tienda vende muchos productos. Suponga que esta aplicación rastrea 50 tiendas grandes. Cada producto se vende en un máximo de 50 tiendas, con cada tienda que vende miles de productos.

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

En primer lugar, consulte las actualizaciones. Con la disposición A, eliminar un producto del inventario de una tienda bloquea la entidad del producto. Si la cuadrícula de datos aloja 10000 productos, solo será necesario bloquear el 1/10000 de la cuadrícula para realizar la actualización. Con la disposición B, la cuadrícula de datos solo contiene 50 tiendas, por lo que se debe bloquear el 1/50 de la cuadrícula para comletar la actualización. Así pues, aunque ambas disposiciones se podrían considerar operaciones de partición única, la disposición A se escala de forma horizontal de forma más eficaz.

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

Escalar con datos normalizados

Un uso legítimo de las transacción entre cuadrícula de datos es escalar el proceso de datos. Si una cuadrícula d e datos tiene 5 sistemas y se envía una transacción entre cuadrículas de datos que clasificar unos 100.000 registros e n cada sistema, esa transacción ordenará unos 500.000 registros. Si el sistema más lento de la cuadrícula de datos pueden realizar 10 de estas transacciones por segundo, la cuadrícula de datos puede ordenar unos 5.000.000 registros por segundo. Si los datos de la cuadrícula se doblan, cada sistema realiza una clasificación entre 200.000 registros y cada transacción realiza una clasificación entre 1.000.000 de registros. Este aumento de datos disminuye el rendimiento del sistema más lento a 5 transacciones por segundo, reducendo de esta forma el rendimiento de la cuadrícula de datos a 5 transacciones por segundo. Sin embargo, la cuadrícula de datos ordena unos 5.000.000 registros por segundo.

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

Las aplicaciones como por ejemplo un motor de búsqueda que es necesario escalar en términos de proceso de datos para alojar el tamaño creciente de Internet y el rendimiento para acomodar el crecimiento en el número de usuarios, debe crear varias cuadrículas de datos, con un turno circular de las solicitudes entre cuadrículas. Si debe escalar el rendimiento, añada sistemas y añada otra cuadrícula de datos a las solicitudes de servicio. Si

es necesario escalar el proceso de datos, añada más sistemas y mantenga constante el número de cuadrículas de datos.

Utilización de bloqueo

Los bloqueos tienen ciclos de vida y tipos de bloqueos diferentes son compatibles con otros de distintas formas. Los bloqueos deben manejarse en el orden correcto para evitar escenarios de punto muerto.

Bloqueos:

Los bloqueos tienen ciclos de vida y tipos de bloqueos diferentes son compatibles con otros de distintas formas. Los bloqueos deben manejarse en el orden correcto para evitar escenarios de punto muerto.

Bloqueos compartidos, actualizables y exclusivos

Cuando una aplicación llama a cualquier método de la interfaz ObjectMap, utiliza los métodos de búsqueda en un índice, o realiza una consulta, eXtreme Scale intenta automáticamente adquirir un bloqueo para la entrada de correlación a la que se está accediendo. WebSphere eXtreme Scale utiliza las siguientes modalidades de bloqueo basadas en el método al que llama la aplicación en la interfaz ObjectMap.

- Los métodos get y getAll en la interfaz ObjectMap, los métodos de índice y las consultas adquieren un *bloqueo S*, o modalidad de bloqueo compartido para la clave de una entrada de correlación. La duración que mantiene el bloqueo S depende del nivel de aislamiento de la transacción utilizado. Una modalidad de bloqueo S permite la simultaneidad de las transacciones que intentan adquirir una modalidad de bloqueo S o de bloqueo actualizable (bloqueo U) para la misma clave, pero bloquea otras transacciones que intenten obtener una modalidad de bloqueo exclusivo (bloqueo X) para la misma clave.
- Los métodos getForUpdate y getAllForUpdate adquieren un *bloqueo U*, o modalidad de bloqueo actualizable para la clave de una entrada de correlación. El bloqueo U se mantiene hasta que se completa la transacción. Una modalidad de bloqueo U permite la simultaneidad de las transacciones que adquieren una modalidad de bloqueo S para la misma clave, pero bloquea otras transacciones que intenten obtener una modalidad de bloqueo U o X para la misma clave.
- Los métodos put, putAll, remove, removeAll, insert, update y touch adquieren un *bloqueo X*, o modalidad de bloqueo exclusivo para la clave de una entrada de correlación. El bloqueo X se mantiene hasta que se completa la transacción. Una modalidad de bloqueo X garantiza que sólo una transacción inserte, actualice o elimine una entrada de correlación de un valor de clave dado. Un bloqueo X bloquea todas las otras transacciones que intenten adquirir una modalidad de bloqueo S, U o X para la misma clave.
- Los métodos globales invalidate e invalidateAll adquieren un bloqueo X para cada entrada de correlación que se invalida. El bloqueo X se mantiene hasta que se completa la transacción. No se adquiere ningún bloqueo para los métodos locales invalidate e invalidateAll porque ninguna de las entradas de BackingMap se invalida mediante llamadas a métodos invalidate locales.

A partir de las definiciones anteriores, es obvio que una modalidad de bloqueo S es más débil que una modalidad de bloqueo U ya que permite que se ejecuten simultáneamente más transacciones al acceder a la misma entrada de correlación. La modalidad de bloqueo U es ligeramente más restrictiva que la modalidad de bloqueo S ya que bloquea las otras transacciones que soliciten una modalidad de bloqueo U o X. La modalidad de bloqueo S sólo bloquea a las otras transacciones

que soliciten una modalidad de bloqueo X. Este pequeña diferencia es importante para evitar situaciones de punto muerto. La modalidad de bloqueo X es la más fuerte porque bloquea todas las otras transacciones que intenten obtener una modalidad de bloqueo S, U o X para la misma entrada de correlación. La modalidad de bloqueo X garantiza que sólo una transacción pueda insertar, actualizar o eliminar una entrada de correlación. De este modo, se evita que se pierdan actualizaciones cuando más de una transacción intenta actualizar la misma entrada de correlación.

En la tabla siguiente se ofrece una matriz de compatibilidad de modalidades de bloqueo que resume las modalidades de bloqueo descritas, que puede utilizar para determinar las modalidades de bloqueo que son compatibles entre sí. La fila de la matriz indica una modalidad de bloqueo que ya se ha otorgado. La columna indica la modalidad de bloqueo que solicita otra transacción. Si en la columna aparece Sí, se otorga la modalidad de bloqueo solicitada por otra transacción porque es compatible con la modalidad de bloqueo que ya se ha otorgado. Si aparece No, indica que la modalidad de bloqueo no es compatible y, por tanto, la otra transacción debe esperar a que la primera transacción libere el bloqueo.

Tabla 4. Matriz de compatibilidad de modalidad de bloqueo

Bloqueo	Tipo de bloqueo S (compartido)	Tipo de bloqueo U (actualizable)	Tipo de bloqueo X (exclusivo)	Fuerza
S (compartido)	Sí	Sí	No	más débil
U (actualizable)	Sí	No	No	normal
X (exclusivo)	No	No	No	más fuerte

Puntos muertos de bloqueo

Considere la siguiente secuencia de peticiones de modalidad de bloqueo:

1. Se otorga el bloqueo X a la transacción 1 para key1.
2. Se otorga el bloqueo X a la transacción 2 para key2.
3. La transacción 1 solicita el bloqueo X para key2. (La transacción 1 se bloquea a la espera del bloqueo en propiedad de la transacción 2).
4. La transacción 2 solicita el bloqueo X para key1. (La transacción 2 se bloquea a la espera del bloqueo en propiedad de la transacción 1).

La secuencia anterior es el ejemplo clásico de punto muerto en el que dos transacciones intentan adquirir más de un solo bloqueo, y cada una de ellas adquiere los bloqueos en un orden diferente. Para evitar esta situación de punto muerto, cada transacción debe obtener los diversos bloqueos en el mismo orden. Si se utiliza la estrategia de bloqueo OPTIMISTIC y la aplicación nunca utiliza el método flush de la interfaz ObjectMap, la transacción sólo solicita las modalidades de bloqueo durante el ciclo de confirmación. Durante este ciclo, eXtreme Scale determina las claves de las entradas de correlación que deben bloquearse y solicita las modalidades de bloqueo en la secuencia de claves (comportamiento determinista). Con este método, eXtreme Scale evita la gran mayoría de los puntos muertos clásicos. No obstante, eXtreme Scale no puede evitar todos los escenarios posibles de punto muerto. Existen unos pocos escenarios que la aplicación debe tener en cuenta. A continuación se muestran algunos de éstos para que la aplicación pueda tomar acciones preventivas.

Se da un escenario en el que eXtreme Scale puede detectar un punto muerto sin tener que esperar a que se produzca un tiempo de espera de bloqueo. Si se da este

escenario, se produce una excepción com.ibm.websphere.objectgrid.LockDeadlockException. Observe el fragmento de código siguiente:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Lynn");
// Ha sido el cumpleaños de Lynn, por lo que es 1 año mayor.
p.setAge( p.getAge() + 1 );
person.put( "Lynn", p );
sess.commit();
```

En esta situación, el novio de Lynn quiere que sea mayor de lo que lo es ahora y tanto Lynn, como su novio ejecutan esta transacción simultáneamente. En esta situación, ambas transacciones poseen una modalidad de bloqueo S en la entrada de Lynn de la correlación PERSON como resultado de la invocación del método person.get("Lynn"). Como resultado de la llamada del método person.put ("Lynn", p), ambas transacciones intentan actualizar la modalidad de bloqueo S a una modalidad de bloqueo X. Las dos transacciones se bloquean a la espera de que la otra transacción libere la modalidad de bloqueo S. Por lo tanto, se produce un punto muerto al darse una condición de espera circular entre las dos transacciones. Esta condición de espera circular se produce cuando más de una transacción intenta promover un bloqueo de una modalidad más débil a una más fuerte para la misma entrada de correlación. En este escenario, se produce una excepción LockDeadlockException en lugar de una excepción LockTimeoutException.

En el ejemplo anterior, la aplicación puede evitar la excepción LockDeadlockException si utiliza una estrategia de bloqueo optimista en lugar de la estrategia de bloqueo pesimista. El uso de una estrategia de bloqueo optimista es la solución preferida cuando básicamente se realizan lecturas en la correlación, y las actualizaciones no son frecuentes. Si debe utilizarse la estrategia de bloqueo pesimista, utilice el método getForUpdate en lugar del método get del ejemplo anterior o un nivel de aislamiento de transacción de TRANSACTION_READ_COMMITTED.

Para obtener más información, consulte el tema sobre las estrategias de bloqueo en la *Visión general del producto*.

El uso del nivel de aislamiento de la transacción TRANSACTION_READ_COMMITTED impide que se obtenga el bloqueo S adquirido por el método get hasta que se complete la transacción. Si nunca se invalida la clave en la memoria caché transaccional, se siguen garantizando las lecturas repetibles.

Consulte el tema sobre el bloqueo de la entrada de correlación en la *Guía de administración* si desea más información.

Un procedimiento alternativo para cambiar el nivel de aislamiento de la transacción es utilizar el método getForUpdate. La primera transacción que llama al método getForUpdate adquiere una modalidad de bloqueo U en lugar de un bloqueo S. Esta modalidad de bloqueo hace que se bloquee la segunda transacción al llamar al método getForUpdate porque sólo se otorga una modalidad de bloqueo U a una transacción. Puesto que la segunda transacción está bloqueada, no posee ninguna modalidad de bloqueo en la entrada de la correlación de Lynn. La primera transacción no se bloquea cuando intenta actualizar la modalidad de bloqueo U a una modalidad de bloqueo X como resultado de la llamada de método put de la primera transacción. Esta característica demuestra por qué la

modalidad de bloqueo U se llama *actualizable*. Cuando se completa la primera transacción, la segunda transacción se desbloquea y se le otorga la modalidad de bloqueo U. Cuando se utiliza una estrategia de bloqueo pesimista, una aplicación puede evitar que se produzca un escenario de punto muerto de promoción de bloqueo si utiliza el método getForUpdate en lugar del método get.

Importante: esta solución no impide que las transacciones de sólo lectura puedan leer una entrada de correlación. Las transacciones de sólo lectura llaman al método get, pero nunca llaman a los métodos put, insert, update ni remove. La simultaneidad es tan alta como cuando el utiliza el método get. La única reducción en la simultaneidad se produce cuando más de una transacción llama al método getForUpdate para la misma entrada de correlación.

Debe saber cuándo una transacción llama al método getForUpdate para más de una entrada de correlación y así garantizar que cada transacción adquiera los bloqueos U en el mismo orden. Por ejemplo, suponga que la primera transacción llama al método getForUpdate para la clave 1 y al método getForUpdate para la clave 2. Otra transacción simultánea llama al método getForUpdate para las mismas claves, pero en orden inverso. Esta secuencia dará lugar a una situación clásica de punto muerto ya que varias transacciones obtienen bloqueos en distinto orden. La aplicación debe asegurarse de que cada transacción accede a varias entradas de correlación en la secuencia de claves para garantizar que no se produzca un punto muerto. Como se obtiene el bloqueo U en el momento en el que se llama al método getForUpdate en lugar de en el ciclo de confirmación, eXtreme Scale no puede ordenar las solicitudes de bloqueo como lo hace durante el ciclo de confirmación. La aplicación debe controlar el orden de los bloqueos en este caso.

El uso del método flush de la interfaz ObjectMap antes de una confirmación presenta consideraciones de orden de bloqueos adicionales. El método flush suele utilizarse para forzar cambios realizados en la correlación para el programa de fondo a través del plug-in Loader. En esta situación, el programa de fondo utiliza su propio gestor de bloqueos para controlar la simultaneidad, de modo que la condición de espera de bloqueos y el punto muerto pueden producirse en el programa de fondo en lugar de en el gestor de bloqueos de eXtreme Scale. Observe la transacción siguiente:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
    sess.begin();
    activeTran = true;
    Person p = (IPerson)person.get("Lynn");
    p.setAge( p.getAge() + 1 );
    person.put( "Lynn", p );
    person.flush();
    ...
    p = (IPerson)person.get("Tom");
    p.setAge( p.getAge() + 1 );
    sess.commit();
    activeTran = false;
}
finally
{
    if ( activeTran ) sess.rollback();
}
```

Suponga que otra transacción también ha actualizado la persona Tom, llamó al método flush y, a continuación, actualizó la persona Lynn. Si se produjera esta situación, el intercalado de las dos transacciones provocaría una condición de punto muerto de la base de datos:

```
Se otorga el bloqueo X a la transacción 1 para "Lynn" cuando se ejecuta el
método flush.
Se otorga el bloqueo X a la transacción 2 para "Tom" cuando se ejecuta el
método flush.
La transacción 1 solicita el bloqueo X para "Tom" durante el proceso de
confirmación. (La transacción 1 se bloquea a la espera del bloqueo en propiedad
de la transacción 2). El bloqueo X solicitado por la transacción 2 para "Lynn"
durante el proceso de confirmación.
(La transacción 2 se bloquea a la espera
del bloqueo en propiedad de la transacción 1).
```

Este ejemplo demuestra que el uso del método flush puede causar un punto muerto en la base de datos en lugar de en eXtreme Scale. Este ejemplo de punto muerto puede ocurrir independientemente del tipo de estrategia de bloqueo utilizado. La aplicación debe evitar que se produzca este punto muerto al utilizar el método flush y cuando un objeto Loader se conecta a BackingMap. El ejemplo anterior también ilustra otra razón por la que eXtreme Scale tiene un mecanismo de tiempo de espera de bloqueo. Una transacción que espera un bloqueo de la base de datos podría estar esperando mientras posee un bloqueo de entrada de correlación de eXtreme Scale. En consecuencia, los problemas a nivel de base de datos pueden ocasionar tiempos de espera excesivos para una modalidad de bloqueo de eXtreme Scale y terminar en una excepción LockTimeoutException.

Tareas relacionadas:

"Resolución de problemas de puntos muertos" en la página 525
Las siguientes secciones describen algunos de los escenarios más comunes de punto muerto y algunas sugerencias para evitarlos.

Implementación de manejo de excepciones en escenarios de bloqueo:

Para evitar que los bloqueos se mantengan durante un tiempo excesivo cuando se produzcan las excepciones LockTimeoutException o LockDeadlockException, una aplicación debe captar las excepciones no esperadas y llamar al método de retrotracción cuando sucede alguna acción inesperada.

Procedimiento

1. Detecte la excepción y visualice el mensaje resultante.

   ```
   try {
   ...
   } catch (ObjectGridException oe) {
   System.out.println(oe);
   }
   ```

   Como resultado, se visualiza la siguiente excepción:

   ```
   com.ibm.websphere.objectgrid.plugins.LockDeadlockException: Mensaje
   ```

   Este mensaje representa la serie que se pasa como parámetro cuando se crea y se emite la excepción.

2. Retrotraiga la transacción después de una excepción:

   ```
   Session sess = ...;
   ObjectMap person = sess.getMap("PERSON");
   boolean activeTran = false;
   try
   {
       sess.begin();
   ```

```
    activeTran = true;
    Person p = (IPerson)person.get("Lynn");
    // Ha sido el cumpleaños de Lynn, por lo que es 1 año mayor.
    p.setAge( p.getAge() + 1 );
    person.put( "Lynn", p );
    sess.commit();
    activeTran = false;
}
finally
{
    if ( activeTran ) sess.rollback();
}
```

El bloque `finally` del fragmento de código garantiza que una transacción se retrotrae cuando se produce una excepción inesperada. No sólo maneja la excepción LockDeadlockException, sino cualquier otra excepción inesperada que pueda producirse. El bloque finally maneja el caso en el que se produce una excepción durante la invocación del método commit. Este ejemplo no es el único modo de tratar las excepciones inesperadas, y podrían darse casos en los que una aplicación desea captar algunas de las excepciones inesperadas que puedan producirse y mostrar una de las excepciones de la aplicación. Puede añadir bloques catch como desee, pero la aplicación debe garantizar que el fragmento de código no salga sin completar la transacción.

Configuración de una estrategia de bloqueo:

Puede definir una estrategia de bloqueo optimista, pesimista o sin bloqueo en cada BackingMap en la configuración de WebSphere eXtreme Scale.

Acerca de esta tarea

Cada instancia de BackingMap se puede configurar para utilizar una de las siguientes estrategias de bloqueo:

1. Modalidad de bloqueo optimista
2. Modalidad de bloqueo pesimista
3. Ninguna

La estrategia de bloqueo predeterminada es OPTIMISTIC. Utilice el bloqueo optimista cuando los datos no se modifican frecuentemente. Los bloqueos sólo se mantienen durante un tiempo breve mientras los datos se leen de la memoria caché y se copian en la transacción. Cuando la memoria caché de la transacción se sincroniza con la memoria caché principal, los objetos de la memoria caché actualizados se comprueban contra la versión original. Si la comprobación falla, la transacción se retrotrae y se produce la excepción OptimisticCollisionException.

La estrategia de bloqueo PESSIMISTIC adquiere bloqueos para las entradas de memoria caché y debe utilizarse cuando los datos se cambian con frecuencia. Cada vez que se lee una entrada de la memoria caché, se adquiere un bloqueo, que puede mantenerse condicionalmente hasta que se complete la transacción. La duración de algunos de los bloqueos pueden ajustarse mediante el uso de niveles de aislamiento para la sesión.

Si el bloqueo no es necesario porque los datos nunca se actualizan o sólo se actualizan durante períodos tranquilos, puede inhabilitar el bloqueo mediante el uso de la estrategia de bloqueo NONE. Esta estrategia es muy rápida porque no se necesita ningún gestor de bloqueos. La estrategia de bloqueo NONE es ideal en tablas de búsqueda o en correlaciones de sólo lectura.

Para obtener más información sobre las estrategias de bloqueo, consulte "Estrategias de bloqueo" en la página 238la información sobre las estrategias de bloqueo en la *Visión general del producto*.

Procedimiento

- **Configure una estrategia de bloqueo optimista**
  - Mediante programación utilizando el método setLockStrategy:

```
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("optimisticMap");
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
```

  - Utilizando el atributo lockStrategy en la :

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

- **Configure una estrategia de bloqueo pesimista**
  - Mediante programación utilizando el método setLockStrategy:

```
especifique la estrategia pesimista a través de programas
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("pessimisticMap");
bm.setLockStrategy( LockStrategy.PESSIMISTIC);
```

  - Utilizando el atributo lockStrategy en la .

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

    <objectGrids>
        <objectGrid name="test">
            <backingMap name="pessimisticMap"
                lockStrategy="PESSIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **Configure una estrategia sin bloqueo**
  - Mediante programación utilizando el método setLockStrategy:

```
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("noLockingMap");
bm.setLockStrategy( LockStrategy.NONE);
```

– Utilizando el atributo lockStrategy en la :

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

    <objectGrids>
        <objectGrid name="test">
            <backingMap name="noLockingMap"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Qué hacer a continuación

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

Configuración del valor de tiempo de espera de bloqueo:

El valor de tiempo de espera de bloqueo en una instancia de BackingMap se utiliza para garantizar que una aplicación no espera indefinidamente que se otorgue una modalidad de bloqueo debido a una condición de punto muerto que se produce por un error de la aplicación.

Antes de empezar

Para configurar el valor de tiempo de espera de bloqueo, la estrategia de bloqueo se debe establecer en OPTIMISTIC o PESSIMISTIC. Si desea más información, consulte “Configuración de una estrategia de bloqueo” en la página 254.

Acerca de esta tarea

Cuando se produce una excepción LockTimeoutException, la aplicación debe determinar si el tiempo de espera se produce porque la aplicación se ejecuta más lentamente de lo esperado, o si el tiempo de espera se ha producido debido a una condición de punto muerto. Si se ha producido una condición de punto muerto, aumentar el valor de tiempo de espera de bloqueo no elimina la excepción. Si se incrementa el valor de tiempo de espera, la excepción tarda más en producirse. No obstante, si al aumentar el valor de tiempo de espera de bloqueo se elimina la excepción, la fuente del problema era que la aplicación se estaba ejecutando más despacio de lo esperado. La aplicación en este caso debe determinar por qué el rendimiento es lento.

Para impedir que se produzcan puntos muertos, el gestor de bloqueos tiene un valor de tiempo de espera predeterminado de 15 segundos. Si se supera el límite de tiempo de espera, se produce una excepción LockTimeoutException. Si el sistema está muy cargado, es posible que el valor de tiempo de espera predeterminado haga que se produzcan excepciones LockTimeoutException cuando no existan condiciones de punto muerto. En esta situación, puede aumentar el

valor de tiempo de espera de bloqueo mediante programación o en el archivo XML de descriptor de ObjectGrid.

Procedimiento

- Configure un valor de tiempo de espera de bloqueo mediante programación en una instancia de BackingMap con el método setLockTimeout.

  El ejemplo siguiente muestra cómo establecer el valor de tiempo de espera de bloqueo para la correlación de respaldo map1 en 60 segundos:

  ```
  import com.ibm.websphere.objectgrid.BackingMap;
  import com.ibm.websphere.objectgrid.LockStrategy;
  import com.ibm.websphere.objectgrid.ObjectGrid;
  import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
  ...
  ObjectGrid og =
   ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
  BackingMap bm = og.defineMap("map1");
  bm.setLockStrategy( LockStrategy.PESSIMISTIC );
  bm.setLockTimeout( 60 );
  ```

  Para evitar una excepción java.lang.IllegalStateException, llame al método setLockStrategy y, también, al método setLockTimeout, antes de llamar a los métodos initialize o getSession en la instancia de ObjectGrid. El parámetro del método setLockTimeout es un entero primitivo Java que especifica el número de segundos que eXtreme Scale espera a que se otorgue una modalidad de bloqueo. Si una transacción espera más tiempo que el especificado en el valor de tiempo de espera de bloqueo configurado para BackingMap, se produce la excepción com.ibm.websphere.objectgrid.LockTimeoutException.

- Configure el valor de tiempo de espera de bloqueo mediante el atributo lockTimeout en el Archivo XML de descriptor ObjectGridarchivo XML de descriptor de ObjectGrid.

  ```
  <?xml version="1.0" encoding="UTF-8"?>
  <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
      xmlns="http://ibm.com/ws/objectgrid/config">
      <objectGrids>
          <objectGrid name="test">
              <backingMap name="optimisticMap"
                  lockStrategy="OPTIMISTIC"
           lockTimeout="60"/>
          </objectGrid>
      </objectGrids>
  </objectGridConfig>
  ```

- Altere temporalmente el tiempo de espera de bloqueo para una única instancia de ObjectMap. Utilice el método ObjectMap.setLockTimeout para sustituir el valor de tiempo de espera de bloqueo para una instancia específica de ObjectMap. El valor de tiempo de espera de bloqueo afecta a todas las transacciones que se inician después de haber establecido el nuevo valor de tiempo de espera. Este método puede ser útil si es posible que se produzcan o se esperen colisiones de bloqueos en transacciones de tipo select.

Bloqueos de entrada de correlación con consultas e índices:

Este tema describe cómo las API de consulta de eXtreme Scale y el plug-in de indexación MapRangeIndex interactúan con bloqueos y algunos procedimientos recomendados para aumentar la simultaneidad y reducir los puntos muertos al utilizar la estrategia de bloqueo pesimista para correlaciones.

Visión general

La API de consulta de ObjectGrid permite consultas SELECT en entidades y objetos de la memoria caché ObjectMap. Cuando se ejecuta una consulta, el motor de consultas utiliza un índice MapRangeIndex, siempre que es posible, para buscar claves coincidentes con los valores de la cláusula WHERE de la consulta o para servir de puente de las relaciones. Si no hay ningún índice disponible, el motor de consultas explorará cada entrada en una o más correlaciones para buscar las entradas correspondientes. El motor de consultas y los plug-ins de índices adquirirán bloqueos para comprobar los datos coherentes, en función de la estrategia de bloqueo, el nivel de aislamiento y el estado de la transacción.

Bloqueo con el plug-in HashIndex

El plug-in HashIndex de eXtreme Scale permite encontrar claves basadas en un único atributo almacenado en el valor de la entrada de la memoria caché. El índice almacena el valor indizado en una estructura de datos independiente de la correlación de memoria caché. El índice valida las claves respecto a las entradas de correlación antes de volver al usuario para intentar conseguir un conjunto de resultados precisos. Cuando se utiliza la estrategia de bloqueo pesimista y se usa el índice en una instancia local de ObjectMap (versus un ObjectMap de cliente/servidor), el índice adquirirá bloqueos para cada entrada coincidente. Si utiliza un bloqueo optimista o un objeto ObjectMap remoto, los bloqueos se liberan de forma inmediata.

El tipo de bloqueo que se adquiere depende del argumento forUpdate pasado al método ObjectMap.getIndex. El argumento forUpdate especifica el tipo de bloqueo que debe adquirir el índice. Si es false, se adquiere un bloqueo compartible (S) y si es true, se adquiere un bloqueo actualizable (U).

Si el bloqueo es de tipo compartible, se aplica el valor del aislamiento de la transacción de la sesión y afecta a la duración del bloqueo. Consulte el tema que trata sobre el aislamiento de transacciones para obtener más detalles sobre cómo se utiliza el aislamiento de transacciones para añadir simultaneidad a las aplicaciones.

Bloqueos compartidos con consulta

El motor de consultas de eXtreme Scale adquiere los bloqueos S cuando se necesita realizar una introspección de las entradas de la memoria caché para descubrir si cumplen los criterios del filtro de la consulta. Si se utiliza el aislamiento de transacciones de lectura repetible con bloqueo pesimista, los bloqueos compartibles S sólo se retienen en los elementos incluidos en el resultado de la consulta y se liberan en todas las entradas que no se incluyen en el resultado. Si utiliza un nivel de aislamiento de transacciones más bajo u optimista, los bloqueos S no se retienen.

Bloqueos compartidos con una consulta de cliente a servidor

Al utilizar la consulta de eXtreme Scale de un cliente, normalmente, la consulta se ejecuta en el servidor, a menos que todas las correlaciones o entidades a las que se hace referencia en la consulta son locales respecto al cliente (por ejemplo: una correlación replicada por el cliente o una entidad de resultado de consulta). Todas las consultas que se ejecutan en una transacción de lectura/grabación retendrán bloqueos S, como se ha descrito en el apartado anterior. Si la transacción no es una transacción de lectura/grabación, una sesión no se retiene en el servidor y los bloqueos S se liberan.

Una transacción de lectura/grabación sólo se direcciona a una partición primaria, y una sesión se mantiene en el servidor para la sesión de cliente. Una transacción puede promocionarse a lectura/grabación de acuerdo con las condiciones siguientes:

1. Se accede a cualquier correlación configurada para usar un bloqueo pesimista mediante los métodos de API get y getAll de ObjectMap o los métodos EntityManager.find.
2. La transacción se vacía, lo que ocasiona que se envíen actualizaciones al servidor.
3. Se accede a cualquier correlación configurada para usar un bloqueo optimista mediante los métodos ObjectMap.getForUpdate o EntityManager.findForUpdate.

Bloqueos actualizables con consulta

Los bloqueos compartidos son útiles cuando es importante la coherencia y simultaneidad. Garantizan que el valor de la entrada no cambie durante la vida de la transacción. Ninguna otra transacción podrá cambiar el valor mientras se mantengan otros bloqueos S, y sólo una transacción puede intentar actualizar la entrada. Para obtener más información sobre las modalidades de bloqueo S, U y X, consulte el tema sobre la modalidad de bloqueo pesimista.

Los bloqueos actualizables se utilizan para identificar el intento de actualizar una entrada de la memoria caché cuando se usa la estrategia de bloqueo pesimista. Permite la sincronización entre las transacciones que desean modificar una entrada de la memoria caché. Las transacciones pueden ver la entrada mediante un bloqueo S, pero otras transacciones no pueden adquirir un bloqueo U o un bloqueo X. En numerosos escenarios, es necesario adquirir un bloqueo U sin adquirir primero un bloqueo S para evitar situaciones de punto muerto. Consulte el tema sobre la modalidad de bloqueo pesimista para obtener ejemplos de situaciones comunes de punto muerto.

Las interfaces ObjectQuery y EntityManager Query proporcionan el método setForUpdate para identificar el uso previsto para el resultado de la consulta. De forma específica, el motor de consultas adquiere bloqueos U en lugar de bloqueos S para cada entrada de correlación del resultado de la consulta:

```
ObjectMap orderMap = session.getMap("Order");
ObjectQuery q = session.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
session.begin();
// Ejecutar la consulta. Cada orden tiene un bloqueo U
Iterator result = q.getResultIterator();
// Para cada orden, actualice el estado.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
    orderMap.update(o.getId(), o);
}
// Cuando se confirma,
session.commit();
```

```
Query q = em.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
emTran.begin();
// Ejecutar la consulta. Cada orden tiene un bloqueo U
Iterator result = q.getResultIterator();
// Para cada orden, actualice el estado.
```

```
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
}
tmTran.commit();
```

Cuando se habilita el atributo **setForUpdate**, la transacción se convierte automáticamente en una transacción de lectura/grabación y los bloqueos se mantienen en el servidor, como se esperaba. Si la consulta no puede utilizar índices, la correlación debe explorarse, lo cual resultará en bloqueos U temporales para las entradas de correlación que no satisfagan el resultado de la consulta, y mantendrá bloqueos U para las entradas incluidas en el resultado.

Aislamiento de transacciones

Para las transacciones, puede configurar cada configuración de correlación de respaldo con una de las tres estrategias de bloqueo: pesimista, optimista o ninguno. Si utiliza el bloqueo pesimista y optimista, eXtreme Scale utiliza bloqueos compartidos (S), actualizables (U) y exclusivos (X) para mantener la coherencia. Este comportamiento de bloqueo es más notable cuando se utiliza el bloqueo pesimista, porque los bloqueos optimistas no se conservan. Puede utilizar uno de los tres niveles de aislamiento de transacción para ajustar la semántica del bloqueo que utiliza eXtreme Scale para mantener la coherencia en cada correlación de memoria caché: lectura repetible, lectura confirmada y lectura no confirmada.

Visión general del aislamiento de transacciones

El aislamiento de transacciones define cómo los cambios realizados por una operación se vuelven visibles para otras operaciones simultáneas.

WebSphere eXtreme Scale soporta tres niveles de aislamiento de transacción con las que puede ajustar de forma adicional la semántica del bloqueo que utiliza eXtreme Scale para mantener la coherencia en cada correlación de memoria caché: lectura repetible, lectura confirmada y lectura no confirmada. El nivel de aislamiento de transacción se establece en la interfaz Session utilizando el método setTransactionIsolation. El aislamiento de transacción se puede modificar en cualquier momento durante el ciclo de vida de la sesión, si una transacción no está actualmente en progreso.

El producto aplica las distintas semánticas de aislamiento de transacción ajustando la forma en la que se solicitan y conservan los bloqueos compartidos (S). El aislamiento de transacciones no tiene ningún efecto en las correlaciones configuradas para usar las estrategias de bloqueo optimista o ningún bloqueo o cuando se adquieren bloqueos actualizables (U).

Lectura repetible con bloqueo pesimista

El nivel de aislamiento de transacción de lectura repetible es el valor predeterminado. Este nivel de aislamiento impide lecturas sucias y lecturas no repetibles, pero no impide las lecturas fantasma. Una lectura sucia es una operación de lectura que se produce en datos que han sido modificados por una transacción, pero no han sido confirmados. Una lectura no repetible se puede producir cuando los bloqueos de lectura no se adquieren al realizar una operación de lectura. Una lectura fantasma se puede producir cuando se realizan dos operaciones de lectura idénticas, pero se devuelven dos conjuntos distintos de resultados porque se ha producido una actualización de los datos entras las operaciones de lectura. El producto consigue una lectura repetible manteniendo los bloqueos S hasta que se complete la transacción que posee el bloqueo. Como un

bloqueo X no se otorga hasta haberse liberado todos los bloqueos S, todas las transacciones que contienen el bloqueo S ven el mismo valor cuando se vuelven a leer.

```
map = session.getMap("Order");
session.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session.begin();

// Se solicita y se mantiene un bloqueo S y el valor se copia en
// la memoria caché transaccional.
Order order = (Order) map.get("100");
// La entrada se desaloja de la memoria caché transaccional.
map.invalidate("100", false);

// Se vuelve a solicitar el mismo valor. Ya contiene el
// bloqueo, por lo que se recupera el mismo valor y se copia en la
// memoria caché transaccional.
Order order2 (Order) = map.get("100");

// Se liberan todos los bloqueos después de sincronizar la transacción
// con la correlación de la memoria caché.
session.commit();
```

Las lecturas fantasmas son posibles si se utilizan consultas o índices, porque los bloqueos no se adquieren para los rangos de datos, sólo para las entradas de memoria caché que coinciden con los criterios de índice o consulta. Por ejemplo:

```
session1.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session1.begin();

// Se ejecuta una consulta que selecciona un intervalo de valores.
ObjectQuery query = session1.createObjectQuery
    ("SELECT o FROM Order o WHERE o.itemName='Widget'");

// En este caso, sólo un pedido coincide con el filtro de consultas.
// El pedido tiene una clave de "100".
// El motor de consultas adquiere automáticamente un bloqueo S para el pedido
"100".
Iterator result = query.getResultIterator();

// Una segunda transacción inserta un pedido que también coincide con la consulta.
Map orderMap = session2.getMap("Order");
orderMap.insert("101", new Order("101", "Widget"));

// Cuando se vuelve a ejecutar la consulta en la transacción actual, el
// nuevo pedido es visible y devolverá los pedidos "100" y "101".
result = query.getResultIterator();

// Se liberan todos los bloqueos después de sincronizar la transacción
// con la correlación de la memoria caché.
session.commit();
```

Lectura confirmada con bloqueo pesimista

El nivel de aislamiento de la transacción de lectura confirmada se puede utilizar con eXtreme Scale, que impide las lecturas sucias, pero no impide lecturas no repetibles ni lecturas fantasma, de forma que eXtreme Scale sigue utilizando los bloqueos S para leer los datos de la correlación de memoria caché, pero libera inmediatamente los bloqueos.

```
map1 = session1.getMap("Order");
session1.setTransactionIsolation(Session.TRANSACTION_READ_COMMITTED);
session1.begin();

// Se solicita un bloqueo S pero se libera inmediatamente y el
//valor se copia en la memoria caché transaccional.
```

```
Order order = (Order) map1.get("100");

// La entrada se desaloja de la memoria caché transaccional.
map1.invalidate("100", false);

// Una segunda transacción actualiza el mismo pedido.
// Adquiere un bloqueo U, actualiza el valor y lo confirma.
// ObjectGrid adquiere correctamente el bloqueo X durante
// la confirmación puesto que la primera transacción utiliza el aislamiento
// de lectura confirmada.

Map orderMap2 = session2.getMap("Order");
session2.begin();
order2 = (Order) orderMap2.getForUpdate("100");
order2.quantity=2;
orderMap2.update("100", order2);
session2.commit();

// Se vuelve a solicitar el mismo valor. Esta ver, se desea
// actualizar el valor, pero ahora refleja
// el nuevo valor
Order order1Copy (Order) = map1.getForUpdate("100");
```

Lectura no confirmada con bloqueo pesimista

El nivel de aislamiento de la transacción de lectura no confirmada se puede utilizar con eXtreme Scale, que es un nivel que permite las lecturas sucias, las lecturas no repetibles y las lecturas fantasma.

Excepción de colisión optimista

Puede recibir una OptimisticCollisionException directamente, o recibirla con una excepción ObjectGridException.

El código siguiente es un ejemplo de cómo obtener la excepción y mostrar después su mensaje:

```
try {
...
} catch (ObjectGridException oe) {
    System.out.println(oe);
}
```

Causa de la excepción

La excepción OptimisticCollisionException se crea en una situación en la que dos clientes diferentes intentan actualizar la misma entrada de correlación prácticamente al mismo tiempo. Por ejemplo, si un cliente intenta confirmar una sesión y actualizar la entrada de correlación después de que otro cliente hay leído los datos antes de la confirmación, los datos no son correctos. La excepción se crea cuando el otro cliente intenta confirmar los datos incorrectos.

Recuperación de la clave que desencadenó la excepción

Podría resultar de utilizad, al resolver dicha excepción, recuperar la clave que corresponde a la entrada que desencadenó la excepción. La ventaja de la excepción OptimisticCollisionException es que contiene el método getKey, que devuelve el objeto que representa esa clave. El ejemplo siguiente muestra cómo recuperar e imprimir la clave al obtener OptimisticCollisionException:

```
try {
...
} catch (OptimisticCollisionException oce) {
    System.out.println(oce.getKey());
}
```

ObjectGridException ocasiona una excepción OptimisticCollisionException

La excepción OptimisticCollisionException podría ser la causa de que se muestre ObjectGridException. Si es así, puede utilizar el código siguiente para determinar el tipo de excepción e imprimir la clave. El siguiente código utiliza el método del programa de utilidad de findRootCause tal como se describe en la siguiente sección.

```
try {
...
}
catch (ObjectGridException oe) {
    Throwable Root = findRootCause( oe );
    if (Root instanceof OptimisticCollisionException) {
        OptimisticCollisionException oce = (OptimisticCollisionException)Root;
        System.out.println(oce.getKey());
    }
}
```

Técnica de manejo de excepciones general

Conocer la causa raíz de un objeto Throwable es útil para aislar el origen de un error. El ejemplo siguiente muestra cómo un manejador de excepciones utiliza un método de programa de utilidad para buscar la causa raíz de un objeto Throwable.

Ejemplo:

```
static public Throwable findRootCause( Throwable t )
{
     // Iniciar con Throwable que se produjo como raíz.
     Throwable root = t;

     // Seguir cadena de causa hasta encontrar el último objeto Throwable
     // en la cadena.
     Throwable cause = root.getCause();
     while ( cause != null )
     {
        root = cause;
        cause = root.getCause();
     }

     // Devolver el último objeto Throwable en la cadena como causa raíz.
     return root;
}
```

Ejecución de lógica empresarial paralela en la cuadrícula de datos (API de DataGrid)

La API de DataGrid API proporciona una interfaz de programación sencilla para ejecutar la lógica empresarial sobre toda la cuadrícula de datos o sobre un subconjunto de esta en paralelo con el lugar donde se encuentran los datos.

API DataGrid y particionamiento:

Con las API DataGrid, un cliente puede enviar solicitudes a una partición, un subconjunto de particiones o a todas las particiones de una cuadrícula de datos. El cliente puede especificar una lista de claves, y WebSphere eXtreme Scale determina

el conjunto de particiones que albergan las claves. La solicitud se envía, a continuación, a todas las particiones del conjunto en paralelo y el cliente espera los resultados. El cliente también puede enviar solicitudes sin especificar las claves, por lo tanto, las solicitudes se envían a todas las particiones.

Los agentes desplegados en la cuadrícula de datos no funcionan en la modalidad de cliente. Estos agentes trabajan directamente en el fragmento primario. De esta manera se obtiene un rendimiento máximo, al permitir decenas de miles o más transacciones por segundo ya que el agente trabaja con los datos a máxima velocidad de memoria. Trabajar directamente con el fragmento primario también significa que un agente sólo puede ver los datos que están dentro de dicho fragmento. Se producen así interesantes oportunidades que no podrían darse con un cliente.

Un cliente eXtreme Scale típico debe poder determinar la partición de la transacción, porque el cliente necesita direccionar la solicitud. Si un agente está directamente conectado a un fragmento, no es necesario realizar un direccionamiento. Todas las solicitudes van a ese fragmento. Como el agente está conectado directamente a un fragmento, se puede acceder a los datos de otras correlaciones del fragmento sin preocuparse por las claves de particionamiento común, etc., porque no se produce ningún direccionamiento.

Agentes DataGrid y correlaciones basadas en entidades:

Una correlación contiene objetos clave y objetos de valor. El objeto clave es un tuple generado, ya que es el objeto de valor. Por norma, un agente está provisto de los objetos clave específicos de la aplicación.

El objeto clave es un tuple generado, ya que es el objeto de valor. Por norma, un agente está provisto de los objetos clave específicos de la aplicación. Éstos serán los objetos clave que utiliza la aplicación o los tuples si se trata de una correlación de entidades. Una aplicación que utiliza las entidades preferirá no trata directamente con los Tuples y preferirá trabajar con los objetos Java correlacionados con la entidad.

Por lo tanto, una clase de agente puede implementar la interfaz EntityAgentMixin. Esto obliga a la clase a implementar otro método más, getClassForEntity(). Éste devuelve la clase de entidad que debe usarse con el agente en el servidor. Las claves se convierten a esta entidad antes de invocar los métodos de proceso y reducción.

Se trata de una semántica distinta a la de un agente no EntityAgentMixin en la que dichos métodos se proporcionan sólo con las claves. Un agente que implementa EntityAgentMixin recibe el objeto Entity que incluye claves y valores en un objeto.

Nota: si la entidad no existe en el servidor, las claves son el formato tuple sin formato de la clave en lugar de la entidad gestionada.

Ejemplo de la API de DataGrid:

Las API de DataGrid admiten dos patrones de programación de cuadrícula comunes: correlación paralela y reducción paralela.

Correlación paralela

La correlación paralela permite que las entradas de un conjunto de claves se procesen y devuelve un resultado para cada entrada procesada. La aplicación efectúa un listado de claves y recibe una correlación de pares de clave/resultado después de invocar la operación de correlación. El resultado es la consecuencia de aplicar una función a la entrada de cada clave. La función la suministra la aplicación.

Flujo de llamadas MapGridAgent

Cuando se invoca el método AgentManager.callMapAgent con una colección de claves, la instancia de MapGridAgent se serializa y se envía a cada partición primaria en la que se resuelven las claves. Esto significa que los datos de la instancia almacenados en el agente pueden enviarse al servidor. Por consiguiente, cada partición primaria tiene una instancia del agente. El método de proceso se invoca para cada instancia una vez por cada clave que se resuelve en la partición. El resultado de cada método de proceso se vuelve a serializar en el cliente y se devuelve al llamante en una instancia de correlación, donde el resultado se representa como el valor en la correlación.

Cuando se invoca el método AgentManager.callMapAgent sin una colección de claves, la instancia de MapGridAgent se serializa y se envía a todas las particiones primarias. Esto significa que los datos de la instancia almacenados en el agente pueden enviarse al servidor. Por consiguiente, cada partición primaria tiene una instancia (partición) del agente. El método processAllEntries se invoca para cada partición. El resultado de cada método processAllEntries se vuelve a serializar en el cliente y se devuelve al llamante en una instancia de correlación. En el siguiente ejemplo se presupone que hay una entidad Person con la forma siguiente:

```
import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;
@Entity
public class Person
{
  @Id String ssn;
  String firstName;
  String surname;
  int age;
}
```

La función proporcionada por la aplicación está escrita como una clase que implementa la interfaz MapAgentGrid. A continuación se ofrece un agente de ejemplo que muestra una función que devuelve la edad de una persona (Person) multiplicada por dos.

```
public class DoublePersonAgeAgent implements MapGridAgent, EntityAgentMixin
{
  private static final long serialVersionUID = -2006093916067992974L;

  int lowAge;
  int highAge;

  public Object process(Session s, ObjectMap map, Object key)
  {
    Person p = (Person)key;
    return new Integer(p.age * 2);
  }

  public Map processAllEntries(Session s, ObjectMap map)
  {
    EntityManager em = s.getEntityManager();
    Query q = em.createQuery("select p from Person p where p.age > ?1 and p.age < ?2");
    q.setParameter(1, lowAge);
    q.setParameter(2, highAge);
    Iterator iter = q.getResultIterator();
    Map<Person, Interger> rc = new HashMap<Person, Integer>();
   while(iter.hasNext())
    {
   Person p = (Person)iter.next();
```

```
   rc.put(p, (Integer)process(s, map, p));
  }
  return rc;
 }
 public Class getClassForEntity()
 {
  return Person.class;
 }
}
```

Este ejemplo muestra el agente de correlación con la edad doblada de una persona. Observemos primero los métodos de proceso. El primer método de proceso proporciona la persona con la que trabajar y devuelve el doble de la edad de esa entrada. El segundo método de proceso se llama para cada partición y se buscan todos los objetos Person con una edad comprendida entre un valor lowAge y un valor highAge, y se devuelve el doble de las edades.

```
Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

DoublePersonAgeAgent agent = new DoublePersonAgeAgent();

// efectúa una lista de las claves
ArrayList<Person> keyList = new ArrayList<Person>();
Person p = new Person();
p.ssn = "1";
keyList.add(p);
p = new Person ();
p.ssn = "2";
keyList.add(p);

// obtiene los resultados de las entradas
Map<Tuple, Object> = amgr.callMapAgent(agent, keyList);
```

Este ejemplo muestra un cliente que obtiene un elemento Session y una referencia a la correlación de persona. La operación del agente se realiza en una correlación específica. La interfaz AgentManager se recupera de dicha correlación. Se crea una instancia del agente que se va a invocar y se añade cualquier estado necesario al objeto mediante el establecimiento de atributos (no hay ninguno, en este caso). Se crea una lista de las claves. Se devuelve una correlación con los valores para la persona 1 doblados y los mismos valores para la persona 2.

Se invoca el agente para ese conjunto de claves. El método de proceso del agente se invoca en cada partición con algunas de las claves especificadas en la cuadrícula en paralelo. Se devuelve una correlación con los resultados fusionados de la clave especificada. En este caso, se devuelve una correlación con los valores que contienen la edad doblada de la persona 1 y lo mismo para la persona 2.

Aunque la clave no exista, se invoca el agente. De esta manera, el agente tiene la oportunidad de crear la entrada de correlación. Si se usa EntityAgentMixin, la clave que se procesará no será la entidad, sino el valor clave del tuple real de la entidad. Si las claves no se conocen, es posible solicitar a todas las particiones que busquen los objetos Person de una forma determinada y que devuelvan el doble de la edad. Observe el ejemplo siguiente:

```
Session s = grid.getSession();
  ObjectMap map = s.getMap("Person");
  AgentManager amgr = map.getAgentManager();

  DoublePersonAgeAgent agent = new DoublePersonAgeAgent();
  agent.lowAge = 20;
  agent.highAge = 9999;

  Map m = amgr.callMapAgent(agent);
```

El ejemplo anterior muestra el AgentManager que se obtiene para la correlación Person (persona) y el agente construido e inicializado con las edades bajas y altas para las personas de interés. A continuación, se invoca el agente utilizando el método callMapAgent. Observe que no se proporciona ninguna clave. Esto hace que ObjectGrid invoque el agente en cada partición de la cuadrícula en paralelo y a continuación devuelva los resultados fusionados al cliente. De esta manera, se buscarán todos los objetos Person en la cuadrícula con una edad comprendida entre los valores bajos y altos y se calculará el doble de la edad de los objetos Person. Esto muestra cómo las API de la cuadrícula pueden utilizarse para ejecutar una consulta que busque entidades que coincidan con una consulta determinada. ObjectGrid serializa y transporta el agente a las particiones con las entradas necesarias. Los resultados se serializan de forma similar y se transportan al cliente. Las API de correlación deben tratarse con cuidado. Si ObjectGrid contuviera terabytes de objetos y se ejecutara en muchos servidores, posiblemente se saturarían los sistemas más grandes que se ejecutasen en el cliente. Utilice esta API para procesar un subconjunto pequeño. Si necesita procesar un subconjunto de gran tamaño, se recomienda usar un agente de reducción para realizar el proceso en la cuadrícula en lugar de en el cliente.

Reducción paralela o agentes de agregación

Este estilo de programación procesa un subconjunto de entradas y calcula un resultado único para el grupo de entradas. Ejemplos de dicho resultado son:

- Valor mínimo
- Valor máximo
- Alguna otra función específica de empresa

Un agente de reducción está codificado y se invoca de una manera similar a la de los agentes de correlación.

Flujo de llamadas ReduceGridAgent

Cuando se invoca el método AgentManager.callReduceAgent con una colección de claves, la instancia de ReduceGridAgent se serializa y se envía a cada partición primaria en la que se resuelven las claves. Esto significa que los datos de la instancia almacenados en el agente pueden enviarse al servidor. Por consiguiente, cada partición primaria tiene una instancia del agente. El método reduce(Session s, ObjectMap map, Collection keys) se invoca una vez por instancia (partición) con el subconjunto de claves que se resuelve en la partición. El resultado de cada método de reducción se vuelve a serializar en el cliente. El método reduceResults se invoca en la instancia de ReduceGridAgent del cliente con la colección de cada resultado de cada invocación de reducción remota. El resultado del método reduceResults se devuelve al llamante del método callReduceAgent.

Cuando se invoca el método AgentManager.callReduceAgent sin una colección de claves, la instancia de ReduceGridAgent se serializa y se envía a todas las particiones primarias. Esto significa que los datos de la instancia almacenados en el agente pueden enviarse al servidor. Por consiguiente, cada partición primaria tiene una instancia del agente. El método reduce(Session s, ObjectMap map) se invoca una vez por instancia (partición). El resultado de cada método de reducción se vuelve a serializar en el cliente. El método reduceResults se invoca en la instancia de ReduceGridAgent del cliente con la colección de cada resultado de cada invocación de reducción remota. El resultado del método reduceResults se devuelve al llamante del método callReduceAgent. A continuación se muestra un ejemplo de un agente de reducción que simplemente añade las edades de las entradas coincidentes.

```
public class SumAgeReduceAgent implements ReduceGridAgent, EntityAgentMixin
{
  private static final long serialVersionUID = 2521080771723284899L;

  int lowAge;
  int highAge;

  public Object reduce(Session s, ObjectMap map, Collection keyList)
  {
    Iterator<Person> iter = keyList.iterator();
  int sum = 0;
  while(iter.hasNext())
  {
   Person p = iter.next();
   sume += p.age;
  }
  return new Integer(sum);
 }

 public Object reduce(Session s, ObjectMap map)
 {
  EntityManager em = s.getEntityManager ();
  Query q = em.createQuery("select p from Person p where p.age > ?1 and p.age < ?2");
  q.setParameter(1, lowAge);
  q.setParameter(2, highAge);
  Iterator<Person> iter = q.getResultIterator();
  int sum = 0;
  while(iter.hasNext())
  {
   sum += iter.next().age;
  }
  return new Integer(sum);
 }

 public Class getClassForEntity()
 {
  return Person.class;
 }
}
```

El ejemplo anterior muestra el agente. El agente tiene tres partes importantes. La primera permite que un conjunto específico de entradas se procesen sin una consulta. Itera sobre el conjunto de entradas al añadir las edades. La suma se devuelve desde el método. La segunda parte utiliza una consulta para seleccionar las entradas que se agregarán. A continuación, se suman todas las edades de Person coincidentes. El tercer método se utiliza para agregar los resultados de cada partición a un único resultado. ObjectGrid realiza la agregación de entradas en paralelo en la cuadrícula. Cada partición produce un resultado intermedio que debe agregarse con otros resultados intermedios de particiones. Este tercer método realiza dicha tarea. En el ejemplo siguiente, se invoca el agente, y se agregan las edades de todas las personas comprendidas entre 10 y 20 exclusivamente:

```
Session s = grid.getSession();
  ObjectMap map = s.getMap("Person");
  AgentManager amgr = map.getAgentManager();

  SumAgeReduceAgent agent = new SumAgeReduceAgent();

  Person p = new Person();
  p.ssn = "1";
  ArrayList<Person> list = new ArrayList<Person>();
 list.add(p);
 p = new Person ();
 p.ssn = "2";
 list.add(p);
 Integer v = (Integer)amgr.callReduceAgent(agent, list);
```

Funciones del agente

El agente puede llevar a cabo cualquier operación de ObjectMap o EntityManager dentro del fragmento local donde se ejecuta. El agente recibe un objeto Session y puede añadir, actualizar, consultar, leer o eliminar datos de la partición que representa el objeto Session. Algunas aplicaciones sólo consultarán los datos de la cuadrícula, pero también podrá escribir un agente para aumentar en 1 todas las

edades de las entidades Person que coincidan con una consulta determinada. Existe una transacción en el objeto Session cuando se llama al agente, y se confirma cuando se devuelve el agente, a menos que se lance una excepción.

Manejo de errores

Si se invoca un agente de correlación con una clave desconocida, el valor devuelto es un objeto de error que implementa la interfaz EntryErrorValue.

Transacciones

Un agente de correlación se ejecuta en una transacción independiente del cliente. Las invocaciones de agente se pueden agrupar en una única transacción. Si se produce una anomalía en un agente (emite una excepción), la transacción se retrotrae. Cualquier agente que se haya ejecutado correctamente en una transacción se retrotraerá con el agente anómalo. AgentManager volverá a ejecutar los agentes retrotraídos que se ejecutaron correctamente en una nueva transacción.

Si desea más información, consulte la documentación de la API DataGrid.

Configuración de clientes mediante programación

Puede configurar un cliente de WebSphere eXtreme Scale en función de sus requisitos, como por ejemplo la necesidad de sustituir valores.

Alteración temporal de los plug-ins

Puede alterar temporalmente los siguientes plug-ins en un cliente:

- **Plug-ins ObjectGrid**
  - Plug-in TransactionCallback
  - Plug-in ObjectGridEventListener
- **Plug-ins de BackingMap**
  - Plug-in Evictor
  - Plug-in MapEventListener
  - Atributo numberOfBuckets
  - Atributo ttlEvictorType
  - Atributo timeToLive

Configuración del cliente mediante programación

También puede alterar temporalmente los valores de ObjectGrid del lado del cliente mediante programación. Cree un objeto ObjectGridConfiguration que sea similar en estructura a la instancia de ObjectGrid del lado del servidor. El código siguiente crea una instancia de ObjectGrid del lado del cliente funcionalmente equivalente a la alteración temporal del cliente en la sección anterior que utiliza un archivo XML.

```
alteración temporal del lado del cliente mediante programación
ObjectGridConfiguration companyGridConfig = ObjectGridConfigFactory
    .createObjectGridConfiguration("CompanyGrid");
Plugin txCallbackPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.TRANSACTION_CALLBACK, "com.company.MyClientTxCallback");
companyGridConfig.addPlugin(txCallbackPlugin);

Plugin ogEventListenerPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.OBJECTGRID_EVENT_LISTENER, "");
companyGridConfig.addPlugin(ogEventListenerPlugin);
```

```
BackingMapConfiguration customerMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("Customer");
customerMapConfig.setNumberOfBuckets(1429);
Plugin evictorPlugin = ObjectGridConfigFactory.createPlugin(PluginType.EVICTOR,
    "com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor");
customerMapConfig.addPlugin(evictorPlugin);

companyGridConfig.addBackingMapConfiguration(customerMapConfig);

BackingMapConfiguration orderLineMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("OrderLine");
orderLineMapConfig.setNumberOfBuckets(701);
orderLineMapConfig.setTimeToLive(800);
orderLineMapConfig.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);

companyGridConfig.addBackingMapConfiguration(orderLineMapConfig);

List ogConfigs = new ArrayList();
ogConfigs.add(companyGridConfig);

Map overrideMap = new HashMap();
overrideMap.put(CatalogServerProperties.DEFAULT_DOMAIN, ogConfigs);

ogManager.setOverrideObjectGridConfigurations(overrideMap);
ClientClusterContext client = ogManager.connect(catalogServerAddresses, null, null);
ObjectGrid companyGrid = ogManager.getObjectGrid(client, objectGridName);
```

La instancia de ogManager de la interfaz ObjectGridManager comprueba si hay alteraciones temporales sólo en los objetos ObjectGridConfiguration y BackingMapConfiguration incluidos en la correlación overrideMap. Por ejemplo, el código anterior altera temporalmente el número de grupos de la correlación OrderLine. No obstante, la correlación Order permanece inalterada en el lado del cliente porque no se incluye ninguna configuración para dicha correlación.

Inhabilitación de la memoria caché cercana del cliente

La memoria caché cercana se habilita de manera predeterminada al configurar el bloqueo como optimista o ninguno. Los clientes no mantienen una memoria caché cercana cuando el valor de bloqueo se configura como pesimista. Para inhabilitar la memoria caché cercana, debe establecer el atributo de numberOfBuckets en 0 en el archivo descriptor de ObjectGrid de alteración temporal del cliente.

Habilitación de la réplica en el cliente

Puede habilitar también la réplica de correlaciones en el cliente para conseguir una disponibilidad de datos más rápida.

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

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

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

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

Acceso a los datos con el servicio de datos REST

Desarrolle las aplicaciones que realizan operaciones con los protocolos del servicio de datos REST.

Conceptos relacionados:

"Operaciones con el servicio de datos REST"
Después de iniciar el servicio de datos REST de eXtreme Scale, puede utilizar cualquier cliente HTTP para interactuar con él. Se puede utilizar un navegador web, un cliente PHP, un cliente Java o un cliente de WCF Data Services para emitir cualquiera de las operaciones de solicitud soportadas.

"Visión general de los servicios de datos REST" en la página 117
El servicio de datos REST de WebSphere eXtreme Scale es un servicio HTTP Java compatible con Microsoft WCF Data Services (anteriormente ADO.NET Data Services) e implementa el Protocolo de datos abierto (OData). Microsoft WCF Data Services es compatible con esta especificación al utilizar Visual Studio 2008 SP1 y .NET Framework 3.5 SP1.

Referencia relacionada:

"Simultaneidad optimista en el servicio de datos REST" en la página 276
El servicio de datos REST de eXtreme Scale sigue un modelo de bloqueo optimista utilizando cabeceras HTTP nativas: If-Match, If-None-Match y ETag. Estas cabeceras se envían en mensajes de solicitud y respuesta para transmitir la información de versión de una entidad del servidor al cliente y del cliente al servidor.

"Protocolos de solicitud para el servicio de datos REST" en la página 277
En general, los protocolos para interactuar con los servicios REST son los mismos que se describen en el protocolo WCF Data Services AtomPub. No obstante, eXtreme Scale proporciona detalles adicionales, de la perspectiva de modelo de entidad de eXtreme Scale. Se espera que los usuarios estén familiarizados con los protocolos de WCF Data Services antes de leer esta sección. Como alternativa, los usuarios pueden leer esta sección con la sección del protocolo WCF Data Services.

"Solicitudes de recuperación con el servicio de datos REST" en la página 278
Un cliente utiliza una solicitud RetrieveEntity para recuperar una entidad de eXtreme Scale. La carga útil de respuesta contiene los datos de la entidad en formato AtomPub o JSON. Además, se puede utilizar el operador del sistema $expand para expandir las relaciones. Las relaciones se representan en línea en la respuesta de servicio de datos como un documento de canal de información Feed, que es una relación a muchos, o un documento de entrada Atom, que es una relación a uno.

"Recuperación de elementos que no sean entidades con los servicios de datos REST" en la página 285
El servicio de datos REST permite recuperar no sólo entidades, sino también elementos como colecciones de entidades y propiedades.

"Solicitudes de inserción con los servicios de datos REST" en la página 291
Se puede utilizar una solicitud InsertEntity para insertar una nueva instancia de entidad de eXtreme Scale, potencialmente con entidades relacionadas nuevas, en el servicio de datos REST de eXtreme Scale.

"Solicitudes de actualización con los servicios de datos REST" en la página 295
El servicio de datos REST de WebSphere eXtreme Scale soporta solicitudes de actualización de entidades, propiedades primitivas de entidades, etc.

"Solicitudes de supresión con los servicios de datos REST" en la página 299
El servicio de datos REST de WebSphere eXtreme Scale suprime entidades, valores de propiedad y enlaces.

Operaciones con el servicio de datos REST

Después de iniciar el servicio de datos REST de eXtreme Scale, puede utilizar cualquier cliente HTTP para interactuar con él. Se puede utilizar un navegador

web, un cliente PHP, un cliente Java o un cliente de WCF Data Services para emitir cualquiera de las operaciones de solicitud soportadas.

El servicio REST implementa un subconjunto de la especificación Microsoft Atom Publishing Protocol: Data Services URI and Payload Extensions, Versión 1.0, que forma parte del protocolo OData. Este tema describe qué características de la especificación están soportadas y cómo se correlacionan con eXtreme Scale.

URI de la raíz de servicio

Microsoft WCF Data Services define normalmente un servicio por origen de datos o modelo de entidad. El servicio de datos REST de eXtreme Scale define un servicio por cada ObjectGrid definida. Cada ObjectGrid definida en el archivo XML de sustitución de cliente ObjectGrid de eXtreme Scale se expone automáticamente como raíz del servicio REST independiente.

El URI de la raíz de servicio es:

`http://host:puerto/raízcontexto/restservice/nombrecuadrícula`

Donde:

- *raízcontexto* se define al desplegar la aplicación de servicio de datos REST y depende del servidor de aplicaciones
- *nombrecuadrícula* es el nombre del ObjectGrid

Tipos de solicitud

La lista siguiente describe los tipos de solicitud de Microsoft WCF Data Services que el servicio de datos REST de eXtreme Scale soporta. Para obtener información detallada sobre cada tipo de solicitud que WCF Data Services soporta, consulte MSDN: Request Types

Tipos de solicitudes de inserción

Los clientes pueden insertar recursos utilizando el verbo HTTP POST con las limitaciones siguientes:

- Solicitud InsertEntity: Soportada.
- Solicitud InsertLink: Soportada.
- Solicitud InsertMediaResource: No soportada debido a la restricción de soporte de recursos de medios.

Para obtener información adicional, consulte: MSDN: Insert Request Types.

Tipos de solicitudes de actualización

Los clientes pueden actualizar recursos utilizando los verbos HTTP PUT y MERGE con las limitaciones siguientes:

- Solicitud UpdateEntity: Soportada.
- Solicitud UpdateComplexType: No soportada debido a la restricción de tipo complejo.
- Solicitud UpdatePrimitiveProperty: Soportada.
- Solicitud UpdateValue: Soportada.
- Solicitud UpdateLink: Soportada.
- Solicitud UpdateMediaResource: No soportada debido a la restricción de soporte de recursos de medios.

Para obtener información adicional, consulte: MSDN: Insert Request types.

Tipos de solicitudes de supresión

Los clientes pueden suprimir recursos utilizando el verbo HTTP DELETE con las limitaciones siguientes:

- Solicitud DeleteEntity: Soportada.
- Solicitud DeleteLink: Soportada.
- Solicitud DeleteValue: Soportada.

Para obtener información adicional, consulte: MSDN: Delete Request Types.

Tipos de solicitudes de recuperación

Los clientes pueden recuperar recursos utilizando el verbo HTTP GET con las limitaciones siguientes:

- Solicitud RetrieveEntitySet: Soportada.
- Solicitud RetrieveEntity: Soportada.
- Solicitud RetrieveComplexType: No soportada debido a la restricción de tipo complejo.
- Solicitud RetrievePrimitiveProperty: Soportada.
- Solicitud RetrieveValue: Soportada.
- Solicitud RetrieveServiceMetadata: Soportada.
- Solicitud RetrieveServiceDocument: Soportada.
- Solicitud RetrieveLink: Soportada.
- Solicitud Retrieve que contiene una correlación de canal de información personalizable: No soportada
- RetrieveMediaResource: No soportada debido a la restricción de recursos de medios.

Para obtener información adicional, consulte: MSDN: Retrieve Request Types.

Opciones de consulta del sistema

Hay consultas soportadas que permiten que los clientes identifiquen una colección de entidades o una sola entidad. Las opciones de consulta del sistema se especifican en un URI de servicio de datos y están soportadas con las limitaciones siguientes:

- $expand: Soportada.
- $filter: Soportada.
- $orderby: Soportada.
- $format: No soportada. El formato aceptable se identifica en la cabecera de solicitud HTTP Accept.
- $skip: Soportada.
- $top: Soportada.

Para obtener información adicional, consulte: MSDN: System Query Options.

Direccionamiento de particiones

El direccionamiento de particiones se basa en la entidad raíz. Un URI de solicitud deduce una entidad raíz si su vía de acceso de recurso empieza con una entidad raíz o con una entidad que tenga una asociación directa o indirecta con la entidad. En un entorno particionado, cualquier solicitud que no pueda deducir una entidad raíz se rechazará. Cualquier solicitud que deduzca una entidad raíz se direccionará a la partición correcta.

Para obtener información adicional sobre la definición de un esquema con asociaciones y entidades raíz, consulte Modelo de datos escalable de eXtreme Scale y Particionamiento.

Solicitud de invocación

Las solicitudes de invocación no están soportadas.Para obtener información adicional, consulte: MSDN: Invoke RequestTypes.

Solicitud por lotes

Los clientes pueden realizar por lotes varios conjuntos de cambios u operaciones de consulta en una sola solicitud. Esto puede reducir el número de transmisiones de ida y vuelta al servidor y permite la participación de varias solicitudes en una sola transacción. Para obtener información adicional, consulte: MSDN: Batch Request.

Solicitudes a través de túnel

Las solicitudes a través de túnel no están soportadas. Para obtener información adicional, consulte: MSDN: Tunneled Requests.

Tareas relacionadas:

"Acceso a los datos con el servicio de datos REST" en la página 271
Desarrolle las aplicaciones que realizan operaciones con los protocolos del servicio de datos REST.

Referencia relacionada:

"Simultaneidad optimista en el servicio de datos REST"
El servicio de datos REST de eXtreme Scale sigue un modelo de bloqueo optimista utilizando cabeceras HTTP nativas: If-Match, If-None-Match y ETag. Estas cabeceras se envían en mensajes de solicitud y respuesta para transmitir la información de versión de una entidad del servidor al cliente y del cliente al servidor.

"Protocolos de solicitud para el servicio de datos REST" en la página 277
En general, los protocolos para interactuar con los servicios REST son los mismos que se describen en el protocolo WCF Data Services AtomPub. No obstante, eXtreme Scale proporciona detalles adicionales, de la perspectiva de modelo de entidad de eXtreme Scale. Se espera que los usuarios estén familiarizados con los protocolos de WCF Data Services antes de leer esta sección. Como alternativa, los usuarios pueden leer esta sección con la sección del protocolo WCF Data Services.

"Solicitudes de recuperación con el servicio de datos REST" en la página 278
Un cliente utiliza una solicitud RetrieveEntity para recuperar una entidad de eXtreme Scale. La carga útil de respuesta contiene los datos de la entidad en formato AtomPub o JSON. Además, se puede utilizar el operador del sistema $expand para expandir las relaciones. Las relaciones se representan en línea en la respuesta de servicio de datos como un documento de canal de información Feed, que es una relación a muchos, o un documento de entrada Atom, que es una relación a uno.

"Recuperación de elementos que no sean entidades con los servicios de datos REST" en la página 285
El servicio de datos REST permite recuperar no sólo entidades, sino también elementos como colecciones de entidades y propiedades.

"Solicitudes de inserción con los servicios de datos REST" en la página 291
Se puede utilizar una solicitud InsertEntity para insertar una nueva instancia de entidad de eXtreme Scale, potencialmente con entidades relacionadas nuevas, en el servicio de datos REST de eXtreme Scale.

"Solicitudes de actualización con los servicios de datos REST" en la página 295
El servicio de datos REST de WebSphere eXtreme Scale soporta solicitudes de actualización de entidades, propiedades primitivas de entidades, etc.

"Solicitudes de supresión con los servicios de datos REST" en la página 299
El servicio de datos REST de WebSphere eXtreme Scale suprime entidades, valores de propiedad y enlaces.

Simultaneidad optimista en el servicio de datos REST

El servicio de datos REST de eXtreme Scale sigue un modelo de bloqueo optimista utilizando cabeceras HTTP nativas: If-Match, If-None-Match y ETag. Estas cabeceras se envían en mensajes de solicitud y respuesta para transmitir la información de versión de una entidad del servidor al cliente y del cliente al servidor.

Para obtener más información sobre la simultaneidad optimista, consulte MSDN Library: Optimistic Concurrency (ADO.NET).

El servicio de datos REST de eXtreme Scale habilita la simultaneidad optimista de una entidad si hay definido un atributo de versión en el esquema de dicha

entidad. Una propiedad de versión se puede definir en el esquema de entidad mediante una anotación @Version para clases Java o un atributo <version/> para entidades definidas utilizando un archivo XML descriptor de entidad. El servicio de datos REST de eXtreme Scale propaga automáticamente el valor de la propiedad version al cliente en la cabecera ETag para respuestas de una sola entidad utilizando un atributo m:etag en la carga útil para respuestas de XML de entidad múltiple y un atributo etag en la carga útil para respuestas de JSON de entidad múltiple.

Para obtener detalles sobre cómo definir un esquema de entidad eXtreme Scale, consulte "Definición de un esquema de entidad" en la página 169.

Conceptos relacionados:

"Operaciones con el servicio de datos REST" en la página 272
Después de iniciar el servicio de datos REST de eXtreme Scale, puede utilizar cualquier cliente HTTP para interactuar con él. Se puede utilizar un navegador web, un cliente PHP, un cliente Java o un cliente de WCF Data Services para emitir cualquiera de las operaciones de solicitud soportadas.

"Visión general de los servicios de datos REST" en la página 117
El servicio de datos REST de WebSphere eXtreme Scale es un servicio HTTP Java compatible con Microsoft WCF Data Services (anteriormente ADO.NET Data Services) e implementa el Protocolo de datos abierto (OData). Microsoft WCF Data Services es compatible con esta especificación al utilizar Visual Studio 2008 SP1 y .NET Framework 3.5 SP1.

Tareas relacionadas:

"Acceso a los datos con el servicio de datos REST" en la página 271
Desarrolle las aplicaciones que realizan operaciones con los protocolos del servicio de datos REST.

Protocolos de solicitud para el servicio de datos REST

En general, los protocolos para interactuar con los servicios REST son los mismos que se describen en el protocolo WCF Data Services AtomPub. No obstante, eXtreme Scale proporciona detalles adicionales, de la perspectiva de modelo de entidad de eXtreme Scale. Se espera que los usuarios estén familiarizados con los protocolos de WCF Data Services antes de leer esta sección. Como alternativa, los usuarios pueden leer esta sección con la sección del protocolo WCF Data Services.

Se proporcionan ejemplos para ilustrar la solicitud y la respuesta. Estos ejemplos se aplican tanto al servicio de datos REST de eXtreme Scale como a WCF Data Services. Puesto que los navegadores web sólo pueden recuperar datos, las operaciones CUD (crear, actualizar y suprimir) debe realizarlas otro cliente como, por ejemplo, Java, JavaScript, RUBY o PHP.

Conceptos relacionados:

"Operaciones con el servicio de datos REST" en la página 272
Después de iniciar el servicio de datos REST de eXtreme Scale, puede utilizar cualquier cliente HTTP para interactuar con él. Se puede utilizar un navegador web, un cliente PHP, un cliente Java o un cliente de WCF Data Services para emitir cualquiera de las operaciones de solicitud soportadas.

"Visión general de los servicios de datos REST" en la página 117
El servicio de datos REST de WebSphere eXtreme Scale es un servicio HTTP Java compatible con Microsoft WCF Data Services (anteriormente ADO.NET Data Services) e implementa el Protocolo de datos abierto (OData). Microsoft WCF Data Services es compatible con esta especificación al utilizar Visual Studio 2008 SP1 y .NET Framework 3.5 SP1.

Tareas relacionadas:

"Acceso a los datos con el servicio de datos REST" en la página 271
Desarrolle las aplicaciones que realizan operaciones con los protocolos del servicio de datos REST.

Solicitudes de recuperación con el servicio de datos REST

Un cliente utiliza una solicitud RetrieveEntity para recuperar una entidad de eXtreme Scale. La carga útil de respuesta contiene los datos de la entidad en formato AtomPub o JSON. Además, se puede utilizar el operador del sistema $expand para expandir las relaciones. Las relaciones se representan en línea en la respuesta de servicio de datos como un documento de canal de información Feed, que es una relación a muchos, o un documento de entrada Atom, que es una relación a uno.

Consejo: Para obtener información detallada sobre el protocolo RetrieveEntity definido en WCF Data Services, consulte MSDN: RetrieveEntity Request.

Recuperación de una entidad

El ejemplo siguiente de RetrieveEntity recupera una entidad Customer con clave.

AtomPub

- Método

  GET
- URI de la solicitud:

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')
- Cabecera de la solicitud:

  Accept: application/atom+xml
- Carga útil de la solicitud:

  Ninguna
- Cabecera de la respuesta:

  Content-Type: application/atom+xml
- Carga útil de la respuesta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/
restservice" xmlns:d= "http://schemas.microsoft.com/ado/2007/
08/dataservices" xmlns:m = "http://schemas.microsoft.com/ado/2007/
08/dataservices/metadata" xmlns = "http://www.w3.org/2005/Atom">

<category term = "NorthwindGridModel.Customer" scheme = "http://
```

```
schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
<id>http://localhost:8080/wxsrestservice/restservice/
    NorthwindGrid/Customer('ACME')</id>
    <title type = "text"/>
    <updated>2009-12-16T19:52:10.593Z</updated>
    <author>
        <name/>
    </author>
    <link rel = "edit" title = "Customer" href = "Customer('ACME')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/
     dataservices/related/
orders" type = "application/atom+xml;type=feed" title =
  "orders" href ="Customer('ACME')/orders"/>
    <content type = "application/xml">
        <m:properties>
            <d:customerId>ACME</d:customerId>
            <d:city m:null = "true"/>
            <d:companyName>RoaderRunner</d:companyName>
            <d:contactName>ACME</d:contactName>
            <d:country m:null = "true"/>
            <d:version m:type = "Edm.Int32">3</d:version>
        </m:properties>
    </content>
</entry>
```

- Código de respuesta:

  200 OK

JSON

- Método

  GET

- URI de la solicitud:

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')

- Cabecera de la solicitud:

  Accept: application/json

- Carga útil de la solicitud:

  Ninguna

- Cabecera de la respuesta:

  Content-Type: application/json

- Carga útil de la respuesta:

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Customer('ACME')",
"type":"NorthwindGridModel.Customer"},
"customerId":"ACME",
"city":null,
"companyName":"RoaderRunner",
"contactName":"ACME",
"country":null,
"version":3,
   "orders":{"__deferred":{"uri":"http://localhost:8080/
      wxsrestservice/restservice/
   NorthwindGrid/Customer('ACME')/orders"}}}}
```

- Código de respuesta:

  200 OK

Consultas

También se puede utilizar una consulta con una solicitud RetrieveEntitySet o RetrieveEntity. Una consulta se especifica mediante el operador $filter del sistema.

Para obtener información detallada sobre el operador $filter, consulte: MSDN: Filter System Query Option ($filter)

El protocolo OData soporta varias expresiones comunes. El servicio de datos REST de eXtreme Scale da soporte a un subconjunto de expresiones definidas en las especificaciones:

- Expresiones booleanas:
  - eq, ne, lt, le, gt, ge
  - negate
  - not
  - parenthesis
  - and, or
- Expresiones aritméticas:
  - add
  - sub
  - mul
  - div
- Literales primitivos
  - String
  - date-time
  - decimal
  - single
  - double
  - int16
  - int32
  - int64
  - binary
  - null
  - byte

Las expresiones siguientes *no* están disponibles:

- Expresiones booleanas:
  - isof
  - cast
- Expresiones de llamada de método
- Expresiones aritméticas:
  - mod
- Literales primitivos:
  - Guid
- Expresiones de miembro

Para ver una lista completa de las expresiones disponibles en Microsoft WCF Data Services, y su descripción, consulte la sección 2.2.3.6.1.1 : Common Expression Syntax.

El ejemplo siguiente ofrece una demostración de una solicitud RetrieveEntity con una consulta. En este ejemplo, se recuperan todos los clientes cuyo nombre de contacto es “RoadRunner”. El único cliente que coincide con este filtro es Customer('ACME'), tal como se muestra en la carga útil de la respuesta.

Restricción: Esta consulta sólo funcionará para entidades no particionadas. Si la entidad Customer está particionada, se necesitará la clave perteneciente al cliente.

AtomPub

- Método: GET
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?$filter=contactName eq 'RoadRunner'
- Cabecera de la solicitud: Accept: application/atom+xml
- Carga útil de entrada: Ninguna
- Cabecera de la respuesta: Content-Type: application/atom+xml
- Carga útil de la respuesta:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<feed
 xml:base="http://localhost:8080/wxsrestservice/restservice"
 xmlns:d="http://schemas.microsoft.com/ado/2007/08/
   dataservices"
xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <title type="text">Customer</title>
<id>http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer </id>
 <updated>2009-09-16T04:59:28.656Z</updated>
 <link rel="self" title="Customer" href="Customer" />
 <entry>
  <category term="NorthwindGridModel.Customer"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  <id>
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/  Customer('ACME')</id>
  <title type="text" />
  <updated>2009-09-16T04:59:28.656Z</updated>
  <author>
   <name />
  </author>
  <link rel="edit" title="Customer" href="Customer('ACME')" />
  <link
   rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/orders"
   type="application/atom+xml;type=feed" title="orders"
   href="Customer('ACME')/orders" />
  <content type="application/xml">
     <m:properties>
            <d:customerId>ACME</d:customerId>
            <d:city m:null = "true"/>
            <d:companyName>RoaderRunner</d:companyName>
            <d:contactName>ACME</d:contactName>
            <d:country m:null = "true"/>
            <d:version m:type = "Edm.Int32">3</d:version>
        </m:properties>
  </content>
 </entry>
</feed>
```

- Código de respuesta: 200 OK

JSON

- Método: GET
- URI de la solicitud:

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?$filter=contactName eq 'RoadRunner'
- Cabecera de la solicitud: Accept: application/json
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/json
- Carga útil de la respuesta:

```
{"d":[{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Customer('ACME')",
"type":"NorthwindGridModel.Customer"},
"customerId":"ACME",
"city":null,
"companyName":"RoaderRunner",
"contactName":"ACME",
"country":null,
"version":3,
   "orders":{"__deferred":{"uri":"http://localhost:8080/
     wxsrestservice/restservice/NorthwindGrid/
     Customer('ACME')/orders"}}}]}
```

- Código de respuesta: 200 OK

Operador del sistema $expand

El operador del sistema $expand se puede utilizar para expandir asociaciones. Las asociaciones se representan en línea en la respuesta del servicio de datos. Las asociaciones con varios valores (a muchos) se representan como un documento de canal de información Atom o una matriz JSON. Las asociaciones con un solo valor (a uno) se representan como un documento de entrada Atom o un objeto JSON.

Para obtener información detallada sobre el operador del sistema $expand, consulte Expand System Query Option ($expand) (Opción de consulta del sistema expand ($expand).

A continuación se ofrece un ejemplo sobre cómo utilizar al operador del sistema $expand. En este ejemplo, recuperamos la entidad Customer('IBM'), que tiene asociados los pedidos 5000, 5001 y otros. La cláusula de $expand se define como “orders”, de modo que la colección de pedidos se expanda como en línea en la carga útil de la respuesta. Aquí sólo se muestran los pedidos 5000 y 5001.

AtomPub

- Método: GET
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders
- Cabecera de la solicitud: Accept: application/atom+xml
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/atom+xml
- Carga útil de la respuesta:

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/restservice"
 xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
   xmlns:m = "http://schemas.microsoft.com/ado/2007/08/dataservices/
   metadata" xmlns = "http://www.w3.org/2005/Atom">
<category term = "NorthwindGridModel.Customer" scheme = "http://schemas.
```

```
microsoft.com/ado/2007/08/dataservices/scheme"/>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
    Customer('IBM')</id>
    <title type = "text"/>
    <updated>2009-12-16T22:50:18.156Z</updated>
    <author>
        <name/>
    </author><link rel = "edit" title = "Customer" href =
    "Customer('IBM')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/dataservices/
    related/orders" type = "application/atom+xml;type=feed" title =
    "orders" href = "Customer('IBM')/orders">
        <m:inline>
            <feed>
                <title type = "text">orders</title>
                <id>http://localhost:8080/wxsrestservice/restservice/
         NorthwindGrid/Customer('IBM')/orders</id>
                <updated>2009-12-16T22:50:18.156Z</updated>
                <link rel = "self" title = "orders" href = "Customer
          ('IBM')/orders"/>
                <entry>
                    <category term = "NorthwindGridModel.Order" scheme =
           "http://schemas.microsoft.com/ado/2007/08/
           dataservices/scheme"/>
                    <id>http://localhost:8080/wxsrestservice/restservice/
         NorthwindGrid/Order(orderId=5000,customer_customerId=
'IBM')</id>
                    <title type = "text"/>
                    <updated>2009-12-16T22:50:18.156Z</updated>
                    <author>
                        <name/>
                    </author>
                    <link rel = "edit" title = "Order" href =
         "Order(orderId=5000,customer_customerId='IBM')"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/08/
         dataservices/related/customer" type = "application/
         atom+xml;type=entry" title ="customer" href =
         "Order(orderId=5000,customer_customerId='IBM')/customer"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/08/
         dataservices/related/orderDetails" type = "application/
          atom+xml;type=feed" title ="orderDetails" href =
          "Order(orderId=5000,customer_customerId='IBM')/orderDetails"/>
                    <content type = "application/xml">
                        <m:properties>
                            <d:orderId m:type = "Edm.Int32">5000</d:orderId>
                            <d:customer_customerId>IBM</d:customer_customerId>
                            <d:orderDate m:type = "Edm.DateTime">
               2009-12-16T19:46:29.562</d:orderDate>
                            <d:shipCity>Rochester</d:shipCity>
                            <d:shipCountry m:null = "true"/>
                            <d:version m:type = "Edm.Int32">0</d:version>
                        </m:properties>
                    </content>
                </entry>
                <entry>
                    <category term = "NorthwindGridModel.Order" scheme =
           "http://schemas.microsoft.com/ado/2007/08/
           dataservices/scheme"/>
                    <id>http://localhost:8080/wxsrestservice/restservice/
         NorthwindGrid/Order(orderId=5001,customer_customerId=
'IBM')</id>
                    <title type = "text"/>
                    <updated>2009-12-16T22:50:18.156Z</updated>
                    <author>
                        <name/></author>
                    <link rel = "edit" title = "Order" href = "Order(
```

```
orderId=5001,customer_customerId='IBM')"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/
           08/dataservices/related/customer" type =
           "application/atom+xml;type=entry" title =
           "customer" href = "Order(orderId=5001,customer_customerId=
         'IBM')/customer"/>
                    <link rel = "http://schemas.microsoft.com/ado/2007/08/
            dataservices/related/orderDetails" type =
            "application/atom+xml;type=feed" title =
            "orderDetails" href = "Order(orderId=5001,
         customer_customerId='IBM')/orderDetails"/>
<content type = "application/xml">
                        <m:properties>
                            <d:orderId m:type = "Edm.Int32">5001</d:orderId>
                            <d:customer_customerId>IBM</d:customer_customerId>
                            <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:
               50:11.125</d:orderDate>
                            <d:shipCity>Rochester</d:shipCity>
                            <d:shipCountry m:null = "true"/>
                            <d:version m:type = "Edm.Int32">0</d:version>
                        </m:properties>
                    </content>
                </entry>
            </feed>
        </m:inline>
    </link>
    <content type = "application/xml">
        <m:properties>
            <d:customerId>IBM</d:customerId>
            <d:city m:null = "true"/>
            <d:companyName>IBM Corporation</d:companyName>
            <d:contactName>John Doe</d:contactName>
            <d:country m:null = "true"/>
            <d:version m:type = "Edm.Int32">4</d:version>
        </m:properties>
    </content>
</entry>
```

- Código de respuesta: 200 OK

JSON

- Método: GET
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders
- Cabecera de la solicitud: Accept: application/json
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/json
- Carga útil de la respuesta:

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Customer('IBM')",
"type":"NorthwindGridModel.Customer"},
"customerId":"IBM",
"city":null,
"companyName":"IBM Corporation",
"contactName":"John Doe",
"country":null,
"version":4,
"orders":[{"__metadata":{"uri":"http://localhost:8080/
    wxsrestservice/restservice/NorthwindGrid/Order(
    orderId=5000,customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
"orderId":5000,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260992789562)\/",
```

```
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(
   orderId=5000,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:
     8080/wxsrestservice/restservice/NorthwindGrid/
     Order(orderId=5000,customer_customerId='IBM')/
     orderDetails"}}},
{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Order(orderId=5001,
     customer_customerId='IBM')","type":
     "NorthwindGridModel.Order"},
"orderId":5001,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260993011125)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:
   8080/wxsrestservice/restservice/
NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(
   orderId=5001, customer_customerId='IBM')/
   orderDetails"}}}]}}
```

- Código de respuesta: 200 OK

Conceptos relacionados:

"Operaciones con el servicio de datos REST" en la página 272
Después de iniciar el servicio de datos REST de eXtreme Scale, puede utilizar cualquier cliente HTTP para interactuar con él. Se puede utilizar un navegador web, un cliente PHP, un cliente Java o un cliente de WCF Data Services para emitir cualquiera de las operaciones de solicitud soportadas.

"Visión general de los servicios de datos REST" en la página 117
El servicio de datos REST de WebSphere eXtreme Scale es un servicio HTTP Java compatible con Microsoft WCF Data Services (anteriormente ADO.NET Data Services) e implementa el Protocolo de datos abierto (OData). Microsoft WCF Data Services es compatible con esta especificación al utilizar Visual Studio 2008 SP1 y .NET Framework 3.5 SP1.

Tareas relacionadas:

"Acceso a los datos con el servicio de datos REST" en la página 271
Desarrolle las aplicaciones que realizan operaciones con los protocolos del servicio de datos REST.

Recuperación de elementos que no sean entidades con los servicios de datos REST

El servicio de datos REST permite recuperar no sólo entidades, sino también elementos como colecciones de entidades y propiedades.

Recuperación de una colección de entidades

Un cliente puede utilizar una solicitud RetrieveEntitySet para recuperar un conjunto de entidades de eXtreme Scale. Las entidades se representan como un documento de canal de información Atom o una matriz JSON en la carga útil de la respuesta. Para obtener información detallada sobre el protocolo RetrieveEntitySet definido en WCF Data Services, consulte: MSDN: RetrieveEntitySet Request.

El siguiente ejemplo de solicitud RetrieveEntitySet recupera todas las entidades Order asociadas con la entidad Customer('IBM'). Aquí sólo se muestran los pedidos 5000 y 5001.

AtomPub

- Método: GET
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders
- Cabecera de la solicitud: Accept: application/atom+xml
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/atom+xml
- Carga útil de la respuesta:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xml:base = "http://localhost:8080/wxsrestservice/restservice"
 xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
   xmlns:m = "http://schemas.microsoft.com/ado/2007/08/dataservices/
   metadata" xmlns = "http://www.w3.org/2005/Atom">
    <title type = "text">Order</title>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
     Order</id>
    <updated>2009-12-16T22:53:09.062Z</updated>
    <link rel = "self" title = "Order" href = "Order"/>
    <entry>
        <category term = "NorthwindGridModel.Order" scheme = "http://
      schemas.microsoft.com/
       ado/2007/08/dataservices/scheme"/>
        <id>http://localhost:8080/wxsrestservice/restservice/
      NorthwindGrid/Order(orderId=5000,customer_customerId=
'IBM')</id>
        <title type = "text"/>
        <updated>2009-12-16T22:53:09.062Z</updated>
        <author>
            <name/>
        </author>
        <link rel = "edit" title = "Order" href = "Order(orderId=5000,
customer_customerId='IBM')"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
      dataservices/related/customer"
       type = "application/atom+xml;type=entry"
      title = "customer" href = "Order(orderId=5000,
      customer_customerId='IBM')/customer"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
      dataservices/related/orderDetails"
      type = "application/atom+xml;type=feed"
      title = "orderDetails" href = "Order(orderId=5000,
         customer_customerId='IBM')/
        orderDetails"/>
        <content type = "application/xml">
            <m:properties>
                <d:orderId m:type = "Edm.Int32">5000</d:orderId>
                <d:customer_customerId>IBM</d:customer_customerId>
                <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:
          46:29.562</d:orderDate>
                <d:shipCity>Rochester</d:shipCity>
                <d:shipCountry m:null = "true"/>
                <d:version m:type = "Edm.Int32">0</d:version>
            </m:properties>
        </content>
    </entry>
    <entry>
        <category term = "NorthwindGridModel.Order" scheme = "http://
schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
        <id>http://localhost:8080/wxsrestservice/restservice/
```

```
      NorthwindGrid/Order(orderId=5001, customer_customerId='IBM')
    </id>
        <title type = "text"/>
        <updated>2009-12-16T22:53:09.062Z</updated>
        <author>
            <name/>
        </author>
        <link rel = "edit" title = "Order" href = "Order(orderId=5001,
customer_customerId='IBM')"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
       dataservices/related/customer"
       type = "application/atom+xml;type=entry"
       title = "customer" href = "Order(orderId=5001,
         customer_customerId='IBM')/customer"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
         dataservices/related/orderDetails"
       type = "application/atom+xml;type=feed"
       title = "orderDetails" href = "Order(orderId=5001,
         customer_customerId='IBM')/orderDetails"/>
        <content type = "application/xml">
            <m:properties>
                <d:orderId m:type = "Edm.Int32">5001</d:orderId>
                <d:customer_customerId>IBM</d:customer_customerId>
                <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:50:
          11.125</d:orderDate>
                <d:shipCity>Rochester</d:shipCity>
                <d:shipCountry m:null = "true"/>
                <d:version m:type = "Edm.Int32">0</d:version>
            </m:properties>
        </content>
    </entry>
</feed>
```

- Código de respuesta: 200 OK

JSON

- Método: GET
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders
- Cabecera de la solicitud: Accept: application/json
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/json
- Carga útil de la respuesta:

```
{"d":[{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Order(orderId=5000,
   customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
"orderId":5000,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260992789562)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5000,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5000,customer_customerId='IBM')/orderDetails"}}},
{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
     restservice/NorthwindGrid/
   Order(orderId=5001,
   customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
```

```
"orderId":5001,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260993011125)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5001,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5001,customer_customerId='IBM')/orderDetails"}}}]}
```

- Código de respuesta: 200 OK

Recuperación de una propiedad

Se puede utilizar una solicitud RetrievePrimitiveProperty para obtener el valor de una propiedad de una instancia de entidad de eXtreme Scale. El valor de la propiedad se representa en formato XML para las solicitudes AtomPub y como un objeto JSON para las solicitudes JSON en la carga útil de la respuesta. Si desea más detalles sobre la solicitud RetrievePrimitiveProperty, consulte: MSDN: RetrievePrimitiveProperty Request.

El siguiente ejemplo de solicitud RetrievePrimitiveProperty recupera la propiedad contactName de la entidad Customer('IBM').

AtomPub

- Método: GET
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Cabecera de la solicitud: Accept: application/xml
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/atom+xml
- Carga útil de la respuesta:

```
<contactName xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
    John Doe
</contactName>
```

- Código de respuesta: 200 OK

JSON

- Método: GET
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Cabecera de la solicitud: Accept: application/json
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/json
- Carga útil de la respuesta: `{"d":{"contactName":"John Doe"}}`
- Código de respuesta: 200 OK

Recuperación del valor de una propiedad

Se puede utilizar una solicitud RetrieveValue para obtener el valor en bruto de una propiedad de una instancia de entidad de eXtreme Scale. El valor de la propiedad se representa como un valor en bruto en la carga útil de la respuesta. Si el tipo de

entidad es uno de los siguientes, el tipo de medio de la respuesta será "text/plain". De lo contrario, el tipo de medio de la respuesta será "application/octet-stream". Estos tipos son los siguientes:

- Tipos primitivos Java y sus respectivos derivadores
- java.lang.String
- byte[]
- Byte[]
- char[]
- Character[]
- enums
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

Si desea más detalles sobre la solicitud RetrieveValue, consulte: MSDN: RetrieveValue Request.

El siguiente ejemplo de solicitud RetrieveValue recupera el valor en bruto de la propiedad contactName de la entidad Customer('IBM').

- Método de solicitud: GET
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName/$value
- Cabecera de la solicitud: Accept: text/plain
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: text/plain
- Carga útil de la respuesta: John Doe
- Código de respuesta: 200 OK

Recuperación de un enlace

Se puede utilizar una solicitud RetrieveLink para obtener el enlace o los enlaces que representan una asociación a uno o una asociación a muchos. Para la asociación a uno, el enlace es de una instancia de entidad de eXtreme Scale a otra y el enlace se representa en la carga útil de la respuesta. Para la asociación a muchos, los enlaces son de una instancia de entidad de eXtreme Scale a todas las demás de una colección de entidades de eXtreme Scale especificada y la respuesta se representa como un conjunto de enlaces en la carga útil de la respuesta. Si desea más detalles sobre la solicitud RetrieveLink, consulte: MSDN: RetrieveLink Request.

A continuación se ofrece un ejemplo de solicitud RetrieveLink. En este ejemplo, recuperamos la asociación entre la entidad Order(orderId=5000,customer_customerId='IBM') y su cliente. La respuesta muestra el URI de la entidad Customer.

AtomPub

- Método: GET
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- Cabecera de la solicitud: Accept: application/xml
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/xml
- Carga útil de la respuesta:

```
<?xml version="1.0" encoding="utf-8"?>
<uri>http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('IBM')</uri>
```

- Código de respuesta: 200 OK

JSON

- Método: GET
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- Cabecera de la solicitud: Accept: application/json
- Carga útil de la solicitud: Ninguna
- Cabecera de la respuesta: Content-Type: application/json
- Carga útil de la respuesta: {"d":{"uri":"http:\/\/localhost:8080\/wxsrestservice\/restservice\/NorthwindGrid\/Customer('IBM')"}}

Recuperación de metadatos de servicio

Se puede utilizar una solicitud RetrieveServiceMetadata para obtener el documento de lenguaje de definición de esquema conceptual (CSDL), que describe el modelo de datos asociado al servicio de datos REST de eXtreme Scale. Si desea más detalles sobre la solicitud RetrieveServiceMetadata, consulte: MSDN: RetrieveServiceMetadata Request.

Recuperación de un documento de servicio

Se puede utilizar una solicitud RetrieveServiceDocument para recuperar el documento de servicio que describe la colección de recursos expuestos por el servicio de datos REST de eXtreme Scale. Si desea más detalles sobre la solicitud RetrieveServiceDocument, consulte: MSDN: RetrieveServiceDocument Request.

Conceptos relacionados:

"Operaciones con el servicio de datos REST" en la página 272
Después de iniciar el servicio de datos REST de eXtreme Scale, puede utilizar cualquier cliente HTTP para interactuar con él. Se puede utilizar un navegador web, un cliente PHP, un cliente Java o un cliente de WCF Data Services para emitir cualquiera de las operaciones de solicitud soportadas.

"Visión general de los servicios de datos REST" en la página 117
El servicio de datos REST de WebSphere eXtreme Scale es un servicio HTTP Java compatible con Microsoft WCF Data Services (anteriormente ADO.NET Data Services) e implementa el Protocolo de datos abierto (OData). Microsoft WCF Data Services es compatible con esta especificación al utilizar Visual Studio 2008 SP1 y .NET Framework 3.5 SP1.

Tareas relacionadas:

"Acceso a los datos con el servicio de datos REST" en la página 271
Desarrolle las aplicaciones que realizan operaciones con los protocolos del servicio de datos REST.

Solicitudes de inserción con los servicios de datos REST

Se puede utilizar una solicitud InsertEntity para insertar una nueva instancia de entidad de eXtreme Scale, potencialmente con entidades relacionadas nuevas, en el servicio de datos REST de eXtreme Scale.

Solicitud de inserción de entidad

Se puede utilizar una solicitud InsertEntity para insertar una nueva instancia de entidad de eXtreme Scale, potencialmente con entidades relacionadas nuevas, en el servicio de datos REST de eXtreme Scale. Al insertar una entidad, el cliente puede especificar si el recurso o la entidad se deben enlazar automáticamente a otras entidades existentes en el servicio de datos.

El cliente debe incluir la información de enlace necesaria en la representación de la relación asociada en la carga útil de la solicitud.

Además de soportar la inserción de una instancia de EntityType nueva (E1), la solicitud InsertEntity también permite insertar entidades nuevas relacionadas con E1 (descritas por una relación de entidad) en una sola solicitud. Por ejemplo, al insertar una entidad Customer('IBM'), podemos insertar todos los pedidos con Customer('IBM'). Esta forma de solicitud InsertEntity también se conoce como una *inserción profunda*. Con una inserción profunda, las entidades relacionadas se deben representar utilizando la representación en línea de la relación asociada con E1 que identifica el enlace a las entidades relacionadas que se deben insertar.

Las propiedades de la entidad que se deben insertar se especifican en la carga útil de la solicitud. El servicio de datos REST analiza las propiedades y luego se define la propiedad correspondiente para éstas en la instancia de la entidad. Para el formato AtomPub, la propiedad se especifica como un elemento XML <d:PROPERTY_NAME>. Para JSON, la propiedad se especifica como una propiedad de un objeto JSON.

Si falta una propiedad en la carga útil de la solicitud, el servicio de datos REST define como valor de la propiedad de entidad el valor predeterminado java. No obstante, el programa de fondo de base de datos puede rechazar este valor predeterminado, por ejemplo, si la columna no puede tener un valor nulo en la base de datos. Entonces, se devolverá un código de respuesta 500 para indicar un error interno del servidor.

Si se especifican propiedades duplicadas en la carga útil, se utilizará la última propiedad. El servicio de datos REST pasa por alto todos los valores anteriores para el mismo nombre de propiedad.

Si la carga útil contiene una propiedad no existente, el servicio de datos REST devuelve un código de respuesta 400 (Bad Request) para indicar que la solicitud enviada por el cliente era sintácticamente incorrecta.

Si faltan las propiedades clave, el servicio de datos REST devuelve un código de respuesta 400 (Bad Request) para indicar que falta una propiedad clave.

Si la carga útil contiene un enlace a una entidad relacionada con una clave inexistente, el servicio de datos REST devuelve un código de respuesta 404 (Not Found) para indicar que la entidad enlazada no se encuentra.

Si la carga útil contiene un enlace a una entidad relacionada con un nombre de asociación incorrecto, el servicio de datos REST devuelve un código de respuesta 400 (Bad Request) para indicar que el enlace no se encuentra.

Si la carga útil contiene más de un enlace a una relación a uno, se utilizará el último enlace. Todos los enlaces anteriores correspondientes a la misma asociación se pasan por alto.

Si desea más detalles sobre la solicitud InsertEntity, consulte MSDN Library: InsertEntity Request (Biblioteca MSDN: solicitud InsertEntity).

Una solicitud InsertEntity inserta una entidad Customer con la clave 'IBM'.

AtomPub

- Método: POST
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Cabecera de la solicitud: Accept: application/atom+xml Content-Type: application/atom+xml
- Carga útil de la solicitud:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<content type="application/xml">
  <m:properties>
   <d:customerId>Rational</d:customerId>
   <d:city>Rochester</d:city>
   <d:companyName>Rational</d:companyName>
   <d:contactName>John Doe</d:contactName>
   <d:country>USA</d:country>
  </m:properties>
 </content>
</entry>
```

- Cabecera de la respuesta: Content-Type: application/atom+xml
- Carga útil de la respuesta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
```

```
 <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<content type="application/xml">
  <m:properties>
   <d:customerId>Rational</d:customerId>
   <d:city>Rochester</d:city>
   <d:companyName>Rational</d:companyName>
   <d:contactName>John Doe</d:contactName>
   <d:country>USA</d:country>
  </m:properties>
 </content>
</entry>
Cabecera de la respuesta:
Content-Type: application/atom+xml
Carga útil de la respuesta:
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/restservice" xmlns:d =
   "http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m =
     "http://schemas.microsoft.com/
   ado/2007/08/dataservices/metadata" xmlns = "http://www.w3.org/2005/Atom">
    <category term = "NorthwindGridModel.Customer" scheme = "http://schemas.
     microsoft.com/ado/2007/08/dataservices/scheme"/>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
     Customer('Rational')</id>
    <title type = "text"/>
    <updated>2009-12-16T23:25:50.875Z</updated>
    <author>
        <name/>
    </author>
    <link rel = "edit" title = "Customer" href = "Customer('Rational')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/dataservices/related/
    orders" type = "application/atom+xml;type=feed"
    title = "orders" href = "Customer('Rational')/orders"/>
    <content type = "application/xml">
        <m:properties>
            <d:customerId>Rational</d:customerId>
            <d:city>Rochester</d:city>
            <d:companyName>Rational</d:companyName>
            <d:contactName>John Doe</d:contactName>
            <d:country>USA</d:country>
            <d:version m:type = "Edm.Int32">0</d:version>
        </m:properties>
    </content>
</entry>
```

- Código de respuesta: 201 Created

JSON

- Método: POST
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer
- Cabecera de la solicitud: Accept: application/json Content-Type: application/json
- Carga útil de la solicitud:

```
{"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country": "USA",}
```

- Cabecera de la respuesta: Content-Type: application/json
- Carga útil de la respuesta:

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('Rational')",
"type":"NorthwindGridModel.Customer"},
```

```
"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country":"USA",
"version":0,
"orders":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('Rational')/orders"}}}}
```

- Código de respuesta: 201 Created

Solicitud de inserción de enlace

Se puede utilizar una solicitud InsertLink para crear un enlace nuevo entre dos instancias de entidad de eXtreme Scale. El URI de la solicitud se debe resolver como una asociación de eXtreme Scale a muchos. La carga útil de la solicitud contiene un solo enlace que apunta a la entidad de destino de la asociación a muchos.

Si el URI de la solicitud InsertLink representa una asociación a uno, el servicio de datos REST devuelve una respuesta 400 (Bad Request).

Si el URI de la solicitud InsertLink apunta a una asociación que no existe, el servicio de datos REST devuelve una respuesta 404 (Not Found) para indicar que el enlace no se encuentra.

Si la carga útil contiene un enlace con una clave que no existe, el servicio de datos REST devuelve una respuesta 404 (Not Found) para indicar que la entidad enlazada no se encuentra.

Si la carga útil contiene más de un enlace, el servicio de datos Rest de eXtreme Scale analizará el primer enlace. Los enlaces restantes se pasan por alto.

Si desea más detalles sobre la solicitud InsertLink, consulte: MSDN Library: InsertLink Request (Biblioteca MSDN: solicitud InsertLink).

El siguiente ejemplo de solicitud InsertLink crea un enlace de Customer('IBM') a Order(orderId=5000,customer_customerId='IBM').

AtomPub

- Método: POST
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$link/orders
- Cabecera de la solicitud: Content-Type: application/xml
- Carga útil de la solicitud:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5000,customer_customerId='IBM')</uri>
```

- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

JSON

- Método: POST
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$links/orders
- Cabecera de la solicitud: Content-Type: application/json

- Carga útil de la solicitud:

```
{"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId
    =5000,customer_customerId='IBM')"}
```

- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Conceptos relacionados:

"Operaciones con el servicio de datos REST" en la página 272
Después de iniciar el servicio de datos REST de eXtreme Scale, puede utilizar cualquier cliente HTTP para interactuar con él. Se puede utilizar un navegador web, un cliente PHP, un cliente Java o un cliente de WCF Data Services para emitir cualquiera de las operaciones de solicitud soportadas.

"Visión general de los servicios de datos REST" en la página 117
El servicio de datos REST de WebSphere eXtreme Scale es un servicio HTTP Java compatible con Microsoft WCF Data Services (anteriormente ADO.NET Data Services) e implementa el Protocolo de datos abierto (OData). Microsoft WCF Data Services es compatible con esta especificación al utilizar Visual Studio 2008 SP1 y .NET Framework 3.5 SP1.

Tareas relacionadas:

"Acceso a los datos con el servicio de datos REST" en la página 271
Desarrolle las aplicaciones que realizan operaciones con los protocolos del servicio de datos REST.

Solicitudes de actualización con los servicios de datos REST

El servicio de datos REST de WebSphere eXtreme Scale soporta solicitudes de actualización de entidades, propiedades primitivas de entidades, etc.

Actualización de una entidad

Se puede utilizar una solicitud UpdateEntity para actualizar una entidad existente de eXtreme Scale. El cliente puede utilizar un método HTTP PUT para sustituir una entidad existente de eXtreme Scale o utilizar un método HTTP MERGE para fusionar los cambios en una entidad existente de eXtreme Scale.

Al actualizar la entidad, el cliente puede especificar si la entidad, además de actualizarse, se debe enlazar automáticamente a otras entidades existentes del servicio de datos que se relacionan mediante asociaciones a uno de un solo valor.

La propiedad de la entidad que se debe actualizar se encuentra en la carga útil de la solicitud. El servicio de datos REST analiza la propiedad y luego se define la propiedad correspondiente para ésta en la entidad. Para el formato AtomPub, la propiedad se especifica como un elemento XML `<d:PROPERTY_NAME>`. Para JSON, la propiedad se especifica como una propiedad de un objeto JSON.

Si falta una propiedad en la carga útil de solicitud, el servicio de datos REST establece el valor de propiedad de entidad en el valor predeterminado Java para el método HTTP PUT. No obstante, el programa de fondo de base de datos puede rechazar este valor predeterminado si, por ejemplo, la columna no puede tener un valor nulo en la base de datos. Entonces se devuelve un código de respuesta de 500 (Error interno de servidor) para indicar un error interno de servidor. Si falta una propiedad en la carga útil de solicitud HTTP MERGE, el servicio de datos REST no cambia el valor de propiedad existente.

Si se han especificado propiedades duplicadas en la carga útil, se utiliza la última propiedad. El servicio de datos REST pasa por alto todos los valores anteriores con el mismo nombre de propiedad.

Si la carga útil contiene una propiedad no existente, el servicio de datos REST devuelve un código de respuesta 400 (Bad Request) para indicar que la solicitud enviada por el cliente era sintácticamente incorrecta.

Como parte de la serialización de un recurso, si la carga útil de una solicitud de actualización contiene cualquiera de las propiedades clave para la entidad, el servicio de datos REST pasa por alto dichos valores clave porque las claves de entidad son inmutables.

Si desea detalles sobre la solicitud UpdateEntity, consulte: MSDN Library: UpdateEntity Request (Biblioteca MSDN: solicitud UpdateEntity).

Una solicitud UpdateEntity actualiza el nombre de ciudad de Customer('IBM') a 'Raleigh'.

AtomPub

- Método: PUT
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Cabecera de la solicitud: Content-Type: application/atom+xml
- Carga útil de la solicitud:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
 <title />
 <updated>2009-07-28T21:17:50.609Z</updated>
 <author>
  <name />
 </author>
 <id />
 <content type="application/xml">
  <m:properties>
   <d:customerId>IBM</d:customerId>
   <d:city>Raleigh</d:city>
   <d:companyName>IBM Corporation</d:companyName>
   <d:contactName>Big Blue</d:contactName>
   <d:country>USA</d:country>
  </m:properties>
 </content>
</entry>
```

- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

JSON

- Método: PUT
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Cabecera de la solicitud: Content-Type: application/json
- Carga útil de la solicitud:

```
{"customerId":"IBM",
"city":"Raleigh",
"companyName":"IBM Corporation",
"contactName":"Big Blue",
"country":"USA",}
```

- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Actualización de una propiedad primitiva de entidad

La solicitud UpdatePrimitiveProperty puede actualizar un valor de propiedad de una entidad de eXtreme Scale. La propiedad y el valor que se deben actualizar están en la carga útil de la solicitud. La propiedad no puede ser una propiedad clave porque eXtreme Scale no permite que los clientes cambien claves de entidad.

Si desea más detalles sobre la solicitud UpdatePrimitiveProperty, consulte: MSDN Library: UpdatePrimitiveProperty Request (Biblioteca MSDN: solicitud UpdatePrimitiveProperty).

A continuación se ofrece un ejemplo de solicitud UpdatePrimitiveProperty. En este ejemplo, actualizamos el nombre de la ciudad de Customer('IBM') a 'Raleigh'.

AtomPub

- Método: PUT
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city
- Cabecera de la solicitud: Content-Type: application/xml
- Carga útil de la solicitud:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<city xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
 Raleigh
</city>
```

- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

JSON

- Método: PUT
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city
- Cabecera de la solicitud: Content-Type: application/json
- Carga útil de la solicitud: {"city":"Raleigh"}
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Actualización de un valor de propiedad primitiva de entidad

La solicitud UpdateValue puede actualizar un valor de propiedad si procesar de una entidad de eXtreme Scale. El valor que se debe actualizar se representa como un valor en bruto en la carga útil de la solicitud. La propiedad no puede ser una propiedad clave porque eXtreme Scale no permite que los clientes cambien claves de entidad.

El tipo de contenido de la solicitud puede ser “text/plain” o “application/octet-stream”, según el tipo de propiedad. Para obtener más información, consulte “Recuperación de elementos que no sean entidades con los servicios de datos REST” en la página 285.

Si desea más detalles sobre la solicitud UpdateValue, consulte: MSDN Library: UpdateValue Request (Biblioteca MSDN: solicitud UpdateValue).

A continuación se ofrece un ejemplo de solicitud UpdateValue. En este ejemplo, actualice el nombre de ciudad de Customer('IBM') a 'Raleigh'.

- Método: PUT
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city/$value
- Cabecera de la solicitud: Content-Type: text/plain
- Carga útil de la solicitud: Raleigh
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Actualización de un enlace

La solicitud UpdateLink se puede utilizar para establecer una asociación entre dos instancias de entidad de eXtreme Scale. La asociación puede ser una relación de un solo valor (a uno) o una relación de varios valores (a muchos).

La actualización de un enlace entre dos instancias de entidad de eXtreme Scale puede establecer asociaciones o eliminar asociaciones. Por ejemplo, si el cliente establece una asociación a uno entre una entidad `Order(orderId=5000,customer_customerId='IBM')` y la instancia `Customer('ALFKI')`, tiene que disociar la entidad `Order(orderId=5000,customer_customerId='IBM')` y la entidad de la instancia Customer asociada actualmente.

Si cualquiera de las instancias de entidad especificadas en la solicitud UpdateLink no se encuentra, el servicio de datos REST devuelve una respuesta 404 (Not Found).

Si el URI de la solicitud UpdateLink especifica una asociación inexistente, el servicio de datos REST devuelve una respuesta 404 (Not Found) para indicar que el enlace no se encuentra.

Si el URI especificado en la carga útil de la solicitud UpdateLink no se resuelve en la misma entidad o en la misma clave especificada en el URI, si existe, entonces el servicio de datos REST de eXtreme Scale devuelve una respuesta 400 (Bad Request).

Si la carga útil de solicitud UpdateLink contiene varios enlaces, el servicio de datos REST sólo analiza el primer enlace. Los enlaces restantes se pasan por alto.

Si desea más detalles sobre la solicitud UpdateLink, consulte: MSDN Library: UpdateLink Request (Biblioteca MSDN: solicitud UpdateLink).

A continuación se ofrece un ejemplo de solicitud UpdateLink. En este ejemplo, actualizamos la relación de cliente de la entidad `Order(orderId=5000,customer_customerId='IBM')` y de Customer('IBM') a Customer('IBM').

Recuerde: El ejemplo anterior sólo es ilustrativo. Como todas las asociaciones son normalmente asociaciones de claves para una cuadrícula particionada, el enlace no se puede cambiar.

AtomPub

- Método: PUT
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer
- Cabecera de la solicitud: Content-Type: application/xml
- Carga útil de la solicitud:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>
 http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
</uri>
```

- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

JSON

- Método: PUT
- URI de solicitud: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer`
- Cabecera de la solicitud: Content-Type: application/xml
- Carga útil de solicitud: `{"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')"}`
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Conceptos relacionados:

"Operaciones con el servicio de datos REST" en la página 272
Después de iniciar el servicio de datos REST de eXtreme Scale, puede utilizar cualquier cliente HTTP para interactuar con él. Se puede utilizar un navegador web, un cliente PHP, un cliente Java o un cliente de WCF Data Services para emitir cualquiera de las operaciones de solicitud soportadas.

"Visión general de los servicios de datos REST" en la página 117
El servicio de datos REST de WebSphere eXtreme Scale es un servicio HTTP Java compatible con Microsoft WCF Data Services (anteriormente ADO.NET Data Services) e implementa el Protocolo de datos abierto (OData). Microsoft WCF Data Services es compatible con esta especificación al utilizar Visual Studio 2008 SP1 y .NET Framework 3.5 SP1.

Tareas relacionadas:

"Acceso a los datos con el servicio de datos REST" en la página 271
Desarrolle las aplicaciones que realizan operaciones con los protocolos del servicio de datos REST.

Solicitudes de supresión con los servicios de datos REST

El servicio de datos REST de WebSphere eXtreme Scale suprime entidades, valores de propiedad y enlaces.

Supresión de una entidad

La solicitud DeleteEntity suprime entidades de eXtreme Scale del servicio de datos REST.

Si cualquier relación con la entidad que se debe suprimir tiene definida la supresión en cascada, el servicio de datos REST de eXtreme Scale suprimirá la entidad o las entidades relacionadas.Si desea más detalles sobre la solicitud DeleteEntity, consulte: MSDN Library: DeleteEntity Request (Biblioteca MSDN: solicitud DeleteEntity).

La siguiente solicitud DeleteEntity suprime el cliente con la clave 'IBM'.

- Método: DELETE
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Carga útil de la solicitud: Ninguna
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Supresión del valor de una propiedad

La solicitud DeleteValue define una propiedad de entidad de eXtreme Scale como nula.

Cualquier propiedad de una entidad de eXtreme Scale se puede definir como nula con una solicitud DeleteValue. Para definir una propiedad como nula, asegúrese de todo lo siguiente:

- Para cualquier tipo de número primitivo y su derivador, BigInteger, o BigDecimal, el valor de la propiedad se define como 0.
- Para el tipo Boolean o boolean, el valor de la propiedad se define como false.
- Para el tipo char o Character, el valor de la propiedad se define con el carácter #X1 (NIL).
- Para el tipo enum, el valor de la propiedad se define con el valor enum con el ordinal 0.
- Para todos los demás tipos, el valor de la propiedad se define como nulo.

No obstante, el programa de fondo de base de datos puede rechazar una solicitud de supresión de este tipo si, por ejemplo, la propiedad no puede tener un valor nulo en la base de datos. En este caso, el servicio de datos REST devuelve una respuesta 500 (Internal Server Error). Si desea más detalles sobre la solicitud DeleteValue, consulte: MSDN Library: DeleteValue Request (Biblioteca MSDN: solicitud DeleteValue).

A continuación se ofrece un ejemplo de solicitud DeleteValue. En este ejemplo, definimos como nulo el nombre de contacto de Customer('IBM').

- Método: DELETE
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Carga útil de la solicitud: Ninguna
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Supresión de un enlace

La solicitud DeleteLink puede eliminar una asociación entre dos instancias de entidad de eXtreme Scale. La asociación puede ser una relación a uno o una relación a muchos.No obstante, el programa de fondo de base de datos puede rechazar una solicitud de supresión de este tipo si, por ejemplo, la restricción de clave foránea está definida. En este caso, el servicio de datos REST devuelve una respuesta 500 (Internal Server Error).Si desea más detalles sobre la solicitud DeleteLink, consulte: MSDN Library: DeleteLink Request (Biblioteca MSDN: solicitud DeleteLink).

La siguiente solicitud DeleteLink elimina la asociación entre Order(101) y su Customer asociado.

- Método: DELETE
- URI de la solicitud: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer
- Carga útil de la solicitud: Ninguna
- Carga útil de la respuesta: Ninguna
- Código de respuesta: 204 No Content

Conceptos relacionados:

"Operaciones con el servicio de datos REST" en la página 272
Después de iniciar el servicio de datos REST de eXtreme Scale, puede utilizar cualquier cliente HTTP para interactuar con él. Se puede utilizar un navegador web, un cliente PHP, un cliente Java o un cliente de WCF Data Services para emitir cualquiera de las operaciones de solicitud soportadas.

"Visión general de los servicios de datos REST" en la página 117
El servicio de datos REST de WebSphere eXtreme Scale es un servicio HTTP Java compatible con Microsoft WCF Data Services (anteriormente ADO.NET Data Services) e implementa el Protocolo de datos abierto (OData). Microsoft WCF Data Services es compatible con esta especificación al utilizar Visual Studio 2008 SP1 y .NET Framework 3.5 SP1.

Tareas relacionadas:

"Acceso a los datos con el servicio de datos REST" en la página 271
Desarrolle las aplicaciones que realizan operaciones con los protocolos del servicio de datos REST.

Plug-ins y API del sistema

Un plug-in es un componente que proporciona una función a los componentes que se pueden conectar, que incluyen ObjectGrid y BackingMap. Para utilizar eXtreme Scale de forma más eficaz como una cuadrícula de datos en memoria o un espacio de proceso de base de datos, debe determinar con atención cómo puede maximizar mejor el rendimiento con los plug-ins disponibles.

Gestión de ciclos de vida de plug-ins

Puede gestionar ciclos de vida de plug-ins con métodos especializados para cada plug-in, que están disponibles para su invocación en puntos funcionales designados. Tanto el método initialize como el método destroy definen el ciclo de vida de los plug-ins, que controlan sus objetos *owner*. Un objeto propietario es el objeto que utiliza realmente el plug-in determinado. Un propietario puede ser un cliente de cuadrícula, un servidor o una correlación de respaldo.

Acerca de esta tarea

De forma similar, todos los plug-ins pueden implementar las interfaces mixin opcionales adecuadas para su objeto de propietario. Cualquier plug-in ObjectGrid puede implementar la interfaz mixin opcional ObjectGridPlugin. Cualquier plug-in BackingMap puede implementar la interfaz mixin opcional BackingMapPlugin. Las interfaces mixin opcionales requieren implementación de varios métodos aparte de los métodos initialize() y destroy() para los plug-ins básicos. Para obtener más información sobre estas interfaces, consulte la documentación de la API.

Cuando están inicializando objetos de propietario, estos objetos establecen atributos en el plug-in y a continuación invocan el método initialize de los plug-ins de los que son propietarios. Durante el ciclo de destrucción de objetos de propietario, en consecuencia se invoca también el método destroy de los plug-ins. Si desea información detallada sobre los métodos initialize y destroy, junto con otros métodos con capacidad para cada uno de los plug-ins, consulte los temas correspondientes para cada plug-in.

Como ejemplo, considere un entorno distribuido. Tanto los ObjectGrids del lado del cliente, como los ObjectGrids del lado del servidor pueden tener sus propios plug-ins. El ciclo de vida de un ObjectGrid del lado del cliente y, por lo tanto, sus instancias de plug-in, son independientes de todas las instancias de plug-in y del ObjectGrid del lado del servidor.

En una topología distribuida de este tipo, suponga que tiene un ObjectGrid denominado `myGrid` definido en el archivo `objectGrid.xml` y configurado con un ObjectGridEventListener personalizado denominado myObjectGridEventListener. El archivo `objectGridDeployment.xml` define la política de despliegue del ObjectGrid `myGrid`. Los archivos `objectGrid.xml` y `objectGridDeployment.xml` se utilizan para iniciar los servidores de contenedor. Durante el inicio del servidor de contenedor, la instancia del ObjectGrid `myGrid` del lado del servidor se inicializa. Entretanto, se invoca el método initialize de la instancia de myObjectGridEventListener de la que es propietaria la instancia de myObjectGrid. Una vez que se ha iniciado el servidor de contenedor, las aplicaciones se pueden conectar a la instancia del ObjectGrid `myGrid` del lado del servidor y obtener una instancia del lado del cliente.

Al obtener la instancia del ObjectGrid myGrid del lado del cliente, la instancia de `myGrid` del lado del cliente pasa por su propio ciclo de inicialización e invoca el método initialize de su instancia de myObjectGridEventListener del lado del cliente. Esta instancia de myObjectGridEventListener del lado del cliente es independiente de la instancia de myObjectGridEventListener del lado del servidor. Su ciclo de vida lo controla su propietario, que es la instancia del ObjectGrid `myGrid` del lado del cliente.

Si la aplicación desconecta o destruye la instancia del ObjectGrid `myGrid` del lado del cliente, el método destroy que pertenece a la instancia de myObjectGridEventListener del lado del cliente se invoca automáticamente. Sin embargo, este proceso no tiene ningún impacto en la instancia de myObjectGridEventListener del lado del servidor. El método destroy de la instancia de myObjectGridEventListener del lado del servidor solo se puede invocar durante el ciclo de vida de destrucción de la instancia de ObjectGrid `myGrid` del lado del servidor, al detener un servidor de contenedor. Específicamente, al detener un servidor de contenedor, las instancias de ObjectGrid contenidas se destruyen y se invoca el método destroy de todos los plug-ins de los que se es propietario.

Aunque el ejemplo anterior se aplica específicamente al caso de una instancia de cliente y servidor de un ObjectGrid, el propietario de un plug-in también puede ser una interfaz BackingMap. Además, determine cuidadosamente las configuraciones de los plug-ins que podría escribir, en función de estas consideraciones de ciclo de vida. Utilice los temas siguientes para escribir plug-ins para proporcionar sucesos de gestión de ciclo de vida ampliados que puede utilizar para configurar o eliminar recursos del entorno:

Conceptos relacionados:

"Visión general de la infraestructura OSGi" en la página 37
OSGi define un sistema de módulo dinámico para Java. La plataforma de servicio OSGi tiene una arquitectura por capas, y está diseñada para ejecutarse en diversos perfiles Java estándar. Puede iniciar servidores y clientes de WebSphere eXtreme Scale en un contenedor OSGi.

Información relacionada:

Documentación de la API

Escritura de un plug-in ObjectGridPlugin

Un ObjectGridPlugin es una interfaz mixin opcional que puede utilizar para proporcionar sucesos de gestión de ciclo de vida ampliados a todos los demás plug-in de ObjectGrid.

Acerca de esta tarea

Cualquier plug-in de ObjectGrid que implementa ObjectGridPlugin recibe el conjunto ampliado de sucesos de ciclo de vida y puede proporcionar más control, que se puede utilizar para configurar o eliminar recursos. En un contenedor para una cuadrícula de datos particionada, habrá una instancia de ObjectGrid (el propietario del plug-in) para cada partición gestionada por el contenedor. Cuando se eliminan particiones individuales, los recursos que utiliza esa instancia de ObjectGrid también se deben eliminar. Por lo tanto, es posible que deba cerrar o finalizar un recurso como, por ejemplo, un archivo de configuración abierto o una hebra en ejecución gestionada por un plug-in, cuando se elimine la partición propietaria de dicho recurso.

La interfaz ObjectGridPlugin proporciona métodos para establecer o modificar el estado del plug-in, así como métodos para inspeccionar el estado actual del plug-in. Todos los métodos se deben implementar correctamente y el entorno de ejecución de WebSphere eXtreme Scale verifica el comportamiento del método en determinadas circunstancias. Por ejemplo, después de llamar al método initialize(), el entorno de ejecución de eXtreme Scale llama al método isInitialized() para garantizar que el método ha completado satisfactoriamente la inicialización adecuada.

Procedimiento

1. Implemente la interfaz ObjectGridPlugin de forma que el plug-in ObjectGridPlugin reciba notificaciones sobre sucesos significativos de eXtreme Scale. Existen tres categorías principales de métodos:

Métodos de propiedades	Finalidad
setObjectGrid()	Se llama para establecer la instancia de ObjectGrid para la que se utiliza el plug-in.
getObjectGrid()	Se llama para obtener o confirmar la instancia de ObjectGrid para la que se utiliza el plug-in.

Métodos de inicialización	Finalidad
initialize()	Se llama para inicializar el ObjectGridPlugin.
isInitialized()	Se llama para obtener o confirmar el estado de inicialización del plug-in.

Métodos de destrucción	Finalidad
destroy()	Se llama para destruir el ObjectGridPlugin.
isDestroyed()	Se llama para obtener o confirmar el estado destruido del plug-in.

Consulte la documentación de la API para obtener más información sobre estas interfaces.

2. Configure un plug-in ObjectGridPlugin con XML. Utilice la clase com.company.org.MyObjectGridPluginTxCallback, que implementa la interfaz TransactionCallback y la interfaz ObjectGridPlugin.

   En el siguiente ejemplo de código, la devolución de llamada de transacción personalizada, que en última instancia recibirá los sucesos de ciclo de vida ampliados, se genera y añade a un ObjectGrid.

   Importante: La interfaz TransactionCallback ya tiene un método inicialize, se añade un nuevo método initialize así como el método destroy y otros métodos ObjectGridPlugin. Se utiliza cada uno de los métodos y los métodos initialize sólo realizan la inicialización una vez. El siguiente XML crea una configuración que utiliza la interfaz TransactionCallback ampliada.

   El siguiente texto debe aparecer en el archivo `myGrid.xml`:

   ```
   ?xml version="1.0" encoding="UTF-8"?>
   <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
       <objectGrids>
           <objectGrid name="myGrid">
               <bean id="TransactionCallback"
                     className="com.company.org.MyObjectGridPluginTxCallback" />
               <backingMap name="Book"/>
           </objectGrid>
       </objectGrids>
   </objectGridConfig>
   ```

   Tenga en cuenta que las declaraciones de bean preceden a las declaraciones `backingMap`.

3. Proporcione el archivo `myGrid.xml` al plug-in ObjectGridManager para facilitar la creación de esta configuración.

Tareas relacionadas:

"Cómo escribir un plug-in BackingMapPlugin"
Un plug-in BackingMap implementa la interfaz mixin BackingMapPlugin, que se puede utilizar para recibir prestaciones ampliadas para la gestión de su ciclo de vida.

Información relacionada:

../com.ibm.websphere.extremescale.javadoc.doc/topics/com/ibm/websphere/objectgrid/management/package-summary.html

Cómo escribir un plug-in BackingMapPlugin

Un plug-in BackingMap implementa la interfaz mixin BackingMapPlugin, que se puede utilizar para recibir prestaciones ampliadas para la gestión de su ciclo de vida.

Acerca de esta tarea

Cualquier plug-in BackingMap existente que implemente también la interfaz BackingMapPlugin recibirá automáticamente el conjunto ampliado de sucesos de ciclo de vida durante su construcción y uso.

La interfaz BackingMapPlugin proporciona métodos para establecer o modificar el estado del plug-in, así como métodos para inspeccionar el estado actual del plug-in.

Todos los métodos se deben implementar correctamente y el entorno de ejecución de WebSphere eXtreme Scale verifica el comportamiento del método en determinadas circunstancias. Por ejemplo, después de llamar al método initialize(), el entorno de ejecución de eXtreme Scale llama al método isInitialized() para garantizar que el método ha completado satisfactoriamente la inicialización adecuada.

Procedimiento

1. Implemente la interfaz BackingMapPlugin de forma que el plug-in BackingMapPlugin reciba notificaciones sobre sucesos significativos de eXtreme Scale. Existen tres categorías principales de métodos:

Métodos de propiedades	Finalidad
setBackingMap()	Se llama para establecer la instancia de BackingMap para la que se utiliza el plug-in.
getBackingMap()	Se llama para obtener o confirmar la instancia BackingMap para la que se utiliza el plug-in.

Métodos de inicialización	Finalidad
initialize()	Se llama para inicializar el plug-in BackingMapPlugin.
isInitialized()	Se llama para obtener o confirmar el estado de inicialización del plug-in.

Métodos de destrucción	Finalidad
destroy()	Se llama para destruir el plug-in BackingMapPlugin.
isDestroyed()	Se llama para obtener o confirmar el estado destruido del plug-in.

   Consulte la documentación de la API para obtener más información sobre estas interfaces.

2. Configure un plug-in BackingMapPlugin con XML. Supongamos que el nombre de clase de un plug-in eXtreme Scale Loader es la clase com.company.org.MyBackingMapPluginLoader, que implementa la interfaz Loader y la interfaz BackingMapPlugin.

   En el siguiente ejemplo de código, la devolución de llamada de transacción personalizada, que en última instancia recibirá los sucesos de ciclo de vida ampliados, se genera y añade a una BackingMap.

   También puede configurar un plug-in BackingMapPlugin mediante XML. 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">
            <backingMap name="Book" pluginCollectionRef="myPlugins" />
        </objectGrid>
    </objectGrids>
```

```
    <backingMapPluginCollections>
        <backingMapPluginCollection id="myPlugins">
            <bean id="Loader"
                  className="com.company.org.MyBackingMapPluginLoader" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

3. Proporcione el archivo `myGrid.xml` al plug-in ObjectGridManager para facilitar la creación de esta configuración.

Resultados

La instancia BackingMap que se crea tiene un cargador que recibe los sucesos de ciclo de vida de BackingMapPlugin.

Tareas relacionadas:

"Escritura de un plug-in ObjectGridPlugin" en la página 303
Un ObjectGridPlugin es una interfaz mixin opcional que puede utilizar para proporcionar sucesos de gestión de ciclo de vida ampliados a todos los demás plug-in de ObjectGrid.

Información relacionada:

../com.ibm.websphere.extremescale.javadoc.doc/topics/com/ibm/websphere/objectgrid/management/package-summary.html

Plug-ins para réplica multimaestro

Considere transformar los objetos almacenados en memoria caché para aumentar el rendimiento de la memoria caché. Puede utilizar el plug-in de ObjectTransformer cuando el uso del procesador es elevado. Se invierte hasta un 60 o 70 por ciento del tiempo total del procesador en serializar y copiar las entradas. Mediante la implementación del plug-in de ObjectTransformer, puede serializar y deserializar los objetos con su propia implementación. Puede utilizar un plug-in de CollisionArbiter para definir cómo se tratan las colisiones de cambio en los dominios.

Desarrollo de árbitros personalizados para la réplica con varios maestros

Se podrían producir colisiones de cambio si se pueden cambiar en dos lugares a la vez los mismos registros. En una topología de réplica multimaestro, los dominios de servicio de catálogo detectan automáticamente las colisiones. Cuando el dominio de servicio de catálogo detecta una colisión, invoca un árbitro. Normalmente, las colisiones se resuelven con el árbitro de colisión predeterminado. No obstante, una aplicación puede proporcionar un árbitro de colisión personalizado.

Antes de empezar

- Consulte "Planificación de topologías de varios centros de datos" en la página 99 para obtener más información sobre la planificación y el diseño de la topología de réplica multimaestro.
- Consulte Configuración de topologías de varios centros de datos para obtener más información sobre cómo configurar enlaces entre dominios de servicio de catálogo.

Acerca de esta tarea

Si un dominio de servicio de catálogo recibe una entrada replicada que colisiona con un registro de colisión, el árbitro predeterminado utiliza los cambios del

dominio de servicio de catálogo nombrado léxicamente inferior. Por ejemplo, si el dominio A y B generan un conflicto de un registro, el cambio del dominio B se pasa por alto. El dominio A mantiene su versión y el registro del dominio B se modifica para que coincida con el registro del dominio A. Los nombres de dominio se convierten a mayúsculas para la comparación.

Una opción alternativa para la topología de réplica con varios maestros es llamar en un plug-in de colisión personalizado para determinar el resultado. Estas instrucciones esbozan cómo desarrollar un árbitro de colisión personalizado y configurar una topología de réplica con varios maestros para utilizarla.

Procedimiento

1. Desarrolle un árbitro de colisión personalizado e intégrelo en la aplicación.

   La clase debe implementar la interfaz:

   `com.ibm.websphere.objectgrid.revision.CollisionArbiter`

   Un plug-in de colisión tiene tres opciones para determinar el resultado de una colisión. Puede seleccionar la copia local o puede proporcionar una versión revisada de la entrada. Un dominio de servicio de catálogo proporciona la siguiente información a un árbitro de colisión personalizado:

   - La versión existente del registro
   - La versión de colisión del registro
   - Un objeto Session que se debe utilizar para crear la versión revisada de la entrada que ha entrado en colisión

   El método del plug-in devuelve un objeto que indica su decisión. El método invocado por el dominio para llamar al plug-in debe devolver true o false, donde false indica que se ignora la colisión. Cuando se ignora la colisión, la versión local permanece sin cambios y el árbitro olvida que alguna vez vió la versión existente. El método devuelve un valor true si el método ha utilizado la sesión proporcionada para crear una versión nueva fusionada del registro, con lo que se reconcilia el cambio.

2. En el archivo `objectgrid.xml`, especifique el plug-in de árbitro personalizado.

   El ID debe ser `CollisionArbiter`.

```
<dgc:objectGrid name="revisionGrid" txTimeout="10">
     <dgc:bean className="com.you.your_application.
      CustomArbiter" id="CollisionArbiter">
      <dgc:property name="property" type="java.lang.String"
      value="propertyValue"/>
     </dgc:bean>
</dgc:objectGrid>
```

Conceptos relacionados:

"Planificación de topologías de varios centros de datos" en la página 99
Mediante la utilización de la réplica asíncrona multimaestro, dos o más cuadrículas de datos pueden convertirse en copias exactas entre ellas. Cada cuadrícula de datos está alojada en un dominio de servicio de catálogo independiente, con su propio servicio de catálogo, servidores de contenedor y un nombre exclusivo. Con la réplica asíncrona multimaestro, puede utilizar enlaces para conectar una colección de dominios de servicio de catálogo. A continuación, los dominios de servicio de catálogo se sincronizan utilizando la réplica mediante los enlaces. Puede construir casi cada topología mediante la definición de enlaces entre los dominios de servicio de catálogo.

"Topologías para réplica multimaestro" en la página 100
Tiene diversas opciones cuando elige una topología para el despliegue que incorpora réplica multimaestro.

"Consideraciones sobre la configuración para topologías multimaestro" en la página 105
Considere los puntos siguientes cuando decida si desea utilizar topologías de réplica multimaestro y cómo utilizarlas.

"Consideraciones sobre el diseño para la réplica multimaestro" en la página 108
Al implementar la réplica multimaestro, debe tener en cuenta aspectos del diseño como los siguientes: arbitraje, enlace y rendimiento.

"Consideraciones sobre el cargador en una topología multimaestro" en la página 106
Cuando se utilizan cargadores en una topología multimaestro, debe considerar los posibles retos de mantenimiento de la información de revisión y colisión. La cuadrícula de atos mantiene información de revisión sobre los elementos de la cuadrícula de datos de forma que se pueden detectar las colisiones cuando otros fragmentos primarios de la configuración graban entradas en la cuadrícula de datos. Cuando se añaden entradas desde un cargador, esta información de revisión no se incluye y la entrada asume una revisión nueva. Debido a que la revisión de la entrada parece una inserción nueva, se produciría una falta colisión si otro fragmento primario también cambia este estado u obtiene la misma información de un cargador.

Plug-ins para el mantenimiento de versiones y la comparación de objetos de memoria caché

Utilice el plug-in OptimisticCallback para personalizar las operaciones de creación de versiones y comparación de los objetos de la memoria caché cuando se utiliza la estrategia de bloqueo optimista.

Puede proporcionar un objeto de devolución de llamada optimista conectable que implementa la interfaz com.ibm.websphere.objectgrid.plugins.OptimisticCallback. En el caso de correlaciones de entidad, se configura automáticamente un plug-in OptimisticCallback de alto rendimiento.

Finalidad

Utilice la interfaz OptimisticCallback para proporcionar operaciones de comparación optimista para los valores de una correlación. Es necesario un plug-in OptimisticCallback al utilizar la estrategia de bloqueo optimista. El producto proporciona una implementación de OptimisticCallback predeterminada. Sin embargo, por lo general, la aplicación debe conectar su propia implementación de la interfaz OptimisticCallback.

Implementación predeterminada

La infraestructura de eXtreme Scale proporciona una implementación predeterminada de la interfaz OptimisticCallback que se utiliza si la aplicación no conecta un objeto OptimisticCallback proporcionado para la aplicación. La implementación predeterminada siempre devuelve el valor especial de NULL_OPTIMISTIC_VERSION como el objeto de versión del valor y nunca actualiza el objeto de versión. Esta acción hace que la comparación optimista sea una función "sin operación". En la mayoría de los casos, conviene que no se produzca la función "sin operación" cuando utilice la estrategia de bloqueo optimista. Las aplicaciones deben implementar la interfaz y conectar sus propias implementaciones OptimisticCallback de modo que no se utilice la implementación predeterminada. No obstante, existe un escenario donde resulta útil la implementación OptimisticCallback predeterminada. Observe la situación siguiente:

- Se ha conectado un cargador para la correlación de respaldo.
- El cargador sabe cómo realizar la comparación optimista sin ayuda de un plug-in OptimisticCallback.

¿Cómo puede realizar el cargador una creación de versiones optimista sin la ayuda de un objeto OptimisticCallback? El cargador conoce el objeto de clase de valor y sabe qué campo del objeto de valor se utiliza como valor de creación de versiones optimista. Por ejemplo, imagine que se utiliza la interfaz siguiente para el objeto de valor de la correlación de empleados.

```
public interface Employee
{
    // Número de secuencia utilizado para la creación de versiones optimista.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Otros métodos get/set para otros campos del objeto Employee.
}
```

En este ejemplo, el cargador sabe que puede utilizar el método getSequenceNumber para obtener la información de la versión actual para un objeto de valor Employee. El cargador incrementa el valor devuelto para generar un nuevo número de versión antes de actualizar el almacenamiento persistente con el nuevo valor Employee. Para un cargador JDBC (Java DataBase Connectivity), se utiliza el número de secuencia actual de la cláusula WHERE de una sentencia de SQL UPDATE sobrecualificada, y utiliza el nuevo número de secuencia generado para establecer la columna de número de secuencia en el nuevo valor de número de secuencia. Otra posibilidad es que el cargador utilice una función, que proporciona el programa de fondo, que actualiza automáticamente una columna oculta que puede utilizarse para la creación de versiones optimista.

En algunas situaciones, posiblemente se puede utilizar un procedimiento almacenado o un desencadenante que ayuda a mantener una columna que aloja información sobre la creación de versiones. Si el cargador utiliza una de estas técnicas para mantener la información de la creación de versiones optimista, la aplicación no necesita proporcionar una implementación OptimisticCallback. Se puede utilizar la implementación predeterminada de OptimisticCallback en este escenario porque el cargador puede manejar la creación de versiones optimista sin la ayuda de un objeto OptimisticCallback.

Implementación predeterminada de entidades

Las entidades se almacenan en ObjectGrid mediante objetos de tuple. El comportamiento predeterminado de la implementación OptimisticCallback es similar al comportamiento para las correlaciones sin entidades. Sin embargo, el

campo de versión de la entidad se identifica a través del uso de la anotación @Version o el atributo de versión en el archivo XML de descriptor de la entidad.

El atributo de versión puede ser de uno de los tipos siguientes: int, Integer, short, Short, long, Long o java.sql.Timestamp. Una entidad sólo debe tener un atributo de versión definido. Establezca el atributo de versión sólo durante la construcción. Después de persistir la entidad, el valor del atributo de versión no se debe modificar.

Si no se configura el atributo de versión y se utiliza la estrategia de bloqueo optimista, se crea una versión implícitamente de todo el tuple mediante el estado completo del tuple, que es más costoso.

En el ejemplo siguiente, la entidad Employee tiene un atributo de versión de tipo long denominado SequenceNumber:

```
@Entity
public class Employee
{
private long sequence;
    // Número de secuencia utilizado para la creación de versiones optimista.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Otros métodos get/set para otros campos del objeto Employee.
}
```

Escritura de un plug-in OptimisticCallback

Un plug-in OptimisticCallback debe implementar la interfaz OptimisticCallback y seguir los convenios comunes de plug-in de ObjectGrid. Consulte la interfaz OptimisticCallback en la documentación de la API si desea más información.

En la lista siguiente se proporciona una descripción o consideración de cada uno de los métodos de la interfaz OptimisticCallback:

NULL_OPTIMISTIC_VERSION

Este valor especial es devuelto por el método getVersionedObjectForValue si la implementación de OptimisticCallback no requiere ninguna comprobación de versiones. La implementación del plug-in incorporada de la clase com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback utiliza este valor porque la creación de versiones está inhabilitada cuando se especifica esta implementación de plug-in.

Método getVersionedObjectForValue

El método getVersionedObjectForValue podría devolver una copia del valor o un atributo del valor que se puede utilizar para la creación de versiones. Este método se llama siempre que se asocia un objeto con una transacción. Si no se conecta ningún cargador a la correlación de respaldo, ésta utiliza este valor durante la fase de confirmación para llevar a cabo una comparación de versiones optimista. La correlación de respaldo utiliza la comparación de versiones optimista para asegurarse de que la versión no ha cambiado desde la primera que vez que esta transacción accedió a la entrada de la correlación que fue modificada por esta

transacción. Si otra transacción hubiera modificado la versión de esta entrada de correlación, se produciría una anomalía en la comparación de versiones y la correlación de respaldo mostraría una excepción OptimisticCollisionException que forzaría la retrotracción de la transacción. Si hay un cargador conectado, la correlación de respaldo no utiliza la información de creación de versiones optimista. En su lugar, el cargador deberá realizar una comparación de versiones optimista y actualizar la información de la creación de versiones cuando sea necesario. El cargador suelte obtener el objeto de versiones inicial del LogElement pasado al método batchUpdate del cargador, que se llama cuando se produce una operación de vaciado o se confirma una transacción.

El código siguiente muestra la implementación que utiliza el objeto EmployeeOptimisticCallbackImpl:

```
public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}
```

Tal como se demuestra en el ejemplo anterior, se devuelve el atributo sequenceNumber en un objeto java.lang.Long tal como espera el cargador, que implica que la misma persona que escribió el cargador, también escribió la implementación de EmployeeOptimisticCallbackImpl, o bien trabajó estrechamente con la persona que implementó el método EmployeeOptimisticCallbackImpl, por ejemplo, acordó el valor devuelto por el método getVersionedObjectForValue. El plug-in predeterminado OptimisticCallback devuelve el valor especial NULL_OPTIMISTIC_VERSION como el objeto de versión.

Método updateVersionedObjectForValue

Este método se llama siempre que una transacción ha actualizado un valor y se necesita un nuevo objeto de versión. Si el método getVersionedObjectForValue devuelve un atributo del valor, este método suele actualizar el valor de atributo con un nuevo objeto de versión. Si el método getVersionedObjectForValue devuelve una copia del valor, este método normalmente no completa ninguna acción. El plug-in predeterminado OptimisticCallback no completa ninguna acción con este método porque la implementación predeterminada de getVersionedObjectForValue siempre devuelve el valor especial NULL_OPTIMISTIC_VERSION como el objeto de la versión. El siguiente ejemplo muestra la implementación utilizada por el objeto EmployeeOptimisticCallbackImpl que se utiliza en la sección OptimisticCallback:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Tal como se demuestra en el ejemplo anterior, el atributo sequenceNumber se incrementa por uno, de forma que la próxima vez que se llama al método getVersionedObjectForValue, el valor java.lang.Long devuelto tiene un valor largo que es el valor del número de secuencia original más uno, por ejemplo, es el valor de la siguiente versión para esta instancia de empleado. Este ejemplo implica que la misma persona que escribió el cargador escribió EmployeeOptimisticCallbackImpl o bien trabajó estrechamente con la persona que implementó EmployeeOptimisticCallbackImpl.

Método serializeVersionedValue

Este método escribe el valor con versión en la corriente especificada. En función de la implementación, el valor con versión puede utilizarse para identificar colisiones de actualización optimista. En algunas implementaciones, el valor con versión es una copia del valor original. Otras implementaciones podrían tener un número de secuencia o algún otro objeto para indicar la versión del valor. Puesto que la implementación real se desconoce, este método se proporciona para realizar la serialización apropiada. La implementación predeterminada llama al método writeObject.

Método inflateVersionedValue

Este método toma la versión serializada del valor con versión y devuelve el objeto de valor con versión real. En función de la implementación, el valor con versión puede utilizarse para identificar colisiones de actualización optimista. En algunas implementaciones, el valor con versión es una copia del valor original. Otras implementaciones podrían tener un número de secuencia o algún otro objeto para indicar la versión del valor. Puesto que se desconoce la implementación real, este método se proporciona para realizar la deserialización apropiada. La implementación predeterminada llama al método readObject.

Uso del objeto OptimisticCallback proporcionado por la aplicación

Dispone de dos enfoques para añadir un objeto OptimisticCallback proporcionado por la aplicación en la configuración de BackingMap: configuración de XML y configuración mediante programa.

Conectar mediante programación un objeto OptimisticCallback

El siguiente ejemplo demuestra cómo una aplicación puede conectar mediante programación un objeto OptimisticCallback para la correlación de respaldo del empleado en la instancia local del ObjectGrid grid1.

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

Enfoque de configuración de XML para conectar un objeto OptimisticCallback

La aplicación puede utilizar un archivo XML para conectar su objeto OptimisticCallback tal como se muestra en el siguiente ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid1">
        <backingMap name="employees" pluginCollectionRef="employees" lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>

<backingMapPluginCollections>
    <backingMapPluginCollection id="employees">
        <bean id="OptimisticCallback" className="com.xyz.EmployeeOptimisticCallbackImpl" />
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Plug-ins para serializar objetos en la memoria caché

WebSphere eXtreme Scale utiliza varios procesos Java para serializar los datos, convirtiendo las instancias de objetos Java en bytes y de nuevo en objetos, según sea necesario, para mover los datos entre los procesos de cliente y servidor.

Para serializar datos en eXtreme Scale, puede utilizar serialización Java, el plug-in ObjectTransformer o los plug-ins DataSerializer.

 La interfaz ObjectTransformer ha sido sustituida por los plug-ins DataSerializer, que puede utilizar para almacenar eficientemente datos arbitrarios en WebSphere eXtreme Scale de modo que las API existentes del producto puedan interactuar eficazmente con los datos.

Conceptos relacionados:

Visión general de serialización
Los datos normalmente se expresan, pero no se almacenan necesariamente, como objetos Java en la cuadrícula de datos. WebSphere eXtreme Scale utiliza varios procesos Java para serializar los datos, convirtiendo las instancias de objeto Java en bytes y de nuevo en objetos, según se requiera, para mover datos entre procesos de cliente y servidor.

Visión general de la programación del serializador

Puede utilizar los plug-ins DataSerializer para grabar serializadores optimizados a fin de almacenar objetos Java y otros datos en formato binario en la cuadrícula. El plug-in también proporciona métodos que puede utilizar para consultar atributos en los datos binarios sin necesidad de que el objeto de datos entero se infle.

Los plug-ins DataSerializer incluyen tres plug-ins principales y varias interfaces mixin opcionales. El plug-in MapSerializerPlugin incluye metadatos acerca de la relación entre una correlación y otras correlaciones. También incluye una referencia a un KeySerializerPlugin y ValueSerializerPlugin. Los plug-ins de serializador de clave y valor incluyen metadatos y código de serialización responsables de interactuar con los respectivos datos de clave y valor para una correlación. Un plug-in MapSerializerPlugin debe incluir uno o ambos serializadores de clave y valor.

El plug-in KeySerializerPlugin proporciona métodos y metadatos para serializar, inflar y examinar claves. El plug-in ValueSerializer proporciona métodos y metadatos para serializar, inflar y examinar claves. Ambas interfaces tienen

requisitos diferentes. Para obtener información detallada sobre los métodos que están disponibles en los plug-ins DataSerializer, consulte la documentación de API para el paquete com.ibm.websphere.objectgrid.plugins.io.

Plug-in MapSerializerPlugin
MapSerializerPlugin es el punto de plug-in principal en la interfaz BackingMap e incluye dos plug-ins anidados: los plug-ins KeySerializerPlugin y ValueSerializerPlugin. Puesto que eXtreme Scale no soporta plug-ins anidados o conectados, el plug-in BasicMapSerializerPlugin accede de forma artificial a estos plug-ins anidados. Cuando se utilizan estos plug-ins con la infraestructura OSGi, el único proxy es el plug-in MapSerializerPlugin. Ninguno de los plug-ins anidados se debe almacenar en memoria caché en otros plug-ins dependientes como, por ejemplo, cargadores, a menos que estos plug-ins también estén a la escucha de sucesos del ciclo de vida de BackingMap. Esto es importante cuando se ejecuta en una infraestructura OSGi, porque las referencias a esos plug-ins se pueden continuar renovándose.

Plug-in KeySerializerPlugin
El plug-in KeySerializerPlugin amplía la interfaz DataSerializer e incluye otras interfaces mixin y metadatos que describen la clave. Utilice este plug-in para serializar e inflar objetos y atributos de datos de clave.

Plug-in ValueSerializerPlugin
El plug-in ValueSerializerPlugin amplía la interfaz DataSerializer, pero no expone métodos adicionales. Utilice este plug-in para serializar e inflar objetos y atributos de datos de valores.

Interfaces mixin opcionales

Las interfaces mixin opcionales proporcionan prestaciones adicionales, por ejemplo:

Mantenimiento de versiones optimista
La interfaz versionable permite que el plug-in ValueSerializerPlugin maneje la comprobación de versión y las actualizaciones de versión cuando se utiliza el bloqueo optimista. Si no se implementa el mantenimiento de versiones y se habilita el bloqueo optimista, la versión es la forma serializada entera del valor de objeto de datos.

Direccionamiento no basado en hashCode
La interfaz particionable permite que las implementaciones de KeySerializerPlugin direccionen las solicitudes a particiones explícitas. Esto es equivalente a la interfaz PartitionableKey, cuando se utiliza con la API ObjectMap sin un KeySerializerPlugin. Sin esta característica, la clave se direcciona a la partición en función del código hash resultante.

Interfaz UserReadable (toString)
La interfaz UserReadable (toString) permite que todas las implementaciones de DataSerializer proporcionen un método alternativo para visualizar datos en los archivos de registro y los depuradores. Con esta posibilidad, se pueden ocultar los datos confidenciales como por ejemplo contraseñas. Si las implementaciones de DataSerializer no implementan esta interfaz, es posible que el entorno de ejecución llame directamente a toString() en el objeto o incluya representaciones alternativas, si es apropiado.

Soporte de la evolución
La interfaz Mergeable se puede implementar en las implementaciones de plug-in ValueSerializerPlugin para permitir la interoperatividad entre

varias versiones de objetos cuando hay diferentes versiones DataSerializer actualizando datos en la cuadrícula durante su tiempo de vida. Los métodos Mergeable permiten que la opción DataSerializer retenga los datos que de otra manera no podría entender.

Tareas relacionadas:

"Evitar el inflado de objetos para recuperar atributos de datos serializados"
Puede utilizar los plug-ins DataSerializer para omitir el inflado automático de objetos y recuperar manualmente los atributos de los datos que ya se han serializado.

"Programación para utilizar la infraestructura OSGi" en la página 395
Puede iniciar servidores y clientes de eXtreme Scale en un contenedor OSGi, lo que le permite añadir y actualizar dinámicamente plug-ins de eXtreme Scale en el entorno de ejecución.

Información relacionada:

Documentación de la API DataSerializer

Evitar el inflado de objetos para recuperar atributos de datos serializados

Puede utilizar los plug-ins DataSerializer para omitir el inflado automático de objetos y recuperar manualmente los atributos de los datos que ya se han serializado.

Acerca de esta tarea

Este tema trata cómo evitar el inflado de todo el objeto para recuperar atributos. Sin embargo, puede inflar todo el objeto inflando los objetos Java a una representación similar a POJO de los datos. Para inflar todo el objeto, cambie la última línea del ejemplo de este tema a la siguiente línea de código:

```
Order order = (Order) sa.getMapSerializerPlugin().getValueSerializerPlugin().
inflateDataObject(serValue.getContext(), bufValue);
```

Esta tarea utiliza la modalidad de copia COPY_TO_BYTES_RAW con los plug-ins MapSerializerPlugin y ValueSerializerPlugin. El MapSerializer es el punto de plug-in principal a la interfaz BackingMap. Incluye dos plug-ins anidados, KeyDataSerializer y ValueDataSerializer. Dado que el producto no da soporte a plug-ins anidados, el BaseMapSerializer da soporte de forma artificial a plug-ins anidados o conectados. Por lo tanto, cuando se utilizan estas API en el contenedor OSGi, el MapSerializer es el único proxy. Ninguno de los plug-ins anidados debe estar almacenado en memoria caché en otros plug-ins dependientes como, por ejemplo, un cargador, a menos que esté también a la escucha de sucesos de ciclo de vida de BackingMap, de manera que pueda renovar sus referencias de soporte.

Procedimiento

1. Recupere la instancia ObjectMap.
2. Establezca la modalidad de copia en COPY_TO_BYTES_RAW.
3. Utilice el método get para recuperar el objeto SerializedValue.
4. Utilice el método SerializedValue.getByteBuffers() para recuperar el formato serializado del valor.
5. Llame al plug-in ValueSerializerPlugin para inflar atributos individuales desde los almacenamientos intermedios de bytes.

Ejemplo

Utilice el siguiente código de ejemplo para recuperar atributos de datos serializados sin inflar todo el objeto Java.

```
// La BackingMap se configura con COPY_TO_BYTES y un MapSerializerPlugin con un ValueSerializerPlugin

Session session = objectGrid.getSession();
ObjectMap orderMap = session.getMap("OrderMap");

// Inflar automáticamente a un POJO como normal
Order order = (Order) orderMap.get(1234);

// Sustituir la CopyMode para recuperar los bytes.  Este proceso afecta a todos los métodos de la API a
partir de este punto
// durante la vida de la sesión.
// Nota:  la matriz de bytes tiene una cabecera específica de eXtreme Scale.
orderMap.setCopyMode(CopyMode.COPY_TO_BYTES_RAW);
SerializedValue serValue = (SerializedValue) orderMap.get(1234);

// Obtener los almacenamientos intermedios de bytes
XsByteBuffer[] bufValue= serValue.getByteBuffers();

// Convertir/obtener la matriz de bytes
Byte[] bytesValue = ByteBufferUtils.asByteArray(bufValue);

// Recuperar un único atributo del almacenamiento intermedio de bytes.
String name = (String) sa.getMapSerializerPlugin().getValueSerializerPlugin().
inflateDataObjectAttributes(serValue.getContext(),
 bufValue, new String[]{"name"};
```

Conceptos relacionados:

"Visión general de la programación del serializador" en la página 313
Puede utilizar los plug-ins DataSerializer para grabar serializadores optimizados a fin de almacenar objetos Java y otros datos en formato binario en la cuadrícula. El plug-in también proporciona métodos que puede utilizar para consultar atributos en los datos binarios sin necesidad de que el objeto de datos entero se infle.

Visión general de serialización
Los datos normalmente se expresan, pero no se almacenan necesariamente, como objetos Java en la cuadrícula de datos. WebSphere eXtreme Scale utiliza varios procesos Java para serializar los datos, convirtiendo las instancias de objeto Java en bytes y de nuevo en objetos, según se requiera, para mover datos entre procesos de cliente y servidor.

Información relacionada:

Documentación de la API DataSerializer

Plug-in ObjectTransformer

Con el plug-in ObjectTransformer puede serializar, deserializar y copiar objetos en la memoria caché para mejorar el rendimiento.

 La interfaz ObjectTransformer ha sido sustituida por los plug-ins DataSerializer, que puede utilizar para almacenar eficientemente datos arbitrarios en WebSphere eXtreme Scale de modo que las API existentes del producto puedan interactuar eficazmente con los datos.

Si observa problemas de rendimiento con el uso del procesador, añada un plug-in ObjectTransformer a cada correlación. Si no proporciona un plug-in ObjectTransformer, se emplea entre un 60 y un 70 por ciento del tiempo total de procesador en serializar y copiar entradas.

Finalidad

Con el plug-in ObjectTransformer, las aplicaciones pueden proporcionar métodos personalizados para las siguientes operaciones:

- Serializar o deserializar la clave de una entrada.

- Serializar o deserializar el valor de una entrada.
- Copiar una clave o valor de una entrada.

Si no se proporciona ningún plug-in ObjectTransformer, debe poder serializar las claves y los valores porque ObjectGrid utiliza una secuencia de serialización y deserialización para copiar los objetos. Éste es un método costoso, por lo que conviene utilizar un plug-in ObjectTransformer si el rendimiento es crítico. La operación de copia se produce cuando una aplicación busca un objeto en una transacción por primera vez. Puede evitar esta copia si establece la modalidad de copia de la correlación en NO_COPY o puede reducir la copia si establece la modalidad de copia en COPY_ON_READ. Optimice la operación de copia cuando sea necesario para la aplicación; para ello, proporcione un método de copia personalizada en este plug-in. Dicho plug-in puede reducir la sobrecarga de copia del 65 al 70 por ciento a 2/3 del porcentaje del tiempo total del procesador.

Las implementaciones predeterminadas del método copyKey y copyValue intentan, en primer lugar, utilizar el método clone, si se ha proporcionado el método. Si no se ha proporcionado ninguna implementación del método clone, la implementación toma el valor predeterminado de la serialización.

La serialización de objetos también se utiliza directamente cuando eXtreme Scale se ejecuta en la modalidad distribuida. El LogSequence utiliza el plug-in ObjectTransformer para ayudar a serializar claves y valores antes de transmitir los cambios a sus iguales en ObjectGrid. Debe actuar con detenimiento cuando proporcione un método de serialización personalizado, en lugar de utilizar la serialización del Java Developer Kit incorporada. La creación de versiones de objetos es un asunto complejo y podría tener problemas con la compatibilidad de las versiones si no se asegura de que sus métodos personalizados se han diseñado para la creación de versiones.

La siguiente lista describe cómo eXtreme Scale intenta serializar tanto las claves, como los valores:

- Si se ha escrito y conectado un plug-in ObjectTransformer personalizado, eXtreme Scale llama a los métodos de la interfaz ObjectTransformer para serializar las claves y los valores y obtener copiadas de claves y valores de objeto.
- Si no se utiliza un plug-in ObjectTransformer personalizado, eXtreme Scale serializa y deserializa los valores de acuerdo con el valor predeterminado. Si se utiliza el plug-in predeterminado, cada objeto se implementa como externalizable o se implementa como serializable.
  - Si el objeto soporta la interfaz Externalizable, se llama al método writeExternal. Los objetos que se implementan como externalizables obtienen un mejor rendimiento.
  - Si el objeto no soporta la interfaz Externalizable e implementa la interfaz Serializable, el objeto se guarda mediante el método ObjectOutputStream.

Utilización de la interfaz ObjectTransformer

Un objeto ObjectTransformer debe implementar la interfaz ObjectTransformer y seguir las convenciones comunes de plug-in de ObjectGrid.

Se utilizan dos enfoques, configuración mediante programa y configuración de XML, para añadir un objeto ObjectTransformer a la configuración de BackingMap tal como se indica a continuación.

Conexión mediante programación de un objeto ObjectTransformer

El siguiente fragmento de código crea el objeto ObjectTransformer personalizado y lo añade a un BackingMap:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap backingMap = myGrid.getMap("myMap");
MyObjectTransformer myObjectTransformer = new MyObjectTransformer();
backingMap.setObjectTransformer(myObjectTransformer);
```

Conexión de ObjectTransformer mediante la configuración del XML

Imagine que el nombre de clase de la implementación ObjectTransformer es la clase com.company.org.MyObjectTransformer. Esta clase implementa la interfaz ObjectTransformer. Una implementación de ObjectTransformer se puede configurar utilizando el siguiente XML:

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

    <backingMapPluginCollections>
 <backingMapPluginCollection id="myMap">
      <bean id="ObjectTransformer" className="com.company.org.MyObjectTransformer" />
 </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Escenarios de uso de ObjectTransformer

Puede utilizar un plug-in ObjectTransformer en las situaciones siguientes:

- Objeto no serializable
- Objeto serializable pero mejora el rendimiento de serialización
- Copia de clave o valor

En el ejemplo siguiente, se utiliza ObjectGrid para almacenar la clase Stock:

```
/**
* Objeto Stock para ObjectGrid
*
*
*/
public class Stock implements Cloneable {
    String ticket;
    double price;
    String company;
    String description;
    int serialNumber;
    long lastTransactionTime;
    /**
    * @return Devuelve la descripción.
    */
    public String getDescription() {
        return description;
    }
    /**
    * @param description La descripción que se debe establecer.
    */
    public void setDescription(String description) {
        this.description = description;
    }
    /**
    * @return Devuelve lastTransactionTime.
    */
    public long getLastTransactionTime() {
        return lastTransactionTime;
```

```
    }
    /**
    * @param lastTransactionTime El lastTransactionTime a establecer.
    */
    public void setLastTransactionTime(long lastTransactionTime) {
        this.lastTransactionTime = lastTransactionTime;
    }
    /**
    * @return Devuelve el precio.
    */
    public double getPrice() {
        return price;
    }
    /**
    * @param price El precio a establecer.
    */
    public void setPrice(double price) {
        this.price = price;
    }
    /**
    * @return Devuelve serialNumber.
    */
    public int getSerialNumber() {
        return serialNumber;
    }
    /**
    * @param serialNumber El serialNumber a establecer.
    */
    public void setSerialNumber(int serialNumber) {
        this.serialNumber = serialNumber;
    }
    /**
    * @return Devuelve el ticket.
    */
    public String getTicket() {
        return ticket;
    }
    /**
    * @param ticket El ticket a establecer.
    */
    public void setTicket(String ticket) {
        this.ticket = ticket;
    }
    /**
    * @return Devuelve la empresa.
    */
    public String getCompany() {
        return company;
    }
    /**
    * @param company La empresa a establecer.
    */
    public void setCompany(String company) {
        this.company = company;
    }
    //clone
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}
```

Puede escribir una clase de ObjectTransformer para la clase Stock:

```
/**
* Implementación personalizada de ObjectGrid ObjectTransformer para el objeto stock
*
*/
public class MyStockObjectTransformer implements ObjectTransformer {
/* (no 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);
}

/* (no 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());
}

/* (no 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;
}

/* (no 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;
}

/* (no 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)
    {
        // mostrar mensaje de excepción }
}

/* (no 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 clave MyStockObjectTransformer personalizada a BackingMap:

```
ObjectGridManager ogf=ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogf.getObjectGrid("NYSE");
BackingMap bm = og.defineMap("NYSEStocks");
MyStockObjectTransformer ot = new MyStockObjectTransformer();
bm.setObjectTransformer(ot);
```

Plug-ins para proporcionar escuchas de sucesos

Puede utilizar los plug-ins ObjectGridEventListener, MapEventListener, ObjectGridLifecycleListener y BackingMapLifecycleListener para configurar notificaciones para diversos sucesos en la memoria caché de eXtreme Scale. Los plug-ins de escucha se registran con una instancia de ObjectGrid o BackingMap como otros plug-ins eXtreme Scale y añaden puntos de integración y personalización para las aplicaciones y los proveedores de memoria caché.

Plug-in ObjectGridEventListener

Un plug-in ObjectGridEventListener proporciona sucesos de ciclo de vida de eXtreme Scale para la instancia, fragmentos y transacciones de ObjectGrid. Utilice el plug-in ObjectGridEventListener para recibir notificaciones cuando se producen sucesos significativos en un ObjectGrid. Estos sucesos incluyen la inicialización de ObjectGrid, el inicio de una transacción, la finalización de una transacción y la destrucción de un ObjectGrid. Para escuchar estos sucesos, cree una clase que implemente la interfaz ObjectGridEventListener y añádala a eXtreme Scale.

Para obtener más información sobre cómo grabar un plug-in ObjectGridEventListener, consulte "Plug-in ObjectGridEventListener" en la página 323. También puede hacer referencia a la documentación de la API si desea más información.

Adición y eliminación de instancias de ObjectGridEventListener

Un ObjectGrid puede tener varios receptores de ObjectGridEventListener. Añada y elimine los escuchas mediante los métodos addEventListener y removeEventListener en la interfaz ObjectGrid. También puede registrar de forma declarativa los plug-ins ObjectGridEventListener con el archivo descriptor de ObjectGrid. Para obtener ejemplos, consulte "Plug-in ObjectGridEventListener" en la página 323.

Plug-in MapEventListener

Un plug-inMapEventListener proporciona notificaciones de devolución de llamada y cambios de estado de memoria caché significativos que se producen para una instancia de BackingMap. Para ver detalles sobre cómo escribir un plug-in MapEventListener, consulte "Plug-in MapEventListener" en la página 322. También puede hacer referencia a la documentación de la API si desea más información.

Adición y eliminación de instancias de MapEventListener

Un eXtreme Scale puede tener varios receptores de MapEventListener. Añada y elimine escuchas con los métodos addMapEventListener y removeMapEventListener en la interfaz BackingMap. También puede registrar de forma declarativa los receptores de MapEventListener con el archivo descriptor de ObjectGrid. Para obtener ejemplos, consulte "Plug-in MapEventListener" en la página 322.

Plug-in BackingMapLifecycleListener

Un plug-in BackingMapLifecycleListener proporciona notificaciones de devolución de llamada para cambios de estado de ciclo de vida que se producen para una instancia BackingMap. La instancia de BackingMap avanza por un conjunto predefinido de estados durante su vida.

Adición y eliminación de instancias de BackingMapLifecycleListener

Un servidor eXtreme Scale puede tener varios escuchas BackingMapLifecycleListener. Añada y elimine escuchas con los métodos addMapEventListener y removeMapEventListener en la interfaz BackingMap. Los plug-ins BackingMap que implementan la interfaz BackingMapLifecycleListener también se añaden automáticamente como un BackingMapLifecycleListener para la instancia de ObjectGrid con la que están registrados. También puede registrar de

forma declarativa escuchas BackingMapLifecycleListener con el archivo de descriptor de ObjectGrid. Para ver ejemplos, consulte la sección del plug-in BackingMapLifecycleListener.

Plug-in ObjectGridLifecycleListener

Un plug-in ObjectGridLifecycleListener proporciona notificaciones de devolución de llamada para cambios de estado de ciclo de vida que se producen para una instancia de ObjectGrid. La instancia de ObjectGrid pasa por un conjunto predefinido de estados durante su vida.

Adición y eliminación de instancias ObjectGridLifecycleListener

Un eXtreme Scale puede tener varios escuchas ObjectGridLifecycleListener. Añada y elimine escuchas con los métodos addEventListener y removeEventListener en la interfaz ObjectGrid. Cualquier plug-in de ObjectGrid que implemente la interfaz ObjectGridLifecycleListener se añade automáticamente como ObjectGridLifecycleListener para la instancia de ObjectGrid con la que está registrado. También puede registrar de forma declarativa escuchasObjectGridLifecycleListener con el archivo de descriptor de despliegue de ObjectGrid. Por ejemplo, consulte la sección del plug-in ObjectGridLifecycleListener.

Plug-in MapEventListener

Un plug-in MapEventListener proporciona notificaciones de devolución de llamada y cambios significativos de estado de memoria caché que se producen para un objeto BackingMap: cuando una correlación ha terminado la precarga o cuando se desaloja una entrada de la correlación. Un determinado plug-in MapEventListener es una clase personalizada que se escribe implementando la interfaz MapEventListener.

Convenios del plug-in MapEventListener

Cuando desarrolle un plug-in MapEventListener, debe seguir los convenios comunes de plug-in. Para obtener más información sobre los convenios de plug-in, consulte"Visión general de los plug-ins" en la página 115. Para otros tipos de plug-ins de escuchas, consulte "Plug-ins para proporcionar escuchas de sucesos" en la página 320.

Después de escribir una implementación de MapEventListener, puede conectarse a éste en la configuración de BackingMap a través de programa o con una configuración de XML.

Grabar una implementación de MapEventListener

La aplicación puede incluir una implementación del plug-in MapEventListener. El plug-in debe implementar la interfaz MapEventListener para recibir sucesos significativos sobre una correlación. Los sucesos se envían al plug-in MapEventListener cuando una entrada se desaloja de la correlación y cuando finaliza la precarga de una correlación.

Conexión a través de programa de una implementación de MapEventListener

El nombre de clase para la clase personalizada MapEventListener es com.company.org.MyMapEventListener. Esta clase implementa la interfaz

MapEventListener. El siguiente fragmento de código crea el objetoMapEventListener personalizado y lo añade a un objeto BackingMap:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap myMap = myGrid.defineMap("myMap");
MyMapEventListener myListener = new MyMapEventListener();
myMap.addMapEventListener(myListener);
```

Conectar una implementación de MapEventListener utilizando XML

Se puede configurar una implementación de MapEventListner utilizando el XML. El siguiente XML debe aparecer en el archivo myGrid.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridconfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="myGrid">
            <backingMap name="myMap" pluginCollectionRef="myPlugins" />
        </objectGrid>
    </objectGrids>
    <backingMapPluginCollections>
        <backingMapPluginCollection id="myPlugins">
            <bean id="MapEventListener" className=
       "com.company.org.MyMapEventListener" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Proporcionar este archivo a la instancia de ObjectGridManager facilita la creación de esta configuración. El siguiente fragmento de código muestra cómo crear una instancia de ObjectGrid utilizando este archivo XML. La instancia de ObjectGrid recién creada tiene un MapEventListener establecido en myMap BackingMap.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
 objectGridManager.createObjectGrid("myGrid", new URL("file:etc/test/myGrid.xml"),
  true, false);
```

Plug-in ObjectGridEventListener

Un plug-in ObjectGridEventListener proporciona sucesos de ciclo de vida de WebSphere eXtreme Scale para ObjectGrid, fragmentos y transacciones. Un plug-in ObjectGridEventListener proporciona notificaciones cuando se inicializa o destruye un ObjectGrid, y cuando se inicia o finaliza una transacción. Los plug-ins ObjectGridEventListener son clases personalizadas que se escriben implementando la interfaz ObjectGridEventListener. De forma opcional, la implementación incluye las subinterfaces ObjectGridEventGroup y sigue las convenciones comunes de plug-in eXtreme Scale.

Visión general

Un plug-in ObjectGridEventListener es útil cuando está disponible un plug-in Loader, y debe inicializar la conexiones JDBC (Java Database Connectivity) o las conexiones a un programa de fondo cuando se inician y finalizan las transacciones. En general, un plug-in ObjectGridEventListener y un plug-in Loader se escriben juntos.

Escritura de un plug-in ObjectGridEventListener

Un plug-in ObjectGridEventListener debe implementar la interfaz ObjectGridEventListener para recibir notificaciones sobre los sucesos significativos de eXtreme Scale. Para recibir notificaciones de sucesos adicionales, puede implementar las siguientes interfaces. Estas subinterfaces se incluyen en la interfaz ObjectGridEventGroup:

- Interfaz ShardEvents
- Interfaz ShardLifecycle
- Interfaz TransactionEvents

Si desea más información sobre estas interfaces, consulte la documentación de la API.

Sucesos de fragmentos

Cuando el servicio de catálogo coloca los fragmentos del primario de la partición o de réplica en una máquina virtual Java (JVM), se crea una nueva instancia de ObjectGrid en dicha JVM para alojar dicho fragmento. Algunas aplicaciones que necesitan iniciar hebras en la JVM alojan la notificación necesaria primaria de estos sucesos. La interfaz ObjectGridEventGroup.ShardEvents declara los métodos shardActivate y shardDeactivate. Estos métodos se invocan sólo cuando un fragmento está activado como primario y cuando el fragmento se desactiva del primario. Estos dos sucesos permiten a la aplicación iniciar hebras adicionales cuando el fragmento es un primario y detenerlas cuando el fragmento vuelve a ser una réplica o se queda fuera de servicio.

Una aplicación puede determinar qué partición ha sido activada por la búsqueda de un BackingMap específico en la referencia de ObjectGrid que se proporciona al método shardActivate mediante el método ObjectGrid#getMap. La aplicación puede ver el número de partición utilizando el método BackingMap#getPartitionId(). Las particiones están numeradas del 0 hasta el número de particiones en el descriptor de despliegue menos una.

Sucesos de ciclo de vida de fragmento

Los sucesos de los métodosObjectGridEventListener.initialize y ObjectGridEventListener.destroy se entregan utilizando la interfaz ObjectGridEventGroup.ShardLifecycle.

Sucesos de transacciones

Los métodos ObjectGridEventListener.transactionBegin y ObjectGridEventListener.transactionEnd se entregan a través de la interfaz ObjectGridEventGroup.TransactionEvents.

Si un plug-in ObjectGridEventListener implementa las interfaces ObjectGridEventListener y ShardLifecycle, los sucesos de ciclo de vida de fragmentos son los únicos sucesos que se entregan al escucha. Después de implementar cualquiera de las nuevas interfaces internas de ObjectGridEventGroup, eXtreme Scale sólo entrega estos sucesos específicos mediante las nuevas interfaces. Con esta implementación, el código puede ser compatible con versiones anteriores. Si utiliza las nuevas interfaces internas ahora puede recibir sólo los sucesos específicos necesarios.

Utilización del plug-in ObjectGridEventListener

Para utilizar un plug-in ObjectGridEventListener personalizado, primero cree una clase que implemente la interfaz ObjectGridEventListener y todas las subinterfaces ObjectGridEventGroup opcionales. Añada el escucha personalizado a un ObjectGrid para recibir una notificación de los sucesos significativos. Dispone de dos procedimientos para añadir un plug-in ObjectGridEventListener en la configuración de eXtreme Scale: configuración programática y configuración XML.

Configurar un plug-in ObjectGridEventListener mediante programación

Presuponga que el nombre de la clase del receptor de sucesos de eXtreme Scale es la clase com.company.org.MyObjectGridEventListener. Esta clase implementa la interfaz ObjectGridEventListener. El siguiente fragmento de código crea el ObjectGridEventListener personalizado y lo añade a un ObjectGrid.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
MyObjectGridEventListener myListener = new MyObjectGridEventListener();
myGrid.addEventListener(myListener);
```

Configurar un plug-in ObjectGridEventListener con XML

También puede configurar un plug-in ObjectGridEventListener mediante XML. El siguiente XML crea una configuración que es equivalente al receptor de sucesos de ObjectGrid descrito creado mediante programación. El siguiente texto debe aparecer en el archivo `myGrid.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="myGrid">
            <bean id="ObjectGridEventListener"
       className="com.company.org.MyObjectGridEventListener" />
            <backingMap name="Book"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Tenga en cuenta que las declaraciones bean se indican antes que las declaraciones backingMap. Proporcione este archivo al plug-in ObjectGridManager para facilitar la creación de esta configuración. El siguiente fragmento de código demuestra cómo crear una instancia de ObjectGrid utilizando este archivo XML. La instancia de ObjectGrid que se crea tiene un receptor ObjectGridEventListener establecido en el ObjectGrid myGrid.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid",
 new URL("file:etc/test/myGrid.xml"), true, false);
```

Plug-in BackingMapLifecycleListener

Un plug-in BackingMapLifecycleListener recibe notificación de sucesos de cambio de estado de ciclo de vida de WebSphere eXtreme Scale para la correlación de respaldo.

El plug-in BackingMapLifecycleListener recibe un suceso que contiene un objeto BackingMapLifecycleListener.State para cada cambio de estado de la correlación de respaldo. Cualquier plug-in BackingMap que implemente también la interfaz BackingMapLifecycleListener se añadirá automáticamente como escucha para la instancia de BackingMap donde se ha registrado el plug-in.

Visión general

Un plug-in BackingMapLifecycleListener es útil cuando un plug-in BackingMap existente necesita realizar actividades relacionadas con las actividades de un plug-in relacionado. Como ejemplo, un plug-in Loader podría necesitar recuperar configuración de un plug-in MapIndexPlugin o DataSerializer colaborativo.

Mediante la implementación de la interfaz BackingMapLifecycleListener y la detección del suceso BackingMapLifecycleListener.State.INITIALIZED, el cargador puede conocer el estado de otros plug-ins de la instancia de BackingMap. El cargador puede recuperar de forma segura información del plug-in MapIndexPlugin o DataSerializer colaborativo, ya que BackingMap está en un estado INITIALIZED, lo que significa que en los otros plug-ins se ha llamado a su método initialize().

Se puede añadir o eliminar un BackingMapLifecycleListener en cualquier momento, antes o después de que se inicialice el ObjectGrid y sus BackingMaps.

Escribir un plug-in BackingMapLifecycleListener

Un plug-in BackingMapLifecycleListener debe implementar la interfaz BackingMapLifecycleListener para recuperar notificaciones sobre sucesos significativos de eXtreme Scale. Cualquier plug-in BackingMap puede implementar la interfaz BackingMapLifecycleListener y añadirse automáticamente como escucha cuando se añade también a la correlación de respaldo.

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

Relaciones de plug-in y suceso de ciclo de vida

BackingMapLifecycleListener recupera el estado de ciclo de vida del suceso en el método backingMapStateChanged; por ejemplo:

```
public void backingMapStateChanged(BackingMap map,
                                   LifecycleEvent event)
throws LifecycleFailedException {
  switch(event.getState()) {
    case INITIALIZED: // Todos los demás plug-ins se inicializan.
      // Recuperar referencia a plug-in X para uso desde correlación.
      break;
    case DESTROYING:  // Se inicia fase de destrucción
      // Eliminar referencia a plug-in X, se puede destruir antes de este plug-in
      break;
  }
}
```

La tabla siguiente describe la relación entre los sucesos de ciclo de vida enviados a un plug-in BackingMapLifecycleListener y los estados de la BackingMap y otros objetos de plug-in.

Valor de BackingMapLifecycleListener.State	Descripción
INITIALIZING	La fase de inicialización de BackingMap se está iniciando. Los plug-ins BackingMap y BackingMap están a punto de inicializarse.
INITIALIZED	La fase de inicialización de BackingMap se ha completado. Todos los plug-ins BackingMap se han inicializado. Es posible que se vuelva a producir el estado INITIALIZED cuando tengan lugar actividades de colocación de fragmentos (de promoción o relegación).

Valor de BackingMapLifecycleListener.State	Descripción
STARTING	La instancia de BackingMap se está activando para su uso como instancia local, instancia de cliente o instancia en un fragmento de réplica o primario del servidor. Todos los plug-ins de ObjectGrid de la instancia de ObjectGrid propietarios de esta instancia de BackingMap se han inicializado. Es posible que se vuelva a producir el estado STARTING cuando tengan lugar actividades de colocación de fragmentos (de promoción o relegación).
PRELOAD	La instancia de BackingMap la establece en el estado PRELOAD la API StateManager para la precarga, o el cargador configurado está precargando datos en la correlación de respaldo.
ONLINE	La instancia de BackingMap está lista para funcionar como instancia local, instancia de cliente o instancia de un fragmento de réplica o primario del servidor. Todos los plug-ins de ObjectGrid de la instancia de ObjectGrid propietarios de esta instancia de BackingMap se han inicializado. Este estado estable es típico de la BackingMap. El estado ONLINE podría volver a producirse cuando se produzcan actividades de colocación de fragmentos (promoción o relegación).
QUIESCE	Se está deteniendo el trabajo en la BackingMap como resultado de la API StateManager o de otro suceso. No se permite ningún trabajo nuevo. El plug-in finaliza cualquier trabajo existente lo antes posible.
OFFLINE	Todo el trabajo se ha detenido en la BackingMap como resultado de la API StateManager o de otro suceso. No se permite ningún trabajo nuevo.
DESTROYING	La instancia de BackingMap está iniciando la fase de destrucción. Los plug-ins BackingMap para la instancia están a punto de ser destruidos.
DESTROYED	La instancia de BackingMap y todos los plug-ins BackingMap se han destruido.

Configurar un plug-in BackingMapLifecycleListener con XML

Supongamos que el nombre de clase del escucha de sucesos de eXtreme Scale es la clase com.company.org.MyBackingMapLifecycleListener. La clase implementa la interfaz BackingMapLifecycleListener.

Puede configurar un plug-in BackingMapLifecycleListener utilizando XML. El texto siguiente debe estar en el archivo XML de la cuadrícula de objetos:

```
<?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="BackingMapLifecycleListener"
                  className="com.company.org.MyBackingMapLifecycleListener" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Proporcione este archivo al plug-in ObjectGridManager para facilitar la creación de esta configuración. La instancia de BackingMap creada tiene un escucha BackingMapLifecycleListener establecido en el ObjectGrid myGrid.

Al igual que BackingMapLifecycleListener, otros plug-ins BackingMap, como por ejemplo Loader o MapIndexPlugin, que especifica mediante XML y que también implementan la interfaz BackingMapLifecycleListener, se añadirán automáticamente como escuchas del ciclo de vida.

Referencia relacionada:

"Plug-in ObjectGridLifecycleListener"
Un plug-in ObjectGridLifecycleListener recibe notificación de sucesos de cambio de estado del ciclo de vida de WebSphere eXtreme Scale de la cuadrícula de datos.

Plug-in ObjectGridLifecycleListener

Un plug-in ObjectGridLifecycleListener recibe notificación de sucesos de cambio de estado del ciclo de vida de WebSphere eXtreme Scale de la cuadrícula de datos.

El plug-in ObjectGridLifecycleListener recibe un suceso que contiene un objeto ObjectGridLifecycleListener.State para cada cambio de estado del ObjectGrid. Cualquier plug-in ObjectGrid que implemente también la interfaz ObjectGridLifecycleListener se añadirá automáticamente como escucha para la instancia de ObjectGrid donde se ha registro el plug-in.

Visión general

Un plug-in ObjectGridLifecycleListener resulta útil cuando un plug-in ObjectGrid existente debe realizar actividades relativas a actividades en un plug-in relacionado. Como ejemplo, el plug-in TransactionCallback podría recuperar la configuración de un plug-in ObjectGridEventListener o ShardListener cooperativo.

Mediante la implementación de la interfaz ObjectGridLifecycleListener, y la detección del suceso ObjectGridLifecycleListener.State.INITIALIZED, el plug-in TransactionCallback puede detectar el estado de otros plug-ins en la instancia de ObjectGrid. El plug-in TransactionCallback puede recuperar con seguridad información del plug-in ObjectGridEventListener o ShardListener cooperativo, lo que significa que en el otro plug-in se ha llamado a su método initialize().

Puede añadir un plug-in ObjectGridLifecycleListener en cualquier momento, antes o después de la inicialización del ObjectGrid.

Escribir un plug-in ObjectGridLifecycleListener

Un plug-in ObjectGridLifecycleListener debe implementar la interfaz ObjectGridLifecycleListener para recibir notificaciones sobre sucesos significativos de eXtreme Scale. Cualquier plug-in ObjectGrid puede implementar la interfaz ObjectGridLifecycleListener y añadirse automáticamente como escucha cuando se añade también al ObjectGrid.

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

Relaciones de plug-in y suceso de ciclo de vida

ObjectGridLifecycleListener recupera el estado de ciclo de vida del suceso en el método objectgridStateChanged; por ejemplo:

```
public void objectGridStateChanged(ObjectGrid grid,
                                   LifecycleEvent event)
throws LifecycleFailedException {
  switch(event.getState()) {
    case INITIALIZED: // Todos los demás plug-ins se inicializan.
```

```
      // Recuperar referencia a plug-in X para uso desde cuadrícula.
      break;
    case DESTROYING:  // Se inicia fase de destrucción
      // Eliminar referencia a plug-in X, se puede destruir antes de este plug-in
      break;
  }
```

La tabla siguiente describe la relación entre sucesos de ciclo de vida enviados al ObjectGridLifecycleListener y los estados del ObjectGrid y otros objetos de plug-in.

Valor de ObjectGridLifecycleListener.State	Descripción
INITIALIZING	Se está iniciando la fase de inicialización del ObjectGrid. El ObjectGrid y los plug-ins del ObjectGrid están a punto de inicializarse.
INITIALIZED	La fase de inicialización del ObjectGrid se ha completado. Todos los plug-ins del ObjectGrid se han inicializado. Es posible que se vuelva a producir el estado INITIALIZED cuando tengan lugar actividades de colocación de fragmentos (de promoción o relegación). Todos los plug-ins BackingMap de las instancias de BackingMap propietarias de esta instancia de ObjectGrid se han inicializado.
STARTING	La instancia de ObjectGrid se está activando para su uso como instancia local, instancia de cliente o como instancia en un fragmento primario o de réplica en el servidor. Es posible que se vuelva a producir el estado STARTING cuando tengan lugar actividades de colocación de fragmentos (de promoción o relegación).
PRELOAD	La instancia de ObjectGrid se establece en el estado PRELOAD mediante la API StateManager u otra configuración.
ONLINE	La instancia de ObjectGrid está lista para funcionar como instancia local, instancia de cliente o como una instancia de un fragmento primario o de réplica en el servidor. Este estado estable es típico del ObjectGrid. El estado ONLINE podría volver a producirse cuando se produzcan actividades de colocación de fragmentos (promoción o relegación).
QUIESCE	El trabajo se está deteniendo en el ObjectGrid como resultado de la API StateManager o de otro suceso. No se permite ningún trabajo nuevo. Finalice cualquier trabajo existente lo antes posible.
OFFLINE	Todo el trabajo se ha detenido en el ObjectGrid como resultado de la API StateManager o de otro suceso. No se permite ningún trabajo nuevo.
DESTROYING	La instancia de ObjectGrid está iniciando la fase de destrucción. Los plug-ins del ObjectGrid para la instancia están a punto de ser destruidos. Durante la fase de destrucción, todas las instancias de BackingMap que son propiedad de esta instancia de ObjectGrid también se destruyen.
DESTROYED	La instancia de ObjectGrid, sus instancias de BackingMap y todos los plug-ins de ObjectGrid se han destruido.

Configure un plug-in ObjectGridLifecycleListener con XML

Supongamos que el nombre de clase del escucha de sucesos de eXtreme Scale es la clase com.company.org.MyObjectGridLifecycleListener. Esta clase implementa la interfaz ObjectGridLifecycleListener.

Puede configurar un plug-in ObjectGridLifecycleListener utilizando XML. El siguiente XML crea una configuración utilizando el ObjectGridLifecycleListener. El texto siguiente debe estar en el archivo XML de la cuadrícula de objetos:

```
<?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="ObjectGridLifecycleListener"
                  className="com.company.org.MyObjectGridLifecycleListener" />
            <backingMap name="Book"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Tenga en cuenta que las declaraciones bean se indican antes que las declaraciones backingMap. Proporcione este archivo al plug-in ObjectGridManager para facilitar la creación de esta configuración.

Al igual que el ObjectGridLifecycleListener registrado en el ejemplo anterior, otros plug-ins de ObjectGrid, CollisionArbiter o TransactionCallback por ejemplo, que se especifican mediante XML que también implementan la interfaz ObjectGridLifecycleListener, se añadirán automáticamente como escuchas de ciclo de vida.

Referencia relacionada:

"Plug-in BackingMapLifecycleListener" en la página 325
Un plug-in BackingMapLifecycleListener recibe notificación de sucesos de cambio de estado de ciclo de vida de WebSphere eXtreme Scale para la correlación de respaldo.

Plug-ins para la indexación de datos

El HashIndex incorporado, la clase com.ibm.websphere.objectgrid.plugins.index.HashIndex, es un plug-in MapIndexPlugin que puede añadir a BackingMap para crear índices estáticos o dinámicos. Esta clase da soporte a las interfaces MapIndex y MapRangeIndex. La definición y la implementación de índices puede mejorar significativamente el rendimiento de la consulta.

Tareas relacionadas:

"Configuración del plug-in HashIndex"
Puede configurar el HashIndex incorporado, la clase com.ibm.websphere.objectgrid.plugins.index.HashIndex, con un archivo XML, programáticamente, o con una anotación de entidad en una correlación de entidad.

"Acceso a datos con índices (API Index)" en la página 144
Utilice la indexación para acceder más eficazmente a los datos.

Referencia relacionada:

"Atributos del plug-in HashIndex" en la página 333
Puede utilizar los atributos siguientes para configurar el plug-in HashIndex. Estos atributos definen propiedades como por ejemplo si utiliza un HashIndex compuesto o de atributos, o si la indexación de rango está habilitada.

Configuración del plug-in HashIndex

Puede configurar el HashIndex incorporado, la clase com.ibm.websphere.objectgrid.plugins.index.HashIndex, con un archivo XML, programáticamente, o con una anotación de entidad en una correlación de entidad.

Acerca de esta tarea

Configurar un índice compuesto equivale a configurar un índice normal con XML, excepto el valor de la propiedad **attributeName**. En un índice compuesto, el valor de la propiedad **attributeName** es una lista delimitada por comas de atributos. Por

ejemplo, la clase de valor Address tiene tres atributos: city, state y zipcode. Un índice compuesto se puede definir con el valor de la propiedad **attributeName** como "city,state,zipcode", lo que indica que city, state y zipcode se incluye en el índice compuesto.

Además, tenga en cuenta que los HashIndexes compuestos no soportan las búsquedas de rango y, por lo tanto, no pueden tener la propiedad RangeIndex establecida en true.

Procedimiento

- Configure un índice compuesto en el archivo XML de descriptor de ObjectGrid.

  Utilice el elemento backingMapPluginCollections para definir el plug-in:

  ```
  <bean id="MapIndexPlugin"  className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Address.CityStateZip"/>
  <property name="AttributeName" type="java.lang.String" value="city,state,zipcode"/>
  </bean>
  ```

- Configure un índice compuesto programáticamente.

  El siguiente código de ejemplo crea el mismo índice compuesto:

  ```
  HashIndex mapIndex = new HashIndex();
      mapIndex.setName("Address.CityStateZip");
      mapIndex.setAttributeName(("city,state,zipcode"));
      mapIndex.setRangeIndex(true);

      BackingMap bm = objectGrid.defineMap("mymap");
      bm.addMapIndexPlugin(mapIndex);
  ```

- Configure un índice compuesto con notaciones de entidad.

  Si utiliza correlaciones de entidad, puede utilizar un enfoque de anotaciones para definir un índice compuesto. Puede definir una lista de CompositeIndex en la anotación CompositeIndexes en el nivel de clase de entidad. El CompositeIndex tiene un nombre y una propiedad **attributeNames**. Cada CompositeIndex está asociado con una instancia de HashIndex aplicada a la correlación de respaldo asociada con la entidad. El índice HashIndex está configurado como un índice de no intervalo.

  ```
  @Entity
  @CompositeIndexes({
      @CompositeIndex(name=" CityStateZip ", attributeNames=" city,state,zipcode"),
      @CompositeIndex(name="lastnameBirthday", attributeNames=" lastname,birthday ")
   })
  public class Address {
      @Id int id;
      String street;
      String city;
      String state;
      String zipcode;
      String lastname;
      Date birthday;
  }
  ```

  La propiedad name de cada índice compuesto debe ser exclusiva en la entidad y la correlación de respaldo. Si no se especifica el nombre, se utilizará un nombre generado. La propiedad **attributeName** se utiliza para llenar el attributeName HashIndex con la lista delimitada por comas de atributos. Los nombres de atributo coinciden con los nombres de campos persistentes, cuando las entidades se configuran para utilizar el acceso a campo, o el nombre de la propiedad se haya definido para los convenios de denominación de JavaBeans para las entidades de acceso a propiedades. Por ejemplo, si el nombre de atributo es `street`, el método de obtención de la propiedad se denomina getStreet.

Ejemplo: Añadir HashIndex a BackingMap

En el ejemplo siguiente, configura el plug-in HashIndex añadiendo plug-ins de índice estático al archivo XML:

```
<backingMapPluginCollection id="person">
   <bean id="MapIndexPlugin"
   className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
         <property name="Name" type="java.lang.String" value="CODE"
description="index name" />
          <property name="RangeIndex" type="boolean" value="true"
                  description="true for MapRangeIndex" />
         <property name="AttributeName" type="java.lang.String" value="employeeCode"
            description="attribute name" />
   </bean>
</backingMapPluginCollection>
```

En este ejemplo de configuración XML, se utiliza la clase HashIndex incorporada como el plug-in de índice. HashIndex da soporte a propiedades que los usuarios pueden configurar como, por ejemplo, Name, RangeIndex y AttributeName.

- La propiedad **Name** se configura como `CODE`, una serie que identifica este plug-in de índice. El valor de la propiedad **Name** debe ser exclusivo en el ámbito de la correlación de respaldo. El nombre se puede utilizar para recuperar el objeto de índice por nombre de la instancia ObjectMap para la BackingMap.
- La propiedad **RangeIndex** se configura como `true`, lo que significa que la aplicación puede difundir el objeto de índice recuperado a la interfaz MapRangeIndex. Si la propiedad RangeIndex se configura como `false`, la aplicación solo puede difundir el objeto de índice recuperado a la interfaz MapIndex. Un MapRangeIndex soporta las funciones para encontrar los datos utilizando las funciones de rango como, por ejemplo, mayor que, menor que, o ambos, mientras que un MapIndex sólo soporta las funciones de igual. Si el índice es por consulta, la propiedad **RangeIndex** se debe configurar como `true` en índices de un solo atributo. Para un índice de relación o un índice compuesto, la propiedad **RangeIndex** se debe configurar como `false`.
- La propiedad **AttributeName** se configura como `employeeCode`, lo que significa que el atributo employeeCode del objeto almacenado en memoria caché se utiliza para crear un índice de un solo atributo. Si una aplicación debe buscar objetos almacenados en memoria caché con varios atributos, la propiedad **AttributeName** se puede establecer en una lista delimitada por comas de atributos, lo que genera un índice compuesto.

En resumen, en el ejemplo anterior se define un rango de atributo único HashIndex. Es un HashIndex de un solo atributo porque el valor de la propiedad **AttributeName** es `employeeCode`, que incluye solo un nombre de atributo. Es además un rango HashIndex.

Conceptos relacionados:

"Plug-ins para la indexación de datos" en la página 330
El HashIndex incorporado, la clase com.ibm.websphere.objectgrid.plugins.index.HashIndex, es un plug-in MapIndexPlugin que puede añadir a BackingMap para crear índices estáticos o dinámicos. Esta clase da soporte a las interfaces MapIndex y MapRangeIndex. La definición y la implementación de índices puede mejorar significativamente el rendimiento de la consulta.

"Plug-ins para la indexación personalizada de los objetos de memoria caché" en la página 336
Con un plug-in o índice MapIndexPlugin, puede escribir estrategias personalizadas de indexación que van más allá de los índices incorporados que proporciona eXtreme Scale.

"Utilización de un índice compuesto" en la página 339
El índice compuesto HashIndex mejora el rendimiento de la consulta y evita la costosa exploración de correlaciones. La característica también proporciona un método práctico para que la API HashIndex encuentre los objetos almacenados en memoria caché cuando los criterios de búsqueda implican muchos atributos.

"Índices" en la página 97
Utilice el plug-in MapIndexPlugin para crear un índice o varios índice en una BackingMap para dar soporte al acceso a datos no de clave.

Referencia relacionada:

"Atributos del plug-in HashIndex"
Puede utilizar los atributos siguientes para configurar el plug-in HashIndex. Estos atributos definen propiedades como por ejemplo si utiliza un HashIndex compuesto o de atributos, o si la indexación de rango está habilitada.

Atributos del plug-in HashIndex:

Puede utilizar los atributos siguientes para configurar el plug-in HashIndex. Estos atributos definen propiedades como por ejemplo si utiliza un HashIndex compuesto o de atributos, o si la indexación de rango está habilitada.

Atributos

Name Especifica el nombre del índice. El nombre debe ser exclusivo para cada correlación. El nombre se utiliza para recuperar el objeto de índice de la instancia de correlación de objeto para la correlación de respaldo.

AttributeName
Especifica los nombres delimitados por comas de los atributos que se van a indexar. Para los índices de acceso de campo, los nombre de atributo son equivalentes a los nombres de campo. Para índices de acceso de propiedad, los nombres de atributo son los nombres de propiedades compatibles JavaBean. Si solo existe un nombre de atributo, el HashIndex es un índice de un solo atributo. Si este atributo es una relación, también es un índice de relación. Si se incluyen varios nombres de atributo en los nombres de atributo, HashIndex es un índice compuesto.

FieldAccessAttribute
Se utiliza para las correlaciones sin entidad. Si tiene el valor `true`, se accede al objeto utilizando los campos directamente. Si no se especifica o es `false`, se utiliza el método de obtención para el atributo para acceder a los datos.

POJOKeyIndex
Se utiliza para las correlaciones sin entidad. Si `true`, el índice hará una

introspección del objeto en la parte de clave de la correlación. Este valor es útil cuando la clave es una clave compuesta y el valor no tiene la clave incorporada en él. Si no se especifica o es `false`, el índice hará una introspección del objeto en la parte de valor de la correlación.

RangeIndex
Si es `true`, la indexación de rango está habilitada y la aplicación puede difundir el objeto de índice recuperado a la interfaz MapRangeIndex. Si la propiedad **`RangeIndex`** se configura como `false`, la aplicación puede difundir el objeto de índice recuperado a la interfaz MapIndex solamente.

HashIndex de atributo único frente a HashIndex compuesto

Cuando la propiedad **`AttributeName`** de HashIndex incluye varios nombres de atributo, el HashIndex es un índice compuesto. De lo contrario, si incluye sólo un nombre de atributo, es un índice de atributo único. Por ejemplo, el valor de propiedad AttributeName de un HashIndex compuesto puede ser `city,state,zipcode`. Incluye tres atributos delimitados por comas. Si el valor de la propiedad **`AttributeName`** es solo `zipcode`, que solo tiene un atributo, es un HashIndex de un solo atributo.

El HashIndex compuesto proporciona una forma eficaz de buscar objetos almacenados en memoria caché, cuando los criterios de búsqueda implican muchos atributos. Sin embargo, no da soporte a índice de rango y su propiedad RangeIndex se debe establecer en `false`.

Consulte el tema sobre un HashIndex compuesto en la *Guía de administración*.

HashIndex de relación

Si el atributo indexado de HashIndex de atributo único es una relación, ya sea con un único valor o con varios, HashIndex es un HashIndex de relación. Para el HashIndex de relación, la propiedad RangeIndex de HashIndex se debe establecer en “false”.

El HashIndex de relación puede acelerar las consultas que utilizan referencias cíclicas o que utilizan los filtros de consulta IS NULL, IS EMPTY, SIZE y MEMBER OF. Para obtener más información, consulte “Optimización de consultas mediante el uso de índices” en la página 460la información sobre la optimización de consultas con índices en la *Guía de programación*.

HashIndex de clave

Para correlaciones no de entidad, cuando la propiedad **`POJOKeyIndex`** de HashIndex se establece en `true`, el HashIndex es un HashIndex de clave y la parte de clave de la entrada se utiliza para la indexación. Cuando no se especifica la propiedad AttributeName de HashIndex, toda la clave se indexa; de lo contrario, HashIndex de clave puede ser solo un HashIndex de atributo único.

Por ejemplo, añadir la propiedad siguiente al ejemplo anterior provoca que HashIndex se convierta en HashIndex de clave porque el valor de la propiedad POJOKeyIndex es `true`.

```
<property name="POJOKeyIndex" type="boolean" value="true"
description="indicates if POJO key HashIndex" />
```

En el ejemplo de índice de clave anterior, debido a que el valor de la propiedad **AttributeName** se especifica como `employeeCode`, el atributo indexado es el campo **employeeCode** de la parte de clave de la entrada de correlación. Si desea crear índice de claves en toda la parte de clave de la entrada de correlación, elimine la propiedad **AttributeName**.

HashIndex de rango

Cuando la propiedad RangeIndex de HashIndex se establece en `true`, el HashIndex es un índice de rango y puede dar soporte a la interfaz MapRangeIndex. Una implementación MapRangeIndex da soporte a funciones para buscar datos utilizando funciones de rango como, por ejemplo, mayor que, menor que, o ambos, mientras que un MapIndex solo da soporte a funciones de igual a. Para un índice de un solo atributo, la propiedad **RangeIndex** se puede establecer en `true` solo si el atributo indexado es de tipo Comparable. Si el índice de atributo único va a ser utilizado por la consulta, la propiedad RangeIndex se debe establecer en `true` y el atributo indexado debe ser del tipo Comparable. Para el HashIndex de relación y el HashIndex compuesto, la propiedad RangeIndex se debe establecer en `false`.

El ejemplo anterior es un HashIndex de rango porque el valor de la propiedad RangeIndex es `true`.

En la tabla siguiente se proporciona un resumen para utilizar un índice de rango.

Tabla 5. Soporte para el índice de rango. Establece si los tipos HashIndex admiten el índice de rango.

Tipo HashIndex	Soporta el índice de rango
HashIndex de atributo único: el atributo indexado o clave es del tipo Comparable	Sí
HashIndex de atributo único: la clave o atributo indexado no es del tipo Comparable	No
Índice compuesto HashIndex	No
HashIndex de relación	No

Optimización de consultas con plug-ins HashIndex

La definición de índices puede mejorar considerablemente el rendimiento de las consultas. Las consultas de WebSphere eXtreme Scale pueden utilizar plug-ins HashIndex incorporados para mejorar el rendimiento de las consultas. Aunque el uso de los índices puede mejorar significativamente el rendimiento de la consulta, podría tener un impacto en el rendimiento en las operaciones de correlación transaccional.

Conceptos relacionados:

"Plug-ins para la indexación de datos" en la página 330
El HashIndex incorporado, la clase com.ibm.websphere.objectgrid.plugins.index.HashIndex, es un plug-in MapIndexPlugin que puede añadir a BackingMap para crear índices estáticos o dinámicos. Esta clase da soporte a las interfaces MapIndex y MapRangeIndex. La definición y la implementación de índices puede mejorar significativamente el rendimiento de la consulta.

"Plug-ins para la indexación personalizada de los objetos de memoria caché"
Con un plug-in o índice MapIndexPlugin, puede escribir estrategias personalizadas de indexación que van más allá de los índices incorporados que proporciona eXtreme Scale.

"Utilización de un índice compuesto" en la página 339
El índice compuesto HashIndex mejora el rendimiento de la consulta y evita la costosa exploración de correlaciones. La característica también proporciona un método práctico para que la API HashIndex encuentre los objetos almacenados en memoria caché cuando los criterios de búsqueda implican muchos atributos.

"Índices" en la página 97
Utilice el plug-in MapIndexPlugin para crear un índice o varios índice en una BackingMap para dar soporte al acceso a datos no de clave.

Tareas relacionadas:

"Configuración del plug-in HashIndex" en la página 330
Puede configurar el HashIndex incorporado, la clase com.ibm.websphere.objectgrid.plugins.index.HashIndex, con un archivo XML, programáticamente, o con una anotación de entidad en una correlación de entidad.

"Acceso a datos con índices (API Index)" en la página 144
Utilice la indexación para acceder más eficazmente a los datos.

Plug-ins para la indexación personalizada de los objetos de memoria caché:

Con un plug-in o índice MapIndexPlugin, puede escribir estrategias personalizadas de indexación que van más allá de los índices incorporados que proporciona eXtreme Scale.

Las implementaciones de MapIndexPlugin deben utilizar la interfaz MapIndexPlugin y seguir las convenciones comunes de plug-in de eXtreme Scale.

Las secciones siguientes incluyen algunos de los métodos importantes de la interfaz index.

Método setProperties

Utilice el método setProperties para inicializar el plug-in index mediante programación. El parámetro del objeto Properties que se pasa en el método debe contener la información de configuración para inicializar el plug-in correctamente. La implementación del método setProperties, junto con la implementación del método getProperties, son necesarias en un entorno distribuido porque la configuración del plug-in index se mueve entre los procesos de cliente y servidor. A continuación se muestra un ejemplo de la implementación de este método.

```
setProperties(Properties properties)

// Código de ejemplo del método setProperties
    public void setProperties(Properties properties) {
        ivIndexProperties = properties;

        String ivRangeIndexString = properties.getProperty("rangeIndex");
```

```
        if (ivRangeIndexString != null && ivRangeIndexString.equals("true")) {
            setRangeIndex(true);
        }
        setName(properties.getProperty("indexName"));
        setAttributeName(properties.getProperty("attributeName"));

        String ivFieldAccessAttributeString = properties.getProperty("fieldAccessAttribute");
        if (ivFieldAccessAttributeString != null && ivFieldAccessAttributeString.equals("true")) {
            setFieldAccessAttribute(true);
        }

        String ivPOJOKeyIndexString = properties.getProperty("POJOKeyIndex");
        if (ivPOJOKeyIndexString != null && ivPOJOKeyIndexString.equals("true")) {
            setPOJOKeyIndex(true);
        }
    }
```

Método getProperties

El método getProperties extrae la configuración del plug-in index de una instancia de MapIndexPlugin. Puede utilizar las propiedades extraídas para inicializar otra instancia de MapIndexPlugin para que tenga los mismos estados internos. Las implementaciones del método getProperties y del método setProperties son necesarias en un entorno distribuido. A continuación, aparece una implementación de ejemplo del método getProperties:

```
getProperties()

// Código de ejemplo del método getProperties
    public Properties getProperties() {
        Properties p = new Properties();
        p.put("indexName", indexName);
        p.put("attributeName", attributeName);
        p.put("rangeIndex", ivRangeIndex ? "true" : "false");
        p.put("fieldAccessAttribute", ivFieldAccessAttribute ? "true" : "false");
        p.put("POJOKeyIndex", ivPOJOKeyIndex ? "true" : "false");
        return p;
    }
```

Método setEntityMetadata

La ejecución de WebSphere eXtreme Scale llama al método setEntityMetadata durante la inicialización para establecer el EntityMetadata del BackingMap asociado en la instancia de MapIndexPlugin. El EntityMetadata es necesario para dar soporte a la indexación de objetos tuple. Un tuple es un conjunto de datos que represente un objeto de entidad o su clave. Si la BackingMap es para una entidad, el usuario debe implementar este método.

El siguiente código de ejemplo implementa el método setEntityMetadata.

```
setEntityMetadata(EntityMetadata entityMetadata)

// Código de ejemplo del método setEntityMetadata
    public void setEntityMetadata(EntityMetadata entityMetadata) {
        ivEntityMetadata = entityMetadata;
        if (ivEntityMetadata != null) {
            // es una correlación de tuples
            TupleMetadata valueMetadata = ivEntityMetadata.getValueMetadata();
            int numAttributes = valueMetadata.getNumAttributes();
            for (int i = 0; i < numAttributes; i++) {
                String tupleAttributeName = valueMetadata.getAttribute(i).getName();
                if (attributeName.equals(tupleAttributeName)) {
                    ivTupleValueIndex = i;
                    break;
                }
            }

            if (ivTupleValueIndex == -1) {
                // no se encontró atributo en tuple de valor, intente encontrarlo en tuple de clave
                // no se encontró en tuple de clave, implica indexación de claves en uno de los atributos
                // de clave de tuple
                TupleMetadata keyMetadata = ivEntityMetadata.getKeyMetadata();
                numAttributes = keyMetadata.getNumAttributes();
                for (int i = 0; i < numAttributes; i++) {
                    String tupleAttributeName = keyMetadata.getAttribute(i).getName();
```

```
                if (attributeName.equals(tupleAttributeName)) {
                    ivTupleValueIndex = i;
                    ivKeyTupleAttributeIndex = true;
                    break;
                }
            }
        }

        if (ivTupleValueIndex == -1) {
            // si entityMetadata no es nulo y no se ha podido encontrar el
   // attributeName en entityMetadata, se trata de un
            // error
            throw new ObjectGridRuntimeException("Invalid attributeName.  Entity: " + ivEntityMetadata.getName());
        }
    }
}
```

Métodos de nombres de atributos

El método setAttributeName establece el nombre del atributo que se va a indexar. La clase de objeto almacenada en la memoria caché debe proporcionar el método get para el atributo indexado. Por ejemplo, si el objeto tiene un atributo employeeName o EmployeeName, el índice llama al método getEmployeeName en el objeto para extraer el valor de atributo. El nombre de atributo debe ser el mismo que el nombre que aparece en el método get, y el atributo debe implementar la interfaz Comparable. Si el atributo es del tipo booleano, también puede utilizar el patrón del método isAttributeName.

El método getAttributeName devuelve el nombre del atributo indexado.

Método getAttribute

El método getAttribute devuelve el valor de atributo indexado del objeto especificado. Por ejemplo, si un objeto Employee tiene un atributo llamado employeeName que está indexado, puede utilizar el método getAttribute para extraer el valor del atributo employeeName de un objeto Employee especificado. Este método es necesario en un entorno distribuido de WebSphere eXtreme Scale.

```
getAttribute(Object value)
```

```
// Código de ejemplo del método getAttribute
    public Object getAttribute(Object value) throws ObjectGridRuntimeException {
        if (ivPOJOKeyIndex) {
            // En el caso de indexación de claves POJO, no es necesario obtener el atributo
            // del objeto de valor.
            // La clave misma es el valor del atributo utilizado para compilar el índice.
            return null;
        }

        try {
            Object attribute = null;
            if (value != null) {
                // manejar valor de tuple si ivTupleValueIndex != -1
                if (ivTupleValueIndex == -1) {
                    // valor regular
                    if (ivFieldAccessAttribute) {
                        attribute = this.getAttributeField(value).get(value);
                    } else {
                        attribute = getAttributeMethod(value).invoke(value, emptyArray);
                    }
                } else {
                    // Nombre de tuple
                    attribute = extractValueFromTuple(value);
                }
            }
            return attribute;
        } catch (InvocationTargetException e) {
             throw new ObjectGridRuntimeException(
                    "Caught unexpected Throwable during index update processing,
                      index name = " + indexName + ": " + t,
                    t);
        } catch (Throwable t) {
             throw new ObjectGridRuntimeException(
                    "Caught unexpected Throwable during index update processing,
                      index name = " + indexName + ": " + t,
                    t);
        }
    }
```

Tareas relacionadas:

"Configuración del plug-in HashIndex" en la página 330
Puede configurar el HashIndex incorporado, la clase com.ibm.websphere.objectgrid.plugins.index.HashIndex, con un archivo XML, programáticamente, o con una anotación de entidad en una correlación de entidad.

"Acceso a datos con índices (API Index)" en la página 144
Utilice la indexación para acceder más eficazmente a los datos.

Referencia relacionada:

"Atributos del plug-in HashIndex" en la página 333
Puede utilizar los atributos siguientes para configurar el plug-in HashIndex. Estos atributos definen propiedades como por ejemplo si utiliza un HashIndex compuesto o de atributos, o si la indexación de rango está habilitada.

Utilización de un índice compuesto:

El índice compuesto HashIndex mejora el rendimiento de la consulta y evita la costosa exploración de correlaciones. La característica también proporciona un método práctico para que la API HashIndex encuentre los objetos almacenados en memoria caché cuando los criterios de búsqueda implican muchos atributos.

Rendimiento mejorado

Un HashIndex compuesto proporciona una forma rápida y práctica para buscar objetos almacenados en memoria caché con varios atributos en los criterios de búsqueda de coincidencia. El índice compuesto soporta búsquedas completas de coincidencia de atributo, pero no soporta las búsquedas de rango.

Nota: Los índices compuestos no soportan el operador BETWEEN en el lenguaje de consulta de ObjectGrid porque BETWEEN requeriría el soporte de rango. Los condicionales mayor que (>) y menor que (<) tampoco funcionan porque requieren los índices de rango.

Un índice compuesto puede mejorar el rendimiento de las consultas si el índice compuesto apropiado está disponible para la condición WHERE. Esto significa que el índice compuesto tiene exactamente los mismos atributos que los implicados en la condición WHERE con todos los atributos coincidentes.

Una consulta podría tener muchos atributos implicados en una condición como en el ejemplo siguiente.

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND
a.zipcode='55901'
```

El índice compuesto puede mejorar el rendimiento de la consulta al evitar tener que explorar la correlación o unir diversos resultados de índice de un único atributo. En el ejemplo, si un índice compuesto se define con atributos (city,state,zipcode), el motor de consultas puede utilizar el índice compuesto para encontrar la entrada con `city='Rochester'`, `state='MN'` y `zipcode='55901'`. Sin un índice compuesto y un índice de atributo en los atributos city, state y zipcode, el motor de consultas tendrá que explorar la correlación o unir varias búsquedas de atributo único, que normalmente tienen una sobrecarga costosa. Además, la consulta del índice compuesto sólo soporta un patrón de coincidencia completa.

Configuración de un índice compuesto

Puede configurar índices compuestos de tres formas: mediante XML, mediante programación o con anotaciones de entidad solo para correlaciones de entidad.

Configuración mediante programa

El código de ejemplo programático siguiente creará el mismo índice compuesto que el XML anterior.

```
HashIndex mapIndex = new HashIndex();
    mapIndex.setName("Address.CityStateZip");
    mapIndex.setAttributeName(("city,state,zipcode"));
    mapIndex.setRangeIndex(true);

    BackingMap bm = objectGrid.defineMap("mymap");
    bm.addMapIndexPlugin(mapIndex);
```

Observe que la configuración de un índice compuesto es igual que la configuración de un índice ordinario con XML excepto por el valor de la propiedad attributeName. En el caso de un índice compuesto, el valor de attributeName es una lista de atributos delimitada por comas. Por ejemplo, la clase de valor Address tiene 3 atributos: city, state y zipcode. Un índice compuesto se puede definir con el valor de propiedad attributeName como `"city,state,zipcode"` que indica que city, state y zipcode se incluyen en el índice compuesto.

Además, tenga en cuenta que los HashIndexes compuestos no soportan las búsquedas de rango y, por lo tanto, no pueden tener la propiedad RangeIndex establecida en true.

Mediante XML

Para poder configurar un índice compuesto con XML, incluya código como, por ejemplo, el que aparece abajo en el elemento backingMapPluginCollections del archivo de configuración.

```
Índice compuesto - Configuración mediante XML
<bean id="MapIndexPlugin"  className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Address.CityStateZip"/>
<property name="AttributeName" type="java.lang.String" value="city,state,zipcode"/>
</bean>
```

Con anotaciones de entidad

En el caso de correlaciones de entidad, el acercamiento de anotaciones puede utilizarse para definir un índice compuesto. Puede definir una lista de CompositeIndex dentro de una anotación CompositeIndexes en el nivel de clase de la entidad. El índice CompositeIndex tiene un nombre y una propiedad attributeNames. Cada índice CompositeIndex está asociado con una instancia HashIndex aplicada al elemento BackingMap asociado de la entidad. El índice HashIndex está configurado como un índice de no intervalo.

```
@Entity
@CompositeIndexes({
    @CompositeIndex(name=" CityStateZip ", attributeNames=" city,state,zipcode"),
    @CompositeIndex(name="lastnameBirthday", attributeNames=" lastname,birthday ")
 })
public class Address {
    @Id int id;
    String street;
    String city;
    String state;
    String zipcode;
    String lastname;
    Date birthday;
}
```

La propiedad name de cada índice compuesto debe ser exclusivo dentro de la entidad y BackingMap. Si no se especifica el nombre, se utilizará un nombre generado. La propiedad attributeNames se utiliza para rellenar la propiedad attributeName de HashIndex con la lista de atributos delimitada por comas. Los nombres de atributo coinciden con los nombres de campos persistentes, cuando las entidades se configuran para utilizar el acceso a campo, o el nombre de la propiedad se haya definido para los convenios de denominación de JavaBeans para las entidades de acceso a propiedades. Por ejemplo: si el nombre de atributo es "street", el método getter de la propiedad es getStreet.

Búsquedas en índices compuestos

Después de configurar un índice compuesto, una aplicación puede utilizar el método findAll(Object) de la interfaz MapIndex para realizar búsquedas, como en el ejemplo siguiente.

```
Session sess = objectgrid.getSession();
ObjectMap map = sess.getMap("MAP_NAME");
MapIndex codeIndex = (MapIndex) map.getIndex("INDEX_NAME");
Object[] compositeValue = new Object[]{ MapIndex.EMPTY_VALUE,
   "MN", "55901"};
   Iterator iter = mapIndex.findAll(compositeValue);
```

MapIndex.EMPTY_VALUE se asigna a compositeValue[0] que indica que el atributo city se excluye de la evaluación. Sólo los objetos con el atributo state igual a "MN" y el atributo zipcode igual a "55901" se incluirán en el resultado.

Las siguientes consultas se benefician de la configuración del índice compuesto anterior:

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND
a.zipcode='55901'

SELECT a FROM Address a WHERE a.state='MN' AND a.zipcode='55901'
```

El motor de consultas encontrará el índice compuesto apropiado y lo utiliza para mejorar el rendimiento de la consulta en casos de coincidencia de atributos total.

En algunos escenarios, la aplicación podría definir varios índices compuestos con atributos solapados para satisfacer todas las consultas con coincidencia total de los atributos. Un inconveniente de aumentar el número de índices es la posible sobrecarga de rendimiento en las operaciones de correlaciones.

Migración e interoperatividad

La única restricción del uso de un índice compuesto es que una aplicación no puede configurarlo en un entorno distribuido con contenedores heterogéneos. Los contenedores antiguos y nuevos no pueden mezclare, ya que los contenedores más antiguos no reconocerán una configuración de índice compuesto. El índice compuesto es como el índice de atributos ordinario existente, sólo que el primero permite la creación de índices sobre varios atributos. Si utiliza el índice de atributos ordinario, el uso del entorno de contenedores mixto es viable.

Tareas relacionadas:

"Configuración del plug-in HashIndex" en la página 330
Puede configurar el HashIndex incorporado, la clase com.ibm.websphere.objectgrid.plugins.index.HashIndex, con un archivo XML, programáticamente, o con una anotación de entidad en una correlación de entidad.

"Acceso a datos con índices (API Index)" en la página 144
Utilice la indexación para acceder más eficazmente a los datos.

Referencia relacionada:

"Atributos del plug-in HashIndex" en la página 333
Puede utilizar los atributos siguientes para configurar el plug-in HashIndex. Estos atributos definen propiedades como por ejemplo si utiliza un HashIndex compuesto o de atributos, o si la indexación de rango está habilitada.

Plug-ins para la comunicación con bases de datos

Con un plug-in Loader, una correlación de ObjectGrid se puede comportar como una memoria caché de memoria para datos que normalmente se mantienen en un almacén persistente en el mismo sistema o en algún otro sistema. Generalmente, se utiliza una base de datos o un sistema de archivos como almacenamiento persistente. También se puede utilizar una máquina virtual Java (JVM) remota como el origen de datos, lo que permite que las memorias caché basadas en hub se creen utilizando el ObjectGrid. Un cargador tiene la lógica para leer y escribir datos en un almacén persistente.

Los cargadores son plug-ins de correlaciones de respaldo que se invocan cuando se realizan cambios en la correlación de respaldo o ésta no puede satisfacer una solicitud de datos (una falta de memoria caché).

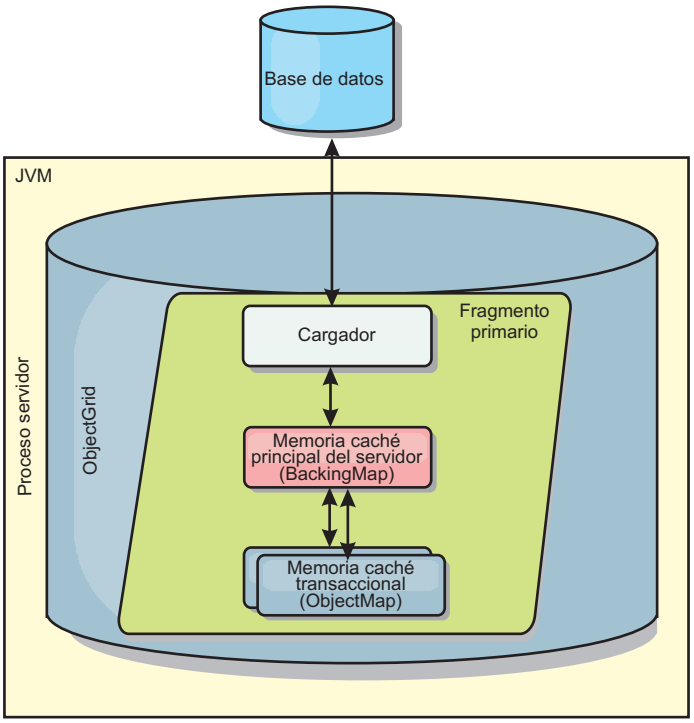


Figura 26. Cargador

WebSphere eXtreme Scale incluye dos cargadores incorporados para integrar con los programas de fondo de la base de datos relacional. Los cargadores JPA (Java Persistence API) utilizan las capacidades de correlación de objetos relacionales (ORM) de ambas implementaciones, OpenJPA e Hibernate, de la especificación JPA.

Utilización de un cargador

Para añadir un cargador a la configuración de BackingMap, puede utilizar la configuración programática o la configuración XML. Un cargador tiene la siguiente relación con una correlación de respaldo:

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

Cargadores en configuraciones multimaestro

Para ver las consideraciones sobre los cargadores en configuraciones multimaestro, consulte "Consideraciones sobre el cargador en una topología multimaestro" en la página 106.

Conexión de un cargador mediante programación

El siguiente fragmento de código demuestra cómo conectar un cargador proporcionado por la aplicación a una correlación de respaldo para map1 utilizando la API de ObjectGrid:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
MyLoader loader = new MyLoader();
loader.setDataBaseName("testdb");
loader.setIsolationLevel("read committed");
bm.setLoader( loader );
```

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

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

Enfoque de configuración XML para conectar un cargador

Un cargador proporcionado por una aplicación también puede conectarse mediante la configuración de un archivo XML. El ejemplo siguiente muestra cómo conectar el cargador MyLoader a la correlación de respaldo map1 con las mismas propiedades de cargador de nivel de aislamiento y nombre de base de datos. Debe especificar el className para el cargador, el nombre de la base de datos y los detalles de la conexión, y las propiedades del nivel de aislamiento. Puede utilizar la misma estructura XML si solo utiliza un precargador especificando el nombre de clase de precargador en lugar de un nombre de clase completo de cargador:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid">
        <backingMap name="map1" pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPluginCollection id="map1">
        <bean id="Loader" className="com.myapplication.MyLoader">
            <property name="dataBaseName"
                      type="java.lang.String"
                      value="testdb"
                      description="database name" />
            <property name="isolationLevel"
                      type="java.lang.String"
                      value="read committed"
                      description="iso level" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Referencia relacionada:

"Consideraciones de programación del cargador JPA" en la página 369
Un cargador Java Persistence API (JPA) es una implementación de plug-in de cargador que utiliza JPA para interactuar con la base de datos. Utilice las siguientes consideraciones cuando desarrolle una aplicación que utiliza un cargador JPA.

Configuración de cargadores de base de datos

Los cargadores son plug-ins de correlaciones de respaldo que se invocan cuando se realizan cambios en la correlación de respaldo o ésta no puede satisfacer una solicitud de datos (una falta de memoria caché).

Consideraciones de precarga

Los cargadores son plug-ins de correlaciones de respaldo que se invocan cuando se realizan cambios en la correlación de respaldo o ésta no puede satisfacer una solicitud de datos (una falta de memoria caché). Para obtener una visión general de cómo interactúa eXtreme Scale con un cargador, consulte "Memoria caché en línea" en la página 83.

Cada correlación de respaldo tiene un atributo preloadMode booleano establecido para indicar si una precarga de una correlación se ejecuta asíncronamente. De manera predeterminada, el atributo preloadMode está establecido en false, que indica que la inicialización de la correlación de respaldo no se completa hasta que la precarga de la correlación haya terminado. Por ejemplo, la inicialización de la correlación de respaldo no se completa hasta que se devuelve el método preloadMap. Si el método preloadMap lee una gran cantidad de datos de su programa de fondo y los carga en la correlación, puede que tarde en completarse. En ese caso, puede configurar una correlación de respaldo de modo que use una

precarga asíncrona de la correlación; para ello, establezca el atributo preloadMode en true. Este valor hace que el código de inicialización de la correlación de respaldo inicie una hebra que invoca el método preloadMap, lo que permite que se complete la inicialización de una correlación de respaldo mientras la precarga de la correlación aún está en curso.

En un caso de ejemplo de eXtreme Scale distribuido, uno de los patrones de precarga es la precarga de cliente. En el patrón de precarga de cliente, un cliente de eXtreme Scale es responsable de recuperar datos del programa de fondo y a continuación de insertar los datos en el servidor de contenedor distribuido utilizando agentes de DataGrid. Además, la precarga de cliente se puede ejecutar en el método Loader.preloadMap en una y sólo una partición específica. En este caso, es muy importante cargar asincrónicamente los datos en la cuadrícula. Si la precarga del cliente se ejecutase en la misma hebra, la correlación de respaldo nunca se inicializaría, de modo que la partición en la que reside nunca estaría ONLINE. Por lo tanto, el cliente de eXtreme Scale no podría enviar la solicitud a la partición y esto acabaría causando una excepción.

Si se utiliza un cliente de eXtreme Scale en el método preloadMap, debe establecer el atributo **preloadMode** en true. La alternativa debe consiste en iniciar una hebra en el código de precarga del cliente.

El fragmento de código siguiente ilustra cómo se establece el atributo preloadMode para habilitar la precarga asíncrona:

```
BackingMap bm = og.defineMap( "map1" );
bm.setPreloadMode( true );
```

El atributo preloadMode también puede establecerse mediante un archivo XML, como se muestra en el ejemplo siguiente:

```
<backingMap name="map1" preloadMode="true" pluginCollectionRef="map1"
lockStrategy="OPTIMISTIC"/>
```

TxID y uso de la interfaz TransactionCallback

El método get y los métodos batchUpdate de la interfaz Loader se pasan a un objeto TxID que representa la transacción Session que requiere que se realice la operación get o batchUpdate. Se puede llamar a los métodos get y batchUpdate más de una vez por transacción. Por lo tanto, los objetos con ámbito de transacción que el cargador necesita se conservan normalmente en una ranura del objeto TxID. Se utiliza un cargador JDBC (Java database connectivity) para ilustrar cómo utiliza un cargador las interfaces TxID y TransactionCallback.

Se pueden almacenar varias correlaciones ObjectGrid en la misma base de datos. Cada correlación tiene su propio cargador, y cada cargador podría conectarse a la misma base de datos. Cuando los cargadores se conectan a la base de datos, deben utilizar la misma conexión JDBC. La utilización de la misma conexión confirma los cambios en cada tabla como parte de la misma transacción de base de datos. Normalmente, la misma persona que escribe la implementación Loader también escribe la implementación TransactionCallback. El mejor método es cuando la amplía la interfaz TransactionCallback para añadir métodos que el cargador necesita para obtener una conexión de base de datos y para almacenar en memoria caché sentencias preparadas. Comprenderá porqué se recomienda este procedimiento en cuanto vea cómo utiliza el cargador las interfaces TransactionCallback y TxID.

En el ejemplo siguiente verá cómo el cargador necesita que la interfaz TransactionCallback se amplíe:

```
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 nuevos métodos,Loader y los métodos batchUpdate pueden obtener una conexión de la forma siguiente:

```
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 y ivOcb son variables de instancia del cargador que se inicializaron como se describió en el apartado sobre las consideraciones de precarga. La variable ivTcb es una referencia a la instancia de MyTransactionCallback e ivOcb es una referencia a la instancia de MyOptimisticCallback. La variable databaseName es una variable de instancia del cargador que se estableció como propiedad de cargador durante la inicialización de la correlación de respaldo. El argumento isolationLevel es una de las constantes JDBC Connection definidas para los diversos niveles de aislamiento que JDBC admite. Si el cargador utiliza una implementación optimista, el método get suele utilizar una conexión JDBC de confirmación automática para captar los datos de la base de datos. En ese caso, el cargador podría tener un método getAutoCommitConnection que se implementase de la siguiente manera:

```
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 siguiente sentencia switch:

```
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. A continuación se muestra un fragmento de código que ilustra cómo el método buildBatchSQLUpdate crea una sentencia update de SQL para actualizar una entrada EmployeeRecord y añadirla a la actualización de proceso 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();
}
```

Una vez que el bucle batchUpdate ha credo todas las sentencias preparadas, llama al método getPreparedStatementCollection. Este método se implementa de la siguiente manera:

```
private Collection getPreparedStatementCollection( TxID tx, Connection conn )
{
    return ( ivTcb.getPreparedStatementCollection( tx, conn, "employee" ) );
}
```

Cuando la aplicación invoca el método commit en Session, el código de Session llama al método commit en el método TransactionCallback después de haber enviado al cargador todos los cambios realizados por la transacción para cada correlación que la transacción modificó. Debido a que todos los cargadores utilizaron el método MyTransactionCallback para obtener las conexiones y las sentencias preparadas que necesitan, el método TransactionCallback sabe qué conexión utilizar para solicitar que el programa de fondo confirme los cambios. Por lo tanto, ampliar la interfaz TransactionCallback con los métodos que necesite cada uno de los cargadores tiene las ventajas siguientes:

- El objeto TransactionCallback encapsula el uso de ranuras de TxID para los datos con ámbito de transacción, y el cargador no requiere información sobre las ranuras de TxID. El cargador sólo necesita saber qué métodos se van a añadir a TransactionCallback mediante la interfaz MyTransactionCallback para las funciones que necesite el cargador.
- El objeto TransactionCallback puede garantizar que la conexión se comparta entre cada cargador que se conecte el mismo programa de fondo, de modo que puede evitarse un protocolo de confirmación de dos fases.
- Con el objeto TransactionCallback, la conexión al programa de fondo se completa a través de una confirmación o retrotracción que se invoca en la conexión cuando se necesita.
- TransactionCallback garantiza que se produzca la limpieza de los recursos de base de datos cuando se completa una transacción.
- TransactionCallback oculta si está obteniendo una conexión gestionada de un entorno gestionado como, por ejemplo, WebSphere Application Server u otro servidor de aplicaciones compatible con Java 2 Platform, Enterprise Edition

(J2EE). Esta ventaja permite que se utilice el mismo código de cargador en entornos gestionados y no gestionados. Sólo debe cambiarse el plug-in TransactionCallback.

- Para obtener más información sobre cómo la implementación TransactionCallback utiliza las ranuras de TxID para los datos con ámbito de transacción, consulte Plug-in TransactionCallback

OptimisticCallback

Como se ha mencionado anteriormente, el cargador puede utilizar un acercamiento optimista para conseguir un control de simultaneidad. En ese caso, el ejemplo del método buildBatchSQLUpdate debe modificarse ligeramente para implementar un acercamiento optimista. Existen diversas formas de utilizar un acercamiento optimista. Puede tener un columna de indicación de hora o una columna de contador de número de secuencia para añadir una versión a cada actualización de la fila. Presuponga que la tabla de empleados tiene una columna de número de secuencia que aumenta cada vez que se actualiza la fila. A continuación, deberá modificar la firma del método buildBatchSQLUpdate de modo que se pase al objeto LogElement en lugar del par clave/valor. También deberá utilizar el objeto OptimisticCallback conectado a la correlación de respaldo para obtener el objeto de versión inicial y para actualizar el objeto de versión. A continuación se muestra un ejemplo de un método buildBatchSQLUpdate modificado que utiliza la variable de instancia ivOcb que se inicializó como se describió en el apartado sobre preloadMap:

Ejemplo de código de método de actualización por lotes modificado

```
private void  buildBatchSQLUpdate( TxID tx, LogElement le, Connection conn )
 throws SQLException, LoaderException
{
    // Obtener el objeto de versión inicial cuando esta entrada de correlación se leyó
    // o actualizó por última vez 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 update de SQL.
    Long currentVersion = (Long)ivOcb.getVersionedObjectForValue( emp );
    long nextVersion = currentVersion.longValue();
    // A continuación cree una operación update de 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 el objeto LogElement para obtener el valor de versión inicial. Cuando la transacción accede por primera vez a la entrada de correlación, se crea un objeto LogElement con el objeto Employee inicial que se obtiene de la correlación. Este objeto Employee inicial se pasa también al método getVersionedObjectForValue en la interfaz OptimisticCallback y el resultado se guarda en el LogElement. Este proceso se produce antes de que se dé a la

aplicación una referencia al objeto Employee inicial, por lo que es posible llamar a algún método que cambie el estado del objeto Employee inicial.

El ejemplo muestra que el cargador utiliza el método getVersiondObjectForValue para obtener el objeto de versión para el objeto Employee actual y actualizado. Antes de llamar al método batchUpdate en la interfaz Loader, eXtreme Scale llama al método updateVersionedObjectForValue en la interfaz OptimisticCallback para provocar que se genere un objeto de una nueva versión para el objecto Employee actualizado. Una vez que el método batchUpdate vuelve a ObjectGrid, se actualiza el objeto LogElement con el objeto de versión actual y pasa a ser el nuevo objeto de versión inicial. Este paso es necesario porque la aplicación podría haber llamado al método flush en la correlación en lugar del método commit en Session. Es posible que una única transacción llame al cargador varias veces para la misma clave. Por dicho motivo, eXtreme Scale se asegura de que se actualice el objeto LogElement con el objeto de la nueva versión, cada vez que se actualiza 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 update de SQL que establezca la columna SEQNO en el valor del objeto de versión siguiente y utilice el valor del objeto de versión inicial en la cláusula where. Este procedimiento suele denominarse sentencia de actualización sobrecualificada. El uso de la sentencia de actualización sobrecualificada permite que la base de datos relacional verifique que ninguna otra transacción modificó la fila entre el tiempo en que esta transacción lee los datos de la base de datos y el tiempo en que actualiza la base de datos. Si otra transacción ha modificado la fila, la matriz de números devuelta por la actualización de proceso por lotes indica que ninguna fila se actualizó para esta clave. El cargador debe verificar que la operación update de SQL ha actualizado verdaderamente la fila. Si no se ha actualizado, el cargador muestra una excepción com.ibm.websphere.objectgrid.plugins.OptimisticCollisionException para informar a Session de que se ha producido una anomalía en el método batchUpdate debido a la existencia de más de una transacción simultánea intentando actualizar la misma fila de la tabla de la base de datos. Al recibir esta excepción, Session se retrotrae y la aplicación debe volver a repetir la transacción entera. La explicación es que esta repetición será correcta, de ahí que este procedimiento se llame optimista. El procedimiento optimista funciona mejor si los datos no se modifican con frecuencia o si transacciones simultáneas rara vez intentan actualizar la misma fila.

Es importante que el cargador utilice el parámetro clave del constructor OptimisticCollisionException para identificar qué clave o conjunto de claves provocó la anomalía en el método batchUpdate optimista. El parámetro clave puede ser el propio objeto clave o una matriz de objetos clave si se obtuvo más de una clave en la anomalía de actualización optimista. eXtreme Scale 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 consiste en desalojar de la correlación cada entrada de correlación obsoleta. El desalojo de las entradas obsoletas es necesario para que las transacciones subsiguientes que accedan a la misma clave o claves llamen al método get de la interfaz Loader para renovar las entradas de correlación con los datos actuales de la base de datos.

Otras maneras que tiene un cargador de implementar un procedimiento optimista son:

- No existe columna de indicación de hora ni columna de número de secuencia. En ese caso, el método getVersionObjectForValue de la interfaz

OptimisticCallback devuelve simplemente el objeto de valor como versión. Con este procedimiento, el cargador necesita crear una cláusula where que incluya cada uno de los campos del objeto de versión inicial. Este procedimiento no es eficaz, y no todos los tipos de columna se pueden utilizar en la cláusula where de una sentencia update de SQL sobrecualificada. No se suele utilizar este procedimiento.

- No existe columna de indicación de hora ni columna de número de secuencia. No obstante, a diferencia del procedimiento anterior, la cláusula where sólo contiene los campos de valor que ha modificado la transacción. Otro método para detectar qué campos se han modificado consiste en establecer la modalidad de copia en la correlación de respaldo como modalidad CopyMode.COPY_ON_WRITE. Esta modalidad de copia requiere que una interfaz de valor se pase al método setCopyMode en la interfaz BackingMap. BackingMap crea objetos de proxy dinámicos que implementan la interfaz de valor proporcionada. Con esta modalidad de copia, el cargador puede difundir cada valor a un objeto com.ibm.websphere.objectgrid.plugins.ValueProxyInfo. La interfaz ValueProxyInfo tiene un método que permite al cargador obtener la lista de los nombres de atributos modificados por la transacción. Este método permite que el cargador llame a los métodos get en la interfaz de valor de los nombres de atributos para obtener los datos modificados y para crear una sentencia update de SQL que sólo establezca los atributos modificados. A continuación, puede crearse la cláusula where de modo que tenga la columna de claves primarias más cada una de las columnas de atributos modificados. Este procedimiento es mucho más eficaz que el anterior, pero requiere escribir más código en el cargador y puede que la memoria caché de las sentencias preparadas necesite ser de gran tamaños para poder manejar las diferentes permutaciones. Sin embargo, si las transacciones sólo modifican unos pocos atributos, esta limitación no supondría un problema.
- Puede que algunas bases de datos relacionales tengan una API que sirva de ayuda en el mantenimiento automático de los datos de columnas que resulten útiles en la creación optimista de versiones. Consulte la documentación de la base de datos para determinar si existe esta posibilidad.

Escribir un cargador

Puede escribir su propia implementación de plug-in de cargador en sus aplicaciones, que debe seguir los convenios de plug-in de WebSphere eXtreme Scale.

Incluir un plug-in de cargador

La definición de la interfaz Loader es la siguiente:

```
public interface Loader
{
    static final SpecialValue KEY_NOT_FOUND;
    List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException;
    void batchUpdate(TxID txid, LogSequence sequence) throws
    LoaderException, OptimisticCollisionException;
    void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
}
```

Si desea más información, consulte “Cargadores” en la página 90.

Método get

La correlación de respaldo llama al método get del cargador para obtener los valores asociados a una lista de claves que se pasa como el argumento keyList. El método get es necesario para devolver una lista java.lang.util.List de valores, un valor para cada clave que aparece en la lista de claves. El primer valor que se

devuelve en la lista de valores corresponde a la primera clave de la lista de claves, el segundo valor devuelto en la lista de valores corresponde a la segunda clave de la lista de claves, etc. Si un cargador no encuentra el valor de una clave en la lista de claves, se solicita al cargador que devuelva el objeto de valor especial KEY_NOT_FOUND que se define en la interfaz Loader. Puesto que se puede configurar una correlación de respaldo para permitir `null` como un valor válido, es muy importante para el cargador devolver el objeto especial KEY_NOT_FOUND cuando el cargador no puede encontrar la clave. Este valor especial permite a la correlación de respaldo distinguir entre un valor nulo y un valor que no existe porque no se encontró. Si una correlación de respaldo no admite valores nulos, se producirá una excepción en un cargador que devuelva un valor nulo en lugar del objeto KEY_NOT_FOUND para una clave que no exista.

El argumento forUpdate indica al cargador si la aplicación llamó a un método get en la correlación o a un método getForUpdate en la correlación. Consulte Interfaz ObjectMap en la documentación de la API si desea más información. El cargador es responsable de implementar una política de control de simultaneidad que controle los accesos simultáneos al almacén persistente. Por ejemplo, numerosos sistemas de gestión de bases de datos relacionales admiten la sintaxis FOR UPDATE de la sentencia select de SQL que se utiliza para leer los datos de una tabla relacional. El cargador puede elegir utilizar la sintaxis FOR UPDATE en la sentencia select de SQL basándose en si se ha pasado `boolean true` como el valor del argumento para el parámetro forUpdate de este método. Normalmente, el cargador utilizar la sintaxis FOR UPDATE sólo cuando se utiliza la política de control de simultaneidad pesimista. Para un control de simultaneidad optimista, el cargador nunca utiliza la sintaxis `for update` en la sentencia select de SQL. El cargador deberá decidir si va a utilizar el argumento forUpdate basado en la política de control de simultaneidad que utiliza el cargador.

Si desea una explicación del parámetro txid, consulte “Plug-ins para gestionar los sucesos del ciclo de vida de transacciones” en la página 385.

Método batchUpdate

El método batchUpdate es importante en la interfaz Loader. Este método se llama siempre que eXtreme Scale necesita aplicar todos los cambios actuales al cargador. Se proporciona al cargador una lista de cambios para la correlación seleccionada. Los cambios se repiten y se aplican al programa de fondo. El método recibe el valor TxID actual y los cambios que se aplicarán. El siguiente ejemplo se repite en el conjunto de cambios y procesa por lotes tres sentencias JDBC (Java database connectivity), una con insert, otra con update y una con delete.

```
import java.util.Collection;
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;

    public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException {
        // Obtener una conexión SQL que vaya a utilizarse.
        Connection conn = getConnection(tx);
        try {
            // Procesar la lista de cambios y crear un conjunto de
            // sentencias preparadas para ejecutar una
            // operación SQL de batch update, insert o delete.
            Iterator iter = sequence.getPendingChanges();
            while (iter.hasNext()) {
                LogElement logElement = (LogElement) iter.next();
                Object key = logElement.getKey();
                Object value = logElement.getCurrentValue();
                switch (logElement.getType().getCode()) {
                case LogElement.CODE_INSERT:
```

```
                buildBatchSQLInsert(tx, key, value, conn);
                break;
            case LogElement.CODE_UPDATE:
                buildBatchSQLUpdate(tx, key, value, conn);
                break;
            case LogElement.CODE_DELETE:
                buildBatchSQLDelete(tx, key, conn);
                break;
            }
        }
        // Ejecutar las sentencias de proceso por lotes creadas mediante el bucle anterior.
        Collection statements = getPreparedStatementCollection(tx, conn);
        iter = statements.iterator();
        while (iter.hasNext()) {
            PreparedStatement pstmt = (PreparedStatement) iter.next();
            pstmt.executeBatch();
        }
    } catch (SQLException e) {
        LoaderException ex = new LoaderException(e);
        throw ex;
    }
}
```

El ejemplo anterior ilustra la lógica de alto nivel de proceso del argumento LogSequence, pero no ilustra los detalles sobre cómo se crea una sentencia insert, update o delete de SQL. Algunos de los puntos claves que se ilustran son:

- Se llama al método getPendingChanges en el argumento LogSequence para obtener un repetidor en la lista de LogElements que debe procesar el cargador.
- El método LogElement.getType().getCode() se utiliza para determinar si el LogElement es para una operación insert, update o delete de SQL.
- Se obtiene una excepción SQLException que se encadena a una excepción LoaderException que se imprime para informar de que se ha producido una excepción durante la actualización de proceso por lotes.
- Se utiliza el soporte de actualización de proceso por lotes JDBC para minimizar el número de consultas que deben hacerse en el programa de fondo.

Método preloadMap

Durante la inicialización de eXtreme Scale, se inicializa cada correlación de respaldo definida. Si se conecta un cargador en una correlación de respaldo, esta correlación invoca el método preloadMap en la interfaz Loader para permitir al cargador que busque previamente los datos en su programa de fondo y los cargue en la correlación. En el ejemplo siguiente se presupone que las primeras 100 filas de una tabla Employee de empleados se leen de la base de datos y se cargan en la correlación. La clase EmployeeRecord es una clase proporcionada por una aplicación que aloja los datos de empleado que se leen en la tabla de empleados.

Nota: Esta muestra capta todos los datos de la base de datos y luego los inserta en la correlación base de una partición. En un caso de ejemplo de despliegue de eXtreme Scale distribuido del mundo real, los datos se deberían distribuir entre todas las particiones. Consulte “Desarrollo de cargadores JPA basados en cliente” en la página 402 para obtener más información.

```
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException

    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException {
        boolean tranActive = false;
        ResultSet results = null;
        Statement stmt = null;
        Connection conn = null;
        try {
            session.beginNoWriteThrough();
            tranActive = true;
            ObjectMap map = session.getMap(backingMap.getName());
            TxID tx = session.getTxID();
            // Obtener una conexión de confirmación automática que esté establecida en
```

```
            // un nivel de aislamiento de lectura confirmada.
            conn = getAutoCommitConnection(tx);
            // Precargar la correlación de empleados con objetos EmployeeRecord
            // . Leer todos los empleados de la tabla, pero
            // limitar la precarga a las primeras 100 filas.
            stmt = conn.createStatement();
            results = stmt.executeQuery(SELECT_ALL);
            int rows = 0;
            while (results.next() && rows < 100) {
                int key = results.getInt(EMPNO_INDEX);
                EmployeeRecord emp = new EmployeeRecord(key);
                emp.setLastName(results.getString(LASTNAME_INDEX));
                emp.setFirstName(results.getString(FIRSTNAME_INDEX));
                emp.setDepartmentName(results.getString(DEPTNAME_INDEX));
                emp.updateSequenceNumber(results.getLong(SEQNO_INDEX));
                emp.setManagerNumber(results.getInt(MGRNO_INDEX));
                map.put(new Integer(key), emp);
                ++rows;
            }
            // Confirmar la transacción.
            session.commit();
            tranActive = false;
        } catch (Throwable t) {
            throw new LoaderException("preload failure: " + t, t);
        } finally {
            if (tranActive) {
                try {
                    session.rollback();
                } catch (Throwable t2) {
                    // Tolerar anomalías de retrotracción y
                    // permitir que se emitan objetos Throwable originales.
                }
            }
            // Limpie otros recursos de base de datos aquí
            // como cierre de sentencias, conjuntos de resultados, etc.
        }
    }
```

Este ejemplo ilustra los puntos clave siguientes:

- La correlación de respaldo preloadMap utiliza el objeto Session que se ha pasado a aquélla como argumento de sesión.
- Se utiliza el método Session.beginNoWriteThrough para iniciar la transacción, en lugar del método begin.
- No se puede llamar al cargador para cada operación put que se produce en este método para cargar la correlación.
- El cargador puede correlacionar las columnas de la tabla de empleados con un campo en el objeto EmployeeRecord Java. El cargador obtiene todas las excepciones throwable que se producen y emite una excepción LoaderException con la excepción throwable obtenida encadenada a él.
- El bloque finally garantiza que cualquier excepción throwable que se produzca entre el momento en que se llama al método beginNoWriteThrough y el momento en que se llama al método commit provoque que el bloque finally retrotraiga la transacción activa. Esta acción es crítica para garantizar que cualquier transacción que haya sido iniciada por el método preloadMap se complete antes de devolverla al llamante. El bloque finally es un buen punto para realizar otras acciones de limpieza que podrían ser necesarias, como el cierre de la conexión JDBC (Java Database Connectivity) y otros objetos JDBC.

El ejemplo de preloadMap utiliza una sentencia select de SQL que selecciona todas las filas de la tabla. En el cargador que proporciona la aplicación, puede que necesite establecer una o más propiedades del cargador para controlar cuánta información de la tabla debe precargarse en la correlación.

Como el método preloadMap sólo se llama una vez durante la inicialización de BackingMap, es un buen momento para ejecutar el código de inicialización del cargador de una sola vez. Aunque el cargador decida no buscar previamente los datos en el programa de fondo y cargar los datos en la correlación, probablemente necesite realizar algunas operaciones de inicialización de una sola vez para hacer más eficaces otros métodos del cargador. El ejemplo siguiente ilustra el

almacenamiento en memoria caché del objeto TransactionCallback y del objeto OptimisticCallback como variables de instancia del cargador. De esta manera, los otros métodos del cargador no tienen que realizar llamadas de método para obtener acceso a estos objetos. Este almacenamiento en memoria caché de los valores del plug-in ObjectGrid puede realizarse porque, después de que BackingMap se haya inicializado, los objetos TransactionCallback y OptimisticCallback no pueden cambiarse ni sustituirse. Es aceptable almacenar en memoria caché estas referencias de objeto como variables de instancia del cargador.

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.OptimisticCallback;
import com.ibm.websphere.objectgrid.plugins.TransactionCallback;

    // Variables de instancia del cargador.
    MyTransactionCallback ivTcb; // MyTransactionCallback

    // amplía TransactionCallback
    MyOptimisticCallback ivOcb; // MyOptimisticCallback

    // implementa OptimisticCallback
    // ...
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
   [Programación de réplica]
        // Almacenar en memoria caché los objetos TransactionCallback y OptimisticCallback
        // en variables de instancia de este cargador.
        ivTcb = (MyTransactionCallback) session.getObjectGrid().getTransactionCallback();
        ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
        // Resto de código de preloadMap (como se muestra en el ejemplo anterior).
    }
```

Para obtener información sobre la precarga y la precarga recuperable en relación a la migración tras error de réplica, consulte Réplica para la disponibilidadla información sobre la réplica en la *Visión general del producto*.

Cargadores con correlaciones de entidad

Si el cargador se conecta a una correlación de entidad, el cargador debe manejar los objetos de tuple. Los objetos de tuple son un formato especial de datos de entidad. El cargador debe realizar la conversión de datos entre tuple y otros formatos de datos. Por ejemplo, el método get devuelve una lista de valores que corresponden al conjunto de claves que se pasan en el método. La claves que se pasan son de tipo Tuple, es decir, tuples de clave. Si se presupone que el cargador persiste la correlación con una base de datos utilizando JDBC, el método get debe convertir cada tuple de clave en una lista de valores de atributo que corresponden a las columnas de clave primaria de la tabla que se correlaciona con la correlación de entidad, ejecute la sentencia SELECT con la cláusula WHERE que utiliza los valores de atributo convertidos como criterios para captar datos en la base de datos y, a continuación, convertir los datos devueltos en tuples de valor. El método get obtiene datos de la base de datos y los convierte en tuples de valor para los tuples de clave pasados y, a continuación, devuelve una lista de tuples de valor correspondientes al conjunto de claves de tuple que se pasan en al llamante. El método get puede realizar una sentencia SELECT para captar todos los datos a la vez, o ejecutar una sentencia SELECT para cada tuple de clave. Si desea ver detalles de programación que muestran cómo utilizar el cargador cuando se almacenan los datos utilizando un gestor de entidades, consulte “Uso de un cargador con correlaciones de entidad y tuples” en la página 375.

Referencia relacionada:

"Consideraciones de programación del cargador JPA" en la página 369
Un cargador Java Persistence API (JPA) es una implementación de plug-in de cargador que utiliza JPA para interactuar con la base de datos. Utilice las siguientes consideraciones cuando desarrolle una aplicación que utiliza un cargador JPA.

Precarga de correlaciones

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

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

Precarga en el MapSet particionado

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

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

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

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

Rendimiento

La implementación de la precarga copia datos del programa de fondo en la correlación almacenando varios objetos en la correlación de una única transacción. El número óptimo de registros para almacenar por transacción depende de varios factores, incluidos la complejidad y el tamaño. Por ejemplo, después de que la transacción incluya bloques de más de 100 entradas, se reduce la ventaja del rendimiento a medida que aumenta el número de entradas. Para determinar el número óptimo, empiece con 100 entradas y, a continuación, aumente el número

hasta que no se detecte más aumento en el rendimiento. Las transacciones de mayor tamaño dan como resultado un mayor rendimiento de duplicación. Recuerde que sólo el fragmento primario ejecuta el código de precarga. Los datos cargados previamente se duplican desde el fragmento primario hasta todas las réplicas que están en línea.

Precarga de MapSets

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

Precarga recuperable

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

Tabla 6. Valor de estado y respuesta

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

Evidentemente, cuando un fragmento primario está cargando la correlación, debe dejar algún estado en una correlación del MapSet que se está duplicando para que la réplica determine qué estado debe devolver. Puede utilizar una correlación adicional llamada, por ejemplo, RecoveryMap. Esta RecoveryMap debe formar parte del mismo MapSet que se está precargando para asegurarse de que la

correlación se duplica coherentemente con los datos que se están precargando. A continuación se muestra una implementación sugerida.

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

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

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

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

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

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

Tabla 7. Secuencia de confirmación del fragmento primario

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

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

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

Durante el proceso de confirmaciónsíncrono, utilice la siguiente secuencia:

Tabla 8. Proceso de confirmación síncrona

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

Para el proceso de réplicas, utilice la siguiente secuencia:

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

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

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

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

Configuración del soporte de cargador de grabación diferida

Puede habilitar el soporte de grabación diferida utilizando el archivo XML de descriptor de ObjectGrid, o a través de programa utilizando la interfaz BackingMap.

Utilice el archivo XML de descriptor de ObjectGrid para habilitar el soporte de grabación diferida, o a través de programa mediante la interfaz BackingMap.

Archivo XML de descriptor ObjectGrid

Cuando se configura un ObjectGrid utilizando un archivo XML de descriptor de ObjectGrid, el cargador de grabación diferida se habilita estableciendo el atributo writeBehind en el código backingMap. A continuación se muestra un ejemplo:

```
<objectGrid name="library" >
     <backingMap name="book" writeBehind="T300;C900" pluginCollectionRef="bookPlugins"/>
```

En el ejemplo anterior, el soporte de grabación diferida de la correlación de respaldo book se habilita con el parámetro T300;C900. El atributo de grabación diferida especifica el tiempo de actualización máximo y/o un recuento máximo de actualizaciones de claves. El formato del parámetro de grabación diferida es:

```
atributo de grabación diferida  ::= <predeterminado> | <hora actualización> | <recuento claves actualización> |
<hora actualización> ";" <recuento claves actualización>
hora actualización ::= "T" <entero positivo>
recuento claves actualización ::= "C" <entero positivo>
valores predeterminados  ::= "" {table}
```

Las actualizaciones en el cargador se producen cuando se produce uno de los siguientes sucesos:

1. Ha transcurrido el tiempo máximo de actualización en segundos desde la última actualización.
2. El número de claves actualizadas en la correlación de colas ha alcanzado el recuento de claves de actualización.

Estos parámetros sólo son sugerencias. El recuento de actualizaciones y la hora de actualización reales estarán en un rango cercano de parámetros. Sin embargo, no se garantiza que el recuento de actualizaciones real o la hora de actualización sean los mismos que se han definido en los parámetros. Además, la primera actualización diferida podría darse hasta con dos veces más de tiempo que la hora de actualización. Esto se debe a que ObjectGrid elige aleatoriamente la hora de inicio de la actualización para que todas las particiones no accedan a la base de datos simultáneamente.

En el ejemplo anterior T300;C900, el cargador escribe los datos en el programa de fondo cuando han transcurrido 300 después de la última actualización o cuando hay 900 claves pendientes para actualizar. La hora de actualización predeterminada es de 300 segundos y el recuento de claves de actualización predeterminado.

Tabla 9. Algunas opciones de escritura diferida

Valor de atributo	Hora
T100	La hora de actualización es 100 segundos y el recuento de claves de actualización predeterminado es 1000 (el valor predeterminado)
C2000	La hora de actualización es 300 segundos (el valor predeterminado) y el recuento de claves de actualización es 2000.
T300;C900	La hora de actualización es 300 segundos y el recuento de claves de actualización es 900.
""	La hora de actualización es 300 segundos (el valor predeterminado) y el recuento de claves de actualización es 1000 (el valor predeterminado). **Nota:** Si configura el cargador de grabación diferida como una serie vacía: `writeBehind=""`, el cargador de grabación diferida se habilita utilizando los valores predeterminados. Por lo tanto, no especifique el atributo writeBehind si no desea que el soporte de grabación anticipada esté habilitado.

Habilitación mediante programación del soporte de grabación diferida

Al crear una correlación de respaldo mediante programación para un eXtreme Scale en memoria local, puede utilizar el método siguiente en la interfaz BackingMap para habilitar e inhabilitar el soporte de grabación diferida.

```
public void setWriteBehind(String writeBehindParam);
```

Para obtener más detalles sobre cómo utilizar el método setWriteBehind, consulte la información sobre la interfaz BackingMap en la *Guía de programación*.

Referencia relacionada:

"Ejemplo: Escribir una clase de volcador de grabación diferida" en la página 366
Este código fuente de ejemplo muestra cómo escribir un observador (volcador) para manejar actualizaciones de grabación diferida anómalas.

Almacenamiento en memoria caché de grabación diferida:

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

Visión general del almacenamiento en memoria caché con grabación diferida

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

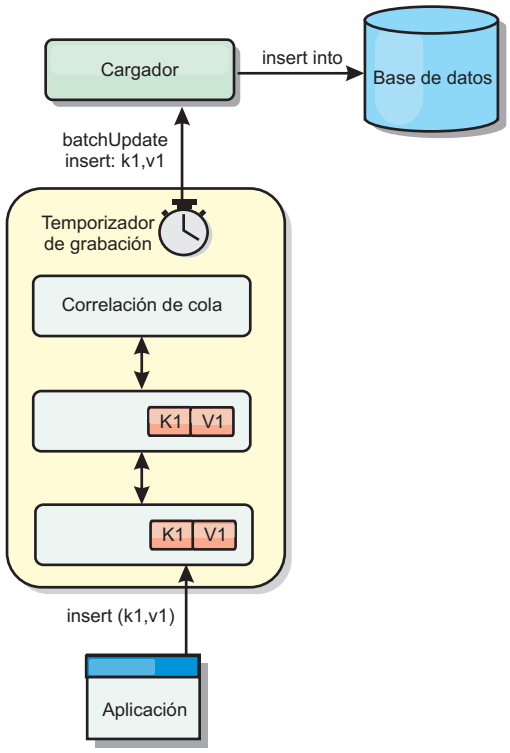


Figura 27. Almacenamiento en memoria caché de grabación diferida

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

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

El soporte de grabación diferida *es* una ampliación del plug-in Loader, que puede utilizar para integrar eXtreme Scale con la base de datos. Por ejemplo, consulte la información del apartado Configuración de cargadores JPA sobre cómo configurar un cargador JPA.

Ventajas

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

- **Aislamiento de anomalía de programa de fondo:** el almacenamiento de grabación diferida proporciona una capa de aislamiento de las anomalías de programa de fondo. Cuando la base de datos de programa de fondo falla, las actualizaciones se ponen en cola en la correlación de cola. Las aplicaciones

pueden continuar con las transacciones a eXtreme Scale. Cuando se recupera el programa de fondo, los datos de la correlación de cola se envían al programa de fondo.

- **Carga reducida de programa de fondo** el cargador de grabación diferida fusiona las actualizaciones según una clave, de forma que sólo existe una actualización fusionada por clave en la correlación de cola. Este procedimiento reduce el número de actualizaciones en la base de datos de programa de fondo.
- **Rendimiento mejorado de transacciones:** los tiempos individuales de las transacciones de eXtreme Scale se reducen porque la transacción no necesita esperar a que los datos se sincronicen con el programa de fondo.

Consideraciones sobre el diseño de aplicaciones

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

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

Restricciones de la integridad referencial

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

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

Comportamiento de bloqueo de correlaciones de cola

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

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

Debido a este comportamiento de correlación de colas adicional, puede ver algunas diferencias en el comportamiento del bloqueo.

- Si la correlación de usuarios está configurada como estrategia de bloqueo pesimista (PESSIMISTIC), no hay mucha diferencia de comportamiento en el bloqueo. Cada vez que se llama a una operación de desecho o confirmación, se coloca un bloqueo S en la misma clave de la misma correlación de colas. Durante la confirmación, no sólo se adquiere un bloqueo X para la clave en la correlación de usuarios, sino que además de adquiere para la clave en la correlación de colas.
- Si la correlación de usuarios está configurada como estrategia de bloqueo optimista (OPTIMISTIC) o ninguna (NONE), la transacción de usuario seguirá el patrón de estrategia de bloqueo pesimista (PESSIMISTIC). Cada vez que se llama a una operación de desecho o confirmación, se adquiere un bloqueo S en la misma clave de la misma correlación de colas. Durante la confirmación se adquiere un bloqueo X para la clave en la correlación de colas utilizando la misma transacción.

Reintentos de transacción de cargador

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

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

Anomalías del cargador

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

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

- Si el cargador de grabación diferida obtiene una excepción LoaderException, el cargador de grabación diferida considerará la anomalía como un error de los datos, como por ejemplo un error de clave duplicada. El cargador de grabación diferida anulará el proceso por lotes de la actualización, e intentará actualizar un registro cada vez para aislar la anomalía de los datos. Si se vuelve a obtener una excepción {{LoaderException}} durante la actualización de un registro, se crea un registro de actualización con errores y se anota en la correlación de actualizaciones con errores.
- Si el cargador de grabación diferida obtiene una excepción LoaderNotAvailableException, el cargador de grabación diferida la considerará como un error porque no puede conectase a la base de datos, por ejemplo, el programa de fondo de la base de datos está inactivo, una conexión de base de datos no está disponible, o la red no está activa. El cargador de grabación diferida esperará 15 segundos y después volverá a intentar realizar la actualización por lotes en la base de datos.

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

Consideraciones sobre el rendimiento

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

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

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

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

Referencia relacionada:

"Ejemplo: Escribir una clase de volcador de grabación diferida" en la página 366
Este código fuente de ejemplo muestra cómo escribir un observador (volcador) para manejar actualizaciones de grabación diferida anómalas.

Manejo de actualizaciones de grabación diferida erróneas:

Puesto que la transacción de WebSphere eXtreme Scale termina antes de que se inicie la transacción de programa de fondo, es posible que se produzca un éxito falso de la transacción.

Si intenta insertar una entrada en una transacción de eXtreme Scale que no existe en la correlación de respaldo, pero existe en el programa de fondo, lo que genera una clave duplicada, la transacción de eXtreme Scale se realiza correctamente. Sin embargo, la transacción en la que la hebra de grabación diferida inserta el objeto en el programa de fondo falla con una excepción de clave duplicada.

Manejo de las actualizaciones de grabación diferida con errores: lado del cliente

Una actualización de este tipo, o cualquier otra actualización de programa de fondo con errores, es una actualización de grabación diferida con errores. Estas actualizaciones de grabación diferida con errores se almacenan en una correlación de actualizaciones de grabación diferida con errores. Esta correlación sirve como cola de sucesos para actualizaciones con errores. La clave de la actualización es un objeto Integer exclusivo, y el valor es una instancia del elemento FailedUpdateElement. La correlación anómala de actualización de escritura diferida se ha configurado con un desalojador, que desaloja los registros one hora después de que se hayan insertado. Por lo tanto, los registros de actualización anómala se perderán si no se recuperan en un plazo da una hora.

La API ObjectMap puede utilizarse para recuperar las entradas de correlación de actualizaciones de grabación diferida con errores. El nombre de la correlación de actualización de grabación diferida es: `IBM_WB_FAILED_UPDATES_<nombre de la correlación>`. Consulte la documentación de la API WriteBehindLoaderConstants para conocer los nombre de los prefijos de cada correlación de sistema de grabación diferida. Lo que aparece a continuación es un ejemplo.

proceso anómalo - código de ejemplo

```
ObjectMap failedMap = session.getMap(
    WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + "Employee");
Object key = null;

session.begin();
while(key = failedMap.getNextKey(ObjectMap.QUEUE_TIMEOUT_NONE)) {
    FailedUpdateElement element = (FailedUpdateElement) failedMap.get(key);
    Throwable throwable = element.getThrowable();
    Object failedKey = element.getKey();
    Object failedValue = element.getAfterImage();
    failedMap.remove(key);
    // Realizar alguna acción con la clave, el valor o la excepción.

}
session.commit();
```

Una llamada al método getNextKey funciona con una partición específica para cada transacción de eXtreme Scale. En un entorno distribuido, para obtener claves de todas las particiones, debe iniciar varias transacciones, como se muestra en el ejemplo siguiente:

obtención de claves de todas las particiones - código de ejemplo

```
ObjectMap failedMap = session.getMap(
    WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + "Employee");
while (true) {
    session.begin();
    Object key = null;
    while(( key = failedMap.getNextKey(5000) )!= null ) {
        FailedUpdateElement element = (FailedUpdateElement) failedMap.get(key);
    Throwable throwable = element.getThrowable();
        Object failedKey = element.getKey();
        Object failedValue = element.getAfterImage();
        failedMap.remove(key);
        // Realizar alguna acción con la clave, el valor o la excepción.
```

```
    }
    Session.commit();
}
```

Nota: La correlación de actualización con anomalía proporciona una forma de supervisar el estado de la aplicación. Si un sistema genera muchos registros en la correlación de actualizaciones con anomalías, es señal de que debe revisarse la aplicación o la arquitectura para utilizar el soporte de grabación diferida. Puede utilizar el mandato **xscmd -showMapSizes** para ver el tamaño de la entrada de correlación de actualizaciones con anomalía.

Manejo de actualizaciones de grabación diferida con anomalía: escucha de fragmentos

Es importante detectar y anotar cuándo falla una transacción de grabación diferida. Cada aplicación que utilice la grabación diferida debe implementar un vigilante que maneje las actualizaciones de grabación diferida con errores. Esto evitará que el sistema se quede sin memoria ya que los registros en la correlación de actualizaciones con errores no se desalojan porque se espera que la aplicación los maneje.

El código siguiente muestra cómo conectar dicho vigilante, o "dumper", que se debe añadir al XML de descriptor de ObjectGrid como en el fragmento de código.

```
<objectGrid name="Grid">
         <bean id="ObjectGridEventListener" className="utils.WriteBehindDumper"/>
```

Puede ver el bean ObjectGridEventListener que se ha añadido, que es el vigilante de grabación diferida mencionado anteriormente. El vigilante interactúa en las correlaciones de todos los fragmentos primarios de una JVM en busca de los que tengan habilitada la grabación diferida. Si encuentra uno, intenta anotar hasta 100 actualizaciones con errores. Sigue vigilando un fragmento primario hasta que éste se mueve a otra JVM . Todas las aplicaciones que usan grabación diferida deben usar un vigilante similar a éste. De lo contrario, las Mäquinas virtuales Java se quedan sin memoria porque esta correlación de errores nunca se desaloja.

Si desea más información, consulte “Ejemplo: Escribir una clase de volcador de grabación diferida”.

Referencia relacionada:

“Ejemplo: Escribir una clase de volcador de grabación diferida”
Este código fuente de ejemplo muestra cómo escribir un observador (volcador) para manejar actualizaciones de grabación diferida anómalas.

Ejemplo: Escribir una clase de volcador de grabación diferida:

Este código fuente de ejemplo muestra cómo escribir un observador (volcador) para manejar actualizaciones de grabación diferida anómalas.

```
//
//Este programa de ejemplo se proporciona TAL CUAL y se puede utilizar, ejecutar, copiar y
//modificar sin que el cliente tenga que pagar derechos (a) para su propia formación,
//(b) para desarrollar aplicaciones diseñadas para ejecutarse con un producto IBM
//WebSphere, ya sea para uso interno propio del cliente o para su redistribución
//por parte del cliente, como parte de una aplicación de este tipo, en los productos propios del cliente. "
//
//5724-J34 (C) COPYRIGHT International Business Machines Corp. 2009
//Reservados todos los derechos * Material bajo licencia - Propiedad de IBM
//
package utils;

import java.util.Collection;
import java.util.Iterator;
```

```
import java.util.concurrent.Callable;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.ScheduledThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
import java.util.logging.Logger;

import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.UndefinedMapException;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventGroup;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener;
import com.ibm.websphere.objectgrid.writebehind.FailedUpdateElement;
import com.ibm.websphere.objectgrid.writebehind.WriteBehindLoaderConstants;

/**
 * La grabación diferida espera que las transacciones para Loader sean satisfactorias. Si una
  * transacción para una clave falla, insertará una entrada en una correlación denominada
 * PREFIJO + nombreCorrelación. La aplicación debe comprobar si en esta correlación hay
 * entradas para volcar anomalías de transacciones de grabación anticipada. La aplicación es
 * responsable de analizar y luego eliminar estas entradas. Estas entradas pueden ser de gran
 * tamaño porque incluyen la clave, las imágenes del valor de antes y después, y la propia
 * excepción. Las excepciones pueden ocupar fácilmente 20k.
 *
  * La clase se registra con la cuadrícula y se crea una instancia por fragmento
 * primario en una JVM.
 * Crea una única hebra una única hebra y dicha hebra comprobará cada correlación
 * cada correlación de errores de grabación diferida para el fragmento, imprimirá
 * el problema y eliminará la entrada.
 *
 * Esto significa que habrá una hebra por fragmento. Si el fragmento se traslada a otra JVM, el
 * método deactivate detiene la hebra.
  * @author bnewport
 *
 */
public class WriteBehindDumper implements ObjectGridEventListener, ObjectGridEventGroup.ShardEvents,
 Callable<Boolean>
{
 static Logger logger = Logger.getLogger(WriteBehindDumper.class.getName());

 ObjectGrid grid;

 /**
  * Agrupación de hebras para manejar verificadores de tablas. Si la aplicación tiene una
  * agrupación propia, cámbiela para reutilizar la agrupación existente
  */
 static ScheduledExecutorService pool = new ScheduledThreadPoolExecutor(2); // dos hebras para volcar registros

 // el futuro para este fragmento
 ScheduledFuture<Boolean> future;

 // true si este fragmento está activo
 volatile boolean isShardActive;

 /**
  * Tiempo normal entre las comprobaciones de correlaciones para ver si hay errores de grabación
  * diferida
  */
 final long BLOCKTIME_SECS = 20L;

 /**
  * Una sesión asignada para este fragmento. No tiene sentido en asignarla una y otra vez
  */
 Session session;
 /**
  * Cuando un fragmento primario se activa, planificar las comprobaciones de forma periódica
  * para comprobar las correlaciones de errores de grabación diferida e imprimir problemas
  */
 public void shardActivated(ObjectGrid grid)
 {
  try
  {
   this.grid = grid;
   session = grid.getSession();

   isShardActive = true;
   future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS); // comprobar cada BLOCKTIME_SECS segundos inicialmente
  }
  catch(ObjectGridException e)
  {
   throw new ObjectGridRuntimeException("Exception activating write dumper", e);
  }
 }

 /**
  * Marcar fragmento como inactivo y luego cancelar el verificador
  */
```

```
public void shardDeactivate(ObjectGrid arg0)
{
 isShardActive = false;
 // si se cancela, la cancelación devuelve true
 if(future.cancel(false) == false)
 {
  // si no, bloquear hasta que se complete el verificador
  while(future.isDone() == false) // esperar a que la tarea finalice de una forma u otra
  {
   try
   {
    Thread.sleep(1000L); // comprobar cada segundo
   }
   catch(InterruptedException e)
   {
   }
  }
 }
}

/**
 * Prueba simple para ver si la correlación está habilitada para la grabación diferida, y si lo
 * está, devolver el nombre de la correlación de errores para la misma.
 * @param mapName La correlación que se va a probar
 * @return El nombre de la correlación de errores de grabación diferida si existe, si no nulo
 */
static public String getWriteBehindNameIfPossible(ObjectGrid grid, String mapName)
{
 BackingMap map = grid.getMap(mapName);
 if(map != null && map.getWriteBehind() != null)
 {
  return WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + mapName;
 }
 else
  return null;
}

/**
 * Se ejecuta para cada fragmento. Comprueba si cada correlación tiene habilitada la grabación
  * diferida y a continuación imprime cualquier error de transacción de grabación
 * y, a continuación, elimina el registro.
 */
public Boolean call()
{
 logger.fine("Called for " + grid.toString());
 try
 {
  // mientras el fragmento primario está presente en esta JVM
  // aquí sólo se devuelven las correlaciones definidas por el usuario, en esta lista no hay
  // ningún correlaciones del sistema como correlaciones de grabación diferida
  Iterator<String> iter = grid.getListOfMapNames().iterator();
  boolean foundErrors = false;
  // iterar en todas las correlaciones actuales
  while(iter.hasNext() && isShardActive)
  {
   String origName = iter.next();

   // si es una correlación de errores de grabación diferida
   String name = getWriteBehindNameIfPossible(grid, origName);
   if(name != null)
   {
    // intentar eliminar bloques de N errores cada vez
    ObjectMap errorMap = null;
    try
    {
     errorMap = session.getMap(name);
    }
    catch(UndefinedMapException e)
    {
     // durante el inicio, las correlaciones de errores pueden todavía no existir, paciencia...
     continue;
    }
    // intentar volcar N registros a la vez
    session.begin();
    for(int counter = 0; counter < 100; ++counter)
    {
     Integer seqKey = (Integer)errorMap.getNextKey(1L);
     if(seqKey != null)
     {
      foundErrors = true;
      FailedUpdateElement elem = (FailedUpdateElement)errorMap.get(seqKey);
      //
      // La aplicación debe anotar el problema aquí
      logger.info("WriteBehindDumper ( " + origName + ") for key (" + elem.getKey() + ") Exception: " +
       elem.getThrowable().toString());
      //
      //
      errorMap.remove(seqKey);
     }
     else
      break;
```

```
     }
     session.commit();
    }
   } // ejecutar correlación siguiente
   // realice un bucle más rápido si hay errores
   if(isShardActive)
   {
    // volver a planificar después de un segundo si había registro anómalos
    // de lo contrario, espere 20 segundos.
    if(foundErrors)
     future = pool.schedule(this, 1L, TimeUnit.SECONDS);
    else
     future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS);
   }
  }
  catch(ObjectGridException e)
  {
   logger.fine("Exception in WriteBehindDumper" + e.toString());
   e.printStackTrace();

   //no dejar una transacción en la sesión.
   if(session.isTransactionActive())
   {
    try { session.rollback(); } catch(Exception e2) {}
   }
  }
  return true;
 }

 public void destroy() {
  // Apéndice de método generado automáticamente TODO

 }

 public void initialize(Session arg0) {
  // Apéndice de método generado automáticamente TODO

 }

 public void transactionBegin(String arg0, boolean arg1) {
  // Apéndice de método generado automáticamente TODO

 }

 public void transactionEnd(String arg0, boolean arg1, boolean arg2,
   Collection arg3) {
  // Apéndice de método generado automáticamente TODO

 }
}
```

Conceptos relacionados:

“Configuración del soporte de cargador de grabación diferida” en la página 359
Puede habilitar el soporte de grabación diferida utilizando el archivo XML de descriptor de ObjectGrid, o a través de programa utilizando la interfaz BackingMap.

“Almacenamiento en memoria caché de grabación diferida” en la página 86
Puede utilizar el almacenamiento en la memoria caché de grabación diferida para reducir la sobrecarga que se produce al actualizar una base de datos utilizada como programa de fondo.

“Manejo de actualizaciones de grabación diferida erróneas” en la página 364
Puesto que la transacción de WebSphere eXtreme Scale termina antes de que se inicie la transacción de programa de fondo, es posible que se produzca un éxito falso de la transacción.

Consideraciones de programación del cargador JPA

Un cargador Java Persistence API (JPA) es una implementación de plug-in de cargador que utiliza JPA para interactuar con la base de datos. Utilice las siguientes consideraciones cuando desarrolle una aplicación que utiliza un cargador JPA.

Entidad eXtreme Scale y entidad JPA

Puede designar cualquier clase POJO como una entidad eXtreme Scale utilizando las anotaciones de entidad eXtreme Scale, la configuración XML, o ambos. También

puede designar la misma clase POJO como entidad JPA mediante el uso de anotaciones de entidad JPA o de la configuración de XML.

Entidad eXtreme Scale: una entidad eXtreme Scale representa los datos persistentes almacenado en correlaciones de ObjectGrid. Un objeto de entidad se transforma en un tuple de clave y un tuple de valor, que después de almacenan como pares de clave-valor en las correlaciones. Un tuple es una matriz de atributos primitivos.

Entidad JPA: una entidad JPA representa los datos persistentes almacenados en una base de datos relacional que utiliza automáticamente la persistencia gestionada por contenedor. Los datos persisten en alguna forma de sistema de almacenamiento de datos con el formato adecuado como ,por ejemplo, los tuples de base de datos.

Cuando persiste una entidad eXtreme Scale, sus relaciones se almacenan en otras correlaciones de entidad. Por ejemplo, cuando se persiste una entidad Consumer con una relación de uno a muchos con una entidad ShippingAddress, si el valor cascade-persist está habilitado, la entidad ShippingAddress se almacena en la correlación shippingAddress en formato de tuple. Si persiste una entidad JPA, las entidades JPA relacionadas también se persisten en las tablas de base de datos, si el valor cascade-persist está habilitado. Cuando se designa una clase POJO como una entidad eXtreme Scale y, también, una entidad JPA, los datos se pueden persistir tanto en correlaciones, como en bases de datos de la entidad ObjectGrid. Los usos comunes son los siguientes:

- **Escenario de precarga**: se carga una entidad desde una base de datos utilizando un proveedor JPA y se conserva en las correlaciones de entidad ObjectGrid.
- **Escenario de cargador**: se conecta una implementación de cargador para las correlaciones de la entidad ObjectGrid, de forma que una entidad almacenada en las correlaciones de la entidad ObjectGrid se puede conservar o cargar desde una base de datos utilizando los proveedores JPA.

También es habitual que una clase POJO se designe únicamente como entidad JPA. En ese caso, lo que se almacena en las correlaciones ObjectGrid son las instancias POJO, frente a los tuples de entidad si se tratara de entidades ObjectGrid.

Consideraciones sobre el diseño de aplicaciones en correlaciones de entidad

Cuando conecta una interfaz JPALoader, las instancias de objeto se almacenan directamente en las correlaciones de ObjectGrid.

Sin embargo, cuando se conecta un JPAEntityLoader, la clase de entidad se designa como entidad eXtreme Scale y, también como una entidad JPA. En este caso, trate a esta entidad como si tuviera dos almacenes persistentes: las correlaciones de entidad ObjectGrid y el almacén de persistencia JPA. La arquitectura es más compleja que el caso de JPALoader.

Si desea más información sobre el plug-in JPAEntityLoader y las consideraciones de diseño de la aplicación, consulte la información sobre el plug-in JPAEntityLoader de la *Guía de administración*. Esta información también puede ayudarle si tiene previsto implementar su propio cargador para las correlaciones de entidad.

Consideraciones sobre el rendimiento

Asegúrese de que establece el tipo Fetch en EAGER o LAZY para las relaciones. Por ejemplo, una relación Consumer bidireccional de uno a muchos con ShippingAddress, con OpenJPA para ayudar a explicar las diferencias en el rendimiento. En este ejemplo, una consulta JPA intenta `select o from Consumer o where . . .` para realizar una carga masiva, y también carga todos los objetos ShippingAddress relacionados. Una relación de uno a muchos se define en la clase Consumer del modo siguiente:

```
@Entity
public class Consumer implements Serializable {

    @OneToMany(mappedBy="consumer",cascade=CascadeType.ALL, fetch =FetchType.EAGER)
    ArrayList <ShippingAddress> addresses;
```

A continuación, se muestra el consumidor de una relación de muchos a uno definida en la clase ShippingAddress:

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.EAGER)
    Consumer consumer;
}
```

Si los tipos Fetch de ambas relaciones se configuran como `eager`, OpenJPA utiliza las consultas N+1+1 para obtener todos los objetos Consumer y objetos ShippingAddress, donde N es el número de objetos ShippingAddress. Sin embargo, si se modifica el ShippingAddress para utilizar el tipo Fetch LAZY del modo siguiente, sólo toma dos consultas para obtener todos los datos.

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.LAZY)
    Consumer consumer;
}
```

Aunque la consulta devuelve los mismos resultados, tener un número inferior de consultas reduce de forma significativa la interacción con la base de datos, que puede aumentar el rendimiento de la aplicación.

Conceptos relacionados:

"Plug-ins para la comunicación con bases de datos" en la página 342
Con un plug-in Loader, una correlación de ObjectGrid se puede comportar como una memoria caché de memoria para datos que normalmente se mantienen en un almacén persistente en el mismo sistema o en algún otro sistema. Generalmente, se utiliza una base de datos o un sistema de archivos como almacenamiento persistente. También se puede utilizar una máquina virtual Java (JVM) remota como el origen de datos, lo que permite que las memorias caché basadas en hub se creen utilizando el ObjectGrid. Un cargador tiene la lógica para leer y escribir datos en un almacén persistente.

"Escribir un cargador" en la página 350
Puede escribir su propia implementación de plug-in de cargador en sus aplicaciones, que debe seguir los convenios de plug-in de WebSphere eXtreme Scale.

"Plug-in JPAEntityLoader"
El plug-in JPAEntityLoader es una implementación de cargador incorporada que utiliza Java Persistence API (JPA) para comunicarse con la base de datos cuando se utiliza la API EntityManager. Al utilizar la API ObjectMap, utilice el cargador JPALoader.

"Uso de un cargador con correlaciones de entidad y tuples" en la página 375
El gestor de entidades convierte todos los objetos de entidad en objetos de tuple antes de que se almacenen en una correlación de WebSphere eXtreme Scale. Cada entidad tiene un tuple de clave y tuple de valor. Este par de clave-valor se almacena en la correlación asociada de eXtreme Scale para la entidad. Al utilizar una correlación eXtreme Scale con un cargador, éste debe interactuar con los objetos de tuple.

"Escribir un cargador con un controlador de precarga de réplica" en la página 380
Un cargador con un controlador de precarga de réplica es un cargador que implementa la interfaz ReplicaPreloadController además de la interfaz del cargador.

"Cargadores" en la página 90
Con un plug-in Loader plug-in, una correlación de cuadrícula de datos puede actuar como una memoria caché de datos para los datos que se mantienen normalmente en un almacén persistente en el mismo sistema o en otro sistema. Generalmente, se utiliza una base de datos o un sistema de archivos como almacenamiento persistente. Una máquina virtual Java (JVM) remota también se puede utilizar como el origen de datos, lo que permite crear memorias caché basadas en hub utilizando eXtreme Scale. Un cargador tiene la lógica para leer y escribir datos en un almacén persistente.

Plug-in JPAEntityLoader:

El plug-in JPAEntityLoader es una implementación de cargador incorporada que utiliza Java Persistence API (JPA) para comunicarse con la base de datos cuando se utiliza la API EntityManager. Al utilizar la API ObjectMap, utilice el cargador JPALoader.

Detalles del cargador

Utilice el plug-in JPALoader cuando almacene los datos utilizando la API ObjectMap. Utilice el plug-in JPAEntityLoader cuando almacene los datos mediante la API EntityManager.

Los cargadores proporcionan dos funciones principales:

1. **get**: en el método get, el plug-in JPAEntityLoader llama en primer lugar al método javax.persistence.EntityManager.find(Class entityClass, Object key) para encontrar la entidad JPA. El plug-in proyecta esta entidad JPA en los tuples de entidad. Durante la proyección, los atributos del tuple y las claves de la asociación se almacenan en el tuple de valor. Después de procesar cada clave, el método get devuelve una lista de tuples de valor de entidad.
2. **batchUpdate**: el método batchUpdate toma un objeto LogSequence que contiene una lista de objetos LogElement. Cada objeto LogElement contiene un tuple de clave y un tuple de valor. Para interactuar con el proveedor de JPA, en primer lugar, debe encontrar la entidad eXtreme Scale basada en el tuple de clave. Basándose en el tipo LogElement, ejecute las siguientes llamadas de JPA:
   - **insert**: javax.persistence.EntityManager.persist(Object o)
   - **update**: javax.persistence.EntityManager.merge(Object o)
   - **remove**: javax.persistence.EntityManager.remove(Object o)

Un LogElement con el tipo **update** realiza la llamada de JPAEntityLoader al método javax.persistence.EntityManager.merge(Object o) para fusionar la entidad. Sin embargo, un tipo **update** de LogElement podría ser el resultado de una llamada a com.ibm.websphere.objectgrid.em.EntityManager.merge(object o) o un cambio de atributo de la instancia gestionada por el EntityManager de eXtreme Scale. Consulte el siguiente ejemplo:

```
com.ibm.websphere.objectgrid.em.EntityManager em = og.getSession().getEntityManager();
em.getTransaction().begin();
Consumer c1 = (Consumer) em.find(Consumer.class, c.getConsumerId());
c1.setName("New Name");
em.getTransaction().commit();
```

En este ejemplo, un tipo update de LogElement se envía al JPAEntityLoader del consumidor de la correlación. Se llama al método javax.persistence.EntityManager.merge(Object o) en el gestor de entidades JPA, en lugar de una actualización de atributo a la entidad gestionada por JPA. Debido a este cambio de comportamiento, existen algunas limitaciones con el uso de este modelo de programación.

Reglas sobre el diseño de aplicaciones

Las entidades tienen relaciones con otras entidades. Para diseñar una aplicación con relaciones y con un JPAEntityLoader conectado debe tenerse en cuenta una serie de consideraciones adicionales. La aplicación debe seguir cuatro reglas, que se describen en los apartados siguientes.

Soporte de profundidad de relaciones limitada

JPAEntityLoader sólo se admite al utilizar entidades sin ninguna relación o entidades con relaciones de un único nivel. No están soportadas las relaciones con más de un nivel como, por ejemplo, `Compañía > Departamento > Empleado`.

Un cargador por correlación

Si utiliza las relaciones de entidad Consumer-ShippingAddress como ejemplo, al cargar Consumer con el tipo FETCH establecido en EAGER, puede cargar todos los objetos ShippingAddress relacionados. Al persistir o fusionar un objeto Consumer, puede persistir o fusionar los objetos ShippingAddress relacionados si se ha habilitado el valor cascade-persist o cascade-merge.

No puede conectar un cargador de la correlación de entidad raíz que almacene los tuples de entidad Consumer. Debe configurar un cargador para cada correlación de entidad.

Mismo tipo de valor cascade para JPA y eXtreme Scale

Vuelva a considerar el escenario en el que la entidad Consumer tiene una relación de uno a muchos con ShippingAddress. Puede consultar el escenario donde se ha habilitado el valor cascade-persist para esta relación. Cuando se persiste un objeto Consumer en eXtreme Scale, el número N asociado de objetos ShippingAddress también se persistirá en eXtreme Scale.

Una llamada de persistencia del objeto Consumer con una relación cascade-persist con ShippingAddress se convierte en una llamada al método javax.persistence.EntityManager.persist(consumer) y el cargador JPAEntityLoader llama *N* veces al método javax.persistence.EntityManager.persist(shippingAddress). Sin embargo, estas N llamadas de persistencia adicionales a los objetos ShippingAddress no son necesarias debido al valor cascade-persist desde el punto de vista del proveedor JPA. Para resolver este problema, eXtreme Scale proporciona un nuevo método isCascaded en la interfaz LogElement. El método isCascaded indica si el LogElement es un resultado de una operación cascade de eXtreme Scale EntityManager. En este ejemplo, el JPAEntityLoader de la correlación de ShippingAddress recibe *N* objetos LogElement debido a las llamadas persistencias en cascada. JPAEntityLoader descubre que el método isCascaded devuelve un valor true y, a continuación, los ignora sin realizar ninguna llamada de JPA. Por lo tanto, desde un punto de vista de JPA, sólo se recibe una llamada del método javax.persistence.EntityManager.persist(consumer).

Este comportamiento se repite si fusiona o elimina una entidad con el valor en cascada habilitado. El plug-in JPAEntityLoader ignora todas las operaciones en cascada.

El diseño del soporte de cascade es reproducir las operaciones de EntityManager de eXtreme Scale para los proveedores JPA. Estas operaciones son persistir, fusionar y eliminar. Para habilitar el soporte de operaciones cascade, verifique que el valor de cascade para el JPA y el EntityManager de eXtreme Scale son iguales.

Use la actualización de entidad con precaución

Como se ha descrito previamente, el diseño del soporte de cascade es reproducir las operaciones EntityManager de eXtreme Scale para los proveedores JPA. Si la aplicación llama al método ogEM.persist(consumer) en el EntityManager de eXtreme Scale, aunque los objetos ShippingAddress asociados se persistan debido al valor de cascade-persist, y JPAEntityLoader sólo llama al método jpAEM.persist(consumer) en los proveedores JPA.

Sin embargo, si la aplicación actualiza una entidad gestionada, el plug-in JPAEntityLoader convertirá esta actualización en una llamada de fusión JPA. En este escenario, no está garantizado el soporte de varios niveles de relaciones y asociaciones de claves. En este caso, el procedimiento recomendado es utilizar el método javax.persistence.EntityManager.merge(o), en lugar de actualizar una entidad gestionada.

Referencia relacionada:

"Consideraciones de programación del cargador JPA" en la página 369
Un cargador Java Persistence API (JPA) es una implementación de plug-in de cargador que utiliza JPA para interactuar con la base de datos. Utilice las siguientes consideraciones cuando desarrolle una aplicación que utiliza un cargador JPA.

Uso de un cargador con correlaciones de entidad y tuples

El gestor de entidades convierte todos los objetos de entidad en objetos de tuple antes de que se almacenen en una correlación de WebSphere eXtreme Scale. Cada entidad tiene un tuple de clave y tuple de valor. Este par de clave-valor se almacena en la correlación asociada de eXtreme Scale para la entidad. Al utilizar una correlación eXtreme Scale con un cargador, éste debe interactuar con los objetos de tuple.

eXtreme Scale contiene plug-ins de cargador que simplifican la integración con las bases de datos relacionales. Los cargadores JPA (Java Persistence) utilizan una Java Persistence API para interactuar con la base de datos y crear los objetos de entidad. Los cargadores JPA son compatibles con las entidades de eXtreme Scale.

Tuples

Un tuple contiene información sobre los atributos y las asociaciones de una entidad. Los valores primitivos se almacenan mediante derivadores primitivos. Otros tipos de objeto admitidos se almacenan con su formato nativo. Las asociaciones a otras entidades se almacenan como una colección de objetos de tuples de clave que representan las claves de las entidades de destino.

Cada atributo o asociación se almacena mediante un índice basado en cero. Puede recuperar el índice de cada atributo utilizando los métodos getAttributePosition o getAssociationPosition. Después de que se recupere la posición, permanecerá sin cambios durante el ciclo de vida de eXtreme Scale. La posición puede cambiar cuando se reinicie eXtreme Scale. Los métodos setAttribute, setAssociation y setAssociations se utilizan para actualizar los elementos en el tuple.

Atención: Al crear o actualizar los objetos de tuple, actualice todos los campos primitivos con un valor que no sea nulo. Los valores primitivos como, por ejemplo, `int` no pueden ser nulos. Si no cambia el valor por un valor predeterminado, se pueden generar problemas de rendimiento bajo, que también afectan a los campos marcados con la anotación @Version o el atributo de versión en el archivo XML de descriptor de entidad.

El siguiente ejemplo explica de forma adicional cómo procesar tuples. Si desea más información sobre cómo definir entidades para este ejemplo, consulte la información sobre el esquema de entidad Order que está en la guía de aprendizaje del gestor de entidades en *Visión general del producto*.WebSphere eXtreme Scale se ha configurado para utilizar cargadores con cada una de las entidades. De forma adicional, sólo se toma la entidad Order y esta entidad específica tiene una relación de muchos a uno con la entidad Customer. El nombre de atributo es `customer`, y tiene una relación de uno a muchos con la entidad OrderLine.

Utilice Projector para crear automáticamente objetos Tuple de las entidades. La utilización de Projector puede simplificar los cargadores cuando se utiliza un programa de utilidad de correlaciones de objetos relacionales como, por ejemplo, Hibernate o JPA.

order.java

```
@Entity
public class Order
{
    @Id  String orderNumber;
    java.util.Date date;
    @OneToOne(cascade=CascadeType.PERSIST) Customer customer;
    @OneToMany(cascade=CascadeType.ALL, mappedBy="order") @OrderBy("lineNumber") List<OrderLine> lines;
}
```

customer.java

```
@Entity
public class Customer {
    @Id String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

orderLine.java

```
@Entity
public class OrderLine
{
    @Id @ManyToOne(cascade=CascadeType.PERSIST) Order order;
    @Id int lineNumber;
    @OneToOne(cascade=CascadeType.PERSIST) Item item;
    int quantity;
    double price;
}
```

Una clase OrderLoader que implementa la interfaz Loader se muestra en el siguiente código. El siguiente ejemplo presupone que se ha definido un plug-in TransactionCallback asociado.

orderLoader.java

```
public class OrderLoader implements com.ibm.websphere.objectgrid.plugins.Loader {

 private EntityMetadata entityMetaData;
 public void batchUpdate(TxID txid, LogSequence sequence)
            throws LoaderException, OptimisticCollisionException {
        ...
        }
        public List get(TxID txid, List keyList, boolean forUpdate)
            throws LoaderException {
        ...
        }
        public void preloadMap(Session session, BackingMap backingMap)
            throws LoaderException {
  this.entityMetaData=backingMap.getEntityMetadata();
 }

}
```

La variable de la instancia entityMetaData se ha inicializado durante la llamada al método preLoadMap desde eXtreme Scale. La variable *entityMetaData* no es nula si la correlación se ha configurado para utilizar entidades. De lo contrario, el valor es nulo.

Método batchUpdate

El método batchUpdate proporciona la capacidad de saber qué acción tiene previsto realizar la aplicación. Basándose en una operación insertar, actualizar o suprimir, se puede abrir una conexión con la base de datos y el trabajo realizado. Puesto que la clave y los valores son del tipo Tuple, se deben transformar de forma que los valores tengan sentido en la sentencia SQL.

La tabla ORDER se creó con la siguiente definición DLL (lenguaje de definición de datos), tal como se muestra en el código siguiente:

```
CREATE TABLE ORDER (ORDERNUMBER VARCHAR(250) NOT NULL, DATE TIMESTAMP, CUSTOMER_ID VARCHAR(250))
ALTER TABLE ORDER ADD CONSTRAINT PK_ORDER PRIMARY KEY (ORDERNUMBER)
```

El código siguiente muestra cómo convertir un tuple en un objeto:

```
public void batchUpdate(TxID txid, LogSequence sequence)
            throws LoaderException, OptimisticCollisionException {
        Iterator iter = sequence.getPendingChanges();
        while (iter.hasNext()) {
            LogElement logElement = (LogElement) iter.next();
            Object key = logElement.getKey();
            Object value = logElement.getCurrentValue();

            switch (logElement.getType().getCode()) {
            case LogElement.CODE_INSERT:

1)              if (entityMetaData!=null) {

// El pedido sólo tiene una clave orderNumber
2)                  String ORDERNUMBER=(String) getKeyAttribute("orderNumber", (Tuple) key);
// Obtener el valor de fecha
3)                  java.util.Date unFormattedDate = (java.util.Date) getValueAttribute("date",(Tuple)value);
// Los valores son 2 asociaciones. Permite el proceso de clientes porque
// la tabla contiene customer.id como clave primaria
4)                  Object[] keys= getForeignKeyForValueAssociation("customer","id",(Tuple) value);
                    //Order para Customer es M para 1. Sólo puede haber 1 clave
5)                  String CUSTOMER_ID=(String)keys[0];
// analizar variable unFormattedDate y darle formato para la base de datos como formattedDate
6)                   String formattedDate = "2007-05-08-14.01.59.780272"; // formateado para DB2
// Por último, la sentencia SQL para insertar el registro
7) //INSERT INTO ORDER (ORDERNUMBER, DATE, CUSTOMER_ID) VALUES(ORDERNUMBER,formattedDate, CUSTOMER_ID)
                }
                break;
            case LogElement.CODE_UPDATE:
                break;
            case LogElement.CODE_DELETE:
                break;
            }
        }

    }
// devuelve el valor al atributo según está almacenado en el tuple de clave
 private Object getKeyAttribute(String attr, Tuple key) {
        //obtener metadatos de clave
        TupleMetadata keyMD = entityMetaData.getKeyMetadata();
        //obtener posición del atributo
        int keyAt = keyMD.getAttributePosition(attr);
        if (keyAt > -1) {
            return key.getAttribute(keyAt);
        } else { // attribute undefined
            throw new IllegalArgumentException("Invalid position index for  "+attr);
        }

    }
// devuelve el valor al atributo según está almacenado en el tuple de valor
    private Object getValueAttribute(String attr, Tuple value) {
        //similar a la operación anterior, excepto que se trabaja con metadatos de valor
        TupleMetadata valueMD = entityMetaData.getValueMetadata();

        int keyAt = valueMD.getAttributePosition(attr);
        if (keyAt > -1) {
            return value.getAttribute(keyAt);
        } else {
            throw new IllegalArgumentException("Invalid position index for  "+attr);
        }
    }
// devuelve una matriz de claves que se refiere a la asociación.
    private Object[] getForeignKeyForValueAssociation(String attr, String fk_attr, Tuple value) {
        TupleMetadata valueMD = entityMetaData.getValueMetadata();
        Object[] ro;

        int customerAssociation = valueMD.getAssociationPosition(attr);
        TupleAssociation tupleAssociation = valueMD.getAssociation(customerAssociation);

        EntityMetadata targetEntityMetaData = tupleAssociation.getTargetEntityMetadata();

        Tuple[] customerKeyTuple = ((Tuple) value).getAssociations(customerAssociation);

        int numberOfKeys = customerKeyTuple.length;
        ro = new Object[numberOfKeys];

        TupleMetadata keyMD = targetEntityMetaData.getKeyMetadata();
        int keyAt = keyMD.getAttributePosition(fk_attr);
        if (keyAt < 0) {
            throw new IllegalArgumentException("Invalid position index for  " + attr);
        }
        for (int i = 0; i < numberOfKeys; ++i) {
```

```
            ro[i] = customerKeyTuple[i].getAttribute(keyAt);
        }

        return ro;

    }
```

1. Asegúrese de que entityMetaData no es nulo, lo cual implica que las entradas de memoria caché de clave y valor son del tipo Tuple. En entityMetaData, se recupera la clave TupleMetaData, que refleja sólo la parte de clave de los metadatos Order.
2. Se procesa KeyTuple y se obtiene el valor del atributo de clave orderNumber
3. Se procesa ValueTuple y se obtiene el valor de la fecha de atributo
4. Se procesa ValueTuple y se obtiene el valor de las claves del cliente de asociación
5. Se extrae CUSTOMER_ID. Según la relación, un objeto Order sólo puede tener un cliente. Tendremos sólo una clave. Por lo tanto, el tamaño de las claves es 1. Se ha pasado por alto el análisis de la fecha para corregir el formato, para que sea más sencillo.
6. Dado que se trata de una operación insert, la sentencia SQL se pasa en la conexión de origen de datos para completar la operación insert.

La demarcación y el acceso de la transacción a la base de datos se cubre en “Escribir un cargador” en la página 350.

Método get

Si no se encuentra la clave en la memoria caché, llame al método get en el plug-in Loader para encontrar la clave.

La clave es un Tuple. El primer paso es convertir el Tuple en valores primitivos que se pueden pasar en la sentencia SELECT de SQL. Después de que se recuperen todos los atributos de la base de datos, debe convertirlos en Tuples. El siguiente código demuestra la clase Order.

```
public List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException {
  System.out.println("OrderLoader: Get called");
  ArrayList returnList = new ArrayList();

1)  if (entityMetaData != null) {
   int index=0;
   for (Iterator iter = keyList.iterator(); iter.hasNext();) {
2)    Tuple orderKeyTuple=(Tuple) iter.next();

// El pedido sólo tiene una clave orderNumber
3)    String ORDERNUMBERKEY = (String) getKeyAttribute("orderNumber",orderKeyTuple);
    //Ejecute una consulta para obtener valores de
4)    // SELECT CUSTOMER_ID, date FROM ORDER WHERE ORDERNUMBER='ORDERNUMBERKEY'

5)    //1) Clave foránea: CUSTOMER_ID
6)    //2) fecha
    // Se presupone que éstos se devuelven como
7)                              String  CUSTOMER_ID = "C001"; // Se presupone recuperación e inicialización
8)    java.util.Date retrievedDate = new java.util.Date();
                                // Se presupone que esta fecha refleja la de la base de datos

    // A continuación, se deben convertir estos datos en un tuple antes de devolver

    //crear un tuple de valor
9)    TupleMetadata valueMD = entityMetaData.getValueMetadata();
    Tuple valueTuple=valueMD.createTuple();


    //añadir objeto retrievedDate a Tuple
    int datePosition = valueMD.getAttributePosition("date");
10)    valueTuple.setAttribute(datePosition, retrievedDate);

    //A continuación se debe añadir la asociación
11)    int customerPosition=valueMD.getAssociationPosition("customer");
    TupleAssociation customerTupleAssociation =
                                  valueMD.getAssociation(customerPosition);
    EntityMetadata customerEMD = customerTupleAssociation.getTargetEntityMetadata();
    TupleMetadata customerTupleMDForKEY=customerEMD.getKeyMetadata();
```

```
12)    int customerKeyAt=customerTupleMDForKEY.getAttributePosition("id");


    Tuple customerKeyTuple=customerTupleMDForKEY.createTuple();
    customerKeyTuple.setAttribute(customerKeyAt, CUSTOMER_ID);
13)    valueTuple.addAssociationKeys(customerPosition, new Tuple[] {customerKeyTuple});


14)    int linesPosition = valueMD.getAssociationPosition("lines");
    TupleAssociation linesTupleAssociation = valueMD.getAssociation(linesPosition);
    EntityMetadata orderLineEMD = linesTupleAssociation.getTargetEntityMetadata();
    TupleMetadata orderLineTupleMDForKEY = orderLineEMD.getKeyMetadata();
    int lineNumberAt = orderLineTupleMDForKEY.getAttributePosition("lineNumber");
    int orderAt = orderLineTupleMDForKEY.getAssociationPosition("order");

    if (lineNumberAt < 0 || orderAt < 0) {
     throw new IllegalArgumentException(
                                       "Invalid position index for lineNumber or order "+
                                       lineNumberAt + " " + orderAt);
    }
15) // SELECT LINENUMBER FROM ORDERLINE WHERE ORDERNUMBER='ORDERNUMBERKEY'
    // Se presupone que dos filas de número de línea se devuelven con los valores 1 y 2

    Tuple orderLineKeyTuple1 = orderLineTupleMDForKEY.createTuple();
    orderLineKeyTuple1.setAttribute(lineNumberAt, new Integer(1));// set Key
    orderLineKeyTuple1.addAssociationKey(orderAt, orderKeyTuple);

    Tuple orderLineKeyTuple2 = orderLineTupleMDForKEY.createTuple();
    orderLineKeyTuple2.setAttribute(lineNumberAt, new Integer(2));// Init Key
    orderLineKeyTuple2.addAssociationKey(orderAt, orderKeyTuple);

16)    valueTuple.addAssociationKeys(linesPosition, new Tuple[]
                                    {orderLineKeyTuple1, orderLineKeyTuple2 });

    returnList.add(index, valueTuple);

    index++;

   }
  }else {
   // no admite tuples
  }
  return returnList;
 }
```

1. Se llama al método get cuando la memoria caché de ObjectGrid no ha podido encontrar la clave y solicita al cargador que la capte. Pruebe el valor de entityMetaData y continué si el valor no es nulo.
2. keyList contiene tuples.
3. Recupere el valor del atributo orderNumber.
4. Ejecute la consulta para recuperar la fecha (valor) y el ID de cliente (clave foránea).
5. CUSTOMER_ID es una clave foránea que se debe establecer en el tuple de asociación.
6. La fecha es un valor y ya debería estar definido.
7. Puesto que este ejemplo no realiza llamadas JDBC, se da por supuesto el CUSTOMER_ID.
8. Dado que este ejemplo no realiza llamadas JDBC, la fecha se da por supuesta.
9. Cree el valor de Tuple.
10. Establezca el valor de la fecha en el Tuple, basándose en su posición.
11. Order tiene dos asociaciones. Empiece con el atributo customer que se refiere a la entidad customer. Debe tener el valor del ID para establecerlo en el tuple.
12. Encuentre la posición del ID en la entidad del cliente.
13. Establezca sólo los valores de las claves de asociación.
14. Además, las líneas son una asociación que se debe configurar como un grupo de claves de asociación, de la misma forma que lo haría para la asociación de cliente.
15. Puesto que debe configurar las claves para el lineNumber asociado con este pedido, ejecute SQL para recuperar los valores de lineNumber.

16. Configure las claves de asociación en el valueTuple. Esto completa la creación del Tuple que se ha devuelto a BackingMap.

En este tema se ofrecen los pasos para crear tuples, y una descripción de la entidad Order solamente. Siga pasos similares para otras entidades, y todo el proceso unido al plug-in TransactionCallback. Consulte “Plug-ins para gestionar los sucesos del ciclo de vida de transacciones” en la página 385 si desea más detalles.

Referencia relacionada:

“Consideraciones de programación del cargador JPA” en la página 369
Un cargador Java Persistence API (JPA) es una implementación de plug-in de cargador que utiliza JPA para interactuar con la base de datos. Utilice las siguientes consideraciones cuando desarrolle una aplicación que utiliza un cargador JPA.

Escribir un cargador con un controlador de precarga de réplica

Un cargador con un controlador de precarga de réplica es un cargador que implementa la interfaz ReplicaPreloadController además de la interfaz del cargador.

La interfaz ReplicaPreloadController se ha diseñado para proporcionar un modo de que la réplica que se convierte en fragmento primario sepa si el fragmento primario anterior ha completado el proceso de precarga. Si la precarga se ha completado parcialmente, se ofrece información sobre el punto en el que se quedó la correlación primaria anterior. Con la implementación de la interfaz ReplicaPreloadController, una réplica que pasa a ser la primaria continúa el proceso de precarga en el punto donde se detuvo el primario anterior y continúa hasta finalizar la operación de precarga general.

En un entorno distribuido de WebSphere eXtreme Scale, una correlación puede tener réplicas y podría precargar un gran volumen de datos durante la inicialización. La precarga es una actividad del cargador y sólo tiene lugar en la correlación primaria durante la inicialización. La operación de precarga tardará en completarse si el volumen de datos que se va a precargar es muy grande. Si la correlación primaria ha precargado una parte considerable de datos, pero se ha detenido durante la inicialización sin motivo aparente, una réplica pasa a ser la réplica primaria. En esta situación, los datos que ha precargado la correlación primaria anterior se pierden porque la nueva réplica primaria normalmente realiza una precarga incondicional. Esto quiere decir que la nueva réplica primaria inicia el proceso de precarga desde el principio y pasa por alto los datos precargados previamente. Si desea que el nuevo primario empiece por el punto en que se detuvo el primario anterior durante el proceso de precarga, proporcione un cargador que implemente la interfaz ReplicaPreloadController. Si desea más información, consulte la documentación de la API.

Para obtener información sobre los cargadores, consulte “Cargadores” en la página 90la información sobre los cargadores en la *Visión general del producto*. Si está interesado en escribir un plug-in Loader regular, consulte “Escribir un cargador” en la página 350.

La interfaz ReplicaPreloadController tiene la siguiente definición:

```
public interface ReplicaPreloadController
{
    public static final class Status
    {
        static public final Status PRELOADED_ALREADY = new Status(K_PRELOADED_ALREADY);
        static public final Status FULL_PRELOAD_NEEDED = new Status(K_FULL_PRELOAD_NEEDED);
        static public final Status PARTIAL_PRELOAD_NEEDED = new Status(K_PARTIAL_PRELOAD_NEEDED);
```

```
    }

    Status checkPreloadStatus(Session session, BackingMap bmap);
}
```

Las siguientes secciones describen algunos de los métodos de la interfaz Loader y ReplicaPreloadController.

Método checkPreloadStatus

Cuando un cargador implementa la interfaz ReplicaPreloadController, se llama al método checkPreloadStatus antes que el método preloadMap durante la inicialización de la correlación. El estado devuelto de este método determina si se llama al método preloadMap. Si este método devuelve `Status#PRELOADED_ALREADY`, se llama al método de precarga. De lo contrario, se ejecuta el método preload. Debido a este comportamiento, este método debe servir como método de inicialización del cargador. Debe inicializar las propiedades del cargador en este método. Este método debe devolver el estado correcto, o la operación de precarga podría no funcionar como se espera.

```
public Status checkPreloadStatus(Session session,
       BackingMap backingMap) {
        // Cuando un cargador implementa la interfaz ReplicaPreloadController,
    // se llamará a este método antes que al método preloadMap durante la
    // inicialización de la correlación. En función del estado devuelto de
    // este método, se llamará o no al método preloadMap. Por ello, este
    // método sirve también como el método de inicialización del cargador.
Este método
// debe devolver el estado correcto, si no, puede que la precarga no
    // funcione como se espera.

        // Nota: debe inicializar esta instancia de cargador a continuación.
        ivOptimisticCallback = backingMap.getOptimisticCallback();
        ivBackingMapName = backingMap.getName();
        ivPartitionId = backingMap.getPartitionId();
        ivPartitionManager = backingMap.getPartitionManager();
        ivTransformer = backingMap.getObjectTransformer();
        preloadStatusKey = ivBackingMapName + "_" + ivPartitionId;

        try {
            // obtener preloadStatusMap para recuperar el estado de precarga
      // que otras JVM podrían haber establecido.
            ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);

            // recuperar el índice de fragmento de datos registrados por última vez.
            Integer lastPreloadedDataChunk = (Integer) preloadStatusMap.get(preloadStatusKey);

            if (lastPreloadedDataChunk == null) {
                preloadStatus = Status.FULL_PRELOAD_NEEDED;
            } else {
                preloadedLastDataChunkIndex = lastPreloadedDataChunk.intValue();
                if (preloadedLastDataChunkIndex == preloadCompleteMark) {
                    preloadStatus = Status.PRELOADED_ALREADY;
                } else {
                    preloadStatus = Status.PARTIAL_PRELOAD_NEEDED;
                }
            }

            System.out.println("TupleHeapCacheWithReplicaPreloadControllerLoader.
       checkPreloadStatus()
     -> map = " + ivBackingMapName + ", preloadStatusKey = " + preloadStatusKey
                    + ", retrieved lastPreloadedDataChunk =" + lastPreloadedDataChunk + ",
                    determined preloadStatus = "
                    + getStatusString(preloadStatus));

        } catch (Throwable t) {
            t.printStackTrace();
        }

        return preloadStatus;
    }
```

Método preloadMap

La ejecución del método preloadMap depende del resultado devuelto del método checkPreloadStatus. Si se llama al método preloadMap, normalmente debe recuperar la información del estado de precarga en la correlación de estado de precarga designada y determinar cómo continuar. Lo ideal es que el método preloadMap sepa si la precarga se ha completado parcialmente y en qué punto debe comenzar exactamente. Durante la precarga de datos, el método preloadMap debe actualizar el estado de precarga en la correlación designada del estado de precarga. El estado de precarga que se almacena en la correlación de estado de precarga es recuperado por el método checkPreloadStatus cuando necesita comprobar el estado de la precarga.

```
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException {
        EntityMetadata emd = backingMap.getEntityMetadata();
        if (emd != null && tupleHeapPreloadData != null) {
            // El método getPreLoadData es similar a captar datos
      //  de una base de datos. Estos datos se pasarán a la memoria caché
      // como proceso de precarga.
            ivPreloadData = tupleHeapPreloadData.getPreLoadData(emd);

            ivOptimisticCallback = backingMap.getOptimisticCallback();
            ivBackingMapName = backingMap.getName();
            ivPartitionId = backingMap.getPartitionId();
            ivPartitionManager = backingMap.getPartitionManager();
            ivTransformer = backingMap.getObjectTransformer();
            Map preloadMap;

            if (ivPreloadData != null) {
                try {
                    ObjectMap map = session.getMap(ivBackingMapName);

                    // obtener preloadStatusMap para registrar el estado de precarga.
                    ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);

                    // Nota: cuando se invoca este método preloadMap, se ha llamado
          // a checkPreloadStatus, se han establecido preloadStatus
          // y preloadedLastDataChunkIndex. Y
          // preloadStatus debe ser PARTIAL_PRELOAD_NEEDED
          // o FULL_PRELOAD_NEEDED que necesitarán una precarga de nuevo.

                    // Si el volumen de datos que debe precargarse es muy grande, los datos
                   // se suelen dividir en fragmentos y el proceso de precarga
          // procesará cada fragmento dentro de su propia transacción.. En este ejemplo sólo
          // se precargan unas pocas entradas, cada una de las cuales representa un fragmento.
                    // Por tanto, la precarga procesa una entrada en una transacción para simular
          //  la precarga de un fragmento.

                    Set entrySet = ivPreloadData.entrySet();
                    preloadMap = new HashMap();
                    ivMap = preloadMap;

                    // dataChunkIndex representa el fragmento de datos
          // en proceso
                    int dataChunkIndex = -1;
                    boolean shouldRecordPreloadStatus = false;
                    int numberOfDataChunk = entrySet.size();
                    System.out.println("    numberOfDataChunk to be preloaded = "
                       + numberOfDataChunk);

                    Iterator it = entrySet.iterator();
                    int whileCounter = 0;
                    while (it.hasNext()) {
                        whileCounter++;
                        System.out.println("preloadStatusKey = " + preloadStatusKey
              + " ,
             whileCounter = " + whileCounter);

                        dataChunkIndex++;

                        // Si dataChunkIndex <= preloadedLastDataChunkIndex
                        // no es necesario realizar el proceso, porque lo ha precargado
                  // antes otra JVM. Sólo se tiene que procesar dataChunkIndex
                  // > preloadedLastDataChunkIndex
```

```
                    if (dataChunkIndex <= preloadedLastDataChunkIndex) {
                        System.out.println("ignore current dataChunkIndex =
                " + dataChunkIndex + " that has been previously
                preloaded.");
                        continue;
                    }

                    // Nota: este ejemplo simula un fragmento de datos como entrada.
                    // Cada clave representa un fragmento de datos por motivos de simplificación.
                    // Si el servidor o fragmento primario se ha detenido por razones desconocidas,
            //  el estado de precarga que indica el progreso
            // de la precarga debe estar disponible en preloadStatusMap. Una
            // réplica que pasa a ser primaria puede obtener el estado de precarga
           //  y determinar cómo realizar la precarga de nuevo.
                    // Nota: la grabación del estado de precarga debe estar en la misma
            // transacción que incluir los datos en memoria caché; por lo que si se produce una
           // retrotracción o un error de la transacción, el estado de precarga grabado es el
           // estado real.

                    Map.Entry entry = (Entry) it.next();
                    Object key = entry.getKey();
                    Object value = entry.getValue();
                    boolean tranActive = false;

                    System.out.println("processing data chunk. map = " +
              this.ivBackingMapName + ", current dataChunkIndex = " +
              dataChunkIndex + ", key = " + key);

                    try {
                        shouldRecordPreloadStatus = false; // re-set to false
                        session.beginNoWriteThrough();
                        tranActive = true;

                        if (ivPartitionManager.getNumOfPartitions() == 1) {
                            // si sólo hay 1 partición, no es necesario realizar ninguna operación
                // con ella.
                            // pase los datos a la memoria caché
                            map.put(key, value);
                            preloadMap.put(key, value);
                            shouldRecordPreloadStatus = true;
                        } else if (ivPartitionManager.getPartition(key) == ivPartitionId) {
                            // si la correlación está particionada, es necesario considerar que la
                // clave de partición sólo precarga los datos que pertenecen
                // a esta partición.
                            map.put(key, value);
                            preloadMap.put(key, value);
                            shouldRecordPreloadStatus = true;
                        } else {
                            // pasar por alto esta entrada porque no pertenece a
                // esta partición.
                        }

                        if (shouldRecordPreloadStatus) {
                            System.out.println("record preload status. map = " +
                   this.ivBackingMapName + ", preloadStatusKey = " +
                   preloadStatusKey + ", current dataChunkIndex ="
+ dataChunkIndex);
                            if (dataChunkIndex == numberOfDataChunk) {
                                System.out.println("record preload status. map = " +
                     this.ivBackingMapName + ", preloadStatusKey = " +
                     preloadStatusKey + ", mark complete =" +
                     preloadCompleteMark);
                                // significa que estamos en el último fragmento de datos,
                                // si la operación se confirma correctamente,
                                // el registro de la precarga se ha completado.
// En este punto, se considera que la precarga ha terminado.
                                // use -99 como marca especial de estado completo de la precarga.

                                preloadStatusMap.get(preloadStatusKey);

                                // si una operación put sigue a una operación get se convierte
                  //  en update si get devuelve un objeto, de lo contrario será insert.
                                preloadStatusMap.put(preloadStatusKey, new
                    Integer(preloadCompleteMark));

                            } else {
                                // registrar dataChunkIndex precargado actual en
                  // preloadStatusMap una operación put sigue a una operación get se convierte
                  //  en update si get devuelve un objeto, de lo contrario
                  //  será insert.
```

```
                                preloadStatusMap.get(preloadStatusKey);
                                preloadStatusMap.put(preloadStatusKey, new
                    Integer(dataChunkIndex));
                            }
                        }

                        session.commit();
                        tranActive = false;

                        // para simular la precarga de un gran volumen de datos
                        // deje inactiva esta hebra durante 30 segundos.
                        // La aplicación real NO debe dejar inactiva esta hebra.
                        Thread.sleep(10000);

                    } catch (Throwable e) {
                        e.printStackTrace();
                        throw new LoaderException("preload failed with exception: " + e, e);
                    } finally {
                        if (tranActive && session != null) {
                            try {
                                session.rollback();
                            } catch (Throwable e1) {
                                // la precarga pasa por alto la excepción en la retrotracción
                            }
                        }
                    }
                }

                // En este punto, se considera que la precarga ha terminado.
                // use -99 como marca especial de estado completo de la precarga.
                // De esta manera se asegura de haber establecido la marca de finalización.
                // Además, al crear particiones, cada partición desconoce cuándo
        //  es su último fragmento de datos. Por lo tanto, el bloque siguiente sirve como
        //  informe de finalización del estado general de la precarga.
                System.out.println("Overall preload status complete -> record
             preload status. map = " + this.ivBackingMapName + ",
             preloadStatusKey = " + preloadStatusKey + ", mark complete =" +
            preloadCompleteMark);
                session.begin();
                preloadStatusMap.get(preloadStatusKey);
                // si una operación put sigue a get, se convierte en update si
                // la operación get devuelve un objeto,
                // de lo contrario, será insert.
                preloadStatusMap.put(preloadStatusKey, new Integer(preloadCompleteMark));
                session.commit();

                ivMap = preloadMap;
              } catch (Throwable e) {
                e.printStackTrace();
                throw new LoaderException("preload failed with exception: " + e, e);
              }
            }
        }
    }
```

Correlación de estado de precarga

Debe utilizar una correlación de estado de precarga para soporta la implementación de la interfaz ReplicaPreloadController. El método preloadMap siempre debe comprobar, en primer lugar, el estado de precarga almacenado en la correlación de estado de precarga y actualizar el estado de precarga en la correlación de estado de precarga siempre que se pasen datos a la memoria caché. El método checkPreloadStatus puede recuperar el estado de precarga de la correlación de estado de precarga, determinar el estado de precarga y devolver el estado al llamante. La correlación de estado de precarga debe estar en el mismo objeto mapSet que otras correlaciones que tienen cargadores de controlador de precarga de réplica.

Referencia relacionada:

"Consideraciones de programación del cargador JPA" en la página 369
Un cargador Java Persistence API (JPA) es una implementación de plug-in de cargador que utiliza JPA para interactuar con la base de datos. Utilice las siguientes consideraciones cuando desarrolle una aplicación que utiliza un cargador JPA.

Plug-ins para gestionar los sucesos del ciclo de vida de transacciones

Utilice el plug-in TransactionCallback para personalizar las operaciones de creación de versiones y de comparación de los objetos de la memoria caché cuando se utiliza la estrategia de bloqueo optimista.

Puede proporcionar un objeto de devolución de llamada optimista conectable que implementa la interfaz com.ibm.websphere.objectgrid.plugins.OptimisticCallback. Para las correlaciones de entidades, se configura automáticamente un plug-in OptimisticCallback de alto rendimiento.

Finalidad

Utilice la interfaz OptimisticCallback para proporcionar operaciones de comparación optimista para los valores de una correlación. Es necesaria una implementación OptimisticCallback cuando se utiliza la estrategia de bloqueo optimista. WebSphere eXtreme Scale proporciona una implementación predeterminada de OptimisticCallback. Sin embargo, normalmente la aplicación debe conectar su propia implementación de la interfaz OptimisticCallback. Consulte "Estrategias de bloqueo" en la página 238la información sobre estrategias de bloqueo en la *Visión general del producto* para obtener más información.

Implementación predeterminada

La infraestructura de eXtreme Scale proporciona una implementación predeterminada de la interfaz OptimisticCallback que se utiliza si la aplicación no se conecta a un objeto OptimisticCallback proporcionado por la aplicación, tal como se demuestra en la sección anterior. La implementación predeterminada siempre devuelve el valor especial de NULL_OPTIMISTIC_VERSION como el objeto de versión del valor y nunca actualiza el objeto de versión. Esta acción realiza una comparación optimista, una operación sin función. En la mayoría de los casos, no desea que se produzca la función sin operación al utilizar la estrategia de bloqueo optimista. Las aplicaciones deben implementar la interfaz OptimisticCallback y conectar sus propias implementaciones de OptimisticCallback, de forma que no se utilice la implementación predeterminada. Sin embargo, como mínimo, existe un escenario donde resulta práctica la implementación proporcionada predeterminada de OptimisticCallback. Observe la situación siguiente:

- Se conecta un cargador para la correlación de respaldo.
- El cargador sabe cómo realizar la comparación optimista sin ayuda de un plug-in OptimisticCallback.

¿Cómo puede saber el cargador cómo tratar con la creación de versiones optimista sin la ayuda de un objeto OptimisticCallback? El cargador conoce el objeto de clase de valor y sabe qué campo del objeto de valor se utiliza como valor de creación de versiones optimista. Por ejemplo, imagine que se utiliza la interfaz siguiente para el objeto de valor de la correlación de empleados.

```
public interface Employee
{
    // Número de secuencia utilizado para la creación de versiones optimista.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Otros métodos get/set para otros campos del objeto Employee.
}
```

En este caso, el cargador sabe que puede utilizar el método getSequenceNumber para obtener la información de creación de versiones actual para un objeto de valor Employee. El cargador incrementa el valor devuelto para generar un nuevo número de versión antes de actualizar el almacenamiento persistente con el nuevo valor Employee. Para un cargador JDBC (Java Database Connectivity), se utiliza el número de secuencia actual en la cláusula where de una sentencia de actualización sobrecualificada de SQL y utiliza el nuevo número de secuencia generado para establecer la columna del número de secuencia en el nuevo valor de número de secuencia.

Otra posibilidad es que el cargador utilice una función, que proporciona el programa de fondo, que actualiza automáticamente una columna oculta que puede utilizarse para la creación de versiones optimista. En algunos casos, puede usarse un procedimiento almacenado o un desencadenante para ayudar a mantener una columna que contiene la información sobre la creación de versiones. Si el cargador utiliza una de estas técnicas para mantener la información de la creación de versiones optimista, la aplicación no necesita proporcionar una implementación OptimisticCallback. Puede utilizar la implementación predeterminada de OptimisticCallback porque el cargador puede manejar la creación de versiones optimista sin la ayuda de un objeto OptimisticCallback.

Implementación predeterminada de entidades

Las entidades se almacenan en ObjectGrid mediante objetos de tuple. La implementación predeterminada de OptimisticCallback se comporta del mismo modo que el comportamiento para las correlaciones sin entidad. Sin embargo, el campo de versión de la entidad se identifica a través del uso de la anotación @Version o el atributo de versión en el archivo XML de descriptor de la entidad.

El atributo de versión puede ser de uno de los tipos siguientes: int, Integer, short, Short, long, Long o java.sql.Timestamp. Una entidad debe tener sólo un atributo de versión definido. El atributo de versión sólo se debe establecer durante la construcción. Después de persistir la entidad, el valor del atributo de versión no se debe modificar.

Si no se configura el atributo de versión y se utiliza la estrategia de bloqueo optimista, se crea una versión implícitamente de todo el tuple utilizando el estado completo del tuple.

En el ejemplo siguiente, la entidad Employee tiene un atributo de versión de tipo long denominado SequenceNumber:

```
@Entity
public class Employee
{
private long sequence;
    // Número de secuencia utilizado para la creación de versiones optimista.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
```

```
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Otros métodos get/set para otros campos del objeto Employee.
}
```

Escritura de una implementación de OptimisticCallback

Una implementación de OptimisticCallback debe implementar la interfaz OptimisticCallback y seguir los convenios comunes de plug-in de ObjectGrid

La siguiente lista proporciona una descripción o una consideración para cada uno de los métodos de la interfaz OptimisticCallback:

NULL_OPTIMISTIC_VERSION

Este valor especial es devuelto por el método getVersionedObjectForValue si se utiliza la implementación predeterminada de OptimisticCallback, en lugar de una predeterminado de OptimisticCallback proporcionada por la aplicación.

Método getVersionedObjectForValue

El método getVersionedObjectForValue podría devolver una copia del valor, o podría devolver un atributo del valor que se puede utilizar para la creación de versiones. Este método se llama siempre que se asocia un objeto con una transacción. Cuando no se establece ningún cargador en una correlación de respaldo, la correlación de respaldo utiliza este valor en la fase de confirmación para realizar una comparación de versiones optimista. La correlación de respaldo utiliza la comparación de versiones optimista para garantizar que la versión no ha cambiado desde la primera vez que esta transacción accedió a la entrada de correlación modificada por esta transacción. Si otra transacción hubiera modificado la versión de esta entrada de correlación, se produciría una anomalía en la comparación de versiones y la correlación de respaldo mostraría una excepción OptimisticCollisionException que forzaría la retrotracción de la transacción. Si hay un cargador conectado, la correlación de respaldo no utiliza la información de creación de versiones optimista. En su lugar, el cargador deberá realizar una comparación de versiones optimista y actualizar la información de la creación de versiones cuando sea necesario. Normalmente, el cargador obtiene el objeto inicial de la creación de versiones del LogElement pasado al método batchUpdate en el cargador, que se llama cuando se produce una operación de vaciado o cuando se confirma una transacción.

El código siguiente muestra la implementación que utiliza el objeto EmployeeOptimisticCallbackImpl:

```
public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}
```

Tal como se ha demostrado en el ejemplo anterior, el atributo sequenceNumber se devuelve en un objeto java.lang.Long tal como esperaba el cargador, que implica que la misma que escribió el cargador, también escribió la implementación de EmployeeOptimisticCallbackImpl, o bien trabajó estrechamente con la persona que implementó la implementación de EmployeeOptimisticCallbackImpl. Por ejemplo, estas personas acordaron el valor devuelto por el método getVersionedObjectForValue. Tal como se ha descrito previamente, la implementación predeterminada de OptimisticCallback devuelve el valor especial NULL_OPTIMISTIC_VERSION como el objeto de la versión.

Método updateVersionedObjectForValue

Se llama al método updateVersionedObjectForValue cuando una transacción ha actualizado un valor y es necesario un nuevo objeto de versión. Si el método getVersionedObjectForValue devuelve un atributo del valor, este método suele actualizar el valor de atributo con un nuevo objeto de versión. Si el método getVersionedObjectForValue devuelve una copia del valor, este método normalmente no se actualiza. El OptimisticCallback predeterminado no se actualiza porque la implementación predeterminada del método getVersionedObjectForValue siempre devuelve el valor especial NULL_OPTIMISTIC_VERSION como el objeto de versión. El siguiente ejemplo muestra la implementación utilizada por el objeto EmployeeOptimisticCallbackImpl que se utiliza en la sección OptimisticCallback:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Como se demuestra en el ejemplo anterior, el atributo sequenceNumber se incrementa por uno, de forma que la próxima vez que se llame al método getVersionedObjectForValue, el valor java.lang.Long devuelto tiene un valor largo que es el valor de número de secuencia original. Más de uno, por ejemplo, es el siguiente valor de versión para esta instancia de empleado. De nuevo, este ejemplo implica que la misma persona que escribió el cargador escribió EmployeeOptimisticCallbackImpl o colaboró con la persona que implementó EmployeeOptimisticCallbackImpl.

Método serializeVersionedValue

Este método escribe el valor con versión en la corriente especificada. En función de la implementación, el valor con versión puede utilizarse para identificar colisiones de actualización optimista. En algunas implementaciones, el valor con versión es una copia del valor original. Otras implementaciones podrían tener un número de secuencia o algún otro objeto para indicar la versión del valor. Puesto que se desconoce la implementación real, este método se proporciona para realizar la serialización correcta. La implementación predeterminada llama al método writeObject.

Método inflateVersionedValue

Este método toma la versión serializada del valor con versión y devuelve el objeto de valor con versión real. En función de la implementación, el valor con versión puede utilizarse para identificar colisiones de actualización optimista. En algunas

implementaciones, el valor con versión es una copia del valor original. Otras implementaciones podrían tener un número de secuencia o algún otro objeto para indicar la versión del valor. Puesto que la implementación real se desconoce, se proporciona este método para realizar la deserialización adecuada. La implementación predeterminada llama al método readObject.

Utilización de una implementación de OptimisticCallback proporcionada por la aplicación

Tiene dos enfoques para añadir un OptimisticCallback proporcionado por la aplicación en la configuración de BackingMap: configuración mediante programa y configuración de XML.

Conexión mediante programación de una implementación de OptimisticCallback

El siguiente ejemplo demuestra cómo una aplicación puede conectar mediante programación un objeto OptimisticCallback para la correlación de respaldo de empleado en la instancia del ObjectGrid grid1:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

Enfoque de configuración de XML para conectar una implementación de OptimisticCallback

El objeto EmployeeOptimisticCallbackImpl del ejemplo anterior debe implementar la interfaz OptimisticCallback. La aplicación también puede utilizar un archivo XML para conectar su objeto OptimisticCallback tal como se muestra en el siguiente ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid1">
        <backingMap name="employees" pluginCollectionRef="employees" lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>

<backingMapPluginCollections>
    <backingMapPluginCollection id="employees">
        <bean id="OptimisticCallback" className="com.xyz.EmployeeOptimisticCallbackImpl" />
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Visión general del proceso de transacciones

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

Para interactuar con los datos, la hebra de la aplicación requiere su propia sesión. Si la aplicación desea utilizar el ObjectGrid en una hebra, llame a uno de los métodos ObjectGrid.getSession para obtener una hebra. Con la sesión, la aplicación puede trabajar con los datos almacenados en las correlaciones de ObjectGrid.

Cuando una aplicación utiliza un objeto Session, la sesión debe estar en el contexto de una transacción. Una transacción empieza o se confirma y retrotrae mediante los métodos begin, commit y rollback en el objeto Session. Las aplicaciones

también pueden funcionar en la modalidad de confirmación automática, en la que Session empieza automáticamente y confirma una transacción, siempre que se realiza una operación en la correlación. Una modalidad de confirmación automática no puede agrupar varias operaciones en una única transacción, de forma que es la opción más lenta si crea un proceso por lotes de varias operaciones en una única transacción. Sin embargo, para las transacciones que sólo contienen una operación, la confirmación automática es la opción más rápida.

Introducción a las ranuras de plug-in

Una ranura de plug-in es un espacio de almacenamiento transaccional que se reserva para los plug-ins que comparten un contexto transaccional. Estas ranuras proporcionan una forma para que los plug-ins de eXtreme Scale se comuniquen entre sí, compartan el contexto transaccional y garanticen que los recursos transaccionales se utilizan de forma correcta y coherente dentro de una transacción.

Un plug-in puede almacenar el contexto transaccional como, por ejemplo, una conexión de base de datos, una conexión JMS (Java Message Service), etc. en una ranura de plug-in. El contexto transaccional almacenado puede recuperarse mediante cualquier plug-in que conozca el número de ranura de plug-in, que sirve como clave para recuperar el contexto transaccional.

Uso de ranuras de plug-in

Las ranuras de plug-in forman parte de la interfaz TxID. Consulte la documentación de la API si desea más información sobre la interfaz. Las ranuras son entradas en una matriz ArrayList. Los plug-ins pueden reservar una entrada en la matriz ArrayList llamando al método ObjectGrid.reserveSlot e indicando que desea una ranura en todos los objetos TxID. Después de reservar las ranuras, los plug-ins pueden colocar contexto transaccional en las ranuras de cada objeto TxID y recuperar el contexto más adelante. Las operaciones put y get se coordinan a través de números de ranura devueltos por el método ObjectGrid.reserveSlot.

Normalmente, un plug-in tiene un ciclo de vida. El uso de ranuras de plug-in debe ajustarse al ciclo de vida del plug-in. Por lo general, el plug-in debe reservar las ranuras de plug-in durante la fase de inicialización y obtener los números de ranura para cada ranura. Durante la ejecución normal, el plug-in coloca el contexto transaccional en la ranura reservada en el objeto TxID en el punto apropiado. Este punto apropiado suele estar el principio de la transacción. A continuación, el plug-in u otros plug-ins pueden obtener el contexto transaccional almacenado mediante el número de ranura desde TxID dentro de la transacción.

Normalmente, el plug-in realiza una limpieza eliminando el contexto transaccional y las ranuras. El siguiente fragmento de código ilustra cómo utilizar las ranuras de plug-in en un plug-in TransactionCallback:

```
public class DatabaseTransactionCallback implements TransactionCallback {
    int connectionSlot;
    int autoCommitConnectionSlot;
    int psCacheSlot;
    Properties ivProperties = new Properties();

    public void initialize(ObjectGrid objectGrid) throws TransactionCallbackException {
        // En la fase de inicialización, reservar las ranuras de plug-in deseadas llamando al
        //método reserveSlot de ObjectGrid y
        // pasar el nombre de ranura designado, TxID.SLOT_NAME.
        // Nota: debe pasarlo en este TxID.SLOT_NAME diseñado para
        // la aplicación.
        try {
            // almacenar en memoria caché los números de ranura devueltos
            connectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
            psCacheSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
            autoCommitConnectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
        }catch (Exception e) {
        }
```

```
}

public void begin(TxID tx) throws TransactionCallbackException {
    // colocar contextos transaccionales en las ranuras reservadas al
    // principio de la transacción.
    try {
        Connection conn = null;
        conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
        tx.putSlot(connectionSlot, conn);
        conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
        conn.setAutoCommit(true);
        tx.putSlot(autoCommitConnectionSlot, conn);
        tx.putSlot(psCacheSlot, new HashMap());
    } catch (SQLException e) {
        SQLException ex = getLastSQLException(e);
        throw new TransactionCallbackException("unable to get connection", ex);
    }
}

public void commit(TxID id) throws TransactionCallbackException {
    // obtener los contextos transaccionales almacenados y utilizarlos
    // después, limpiar todos los recursos transaccionales.
    try {
        Connection conn = (Connection) id.getSlot(connectionSlot);
        conn.commit();
        cleanUpSlots(id);
    } catch (SQLException e) {
        SQLException ex = getLastSQLException(e);
        throw new TransactionCallbackException("commit failure", ex);
    }
}

void cleanUpSlots(TxID tx) throws TransactionCallbackException {
    closePreparedStatements((Map) tx.getSlot(psCacheSlot));
    closeConnection((Connection) tx.getSlot(connectionSlot));
    closeConnection((Connection) tx.getSlot(autoCommitConnectionSlot));
}

/**
 * @param map
 */
private void closePreparedStatements(Map psCache) {
    try {
        Collection statements = psCache.values();
        Iterator iter = statements.iterator();
        while (iter.hasNext()) {
            PreparedStatement stmt = (PreparedStatement) iter.next();
            stmt.close();
        }
    } catch (Throwable e) {
    }

}

/**
 * Cerrar conexión y aceptar cualquier objeto Throwable que se produzca.
 *
 * @param connection
 */
private void closeConnection(Connection connection) {
    try {
        connection.close();
    } catch (Throwable e1) {
    }
}

public void rollback(TxID id) throws TransactionCallbackException
    // obtener los contextos transaccionales almacenados y utilizarlos
    // después, limpiar todos los recursos transaccionales.
    try {
        Connection conn = (Connection) id.getSlot(connectionSlot);
        conn.rollback();
        cleanUpSlots(id);
    } catch (SQLException e) {
    }
}

public boolean isExternalTransactionActive(Session session) {
    return false;
}

// Métodos getter para los números de ranura, otro plug-in puede obtener los números de ranura
// de estos métodos getter.

public int getConnectionSlot() {
    return connectionSlot;
}
public int getAutoCommitConnectionSlot() {
    return autoCommitConnectionSlot;
```

```
    }
    public int getPreparedStatementSlot() {
        return psCacheSlot;
    }
```

El siguiente fragmento de código ilustra cómo un cargador puede obtener el contexto transaccional almacenado colocado por un plug-in de ejemplo TransactionCallback anterior:

```
public class DatabaseLoader implements Loader
{
    DatabaseTransactionCallback tcb;
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
    {
        // El método preload es el método de inicialización de Loader.
        // Obtener el plug-in deseado de la instancia de Session u ObjectGrid.
        tcb =
   (DatabaseTransactionCallback)session.getObjectGrid().getTransactionCallback();
    }
    public List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException
    {
        // obtener los contextos transaccionales almacenados que coloca el  método begin de tcb.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implementar get aquí
        return null;
    }
    public void batchUpdate(TxID txid, LogSequence sequence) throws LoaderException,
   OptimisticCollisionException
    {
        // obtener los contextos transaccionales almacenados que coloca el  método begin de tcb.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implementar actualización de proceso por lotes aquí...
    }
}
```

Gestores de transacciones externas

Normalmente, las transacciones eXtreme Scale empiezan con el método Session.begin y finalizan con el método Session.commit. Sin embargo, cuando se incorpora un ObjectGrid, un coordinador de transacciones externas puede iniciar y finalizar transacciones. En este caso, no es necesario llamar a los métodos begin o commit.

Coordinación de transacciones externas

El plug-in TransactionCallback se amplía con el método isExternalTransactionActive(Session session) que asocia la sesión de eXtreme Scale con una transacción externa. La cabecera del método es la siguiente:

```
public synchronized boolean isExternalTransactionActive(Session session)
```

Por ejemplo, eXtreme Scale se puede configurar para integrarse con WebSphere Application Server y WebSphere Extended Deployment.

Además, eXtreme Scale proporciona un plug-in incorporado llamado WebSphere “Plug-ins para gestionar los sucesos del ciclo de vida de transacciones” en la página 385, que describe cómo generar el plug-in para los entornos de WebSphere Application Server, pero puede adaptar el plug-in para otras infraestructuras.

La clave de esta integración sin fisuras es la explotación de la API ExtendedJTATransaction en WebSphere Application Server versión 5.x y versión 6.x. Sin embargo, si utiliza WebSphere Application Server versión 6.0.2, debe aplicar el APAR PK07848 para soportar este método. Utilice el siguiente código de ejemplo para asociar a un objeto ObjectGrid con un ID de transacción de WebSphere Application Server:

```
/**
* Este método es necesario para asociar una sesión de objectGrid con un ID de
* transacción de WebSphere Application Server.
*/
Map/**/ localIdToSession;
```

```
public synchronized boolean isExternalTransactionActive(Session session)
{
    // recuerde que este localid significa que la sesión se ha guardado para
ser utilizada más tarde.
    localIdToSession.put(new Integer(jta.getLocalId()), session);
    return true;
}
```

Recuperación de una transacción externa

A veces, es posible que tenga que recuperar un objeto de servicio de transacción externa para que lo utilice el plug-in TransactionCallback. En el servidor WebSphere Application Server, busque el objeto ExtendedJTATransaction en su espacio de nombres, tal como se muestra en el ejemplo siguiente:

```
public J2EETransactionCallback() {
    super();
    localIdToSession = new HashMap();
    String lookupName="java:comp/websphere/ExtendedJTATransaction";
    try
    {
        InitialContext ic = new InitialContext();
        jta = (ExtendedJTATransaction)ic.lookup(lookupName);
        jta.registerSynchronizationCallback(this);
    }
    catch(NotSupportedException e)
    {
        throw new RuntimeException("Cannot register jta callback", e);
    }
    catch(NamingException e){
        throw new RuntimeException("Cannot get transaction object");
    }
}
```

Para otros productos, puede utilizar un acercamiento similar para recuperar el objeto de servicio de transacción.

Control de la confirmación mediante la devolución de llamada externa

El plug-in TransactionCallback debe recibir una señal externa para confirmar o retrotraer la sesión de eXtreme Scale. Para recibir esta señal externa, utilice la devolución de llamada del servicio de transacción externa. Implemente la interfaz de devolución de llamada externa y regístrela con el servicio de transacción externa. Por ejemplo, con WebSphere Application Server, implemente la interfaz SynchronizationCallback, tal como se muestra en el ejemplo siguiente:

```
public class J2EETransactionCallback implements
 com.ibm.websphere.objectgrid.plugins.TransactionCallback, SynchronizationCallback {
   public J2EETransactionCallback() {
      super();
      String lookupName="java:comp/websphere/ExtendedJTATransaction";
      localIdToSession = new HashMap();
      try {
         InitialContext ic = new InitialContext();
         jta = (ExtendedJTATransaction)ic.lookup(lookupName);
         jta.registerSynchronizationCallback(this);
      } catch(NotSupportedException e) {
         throw new RuntimeException("Cannot register jta callback", e);
      }
      catch(NamingException e) {
         throw new RuntimeException("Cannot get transaction object");
      }
   }

   public synchronized void afterCompletion(int localId, byte[] arg1,boolean didCommit) {
      Integer lid = new Integer(localId);
      // buscar localId de Session
      Session session = (Session)localIdToSession.get(lid);
      if(session != null) {
          try {
            // si WebSphere Application Server se confirma al
```

```
            // proteger la transacción para backingMap.
            // Se realizó un vaciado en beforeCompletion
            if(didCommit) {
               session.commit();
            } else {
               // de lo contrario, retrotraer
               session.rollback();
            }
         } catch(NoActiveTransactionException e) {
            // imposible en teoría
         } catch(TransactionException e) {
            // como ya se ha hecho un vaciado, no debería producirse error
         } finally {
            // siempre borra la sesión de la correlación
            localIdToSession.remove(lid);
         }
      }
   }

   public synchronized void beforeCompletion(int localId, byte[] arg1) {
      Session session = (Session)localIdToSession.get(new Integer(localId));
      if(session != null) {
         try {
            session.flush();
         } catch(TransactionException e) {
            // WebSphere Application Server no define formalmente
            // una manera de indicar que la
            // transacción ha fallado, por lo que debe hacer esto
            throw new RuntimeException("Cache flush failed", e);
         }
      }
   }
}
```

Utilice las API de eXtreme Scale con el plug-in TransactionCallback

El plug-in TransactionCallback inhabilita la confirmación automática en eXtreme Scale. El patrón de uso normal para un eXtreme Scale es el siguiente:

```
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
ogSession.begin();
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
ogSession.commit();
```

Cuando se utiliza este plug-in TransactionCallback, eXtreme Scale presupone que la aplicación utiliza eXtreme Scale cuando está presente una transacción gestionada por contenedor. El fragmento de código anterior cambia el siguiente código en este entorno:

```
public void myMethod() {
   UserTransaction tx = ...;
   tx.begin();
   Session ogSession = ...;
   ObjectMap myMap = ogSession.getMap("MyMap");
   yObject v = myMap.get("key");
   v.setAttribute("newValue");
   myMap.update("key", v);
   tx.commit();
}
```

El método myMethod es similar a un escenario de aplicación web. La aplicación utiliza la interfaz UserTransaction normal para empezar, confirmar y retrotraer transacciones. eXtreme Scale se inicia y se confirma automáticamente cerca de la transacción de contenedor. Si el método es un método EJB (Enterprise JavaBeans) que utiliza el atributo TX_REQUIRES, elimine la referencia de UserTransaction y las llamadas para iniciar y confirmar transacciones y el método funcionan del mismo modo. En este caso, el contenedor es responsable de iniciar y terminar la transacción.

Plug-in WebSphereTransactionCallback

Cuando utilice el plug-in WebSphereTransactionCallback, las aplicaciones empresariales que se ejecutan en un entorno de WebSphere Application Server puede gestionar las transacciones de ObjectGrid.

Cuando se utiliza una sesión ObjectGrid dentro de un método que está configurado para utilizar las transacciones gestionadas por contenedor, se inicia el contenedor empresarial, confirma o retrotrae automáticamente la transacción ObjectGrid. Si utiliza objetos UserTransaction de JTA (Java Transaction API), la transacción de ObjectGrid es gestionada automáticamente por el objeto UserTransaction.

Si desea una descripción detallada de la implementación de este plug-in, consulte "Gestores de transacciones externas" en la página 392.

Nota: ObjectGrid no admite transacciones XA de dos fases. Este plug-in no lista la transacción ObjectGrid con el gestor de transacciones. Por lo tanto, si ObjectGrid no puede realizar la confirmación, todos los recursos gestionados por la transacción XA no se retrotraen.

Conexión a través de programas del objeto WebSphereTransactionCallback

Puede habilitar WebSphereTransactionCallback en la configuración de ObjectGrid con la configuración programática o la configuración de XML. El siguiente fragmento de código utiliza la aplicación para crear el objeto WebSphereTransactionCallback y lo añade a un ObjectGrid:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
WebSphereTransactionCallback wsTxCallback= new WebSphereTransactionCallback ();
myGrid.setTransactionCallback(wsTxCallback);
```

Enfoque de configuración de XML para conectarse al objeto WebSphereTransactionCallback

La siguiente configuración de XML crea el objeto WebSphereTransactionCallback y lo añade a un ObjectGrid. El siguiente texto debe aparecer en el archivo `myGrid.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
    xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="myGrid">
            <bean id="TransactionCallback" className=
 "com.ibm.websphere.objectgrid.plugins.builtins.WebSphereTransactionCallback" />

        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Programación para utilizar la infraestructura OSGi

Puede iniciar servidores y clientes de eXtreme Scale en un contenedor OSGi, lo que le permite añadir y actualizar dinámicamente plug-ins de eXtreme Scale en el entorno de ejecución.

Conceptos relacionados:

"Visión general de la programación del serializador" en la página 313
Puede utilizar los plug-ins DataSerializer para grabar serializadores optimizados a fin de almacenar objetos Java y otros datos en formato binario en la cuadrícula. El plug-in también proporciona métodos que puede utilizar para consultar atributos en los datos binarios sin necesidad de que el objeto de datos entero se infle.

Visión general de serialización
Los datos normalmente se expresan, pero no se almacenan necesariamente, como objetos Java en la cuadrícula de datos. WebSphere eXtreme Scale utiliza varios procesos Java para serializar los datos, convirtiendo las instancias de objeto Java en bytes y de nuevo en objetos, según se requiera, para mover datos entre procesos de cliente y servidor.

Información relacionada:

Documentación de la API DataSerializer

Creación de plug-ins dinámicos de eXtreme Scale

WebSphere eXtreme Scale incluye los plug-ins ObjectGrid y BackingMap. Estos plug-ins se implementan en Java y se configuran utilizando el archivo XML de descriptor de ObjectGrid. Para crear un plug-in dinámico que se pueda actualizar dinámicamente, es necesario estar al corriente de los sucesos de ciclo de vida de ObjectGrid y BackingMap porque es posible que sea necesario completar algunas acciones durante la actualización. La ampliación de un paquete de plug-in con métodos de devolución de llamada de ciclo de vida, escuchas de sucesos, o ambos, permite al plug-in completar estas acciones en los momentos adecuados.

Antes de empezar

En este tema se supone que ha creado el plug-in apropiado. Para obtener más información sobre el desarrollo de plug-ins de eXtreme Scale, consulte el tema Plug-ins y API del sistema.

Acerca de esta tarea

Todos los plug-ins de eXtreme Scale se aplican a una instancia BackingMap u ObjectGrid. Muchos plug-ins también interactúan con otros plug-ins. Por ejemplo, un cargador y un plug-in TransactionCallback trabajan juntos para interactuar correctamente con una transacción de base de datos y las diversas llamadas JDBC de base de datos. Es posible que algunos plug-ins requieran también que se almacenen en la memoria caché datos de configuración de otros plug-ins a fin de mejorar el rendimiento.

Los plug-ins BackingMapLifecycleListener y ObjectGridLifecycleListener proporcionan operaciones de ciclo de vida para las instancias BackingMap y ObjectGrid respectivas. Este proceso permite notificar a los plug-ins cuando es posible que se cambien la BackingMap o la ObjectGrid padre y sus respectivos plug-ins. Los plug-ins BackingMap implementan la interfaz BackingMapLifecyleListener y los plug-ins ObjectGrid implementan la interfaz ObjectGridLifecycleListener. Estos plug-ins se invocan automáticamente cuando cambia el ciclo de vida de la BackingMap o ObjectGrid padre. Para obtener más información sobre los plug-ins de ciclo de vida, consulte el tema "Gestión de ciclos de vida de plug-ins" en la página 301.

Puede esperar ampliar los paquetes utilizando los métodos de ciclo de vida o escuchas de suceso en las siguientes tareas comunes:

- Inicio y detención de recursos, como por ejemplo hebras o suscriptores de mensajería.
- Si se especifica que se produzca una notificación cuando los plug-ins de igual se actualicen, lo que permite acceso directo al plug-in y la detección de los cambios.

Siempre que acceda a otro plug-in directamente, acceda a ese plug-in mediante el contenedor OSGi para asegurarse de que todas las partes del sistema hagan referencia al plug-in correcto. Si, por ejemplo, algún componente de la aplicación almacena en la memoria caché o hace referencia directamente a una instancia de un plug-in, mantendrá su referencia a esa versión del plug-in, incluso después de que el plug-in se haya actualizado dinámicamente. Este comportamiento puede causar problemas relacionados con la aplicación así como fugas de memoria. Por consiguiente, escriba código que dependa de plug-ins dinámicos que obtienen la referencia utilizando la semántica OSGi, getService(). Si la aplicación debe almacenar en memoria caché uno o varios plug-ins, escucha los sucesos de ciclo de vida utilizando las interfaces ObjectGridLifecycleListener y BackingMapLifecycleListener. La aplicación debe poder renovar también su memoria caché cuando sea necesario, en modalidad de seguridad de hebra.

Todos los plug-ins de eXtreme Scale utilizados con OSGi también deben implementar las interfaces BackingMapPlugin u ObjectGridPlugin respectivas. Los plug-ins nuevos, como la interfaz MapSerializerPlugin, imponen esta práctica. Estas interfaces proporcionan al entorno de ejecución de eXtreme Scale y a OSGi una interfaz coherente para inyectar el estado en el plug-in y controlar su ciclo de vida.

Utilice esta tarea para especificar que se produzca una notificación cuando se actualicen plug-ins de igual. Puede crear una fábrica de escuchas que genere una instancia de escucha.

Procedimiento

- Actualice la clase de plug-in ObjectGrid para implementar la interfaz ObjectGridPlugin. Esta interfaz incluye métodos que permiten a eXtreme Scale inicializar, establecer la instancia de ObjectGrid y destruir el plug-in. Consulte el siguiente código de ejemplo:

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridPlugin;
...

public class MyTranCallback implements TransactionCallback, ObjectGridPlugin {

    private ObjectGrid og = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setObjectGrid(ObjectGrid grid) {
        this.og = grid;
    }

    public ObjectGrid getObjectGrid() {
        return this.og;
    }
    void initialize() {
        // Manejar la inicialización de plug-in aquí. Lo llama
        // eXtreme Scale y no el gestor de beans OSGi.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destruir el plug-in y liberar los recursos. A éste
        // lo puede llamar el gestor de beans OSGi o eXtreme Scale.
        state = State.DESTROYED;
```

```
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- Actualice la clase de plug-in ObjectGrid para implementar la interfaz ObjectGridLifecycleListener. Consulte el siguiente código de ejemplo:

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener;
      import com.ibm.websphere.objectgrid.plugins.ObjectGridLifecycleListener.LifecycleEvent;
      ...

      public class MyTranCallback implements TransactionCallback, ObjectGridPlugin, ObjectGridLifecycleListener{
          public void objectGridStateChanged(LifecycleEvent event) {
              switch(event.getState()) {
              case NEW:
              case DESTROYED:
              case DESTROYING:
              case INITIALIZING:
                  break;
              case INITIALIZED:
                  // Buscar un cargador o MapSerializerPlugin utilizando
                  // OSGi o directamente desde la instancia de ObjectGrid.
                  lookupOtherPlugins()
                  break;
              case STARTING:
              case PRELOAD:
                  break;
              case ONLINE:
                  if (event.isWritable()) {
                      startupProcessingForPrimary();
                  } else {
                      startupProcessingForReplica();
                  }
                  break;
              case QUIESCE:
                  if (event.isWritable()) {
                      quiesceProcessingForPrimary();
                  } else {
                      quiesceProcessingForReplica();
                  }
                  break;
              case OFFLINE:
                  shutdownShardComponents();
                  break;
              }
          }
          ...
      }
```

- Actualice un plug-in BackingMap. Actualice la clase de plug-in BackingMap para implementar la interfaz de plug-in BackingMap. Esta interfaz incluye métodos que permiten a eXtreme Scale inicializar, establecer la instancia de BackingMap y destruir el plug-in. Consulte el siguiente código de ejemplo:

```
package com.mycompany;
import com.ibm.websphere.objectgrid.plugins.BackingMapPlugin;
...

public class MyLoader implements Loader, BackingMapPlugin {

    private BackingMap bmap = null;

    private enum State {
        NEW, INITIALIZED, DESTROYED
    }

    private State state = State.NEW;

    public void setBackingMap(BackingMap map) {
        this.bmap = map;
    }

    public BackingMap getBackingMap() {
        return this.bmap;
    }
    void initialize() {
        // Manejar la inicialización de plug-in aquí. Lo llama
        // eXtreme Scale y no el gestor de beans OSGi.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destruir el plug-in y liberar los recursos. A éste
        // lo puede llamar el gestor de beans OSGi o eXtreme Scale.
        state = State.DESTROYED;
```

```
    }

    public boolean isDestroyed() {
        return state == State.DESTROYED;
    }
}
```

- Actualice la clase de plug-in BackingMap para implementar la interfaz BackingMapLifecycleListener. Consulte el siguiente código de ejemplo:

```
package com.mycompany;

    import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener;
    import com.ibm.websphere.objectgrid.plugins.BackingMapLifecycleListener.LifecycleEvent;
    ...

    public class MyLoader implements Loader, ObjectGridPlugin, ObjectGridLifecycleListener{
        ...
        public void backingMapStateChanged(LifecycleEvent event) {
            switch(event.getState()) {
            case NEW:
            case DESTROYED:
            case DESTROYING:
            case INITIALIZING:
                break;
            case INITIALIZED:
                // Buscar un MapSerializerPlugin utilizando
                // OSGi o directamente desde la instancia de ObjectGrid.
                lookupOtherPlugins()
                break;
            case STARTING:
            case PRELOAD:
                break;
            case ONLINE:
                if (event.isWritable()) {
                    startupProcessingForPrimary();
                } else {
                    startupProcessingForReplica();
                }
                break;
            case QUIESCE:
                if (event.isWritable()) {
                    quiesceProcessingForPrimary();
                } else {
                    quiesceProcessingForReplica();
                }
                break;
            case OFFLINE:
                shutdownShardComponents();
                break;
            }
        }
        ...
    }
```

Resultados

Al implementar la interfaz ObjectGridPlugin o BackingMapPlugin, eXtreme Scale puede controlar el ciclo de vida del plug-in en los momentos correctos.

Al implementar la interfaz ObjectGridLifecycleListener o BackingMapLifecycleListener, el plug-in se registra automáticamente como escucha de los sucesos de ciclo de vida ObjectGrid o BackingMap asociados. El suceso INITIALIZING se utiliza para señalar que todos los plug-ins ObjectGrid y BackingMap se han inicializado y están disponibles para buscarse y utilizarse. El suceso ONLINE se utiliza para señalar que el ObjectGrid está en línea y listo para iniciar el proceso de sucesos.

Programación de la integración JPA

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

Para utilizar JPA, debe tener un proveedor JPA soportado como, por ejemplo, OpenJPA o Hibernate, archivos JAR y un archivo `META-INF/persistence.xml` en la classpath.

Tareas relacionadas:

"Resolución de problemas de los cargadores" en la página 524
Utilice esta información para resolver problemas de los cargadores de base de datos.

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

Cargadores JPA

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

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

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

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

Arquitectura del cargador JPA

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

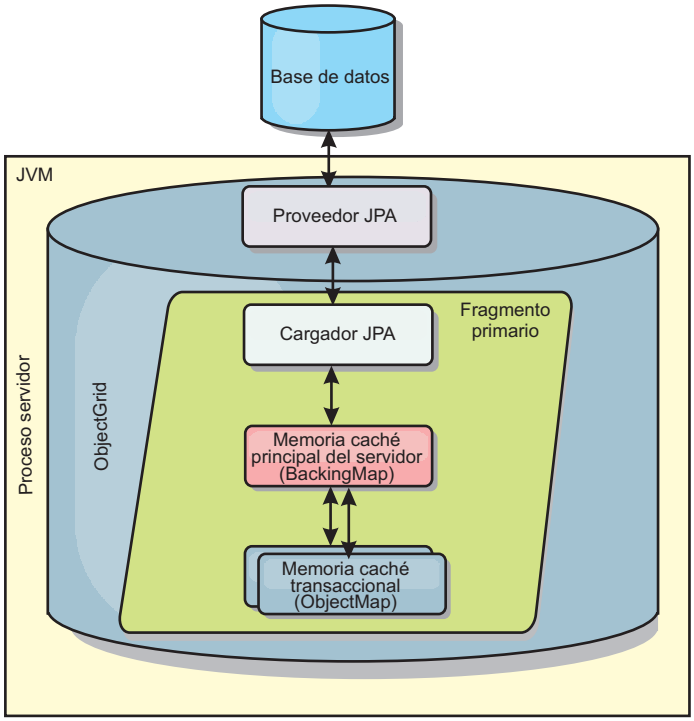


Figura 28. Arquitectura del cargador JPA

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

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

En JPAEntityLoader, también están implicadas las mismas cuatro capas. Sin embargo, dado que se utiliza el plug-in JPAEntityLoader para las correlaciones que almacenan las entidades de eXtreme Scale, las relaciones entre las entidades podrían complicar el escenario de uso. Se distingue una entidad eXtreme Scale de la entidad JPA. Para obtener más detalles, consulte “Plug-in JPAEntityLoader” en la página 372.

Métodos

Los cargadores proporcionan tres métodos principales:

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

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

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

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

Configuración y programación

Si está utilizando cargadores JPA en un entorno multimaestro, consulte "Consideraciones sobre el cargador en una topología multimaestro" en la página 106. Para obtener más información sobre cómo configurar cargadores JPA, consulte la información sobre los cargadores JPA en la *Guía de administración*. Si desea más información sobre cómo programar cargadores JPA, consulte *Guía de programación*.

Desarrollo de cargadores JPA basados en cliente

Puede implementar la precarga y recarga de datos en la aplicación con el programa de utilidad JPA (Java Persistence API). Esta prestación puede simplificar la carga de correlaciones cuando no se pueden particionar las consultas a la base de datos.

Antes de empezar

- Debe estar utilizando un proveedor JPA con una base de datos soportada.
- Antes de precargar o recargar las correlaciones, debe establecer el estado de disponibilidad del ObjectGrid en `PRELOAD`. Puede establecer el estado de disponibilidad con el método setObjectGridState de la interfaz StateManager. La interfaz StateManager impide a otros clientes acceder al ObjectGrid cuando todavía no está en línea. Después de precargar o recargar la correlación, puede establecer el estado de nuevo en ONLINE.
- Al precargar distintas correlaciones en un ObjectGrid, establezca el estado de ObjectGrid en `PRELOAD` una vez y vuelva a establecer el valor en `ONLINE` después de que todas las correlaciones acaben de cargar los datos. Esta coordinación se puede realizar mediante la interfaz ClientLoadCallback. Establezca el estado de ObjectGrid en `PRELOAD` después de recibir la primera notificación de preStart de la instancia de ObjectGrid y vuelva a establecerlo en `ONLINE` después de recibir la última notificación de postFinish.
- Si tiene que precargar las correlaciones de distintas máquinas virtuales Java, se requiere la coordinación entre varias máquinas virtuales Java. Establezca el estado de ObjectGrid en `PRELOAD` una vez vez antes de que se precargue la primera correlación en cualquiera de las máquinas virtuales Java y establezca el

valor de nuevo en `ONLINE` después de que todas las correlaciones finalicen la carga de datos en todas las máquinas virtuales Java. Para obtener más información, consulte Gestión de la disponibilidad del ObjectGrid.

Acerca de esta tarea

Al ejecutar una operación de precarga o recarga en la correlación, se producen las acciones siguientes:

1. La acción inicial que se realiza depende de si está ejecutando una operación de precarga o recarga.
   - **Operación de precarga:** la correlación que se deben precargar se borra. Para una correlación de entidad, si alguna relación se ha configurado como cascade-remove, las correlaciones relacionadas se borran.
   - **Operación de recarga:** la consulta proporcionada se ejecuta en la correlación y los resultados se invalidan. Para una correlación de entidad, si alguna relación se configura con la opción **`CascadeType.INVALIDATE`**, las entidades relacionadas también se invalidan desde sus correlaciones.
2. Ejecute la consulta en JPA para las entidades de un proceso por lotes.
3. Para cada lote, se crea una lista de claves y una lista de valores para cada partición.
4. Para cada partición, se llama al agente de cuadrícula de datos para insertar o actualizar los datos en el lado del servidor directamente si es un cliente de eXtreme Scale. Si la cuadrícula de datos es una instancia local, los datos de las correlaciones se insertan o actualizan directamente.

Conceptos relacionados:

"Visión general del programa de utilidad de precarga JPA basada en cliente"
El programa de utilidad de precarga de JPA (Java Persistence API) basado en cliente carga los datos en las correlaciones de respaldo de eXtreme Scale utilizando una conexión cliente con ObjectGrid.

Referencia relacionada:

"Ejemplo: Precarga de una correlación con la interfaz ClientLoader" en la página 405
Puede precargar una correlación para llenar la correlación con datos antes de que los clientes puedan acceder a la correlación.

"Ejemplo: Recarga de una correlación con la interfaz ClientLoader" en la página 406
Recargar una correlación equivale a precargar una correlación, excepto que el argumento **`isPreload`** se establece en false en el método ClientLoader.load.

"Ejemplo: Llamar a un cargador de clientes" en la página 407
Puede utilizar el método de precarga en la interfaz Loader para llamar a un cargador de clientes.

Información relacionada:

Interfaz ClientLoader

Interfaz StateManager

Visión general del programa de utilidad de precarga JPA basada en cliente

El programa de utilidad de precarga de JPA (Java Persistence API) basado en cliente carga los datos en las correlaciones de respaldo de eXtreme Scale utilizando una conexión cliente con ObjectGrid.

Esta prestación puede simplificar la carga de correlaciones cuando no se pueden particionar las consultas a la base de datos. También se puede utilizar un cargador como, por ejemplo, un cargador JPA, y es ideal cuando los datos se pueden cargar en paralelo.

El programa de utilidad de precarga JPA basado en cliente puede utilizar las implementaciones JPA de OpenJPA o Hibernate para cargar el ObjectGrid desde una base de datos. Puesto que WebSphere eXtreme Scale no interactúa directamente con la base de datos o JDBC (Java Database Connectivity), se puede utilizar cualquier base de datos que soporte OpenJPA o Hibernate para cargar el ObjectGrid.

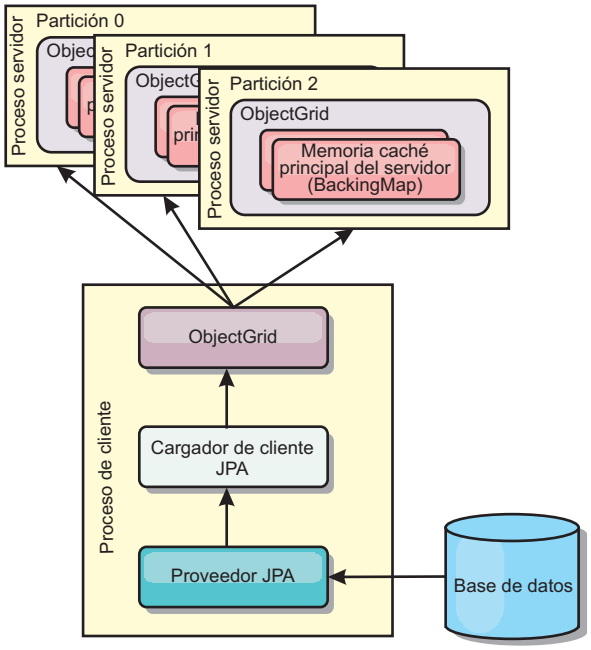


Figura 29. El cargador de cliente que utiliza la implementación JPA para cargar el ObjectGrid

Por lo general, una aplicación de usuario proporciona un nombre de unidad de persistencia, un nombre de clase de entidad y una consulta JPA al cargador de cliente. El cargador de cliente recupera el gestor de entidades JPA de acuerdo con el nombre de la unidad de persistencia, utiliza el gestor de entidades para consultar los datos de la base de datos con la clase de entidad y la consulta JPA proporcionadas, y finalmente carga los datos en las correlaciones ObjectGrid distribuidas. Cuando hay implicadas relaciones de varios niveles en la consulta, puede utilizar una serie de consulta personalizada para optimizar el rendimiento. De forma opcional, puede proporcionarse una correlación de propiedades de persistencia para alterar temporalmente las propiedades de persistencia configuradas.

Un cargador de cliente puede cargar los datos en dos modalidades distintas, tal como se muestra en la tabla siguiente:

Tabla 10. Modalidades del cargador de cliente

Modalidad	Descripción
Precarga	Borra todas las entradas y las carga en la correlación de respaldo. Si la correlación es una correlación de entidad, se borrarán también todas las correlaciones de entidad relacionadas si se ha habilitado la opción de ObjectGrid CascadeType.REMOVE.
Recarga	La consulta JPA se ejecuta en el objeto ObjectGrid para invalidar todas las entradas de la correlación que coincidan con la consulta. Si la correlación es una correlación de entidad, se borrarán también todas las correlaciones de entidad relacionadas si se ha habilitado la opción de ObjectGrid CascadeType.INVALIDATE.

En cualquier caso, una consulta JPA se utiliza para seleccionar y cargar las entidades deseadas desde la base de datos y para almacenarlas en las correlaciones ObjectGrid. Si la correlación ObjectGrid es una correlación que no es de entidad, las entidades JPA se separarán y se almacenarán directamente. Si la correlación ObjectGrid es una correlación de entidades, las entidades JPA se almacenan como tuples de entidad ObjectGrid. Puede proporcionar una consulta JPA o utilizar la consulta predeterminada `select o from EntityName o`.

Si desea más información sobre cómo configurar el programa de utilidad de precarga de JPA basado en cliente, consulte "Desarrollo de cargadores JPA basados en cliente" en la página 402la información de la *Guía de programación*

Tareas relacionadas:

"Desarrollo de cargadores JPA basados en cliente" en la página 402
Puede implementar la precarga y recarga de datos en la aplicación con el programa de utilidad JPA (Java Persistence API). Esta prestación puede simplificar la carga de correlaciones cuando no se pueden particionar las consultas a la base de datos.

Referencia relacionada:

"Ejemplo: Precarga de una correlación con la interfaz ClientLoader"
Puede precargar una correlación para llenar la correlación con datos antes de que los clientes puedan acceder a la correlación.

"Ejemplo: Recarga de una correlación con la interfaz ClientLoader" en la página 406
Recargar una correlación equivale a precargar una correlación, excepto que el argumento **`isPreload`** se establece en false en el método ClientLoader.load.

"Ejemplo: Llamar a un cargador de clientes" en la página 407
Puede utilizar el método de precarga en la interfaz Loader para llamar a un cargador de clientes.

Información relacionada:

Interfaz ClientLoader

Interfaz StateManager

Ejemplo: Precarga de una correlación con la interfaz ClientLoader

Puede precargar una correlación para llenar la correlación con datos antes de que los clientes puedan acceder a la correlación.

Ejemplo de precarga basada en cliente

El siguiente fragmento de código de ejemplo muestra una carga sencilla de cliente. En este ejemplo, la correlación CUSTOMER se configura como correlación de entidad. La clase de entidad Customer, que se configura en el archivo XML de descriptor de metadatos de entidad ObjectGrid, tiene una relación de uno a muchos con las entidades Order. La entidad Customer tiene la opción CascadeType.ALL habilitada en la relación con la entidad Order.Antes de que se llame al método ClientLoader.load, el estado de ObjectGrid se establece en PRELOAD. El parámetro **isPreload** en el método de carga se establece en `true`.

```
// Obtener StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();

// Establecer el estado de ObjectGrid en PRELOAD antes de llamar a
ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Cargar los datos
c.load(objectGrid, "CUSTOMER", "customerPU", null,
    null, null, null, true, null);

// Volver a establecer el estado de ObjectGrid en ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Conceptos relacionados:

"Visión general del programa de utilidad de precarga JPA basada en cliente" en la página 403
El programa de utilidad de precarga de JPA (Java Persistence API) basado en cliente carga los datos en las correlaciones de respaldo de eXtreme Scale utilizando una conexión cliente con ObjectGrid.

Tareas relacionadas:

"Desarrollo de cargadores JPA basados en cliente" en la página 402
Puede implementar la precarga y recarga de datos en la aplicación con el programa de utilidad JPA (Java Persistence API). Esta prestación puede simplificar la carga de correlaciones cuando no se pueden particionar las consultas a la base de datos.

Información relacionada:

Interfaz ClientLoader

Interfaz StateManager

Ejemplo: Recarga de una correlación con la interfaz ClientLoader

Recargar una correlación equivale a precargar una correlación, excepto que el argumento **isPreload** se establece en false en el método ClientLoader.load.

Ejemplo de recarga basada en cliente

El ejemplo siguiente muestra cómo recargar correlaciones. Comparado con el ejemplo de precarga, la principal diferencia es que se proporcionan loadSql y parámetros. Este ejemplo solo recarga los datos de clientes con un ID entre `1000` y `2000`. El parámetro **isPreload** en el método de carga se establece en `false`.

```
// Obtener StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();

// Establecer el estado de ObjectGrid en PRELOAD antes de llamar a
ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);
```

```
ClientLoader c = ClientLoaderFactory.getClientLoader();

// Cargar los datos
String loadSql = "select c from CUSTOMER c
    where c.custId >= :startCustId and c.custId < :endCustId ";
Map<String, Long> params = new HashMap<String, Long>();
params.put("startCustId", 1000L);
params.put("endCustId", 2000L);

c.load(objectGrid, "CUSTOMER", "customerPU", null, null,
    loadSql, params, false, null);

// Volver a establecer el estado de ObjectGrid en ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Recuerde: Esta serie de consulta cumple tanto la sintaxis de consulta de JPA como la sintaxis de consulta de entidad de eXtreme Scale. Esta serie de consulta es importante porque se ejecuta dos veces: para invalidar las entidades ObjectGrid coincidentes y para cargar las entidades JPA coincidentes.

Conceptos relacionados:

"Visión general del programa de utilidad de precarga JPA basada en cliente" en la página 403
El programa de utilidad de precarga de JPA (Java Persistence API) basado en cliente carga los datos en las correlaciones de respaldo de eXtreme Scale utilizando una conexión cliente con ObjectGrid.

Tareas relacionadas:

"Desarrollo de cargadores JPA basados en cliente" en la página 402
Puede implementar la precarga y recarga de datos en la aplicación con el programa de utilidad JPA (Java Persistence API). Esta prestación puede simplificar la carga de correlaciones cuando no se pueden particionar las consultas a la base de datos.

Información relacionada:

Interfaz ClientLoader

Interfaz StateManager

Ejemplo: Llamar a un cargador de clientes

Puede utilizar el método de precarga en la interfaz Loader para llamar a un cargador de clientes.

Utilice el método de precarga en la interfaz Loader para llamar a un cargador de cliente:

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

Este método indica al cargador que puede precargar los datos en la correlación. Una implementación de cargador puede utilizar un cargador de cliente para precargar los datos en todas las particiones. Por ejemplo, el cargador JPA utiliza el cargador de cliente para precargar los datos en la correlación.

Si desea más información, consulte el tema de visión general de cargadores JPA en la *Visión general del producto*.

Ejemplo: Llamar a un cargador de clientes con el método preloadMap

A continuación, se muestra un ejemplo sobre cómo precargar la correlación utilizando el cargador de clientes en el método preloadMap. El ejemplo, en primer lugar, comprueba si el número de partición actual es el mismo que el de la partición de precarga. Si el número de partición no es el mismo que el de la

partición de precarga, no se produce ninguna acción. Si los números de partición coinciden, se llama al cargador de cliente para cargar los datos en las correlaciones. Debe llamar al cargador de clientes en solo una partición.

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

....
 ObjectGrid objectGrid = session.getObjectGrid();
int partitionId = backingMap.getPartitionId();
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
// Llamar al cargador de cliente para cargar datos en sólo una partición
 if (partitionId == preloadPartition) {
  ClientLoader c = ClientLoaderFactory.getClientLoader();
// Llamar al cargador de cliente para cargar los datos
    try {
        c.load(objectGrid, "CUSTOMER", "customerPU",
     null, entityClass, null, null, true, null);
    } catch (ObjectGridException e) {
        LoaderException le = new LoaderException("Exception caught in
ObjectMap " + ogName + "." + mapName);
        le.initCause(e);
        throw le;
    }
 }
}
```

Recuerde: Configure el atributo de backingMap "preloadMode" en `true`, de modo que el método de precarga se ejecute de forma asíncrona. De lo contrario, el método de precarga bloquea la activación de la instancia de ObjectGrid.

Conceptos relacionados:

"Visión general del programa de utilidad de precarga JPA basada en cliente" en la página 403
El programa de utilidad de precarga de JPA (Java Persistence API) basado en cliente carga los datos en las correlaciones de respaldo de eXtreme Scale utilizando una conexión cliente con ObjectGrid.

Tareas relacionadas:

"Desarrollo de cargadores JPA basados en cliente" en la página 402
Puede implementar la precarga y recarga de datos en la aplicación con el programa de utilidad JPA (Java Persistence API). Esta prestación puede simplificar la carga de correlaciones cuando no se pueden particionar las consultas a la base de datos.

Información relacionada:

Interfaz ClientLoader

Interfaz StateManager

Ejemplo: Creación de un cargador JPA basado en cliente personalizado

El método ClientLoader.load de la interfaz Loader proporciona una función de carga de cliente que satisface la mayoría de los escenarios. Sin embargo, si desea cargar datos sin el método ClientLoader.load, puede implementar su propio programa de utilidad de precarga.

Plantilla de cargador personalizado

Utilice la siguiente plantilla para desarrollar el cargador:

```
// Obtener StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();

// Establecer el estado de ObjectGrid en PRELOAD antes de llamar a
ClientLoader.loader
```

```
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);


// Cargar los datos
...<la implementación del programa de utilidad de precarga>...

// Volver a establecer el estado de ObjectGrid en ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Desarrollo de un cargador JPA basado en cliente con un agente DataGrid

Al cargar desde el lado del cliente, la utilización de un agente DataGrid podría aumentar el rendimiento. Al utilizar un agente DataGrid, todas las lecturas y escrituras de datos se producen en el proceso de servidor. También puede diseñar la aplicación para asegurarse de que los agentes DataGrid de varias particiones se ejecuten en paralelo para aumentar más el rendimiento.

Acerca de esta tarea

Para obtener más información sobre el agente DataGrid, consulte "API DataGrid y particionamiento" en la página 263.

Después de crear la implementación de precarga de datos, puede crear un cargador genérico para completar las tareas siguientes:

- Consulte los datos de la base de datos en lotes.
- Cree una lista de claves y una lista de valores para cada partición.
- Para cada partición, llame al método agentMgr.callReduceAgent(agent, aKey) para ejecutar el agente en el servidor de una hebra. Al realizar la ejecución en una hebra, puede ejecutar agentes de forma simultánea en varias particiones.

Ejemplo

El siguiente fragmento de código es un ejemplo de cómo realizar la carga mediante un agente DataGrid:

```
import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

import com.ibm.websphere.objectgrid.NoActiveTransactionException;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TransactionException;
import com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent;
import com.ibm.websphere.objectgrid.em.EntityManager;

public class InsertAgent implements ReduceGridAgent, Externalizable {

    private static final long serialVersionUID = 6568906743945108310L;

    private List keys = null;

    private List vals = null;

    protected boolean isEntityMap;
```

```
public InsertAgent() {
}

public InsertAgent(boolean entityMap) {
    isEntityMap = entityMap;
}

public Object reduce(Session sess, ObjectMap map) {
    throw new UnsupportedOperationException(
        "ReduceGridAgent.reduce(Session, ObjectMap)");
}

public Object reduce(Session sess, ObjectMap map, Collection arg2) {
    Session s = null;
    try {
        s = sess.getObjectGrid().getSession();
        ObjectMap m = s.getMap(map.getName());
        s.beginNoWriteThrough();
        Object ret = process(s, m);
        s.commit();
        return ret;
    } catch (ObjectGridRuntimeException e) {
        if (s.isTransactionActive()) {
            try {
                s.rollback();
            } catch (TransactionException e1) {
            } catch (NoActiveTransactionException e1) {
            }
        }
        throw e;
    } catch (Throwable t) {
        if (s.isTransactionActive()) {
            try {
                s.rollback();
            } catch (TransactionException e1) {
            } catch (NoActiveTransactionException e1) {
            }
        }
        throw new ObjectGridRuntimeException(t);
    }

}

public Object process(Session s, ObjectMap m) {
    try {

        if (!isEntityMap) {
            // En el caso POJO, es una operación directa,
            // se coloca todo en la
            // correlación mediante la operación de insertar
            insert(m);
        } else {
            // 2. Caso Entity.
            // En el caso Entity, pueden persistirse las entidades
            EntityManager em = s.getEntityManager();
            persistEntities(em);

        }
        return Boolean.TRUE;
    } catch (ObjectGridRuntimeException e) {
        throw e;
    } catch (ObjectGridException e) {
        throw new ObjectGridRuntimeException(e);
    } catch (Throwable t) {
        throw new ObjectGridRuntimeException(t);
    }
```

```
}

/**
 * Básicamente se trata de una nueva carga.
 * @param s
 * @param m
 * @throws ObjectGridException
 */
protected void insert(ObjectMap m) throws ObjectGridException {

    int size = keys.size();

    for (int i = 0; i < size; i++) {
        m.insert(keys.get(i), vals.get(i));
    }

}


protected void persistEntities(EntityManager em) {
    Iterator<Object> iter = vals.iterator();

    while (iter.hasNext()) {
        Object value = iter.next();
        em.persist(value);
    }
}

public Object reduceResults(Collection arg0) {
    return arg0;
}

public void readExternal(ObjectInput in)
  throws IOException, ClassNotFoundException {
    int v = in.readByte();
    isEntityMap = in.readBoolean();
    vals = readList(in);
    if (!isEntityMap) {
        keys = readList(in);
    }
}

public void writeExternal(ObjectOutput out) throws IOException {
    out.write(1);
    out.writeBoolean(isEntityMap);

    writeList(out, vals);
    if (!isEntityMap) {
        writeList(out, keys);
    }

}

public void setData(List ks, List vs) {
    vals = vs;
    if (!isEntityMap) {
        keys = ks;
    }
}

/**
 * @return Devuelve isEntityMap.
 */
public boolean isEntityMap() {
    return isEntityMap;
}
```

```
    static public void writeList(ObjectOutput oo, Collection l)
      throws IOException {
        int size = l == null ? -1 : l.size();
        oo.writeInt(size);
        if (size > 0) {
            Iterator iter = l.iterator();
            while (iter.hasNext()) {
                Object o = iter.next();
                oo.writeObject(o);
            }
        }
    }

    public static List readList(ObjectInput oi)
      throws IOException, ClassNotFoundException {
        int size = oi.readInt();
        if (size == -1) {
            return null;
        }


        ArrayList list = new ArrayList(size);
        for (int i = 0; i < size; ++i) {
            Object o = oi.readObject();
            list.add(o);
        }
        return list;
    }
}
```

Ejemplo: Utilización del plug-in Hibernate para precargar datos en la memoria caché de ObjectGrid

Puede utilizar el método preload de la clase ObjectGridHibernateCacheProvider para precargar los datos en la memoria caché de ObjectGrid para una clase de entidad.

Ejemplo: Utilización de la clase EntityManagerFactory

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("testPU");
ObjectGridHibernateCacheProvider.preload("objectGridName", emf, TargetEntity.class, 100, 100);
```

Importante: De forma predeterminada, las entidades no forman parte de la memoria caché de segundo nivel. En las clases de entidad que requieren almacenamiento en memoria caché, añada la anotación @cache. A continuación se muestra un ejemplo:

```
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;
@Entity
@Cache(usage=CacheConcurrencyStrategy.TRANSACTIONAL)
public class HibernateCacheTest { ... }
```

Puede sustituir este valor predeterminado estableciendo el elemento shared-cache-mode en el archivo `persistence.xml` o mediante la propiedad javax.persistence.sharedCache.mode.

Ejemplo: Utilización de la clase SessionFactory

```
org.hibernate.cfg.Configuration cfg = new Configuration();
// utilizar el método addResource, addClass y setProperty de Configuration para preparar
// la configuración necesaria para crear SessionFactor
SessionFactory sessionFactory= cfg.buildSessionFactory();
ObjectGridHibernateCacheProvider.preload("objectGridName", sessionFactory,
TargetEntity.class, 100, 100);
```

Nota:

1. En un sistema distribuido, este mecanismo de precarga sólo se puede invocar desde una máquina virtual Java. El mecanismo de precarga no se puede ejecutar de forma simultánea desde varias Mäquinas virtuales Java.
2. Antes de ejecutar la precarga, debe inicializar la memoria caché de eXtreme Scale creando EntityManager mediante EntityManagerFactory para crear todas las BackingMaps correspondientes; de lo contrario, la precarga obliga a que se inicialice la memoria caché con solo una BackingMap predeterminada para dar soporte a todas las entidades. Esto significa que todas las entidades deberán compartir un objeto BackingMap.

Inicio del actualizador basado en la hora de JPA

Cuando inicie el actualizador basado en la hora de JPA (Java Persistence API), las correlaciones de ObjectGrid se actualizan con los últimos cambios de la base de datos.

Antes de empezar

Configure el actualizador basado en la hora. Consulte Configuración de un actualizador de datos basado en la hora de JPA la información sobre cómo configurar el actualizador de datos basado en tiempo JPA en la *Guía de administración*.

Acerca de esta tarea

Si desea más información sobre cómo funciona el actualizador de datos basado en la hora de Java Persistence API (JPA), consulte "Actualizador de datos basado en la hora de JPA" en la página 416.

Procedimiento

- Inicie un actualizador de base de datos basado en la hora.
  - **De forma automática para el eXtreme Scale distribuido:**

    Si crea la configuración de timeBasedDBUpdate para la correlación de respaldo, el actualizador de la base de datos basado en la hora se inicia automáticamente cuando se activa un fragmento distribuido de ObjectGrid. Para un ObjectGrid que tiene varias particiones, el actualizador de la base de datos basado en la hora sólo se inicia en la partición 0.
  - **De forma automática para el eXtreme Scale local:**

    Si crea la configuración de timeBasedDBUpdate para la correlación de respaldo, el actualizador de la base de datos basado en la hora se inicia automáticamente cuando se activa la correlación local.
  - **Manualmente:**

    También puede iniciar o detener manualmente un actualizador de base de datos basado en la hora mediante la API TimeBasedDBUpdater.

    ```
    public synchronized void startDBUpdate(ObjectGrid objectGrid, String mapName,
     String punitName, Class entityClass, String timestampField, DBUpdateMode mode) {
    ```

    1. **ObjectGrid**: instancia de ObjectGrid (local o cliente).
    2. **mapName**: nombre de la correlación de respaldo para la cual se inicia el actualizador de base de datos basado en la hora.
    3. **punitName**: el nombre de la unidad de persistencia JPA para crear una fábrica del gestor de entidades JPA; el valor predeterminado es el nombre de la primera unidad de persistencia definida en el archivo `persistence.xml`.

4. **entityClass**: el nombre de la clase de entidad utilizado para interactuar con el proveedor de JPA (Java Persistence API); el nombre de la clase de entidad se utiliza para obtener entidades JPA utilizando las consultas de entidad.
5. **timestampField**: un campo de indicación de fecha y hora de la clase de entidad para identificar la hora o la secuencia cuando se actualizó o insertó por última vez un registro de programa de fondo de una base de datos.
6. **mode**: modalidad de actualización de la base datos basada en tiempo; un tipo INVALIDATE_ONLY invalida las entradas en la correlación ObjectGrid si se han modificado los registros correspondientes en la base de datos; un tipo UPDATE_ONLY indica sólo la actualización de las entradas existentes en la correlación ObjectGrid con los valores más recientes de la base datos; no obstante, todos los registros recientemente insertados en la base de datos se pasan por alto; un tipo INSERT_UPDATE indica la actualización de las entradas existentes en la correlación ObjectGrid con los últimos valores de la base de datos; además, todos los registros recién insertados en la base de datos se insertan en la correlación ObjectGrid.

Si desea detener el actualizador de base de datos basado en la hora, puede llamar al siguiente método para detener el actualizador:

```
public synchronized void stopDBUpdate(ObjectGrid objectGrid, String mapName)
```

Los parámetros ObjectGrid y mapName deben ser los mismos que los que se pasan en el método startDBUpdate.

- Cree el campo de indicación de fecha y hora en la base de datos.
  - **DB2**

    Como parte de la características de bloqueo optimista, DB2 9.5 proporciona una función de indicación de fecha y hora de cambio de fila. Puede definir una columna ROWCHGTS mediante el formato ROW CHANGE TIMESTAMP de la siguiente manera:

    ```
    ROWCHGTS TIMESTAMP NOT NULL
         GENERATED ALWAYS
         FOR EACH ROW ON UPDATE AS
         ROW CHANGE TIMESTAMP
    ```

    A continuación, puede indicar el campo de entidad que corresponde a esta columna como campo de indicación de hora mediante una anotación o configuración. A continuación se muestra un ejemplo:

    ```
    @Entity(name = "USER_DB2")
    @Table(name = "USER1")
    public class User_DB2 implements Serializable {

        private static final long serialVersionUID = 1L;

        public User_DB2() {
        }

        public User_DB2(int id, String firstName, String lastName) {
            this.id = id;
            this.firstName = firstName;
            this.lastName = lastName;
        }

        @Id
        @Column(name = "ID")
        public int id;

        @Column(name = "FIRSTNAME")
    ```

```
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ROWCHGTS", updatable = false, insertable = false)
    public Timestamp rowChgTs;
}
```

– **Oracle**

  En Oracle, hay una pseudocolumna ora_rowscn para el número de cambio del sistema del registro. Puede utilizar esta columna para el mismo propósito. A continuación, aparece un ejemplo de la entidad que utiliza el campo ora_rowscn como el campo de indicación de fecha y hora de actualización de la base de datos basada en tiempo:

```
@Entity(name = "USER_ORA")
@Table(name = "USER1")
public class User_ORA implements Serializable  {

    private static final long serialVersionUID = 1L;

    public User_ORA() {
    }

    public User_ORA(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ora_rowscn", updatable = false, insertable = false)
    public long rowChgTs;
}
```

– **Otras bases de datos**

  Para otros tipos de bases de datos, puede crear una columna de tabla para realizar un seguimiento de los cambios. Los valores de la columna debe gestionarlos manualmente la aplicación que actualiza la tabla.

  Tome una base de datos Apache Derby como un ejemplo: puede crear una columna "ROWCHGTS" para rastrear los números de cambio. Además, se realiza un seguimiento del último número de cambio en esta tabla. Cada vez que se inserta o actualiza un registro, se aumenta el número de cambio más reciente en la tabla, y el valor de la columna ROWCHGTS para el registro se actualiza con este número incrementado.

```
@Entity(name = "USER_DER")
@Table(name = "USER1")
public class User_DER implements Serializable {

    private static final long serialVersionUID = 1L;

    public User_DER() {
    }
```

```
    public User_DER(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ROWCHGTS", updatable = true, insertable = true)
    public long rowChgTs;
}
```

Actualizador de datos basado en la hora de JPA

Un actualizador de base de datos basado en la hora de JPA (Java Persistence API) actualiza las correlaciones de ObjectGrid con los últimos cambios de la base de datos.

Cuando los cambios se realizan directamente en una base de datos que es atendida por WebSphere eXtreme Scale, estos cambios no se reflejan de forma simultánea en la cuadrícula de eXtreme Scale. Para implementar correctamente eXtreme Scale como un espacio de proceso de base de datos en memoria, tenga en cuenta que la cuadrícula puede perder la sincronización con la base de datos. El actualizador de base de datos basado en la hora utiliza la capacidad SCN (System Change Number) en Oracle 10g la indicación de fecha y hora de cambio de fila en DB2 9.5 para supervisar los cambios en la base de datos para la invalidación y la actualización. El actualizador también permite a las aplicaciones tener un campo definido por el usuario con el mismo propósito.

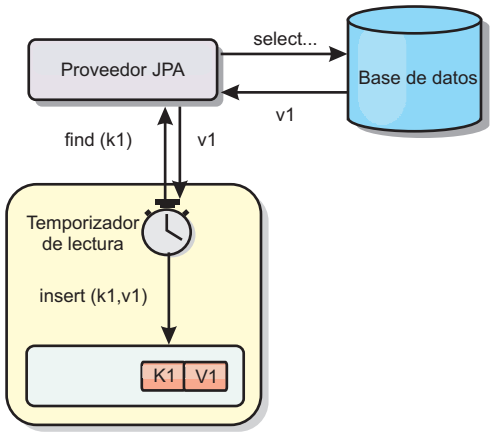


Figura 30. Renovación periódica

El actualizador de la base de datos basado en la hora consulta periódicamente la base de datos utilizando interfaces JPA para obtener las entidades JPA que representan los registros recién insertados y actualizados en la base de datos. Para actualizar de forma periódica los registros, cada registro de la base de datos debe tener una indicación de fecha y hora para identificar la hora o secuencia en la que

se actualizó o insertó el registro por última vez. No es necesarios que la indicación de fecha y hora esté en un formato de indicación de fecha y hora. El valor de indicación de fecha y hora puede ser tener un formato de entero o largo, si genera un valor único creciente.

Esta prestación la proporcionan varias bases de datos comerciales.

Por ejemplo, en DB2 9.5, puede definir una columna utilizando el formato ROW CHANGE TIMESTAMP del modo siguiente:

```
ROWCHGTS TIMESTAMP NOT NULL
      GENERATED ALWAYS
      FOR EACH ROW ON UPDATE AS
      ROW CHANGE TIMESTAMP
```

En Oracle, puede utilizar la pseudo-columna **ora_rowscn**, que representa el número de cambio de sistema del registro.

El actualizador de base de datos basado en la hora actualiza las correlaciones ObjectGrid de tres maneras diferentes:

1. INVALIDATE_ONLY. Invalida las entradas de la correlación ObjectGrid si han cambiado los registros correspondientes de la base de datos.
2. UPDATE_ONLY. Actualiza las entradas de la correlación ObjectGrid si han cambiado los registros correspondientes de la base de datos. Sin embargo, todos los registros recién insertados en la base de datos se ignoran.
3. INSERT_UPDATE. Actualiza las entradas existentes en la correlación ObjectGrid con los valores más recientes de la base de datos. Además, todos los registros recién insertados en la base de datos se insertan en la correlación de ObjectGrid.

Si desea más información sobre cómo configurar el actualizador de datos basado en tiempo de JPA, consulte la información de la *Guía de administración*.

Desarrollo de aplicaciones con la infraestructura Spring

Obtenga información sobre cómo integrar las aplicaciones de eXtreme Scale con la conocida infraestructura Spring.

Conceptos relacionados:

"Visión general de la infraestructura Spring" en la página 120
Spring es una infraestructura de desarrollo de aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para permitir a Spring gestionar transacciones y configurar los clientes y servidores que conforman una cuadrícula de datos en memoria desplegada.

"Beans de ampliación de Spring y soporte de espacio de nombres" en la página 425
WebSphere eXtreme Scale proporciona una característica para declarar objetos POJO (Plain Old Java Object) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml` y un método para denominar los beans y, a continuación, especificar el nombre de la clase. Normalmente, se crean las instancias de la clase especificada y estos objetos se utilizan como los plug-ins. Ahora, eXtreme Scale puede delegar en Spring para obtener las instancias de estos objetos de plug-in. Si una aplicación utiliza Spring en general será necesario que los POJO se conecten al resto de la aplicación.

Referencia relacionada:

"Beans de ampliación gestionados de Spring" en la página 423
Puede declarar POJO (Plain Old Java Objects) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml`. Si denomina los beans y luego especifica el nombre de clase, eXtreme Scale suele crear instancias de la clase especificada y utiliza esas instancias como plug-in. Ahora, WebSphere eXtreme Scale ObjectGrid puede delegar en Spring para actuar como la fábrica de beans para obtener instancias de estos objetos de plug-in.

Archivo XML de descriptor Spring
Utilice un archivo XML de descriptor Spring para configurar e integrar eXtreme Scale con Spring.

Archivo Spring `objectgrid.xsd`
Utilice el archivo Spring `objectgrid.xsd` para integrar eXtreme Scale con Spring para gestionar las transacciones eXtreme Scale y configurar clientes y servidores.

Visión general de la infraestructura Spring

Spring es una infraestructura de desarrollo de aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para permitir a Spring gestionar transacciones y configurar los clientes y servidores que conforman una cuadrícula de datos en memoria desplegada.

Transacciones nativas gestionadas de Spring

Spring proporciona transacciones gestionadas por contenedor que son similares al servidor de aplicaciones Java Platform, Enterprise Edition. Sin embargo, el mecanismo Spring puede utilizar distintas implementaciones. WebSphere eXtreme Scale proporciona una integración del gestor de transacciones que permite a Spring gestionar los ciclos de vida de transacción de ObjectGrid. Consulte la información sobre transacciones nativas en la *Guía de programación* para obtener más información.

Beans de ampliación gestionados de Spring y soporte de espacio de nombres

Además, eXtreme Scale se integra con Spring para habilitar a los beans de estilo Spring definidos para los puntos o plug-ins de ampliación. Esta característica proporciona configuraciones más sofisticadas y más flexibilidad para configurar los puntos de ampliación.

Además de los beans de ampliación gestionados de Spring, eXtreme Scale proporciona un espacio de nombres Spring denominado "objectgrid". Los beans y las implementaciones incorporadas están definidos previamente en este espacio de nombres, que hace que sea más fácil para los usuarios configurar eXtreme Scale.

Soporte de ámbito de fragmento

Con la configuración de Spring de estilo tradicional, un bean ObjectGrid puede ser un tipo singleton o un tipo de prototipo. Además, ObjectGrid soporta un nuevo ámbito denominado el ámbito de "fragmento". Si un bean está definido como ámbito de fragmento, sólo se crea un bean por fragmento. Todas las solicitudes para los beans con un ID o varios ID que coincidan con dicha definición de bean en el mismo fragmento producirán que una instancia de bean específica sea devuelta por el contenedor Spring.

El siguiente ejemplo muestra que un bean com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl está definido con el ámbito establecido en shard (fragmento). Por lo tanto, sólo se crea una instancia de la clase JPAPropFactoryImpl por fragmento.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Flujo web de Spring

El flujo web de Spring almacena su estado de sesión en una sesión HTTP de forma predeterminada. Si una aplicación web utiliza eXtreme Scale para la gestión de sesiones, Spring almacena automáticamente el estado con eXtreme Scale. Además, la tolerancia a errores está habilitada de la misma forma que la sesión.

Consulte la información de gestión de sesiones HTTP en *Visión general del producto* para obtener detalles adicionales.

Empaquetado

Las extensiones Spring de eXtreme Scale están en el archivo `ogspring.jar`. Este archivo Java (JAR) debe estar en la classpath para trabajar con el soporte de Spring. Si una aplicación Java EE en ejecución en un WebSphere Extended Deployment ha aumentado WebSphere Application Server Network Deployment, coloque el archivo `spring.jar` y sus archivos asociados en los módulos de archivadores empresariales (EAR). También debe colocar el archivo `ogspring.jar` en la misma ubicación.

Tareas relacionadas:

"Desarrollo de aplicaciones con la infraestructura Spring" en la página 417
Obtenga información sobre cómo integrar las aplicaciones de eXtreme Scale con la conocida infraestructura Spring.

"Inicio de un servidor de contenedor con Spring" en la página 428
Puede iniciar un servidor de contenedor utilizando beans de ampliación gestionados Spring y soporte de espacio de nombres.

"Gestión de transacciones con Spring"
Spring es una infraestructura popular para desarrollar las aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para que Spring pueda gestionar transacciones de eXtreme Scale y configurar clientes y servidores de eXtreme Scale.

Referencia relacionada:

"Beans de ampliación gestionados de Spring" en la página 423
Puede declarar POJO (Plain Old Java Objects) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml`. Si denomina los beans y luego especifica el nombre de clase, eXtreme Scale suele crear instancias de la clase especificada y utiliza esas instancias como plug-in. Ahora, WebSphere eXtreme Scale ObjectGrid puede delegar en Spring para actuar como la fábrica de beans para obtener instancias de estos objetos de plug-in.

Archivo XML de descriptor Spring
Utilice un archivo XML de descriptor Spring para configurar e integrar eXtreme Scale con Spring.

Archivo Spring `objectgrid.xsd`
Utilice el archivo Spring `objectgrid.xsd` para integrar eXtreme Scale con Spring para gestionar las transacciones eXtreme Scale y configurar clientes y servidores.

Gestión de transacciones con Spring

Spring es una infraestructura popular para desarrollar las aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para que Spring pueda gestionar transacciones de eXtreme Scale y configurar clientes y servidores de eXtreme Scale.

Acerca de esta tarea

Spring Framework es altamente integrable con eXtreme Scale, tal como se describe en las secciones siguientes.

Procedimiento

- **Transacciones nativas:** Spring proporciona transacciones gestionadas por contenedor junto con el estilo de un servidor de aplicaciones Java Platform, Enterprise Edition, pero tiene la ventaja de que el mecanismo Springs puede tener distintas implementaciones conectadas. Este tema describe un gestor de transacciones de la plataforma eXtreme Scale que se puede utilizar con Spring. Esto permite a los programadores anotar sus POJO (Plain Old Java Object) y, a continuación, hacer que Spring adquiera automáticamente los objetos Sessions de eXtreme Scale, así como empezar, confirmar, retrotraer, suspender y reanudar transacciones eXtreme Scale. Las transacciones Spring se describen de forma más completa en el Capítulo 10 de la documentación de referencia oficial de Spring. A continuación se explica cómo crear un gestor de transacciones eXtreme Scale y utilizarlo con los POJO anotados. También explica cómo utilizar este enfoque con eXtreme Scale local o cliente así como una aplicación de estilo Data Grid con ubicación compartida.
- **Gestor de transacciones:** para trabajar con Spring, eXtreme Scale proporciona una implementación de un PlatformTransactionManager de Spring. Este gestor

puede proporcionar sesiones deeXtreme Scale gestionadas a los POJO gestionados por Spring. A través del uso de anotaciones, Spring gestiona estas sesiones para los POJO en términos de ciclo de vida de transacción. El siguiente fragmento de código XML muestra cómo crear un gestor de transacciones:

```
<aop:aspectj-autoproxy/>
 <tx:annotation-driven transaction-manager="transactionManager"/>

 <bean id="ObjectGridManager"
    class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
    factory-method="getObjectGridManager"/>

 <bean id="ObjectGrid"
    factory-bean="ObjectGridManager"
    factory-method="createObjectGrid"/>

 <bean id="transactionManager"
    class="com.ibm.websphere.objectgrid.spring.ObjectGridSpringFactory"
    factory-method="getLocalPlatformTransactionManager"/>
 </bean>

 <bean id="Service" class="com.ibm.websphere.objectgrid.spring.test.TestService">
  <property name="txManager" ref+"transactionManager"/>
 </bean>
```

Esto muestra el bean transactionManager declarándose y conectándose al bean Service que utilizará transacciones Spring. Los demostraremos utilizando anotaciones y por este motivo existe la cláusula tx:annotation al principio.

- **Obtención de una sesión de ObjectGrid:** Un POJO que tiene métodos gestionados por Spring ahora puede obtener la Session de ObjectGrid para la transacción actual utilizando

  ```
  Session s = txManager.getSession();
  ```

  Esto devuelve la sesión para que lo utilice POJO. Los beans que participan en la misma transacción recibirán la misma sesión cuando llame a este método. Spring manejará automáticamente el inicio de Session y también invocará automáticamente la confirmación o la retrotracción cuando sea necesario. Puede obtener un EntityManager de ObjectGrid llamando simplemente a getEntityManager desde el objeto Session.
- **Establecimiento de la instancia de ObjectGrid para una hebra:** una sola máquina virtual Java (JVM) puede alojar muchas instancias de ObjectGrid. Cada fragmento primario colocado en una JVM tiene su propia instancia de ObjectGrid. Una JVM que funcione como cliente para un ObjectGrid remoto utiliza una instancia de ObjectGrid devuelta de ClientClusterContext del método de conexión para interactuar con Grid. Antes de invocar un método en un POJO que utilicen transacciones Spring para ObjectGrid, la hebra debe prepararse con la instancia de ObjectGrid a utilizar. La instancia TransactionManager tiene un método que permite especificar una instancia de ObjectGrid concreta. Una vez que se ha especificado, todas las llamadas a txManager.getSession posteriores devolverán Sessions para esa instancia de ObjectGrid.

  El siguiente ejemplo muestra un ejemplo para utilizar esta prestación:

  ```
  ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext(new String[]
             {"applicationContext.xml"});
      SpringLocalTxManager txManager = (SpringLocalTxManager)ctx.getBean("transactionManager");
      txManager.setObjectGridForThread(og);

      ITestService s = (ITestService)ctx.getBean("Service");
      s.initialize();
      assertEquals(s.query(), "Billy");
      s.update("Bobby");
      assertEquals(s.query(), "Bobby");
      System.out.println("Requires new test");
      s.testRequiresNew(s);
      assertEquals(s.query(), "1");
  ```

  Aquí utilizamos un Spring ApplicationContext. ApplicationContext se utiliza para obtener una referencia para txManager y especificar un ObjectGrid que se va a utilizar con esta hebra. A continuación, el código obtiene una referencia

para el servicio e invoca métodos del mismo. Cada llamada al método de este nivel hace que Spring cree una Session y realice llamadas de inicio/confirmación alrededor de la llamada al método. Todas las excepciones causarán una retrotracción.

- **Interfaz de SpringLocalTxManager:** la interfaz de SpringLocalTxManager se implementa mediante Platform Transaction Manager de ObjectGrid y tiene todas las interfaces públicas. Los métodos de esta interfaz sirven para seleccionar la instancia de ObjectGrid para utilizar en una hebra y obtener una Session para la hebra. Todos los POJO que utilizan las transacciones locales de ObjectGrid deben incluirse con una referencia a esta instancia del gestor y sólo se debe crear una única instancia, es decir, su ámbito debe ser singleton. Esta instancia se crea utilizando un método estático en ObjectGridSpringFactory. getLocalPlatformTransactionManager().

  Restricción: WebSphere eXtreme Scale no da soporte a la confirmación en dos fases o JTA por diversas razones, principalmente que tienen que ver con la escalabilidad. Por ello, excepto en el último participante de una única fase, ObjectGrid no interactúa con transacciones globales de tipo XA o JTA. Este gestor de plataformas está pensado para hacer que la utilización de transacciones de ObjectGrid locales sea lo más fácil posible para los desarrolladores de Spring.

Conceptos relacionados:

"Visión general de la infraestructura Spring" en la página 120
Spring es una infraestructura de desarrollo de aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para permitir a Spring gestionar transacciones y configurar los clientes y servidores que conforman una cuadrícula de datos en memoria desplegada.

"Beans de ampliación de Spring y soporte de espacio de nombres" en la página 425
WebSphere eXtreme Scale proporciona una característica para declarar objetos POJO (Plain Old Java Object) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml` y un método para denominar los beans y, a continuación, especificar el nombre de la clase. Normalmente, se crean las instancias de la clase especificada y estos objetos se utilizan como los plug-ins. Ahora, eXtreme Scale puede delegar en Spring para obtener las instancias de estos objetos de plug-in. Si una aplicación utiliza Spring en general será necesario que los POJO se conecten al resto de la aplicación.

Referencia relacionada:

"Beans de ampliación gestionados de Spring"
Puede declarar POJO (Plain Old Java Objects) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml`. Si denomina los beans y luego especifica el nombre de clase, eXtreme Scale suele crear instancias de la clase especificada y utiliza esas instancias como plug-in. Ahora, WebSphere eXtreme Scale ObjectGrid puede delegar en Spring para actuar como la fábrica de beans para obtener instancias de estos objetos de plug-in.

Archivo XML de descriptor Spring
Utilice un archivo XML de descriptor Spring para configurar e integrar eXtreme Scale con Spring.

Archivo Spring `objectgrid.xsd`
Utilice el archivo Spring `objectgrid.xsd` para integrar eXtreme Scale con Spring para gestionar las transacciones eXtreme Scale y configurar clientes y servidores.

Beans de ampliación gestionados de Spring

Puede declarar POJO (Plain Old Java Objects) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml`. Si denomina los beans y luego especifica el nombre de clase, eXtreme Scale suele crear instancias de la clase especificada y utiliza esas instancias como plug-in. Ahora, WebSphere eXtreme Scale ObjectGrid puede delegar en Spring para actuar como la fábrica de beans para obtener instancias de estos objetos de plug-in.

Si una aplicación utiliza Spring, los POJO tienen como requisito ser accesibles al resto de la aplicación.

Una aplicación puede registrar una instancia de fábrica de beans Spring para utilizar para un ObjectGrid especificado por nombre. La aplicación crea una instancia de BeanFactory o un contexto de aplicación de Spring y a continuación la registra en ObjectGrid mediante el siguiente método estático:

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object springBeanFactory)
```

El método anterior se aplica al caso cuando eXtreme Scale encuentra un bean de ampliación cuyo className empieza con el prefijo {spring}. Un bean de ampliación de este tipo, que podría ser un ObjectTransformer, Loader, TransactionCallback, etc., utiliza el resto del nombre como un nombre de bean Spring. A continuación, obtiene la instancia de bean utilizando la fábrica de beans Spring.

El entorno de despliegue de eXtreme Scale también puede crear una fábrica de beans Spring desde un archivo de configuración XML Spring predeterminado. Si no se ha registrado ninguna fábrica de beans para un ObjectGrid determinado, el despliegue busca automáticamente un archivo XML denominado "/<ObjectGridName>_spring.xml". Por ejemplo, si la cuadrícula de datos se denomina GRID, el archivo XML se denomina "/GRID_spring.xml' y aparece en la classpath del paquete raíz. ObjectGrid construye un ApplicationContext utilizando el archivo "/<ObjectGridName>_spring.xml y construye beans desde esa fábrica de beans.

A continuación se muestra un nombre de clase de ejemplo:

```
"{spring}MyLoaderBean"
```

La utilización del nombre de clase anterior permite a eXtreme Scale utilizar Spring para buscar un bean denominado "MyLoaderBean". Puede especificar POJO gestionados por Spring para cualquier punto de ampliación si se ha registrado la fábrica de beans. Las ampliaciones Spring se encuentran en el archivo ogspring.jar. Este archivo JAR debe estar en la classpath para el soporte de Spring. Si una aplicación J2EE se ejecuta en WebSphere Application Server Network Deployment aumentado con WebSphere Extended Deployment, la aplicación debe colocar el archivo spring.jar y sus archivos asociados en los módulos EAR. El archivo ogspring.jar también debe colocarse en la misma ubicación.

Conceptos relacionados:

"Visión general de la infraestructura Spring" en la página 120
Spring es una infraestructura de desarrollo de aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para permitir a Spring gestionar transacciones y configurar los clientes y servidores que conforman una cuadrícula de datos en memoria desplegada.

"Beans de ampliación de Spring y soporte de espacio de nombres"
WebSphere eXtreme Scale proporciona una característica para declarar objetos POJO (Plain Old Java Object) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml` y un método para denominar los beans y, a continuación, especificar el nombre de la clase. Normalmente, se crean las instancias de la clase especificada y estos objetos se utilizan como los plug-ins. Ahora, eXtreme Scale puede delegar en Spring para obtener las instancias de estos objetos de plug-in. Si una aplicación utiliza Spring en general será necesario que los POJO se conecten al resto de la aplicación.

Tareas relacionadas:

"Desarrollo de aplicaciones con la infraestructura Spring" en la página 417
Obtenga información sobre cómo integrar las aplicaciones de eXtreme Scale con la conocida infraestructura Spring.

"Inicio de un servidor de contenedor con Spring" en la página 428
Puede iniciar un servidor de contenedor utilizando beans de ampliación gestionados Spring y soporte de espacio de nombres.

"Gestión de transacciones con Spring" en la página 420
Spring es una infraestructura popular para desarrollar las aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para que Spring pueda gestionar transacciones de eXtreme Scale y configurar clientes y servidores de eXtreme Scale.

Beans de ampliación de Spring y soporte de espacio de nombres

WebSphere eXtreme Scale proporciona una característica para declarar objetos POJO (Plain Old Java Object) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml` y un método para denominar los beans y, a continuación, especificar el nombre de la clase. Normalmente, se crean las instancias de la clase especificada y estos objetos se utilizan como los plug-ins. Ahora, eXtreme Scale puede delegar en Spring para obtener las instancias de estos objetos de plug-in. Si una aplicación utiliza Spring en general será necesario que los POJO se conecten al resto de la aplicación.

En algunos escenarios, debe utilizar Spring para configurar un plug-in, como en el ejemplo siguiente:

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
        <property name="persistenceUnitName" type="java.lang.String"  value="employeePU" />
    </bean>
    ...
</objectGrid>
```

La implementación de TransactionCallback incorporada, la clase com.ibm.websphere.objectgrid.jpa.JPATxCallback, se configura como la clase TransactionCallback. Esta clase se configura con la propiedad **`persistenceUnitName`**, tal como se muestra en el ejemplo anterior. La clase JPATxCallback también tiene el atributo JPAPropertyFactory, que es del tipo java.lang.Object. La configuración XML de ObjectGrid no puede soportar este tipo de configuración.

La integración de Spring eXtreme Scale resuelve este problema delegando la creación de bean en la infraestructura Spring. La configuración revisada es la siguiente:

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="{spring}jpaTxCallback"/>
    ...
</objectGrid>
```

El archivo spring para el objeto "Grid" contiene la siguiente información:

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.
JPAPropFactoryImpl" scope="shard">
</bean>
```

Aquí, TransactionCallback se especifica como `{spring}jpaTxCallback`, y los beans jpaTxCallback y jpaPropFactory se configuran en el archivo spring tal como se indica en el ejemplo anterior. La configuración de Spring hace posible configurar un bean JPAPropertyFactory como un parámetro del objeto JPATxCallback.

Fábrica de beans Spring predeterminada

Cuando eXtreme Scale encuentra un plug-in o un bean de ampliación (como ObjectTransformer, Loader, TransactionCallback, etc.) con un valor de classname que empieza con el prefijo `{spring}`, eXtreme Scale utiliza el resto del nombre como un nombre de bean Spring y obtenga la instancia del bean mediante la fábrica de beans de Spring.

De forma predeterminada, si no se registró ninguna fábrica de beans para un ObjectGrid determinado, intenta encontrar un archivo `ObjectGridName_spring.xml`. Por ejemplo, si la cuadrícula de datos se denomina "Grid", el archivo XML se denominará `/Grid_spring.xml`. Este archivo debe estar en la classpath o en un directorio `META-INF` que está en la classpath. Si no se encuentra este archivo, eXtreme Scale construye un ApplicationContext utilizando dicho archivo y construye beans desde esa fábrica de beans.

Fábrica de beans Spring personalizada

WebSphere eXtreme Scale también proporciona una API ObjectGridSpringFactory para registrar una instancia de fábrica de beans Spring para utilizar para un ObjectGrid con un nombre específico. Esta A^PI registra una instancia de BeanFactory con eXtreme Scale utilizando el siguiente método estático:

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object
springBeanFactory)
```

Soporte de espacio de nombres

Desde la versión 2.0, Spring tiene un mecanismo para las ampliaciones basadas en esquema del formato XML de Spring básico y para definir y configurar beans. ObjectGrid utiliza esta nueva características para definir y configurar beans ObjectGrid. Con la ampliación del esquema XML de Spring, algunas de las implementaciones incorporadas de los plug-ins eXtreme Scale y algunos beans ObjectGrid están definidos previamente en el espacio de nombres "objectgrid". Al

escribir los archivos de configuración de Spring, no tiene que especificar el nombre de clase completo de las implementaciones incorporadas. En lugar de esto, puede hacer referencia a los beans predefinidos.

Además, con los atributos de los beans definidos en el esquema XML, es menos probable que proporcione un nombre de atributo erróneo. La validación XML basada en el esquema XML puede capturar antes los errores de este tipo en el ciclo de desarrollo.

Estos beans definidos en las ampliaciones del esquema XML son:

- transactionManager
- register
- server
- catalog
- catalogServerProperties
- container
- JPALoader
- JPATxCallback
- JPAEntityLoader
- LRUEvictor
- LFUEvictor
- HashIndex

Estos beans están definidos en el esquema XML objectgrid.xsd. Este archivo XSD se suministra como un archivo `com/ibm/ws/objectgrid/spring/namespace/objectgrid.xsd` en el archivo `ogspring.jar`. Para ver descripciones detalladas del archivo XSD y los beans definidos en el archivo XSD, consulte Archivo XML de descriptor Spring la información sobre el archivo de descriptor de Spring en la *Guía de administración*.

Utilice el ejemplo de JPATxCallback de la sección anterior. En la sección anterior, se configura el bean JPATxCallback del modo siguiente:

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard">
</bean>
```

Mediante esta característica del espacio de nombres, la configuración XML de spring se puede escribir del modo siguiente:

```
<objectgrid:JPATxCallback id="jpaTxCallback" persistenceUnitName="employeeEMPU"
 jpaPropertyFactory="jpaPropFactory" />

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl"
 scope="shard">
</bean>
```

Tenga en cuenta que en lugar de especificar la clase the com.ibm.websphere.objectgrid.jpa.JPATxCallback como en el ejemplo anterior, utilizamos directamente el bean objectgrid:JPATxCallback predefinido. Como puede ver, esta configuración es menos verbosa y más apta para la comprobación de errores.

Para ver una descripción de cómo trabajar con beans Spring, consulte “Inicio de un servidor de contenedor con Spring” en la página 428.

Tareas relacionadas:

"Desarrollo de aplicaciones con la infraestructura Spring" en la página 417
Obtenga información sobre cómo integrar las aplicaciones de eXtreme Scale con la conocida infraestructura Spring.

"Inicio de un servidor de contenedor con Spring"
Puede iniciar un servidor de contenedor utilizando beans de ampliación gestionados Spring y soporte de espacio de nombres.

"Gestión de transacciones con Spring" en la página 420
Spring es una infraestructura popular para desarrollar las aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para que Spring pueda gestionar transacciones de eXtreme Scale y configurar clientes y servidores de eXtreme Scale.

Referencia relacionada:

"Beans de ampliación gestionados de Spring" en la página 423
Puede declarar POJO (Plain Old Java Objects) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml`. Si denomina los beans y luego especifica el nombre de clase, eXtreme Scale suele crear instancias de la clase especificada y utiliza esas instancias como plug-in. Ahora, WebSphere eXtreme Scale ObjectGrid puede delegar en Spring para actuar como la fábrica de beans para obtener instancias de estos objetos de plug-in.

Archivo XML de descriptor Spring
Utilice un archivo XML de descriptor Spring para configurar e integrar eXtreme Scale con Spring.

Archivo Spring `objectgrid.xsd`
Utilice el archivo Spring `objectgrid.xsd` para integrar eXtreme Scale con Spring para gestionar las transacciones eXtreme Scale y configurar clientes y servidores.

Inicio de un servidor de contenedor con Spring

Puede iniciar un servidor de contenedor utilizando beans de ampliación gestionados Spring y soporte de espacio de nombres.

Acerca de esta tarea

Con varios archivos XML configurados para Spring, puede iniciar servidores de contenedor de eXtreme Scale básicos.

Procedimiento

1. **Archivo XML de ObjectGrid:**

   En primer lugar, defina un archivo XML ObjectGrid muy sencillo que contenga una "Grid" de ObjectGrid "Grid" y una correlación "Test". ObjectGrid tiene un plug-in ObjectGridEventListener llamado "partitionListener", y la correlación "Test" tiene un desalojador conectado llamado "testLRUEvictor". Tenga en cuenta que el plug-in ObjectGridEventListener y el plug-in Evictor se han configurado ambos utilizando Spring ya que sus nombres contienen "{spring}".

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="Grid">
     <bean id="ObjectGridEventListener" className="{spring}partitionListener" />
            <backingMap name="Test" pluginCollectionRef="test" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
```

```
        <backingMapPluginCollection id="test">
            <bean id="Evictor" className="{spring}testLRUEvictor"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

2. **Archivo XML de despliegue de ObjectGrid:**

   Ahora, cree un archivo XML de despliegue de ObjectGrid sencillo del modo siguiente. Divida ObjectGrid en 5 particiones, no es necesaria ninguna réplica.

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numInitialContainers="1" numberOfPartitions="5" minSyncReplicas="0"
            maxSyncReplicas="1" maxAsyncReplicas="0">
            <map ref="Test"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

3. **Archivo XML de Spring de ObjectGrid:**

   Ahora se utilizarán ambas características, los beans de ampliación gestionados Spring de ObjectGrid y el soporte de espacio de nombres, para configurar los beans ObjectGrid. El archivo XML de Spring se denomina `Grid_spring.xml`. Tenga en cuenta que se incluyen dos esquemas en el archivo XML: `spring-beans-2.0.xsd` es para la utilización de los beans gestionados Spring, y `objectgrid.xsd` para la utilización de los beans predefinidos en el espacio de nombres de objectgrid.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
       xsi:schemaLocation="
       http://www.ibm.com/schema/objectgrid
   http://www.ibm.com/schema/objectgrid/objectgrid.xsd
       http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <objectgrid:register id="ogregister" gridname="Grid"/>

    <objectgrid:server id="server" isCatalog="true" name="server">
        <objectgrid:catalog host="localhost" port="2809"/>
    </objectgrid:server>

    <objectgrid:container id="container"
   objectgridxml="com/ibm/ws/objectgrid/test/springshard/objectgrid.xml"
       deploymentxml="com/ibm/ws/objectgrid/test/springshard/deployment.xml"
   server="server"/>

    <objectgrid:LRUEvictor id="testLRUEvictor" numberOfLRUQueues="31"/>

    <bean id="partitionListener"
   class="com.ibm.websphere.objectgrid.springshard.ShardListener" scope="shard"/>
</beans>
```

   Existen seis beans definidos en este archivo XML Spring:

   a. *objectgrid:register*: registra la fábrica de beans predeterminada para la "Grid" de ObjectGrid.
   b. *objectgrid:server*: define un servidor ObjectGrid con el nombre "server". Este servidor también proporcionará un servicio de catálogos puesto que tiene un bean objectgrid:catalog que está anidado ahí.
   c. *objectgrid:catalog*: define un punto final de servicio de catálogos ObjectGrid, que se establece en "localhost:2809".

d. *objectgrid:container*: define un contenedor ObjectGrid con un archivo XML objectgrid especificado y un archivo XML de despliegue, tal como se indicó antes. La propiedad de servidor especifica en qué servidor está alojado este contenedor.

e. *objectgrid:LRUEvictor*: define un LRUEvictor con el número de colas LRU para utilizar establecido en 31.

f. *bean partitionListener*: define un plug-in ShardListener. Debe proporcionar una implementación de este plug-in, de este modo no puede utilizar los beans predefinidos. Además, este ámbito del bean está establecido en "shard", que indica que sólo hay una instancia de este ShardListener por fragmento de ObjectGrid.

4. **Inicio del servidor:**

   El fragmento siguiente inicia el servidor ObjectGrid, que aloja tanto el servicio de contenedor, como el servicio de catálogos. Como se puede ver, el único método que se necesita llamar para iniciar el servidor es obtener un "container" de la fábrica de beans. Así se simplifica la complejidad de la programación moviendo la mayoría de la lógica a la configuración de Spring.

```
public class ShardServer extends TestCase
{
    Container container;
    org.springframework.beans.factory.BeanFactory bf;

    public void startServer(String cep)
    {
        try
        {
            bf = new org.springframework.context.support.ClassPathXmlApplicationContext(
                "/com/ibm/ws/objectgrid/test/springshard/Grid_spring.xml", ShardServer.class);
            container = (Container)bf.getBean("container");
        }
        catch (Exception e) {
            throw new ObjectGridRuntimeException("Cannot start OG container", e);
        }
    }

    public void stopServer()
    {
        if(container != null)
            container.teardown();
    }
}
```

Conceptos relacionados:

"Visión general de la infraestructura Spring" en la página 120
Spring es una infraestructura de desarrollo de aplicaciones Java. WebSphere eXtreme Scale proporciona soporte para permitir a Spring gestionar transacciones y configurar los clientes y servidores que conforman una cuadrícula de datos en memoria desplegada.

"Beans de ampliación de Spring y soporte de espacio de nombres" en la página 425
WebSphere eXtreme Scale proporciona una característica para declarar objetos POJO (Plain Old Java Object) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml` y un método para denominar los beans y, a continuación, especificar el nombre de la clase. Normalmente, se crean las instancias de la clase especificada y estos objetos se utilizan como los plug-ins. Ahora, eXtreme Scale puede delegar en Spring para obtener las instancias de estos objetos de plug-in. Si una aplicación utiliza Spring en general será necesario que los POJO se conecten al resto de la aplicación.

Referencia relacionada:

"Beans de ampliación gestionados de Spring" en la página 423
Puede declarar POJO (Plain Old Java Objects) para utilizarlos como puntos de ampliación en el archivo `objectgrid.xml`. Si denomina los beans y luego especifica el nombre de clase, eXtreme Scale suele crear instancias de la clase especificada y utiliza esas instancias como plug-in. Ahora, WebSphere eXtreme Scale ObjectGrid puede delegar en Spring para actuar como la fábrica de beans para obtener instancias de estos objetos de plug-in.

Archivo XML de descriptor Spring
Utilice un archivo XML de descriptor Spring para configurar e integrar eXtreme Scale con Spring.

Archivo Spring `objectgrid.xsd`
Utilice el archivo Spring `objectgrid.xsd` para integrar eXtreme Scale con Spring para gestionar las transacciones eXtreme Scale y configurar clientes y servidores.

Configuración de clientes en la infraestructura Spring

Puede sustituir los valores de ObjectGrid del lado del cliente con la infraestructura Spring

Acerca de esta tarea

. El siguiente archivo XML de ejemplo muestra cómo crear un elemento ObjectGridConfiguration y utilizarlo para alterar temporalmente algunos valores del lado del cliente. Puede crear una configuración similar utilizando configuración programática o configurando el archivo XML de descriptor de ObjectGrid.

Procedimiento

1. Cree un archivo XML para configurar clientes con la infraestructura Spring.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
  "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="companyGrid" factory-bean="manager" factory-method="getObjectGrid"
    singleton="true">
    <constructor-arg type="com.ibm.websphere.objectgrid.ClientClusterContext">
      <ref bean="client" />
    </constructor-arg>
    <constructor-arg type="java.lang.String" value="CompanyGrid" />
</bean>

  <bean id="manager" class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
    factory-method="getObjectGridManager" singleton="true">
```

```
    <property name="overrideObjectGridConfigurations">
      <map>
        <entry key="DefaultDomain">
          <list>
            <ref bean="ogConfig" />
          </list>
        </entry>
      </map>
    </property>
  </bean>

  <bean id="ogConfig"
    class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
    factory-method="createObjectGridConfiguration">
    <constructor-arg type="java.lang.String">
      <value>CompanyGrid</value>
    </constructor-arg>
    <property name="plugins">
      <list>
   <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
         factory-method="createPlugin">
         <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
           value="TRANSACTION_CALLBACK" />
         <constructor-arg type="java.lang.String"
           value="com.company.MyClientTxCallback" />
       </bean>
          <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
         factory-method="createPlugin">
         <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
           value="OBJECTGRID_EVENT_LISTENER" />
         <constructor-arg type="java.lang.String" value="" />
       </bean>
      </list>
 </property>
      <property name="backingMapConfigurations">
        <list>
   <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
           factory-method="createBackingMapConfiguration">
           <constructor-arg type="java.lang.String" value="Customer" />
             <property name="plugins">
   <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
                 factory-method="createPlugin">
                 <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
     value="EVICTOR" />
<constructor-arg type="java.lang.String"
                   value="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
               </bean>
             </property>
             <property name="numberOfBuckets" value="1429" />
           </bean>
              <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
             factory-method="createBackingMapConfiguration">
               <constructor-arg type="java.lang.String" value="OrderLine" />
                 <property name="numberOfBuckets" value="701" />
   <property name="timeToLive" value="800" />
   <property name="ttlEvictorType">
                  <value type="com.ibm.websphere.objectgrid.
          TTLType">LAST_ACCESS_TIME</value>
   </property>
           </bean>
         </list>
       </property>
     </bean>

     <bean id="client" factory-bean="manager" factory-method="connect"
       singleton="true">
       <constructor-arg type="java.lang.String">
     <value>localhost:2809</value>
       </constructor-arg>
   <constructor-arg
         type="com.ibm.websphere.objectgrid.security.
     config.ClientSecurityConfiguration">
         <null />
       </constructor-arg>
   <constructor-arg type="java.net.URL">
     <null />
       </constructor-arg>
     </bean>
</beans>
```

2. Cargue el archivo XML que ha creado y cree el ObjectGrid.

```
BeanFactory beanFactory = new XmlBeanFactory(newUrlResource
  ("file:test/companyGridSpring.xml"));
ObjectGrid companyGrid = (ObjectGrid) beanFactory.getBean("companyGrid");
```

Lea sobre “Visión general de la infraestructura Spring” en la página 120 para obtener más información sobre la creación de un archivo de descriptor XML.

Capítulo 6. Ajuste del rendimiento

Puede ajustar los valores de su entorno para aumentar el rendimiento global de su entorno de WebSphere eXtreme Scale.

Ajuste del agente de dimensionamiento de memoria caché para obtener estimaciones precisas del consumo de memoria

WebSphere eXtreme Scale da soporte al dimensionamiento del consumo de memoria de instancias de BackingMap en cuadrículas de datos distribuidas. No se da soporte al dimensionamiento de consumo de memoria para instancias de cuadrícula de datos locales. El valor que notifica WebSphere eXtreme Scale para una cuadrícula determinada se aproxima mucho al valor notificado por los análisis de volcado de almacenamiento dinámico. Si el objeto de correlación es complejo, es posible que los dimensionamientos sean menos precisos. Se visualiza el mensaje CWOBJ4543 en el registro para cada objeto de entrada de memoria caché que no se pueda dimensionar de forma precisa debido a que es demasiado complejo. Puede obtener una medición más precisa evitando la complejidad innecesaria de la correlación.

Procedimiento

- Habilite el agente de dimensionamiento.

  Si utiliza una máquina virtual Java (JVM) 5 o posterior, utilice el agente de dimensionamiento. Si utiliza el agente de dimensionamiento, WebSphere eXtreme Scale puede obtener información adicional de la JVM para mejorar sus estimaciones. Se puede cargar el agente añadiendo el argumento siguiente a la línea de mandatos de la JVM:

  ```
  -javaagent:directorio lib de WXS/wxssizeagent.jar
  ```

  Para una topología incorporada, añada el argumento a la línea de mandatos del proceso de WebSphere Application Server.

  Para una topología distribuida, añada el argumento a la línea de mandatos de los procesos (contenedores) de eXtreme Scale y al proceso de WebSphere Application Server.

  Cuando se carga correctamente, se graba el mensaje siguiente en el archivo `SystemOut.log`.

  ```
  CWOBJ4541I: Se ha habilitado el dimensionamiento de memoria BackingMap mejorada.
  ```

- Elija tipos de datos Java en lugar de tipos de datos personalizados, donde sea posible.

  WebSphere eXtreme Scale dimensiona con precisión el coste de memoria de los tipos siguientes:

  - java.lang.String y matrices donde String es la clase de componente (String[])
  - Todos los tipos de derivador primitivos (Byte, Short, Character, Boolean, Long, Double, Float, Integer) y las matrices donde los derivadores primitivos son el tipo de componente (por ejemplo, Integer[], Character[])
  - java.math.BigDecimal y java.math.BigInteger y las matrices donde estas dos clases son el tipo de componente (BigInteger[] y BigDecimal[])
  - Tipos temporales (java.util.Date, java.sql.Date, java.util.Time, java.sql.Timestamp)

  - java.util.Calendar y java.util.GregorianCalendar
- Evite internación de objetos, cuando sea posible.

  Cuando se inserta un objeto en una correlación, WebSphere eXtreme Scale asume que aloja la única referencia al objeto y todos los objetos a los que el objeto se refiere directamente. Si inserta 1000 objetos personalizados en una correlación y cada uno de ellos tiene una referencia a la misma instancia de serie, WebSphere eXtreme Scale dimensiona esa instancia de serie 1000 veces, sobrestimando el tamaño real de la correlación en el almacenamiento dinámico. Sin embargo, WebSphere eXtreme Scale compensa correctamente en los escenarios de internación común siguientes:
  - Referencias a las enumeraciones de Java 5
  - Referencias a las clases que siguen el patrón de enumeración de Typesafe. Las clases que siguen este patrón solo tienen definidos constructores privados, tienen como mínimo un campo final estático privado de su propio tipo y si implementan Serializable, la clase implementa el método readResolve().
  - Internación del derivador primitivo de Java 5. Por ejemplo, utilizando Integer.valueOf(1) en lugar de nuevo Integer(1)

  Si debe utilizar internación, utilice una de las técnicas siguientes para obtener estimaciones más precisas.
- Utilice los tipos personalizados cuidadosamente.

  Cuando utilice tipos personalizados, elija tipos de datos primitivos para los campos antes que tipos de objeto.

  Además, consulte los tipos de objeto enumerados en la entrada 2 en sus propias implementaciones personalizadas.

  Cuando utilice los tipos personalizados, conserve el árbol de objeto en un nivel. Cuando inserte un objeto personalizado en un correlación, WebSphere eXtreme Scale solo calculará el coste del objeto insertado, que incluye los campos primitivos y todos los objetos a los que hace referencia directamente. WebSphere eXtreme Scale no seguirá las referencias más abajo en el árbol de objeto. Si inserta un objeto en la correlación, y WebSphere eXtreme Scale detecta que no se han seguido las referencias durante el proceso de dimensionamiento, recibirá un mensaje con el código CWOBJ4543 que incluirá el nombre de la clase que no se ha dimensionado completamente. Cuando se produzca este error, trate las estadísticas sobre el tamaño de la correlación como datos de tendencias, en lugar de confiar en las estadísticas de tamaño como un total preciso.
- Utilice la modalidad de copia CopyMode.COPY_TO_BYTES, si es posible.

  Utilice la modalidad de copia CopyMode.COPY_TO_BYTES para eliminar cualquier duda resultante de dimensionar el valor Objects que se está insertando en la correlación, incluso si el árbol Object tiene demasiados niveles para que se dimensione normalmente (lo que resulta en el mensaje CWOBJ4543).

Conceptos relacionados:

"Dimensionamiento del consumo de memoria caché"
WebSphere eXtreme Scale puede estimar con precisión el uso de memoria de almacenamiento dinámico de Java de un BackingMap determinado en bytes. Aproveche esta posibilidad para ayudar a dimensionar correctamente los valores de almacenamiento dinámico de la máquina virtual Java y las políticas de desalojo. El comportamiento de esta característica varía con la complejidad de los objetos colocados en la correlación de respaldo y con el modo en que se configura la correlación. Actualmente, esta característica está soportada solo para cuadrículas de datos distribuidas. Las instancias de cuadrículas de datos locales no dan soporte al dimensionamiento de bytes utilizados.

Dimensionamiento del consumo de memoria caché

WebSphere eXtreme Scale puede estimar con precisión el uso de memoria de almacenamiento dinámico de Java de un BackingMap determinado en bytes. Aproveche esta posibilidad para ayudar a dimensionar correctamente los valores de almacenamiento dinámico de la máquina virtual Java y las políticas de desalojo. El comportamiento de esta característica varía con la complejidad de los objetos colocados en la correlación de respaldo y con el modo en que se configura la correlación. Actualmente, esta característica está soportada solo para cuadrículas de datos distribuidas. Las instancias de cuadrículas de datos locales no dan soporte al dimensionamiento de bytes utilizados.

Consideraciones sobre el consumo del almacenamiento dinámico

eXtreme Scale almacena todos sus datos en el espacio de almacenamiento dinámico de los procesos de JVM que componen la cuadrícula de datos. Para una correlación determinada, el espacio de almacenamiento dinámico que consume se puede dividir en los componentes siguientes:

- El tamaño de todos los objetos clave que están actualmente en la correlación
- El tamaño de todos los objetos de valores que están actualmente en la correlación
- El tamaño de todos los objetos EvictorData que están siendo utilizados por los plug-ins Evictor de la correlación
- La sobrecarga de la estructura de datos subyacente

El número de bytes utilizados notificado por las estadísticas de tamaño es la suma de estos cuatro componentes. Estos valores se calculan por cada entrada en las operaciones de insertar, actualizar y eliminar correlación, lo que significa que eXtreme Scale siempre tiene un valor actual para el número de bytes que consume una correlación de respaldo determinada.

Cuando se particionan las cuadrículas de datos, cada partición contiene una parte de la correlación de respaldo. Dado que las estadísticas de tamaño se calculan en el nivel bas bajo del código de eXtreme Scale, cada partición de una correlación de respaldo realiza un seguimiento de su propio tamaño. Puede utilizar las API de estadísticas de eXtreme Scale para realizar un seguimiento del tamaño acumulativo de la correlación , así como del tamaño de sus particiones individuales.

En general, utilice los datos de tamaño como medida de las tendencias de los datos a lo largo del tiempo, no como una medida precisa del espacio de almacenamiento dinámico que está utilizando la correlación. Por ejemplo, si el tamaño notificado de una correlación se hace el doble de 5 MB a 10 MB, vea el consumo de memoria de

la correlación como que se ha duplicado. La medida actual de 10 MB podría ser imprecisa por diversas razones. Si tiene en cuenta las razones y sigue los métodos recomendados, la precisión de las mediciones de tamaño se aproxima a la del proceso posterior de un volcado de almacenamiento dinámico Java.

El problema principal con la precisión es que el modelo de memoria Java no es lo suficientemente restrictivo para permitir mediciones de memoria que es seguro que son precisas. El problema fundamental es que un objeto puede estar activo en el almacenamiento dinámico debido a varias referencias. Por ejemplo, si se inserta la misma instancia de objeto de 5 KB en tres correlaciones distintas, cualquiera de estas tres correlaciones impide que el objeto sea objeto de la recogida de basura. En esta situación, cualquiera de las mediciones siguientes sería justificable:

- El tamaño de cada correlación aumenta en 5 KB.
- El tamaño de la primera correlación en la que se coloca el Objeto aumenta en 5 KB.
- El tamaño de las otras dos correlaciones no aumenta. El tamaño de cada correlación aumenta en una fracción del tamaño del objeto.

Esta ambigüedad es por lo que estas medidas se deben considerar datos de tendencia, a menos que haya eliminado la ambigüedad mediante opciones de diseño, métodos recomendados y la comprensión de las opciones de implementación que pueden proporcionar estadísticas más precisas.

eXtreme Scale asume que una correlación determinada mantiene la única referencia de larga duración a los objetos clave y valor que contiene. Si el mismo objeto de 5 KB se coloca en tres correlaciones, el tamaño de cada correlación aumenta en 5 KB. El aumento no suele ser un problema, porque la característica solo está soportada para cuadrículas de datos distribuidas. Si inserta el mismo Objeto en tres correlaciones distintas en un cliente remoto, cada correlación recibe su propia copia del Objeto. Los valores de COPY MODE transaccionales predeterminados suelen garantizar también que cada correlación tiene su propia copia de un objeto determinado.

Internación de objetos

La internación de objetos puede presentar un reto al estimar el uso de memoria de almacenamiento dinámico. Al implementar la internación de objetos, el código de la aplicación garantiza deliberadamente que todas las referencias a un valor de objeto determinado apunten realmente a la misma instancia de objeto en el almacenamiento dinámico y, por lo tanto, la misma ubicación en la memoria. Un ejemplo de esto podría ser la clase siguiente:

```
public class ShippingOrder implements Serializeable,Cloneable{

    public static final STATE_NEW = "new";
    public static final STATE_PROCESSING = "processing";
    public static final STATE_SHIPPED = "shipped";

    private String state;
    private int orderNumber;
private int customerNumber;

public Object clone(){
       ShippingOrder toReturn = new ShippingOrder();
       toReturn.state = this.state;
       toReturn.orderNumber = this.orderNumber;
       toReturn.customerNumber = this.customerNumber;
       return toReturn;
    }
```

```
    private void readResolve(){
        if (this.state.equalsIgnoreCase("new")
            this.state = STATE_NEW;
        else if (this.state.equalsIgnoreCase("processing")
            this.state = STATE_PROCESSING;
        else if (this.state.equalsIgnoreCase("shipped")
            this.state = STATE_SHIPPED:
    }
}
```

La internación de objetos causa una sobrestimación de las estadísticas de tamaño porque eXtreme Scale supone que los objetos utilizan distintas ubicaciones de memoria. Si existe un millón de objetos ShippingOrder, las estadísticas de tamaño visualizan el coste de un millón de series que contienen la información de estado. En realidad, solo existen tres series que son miembros de clase estática. El coste de memoria de los miembros de clase estática nunca se debe añadir a ninguna correlación eXtreme Scale. Sin embargo, esta situación no se puede detectar durante el tiempo de ejecución. Existen docenas de formas en las que se puede implementar internación de objetos similar, y por esto es tan difícil de detectar. No es práctico para eXtreme Scale protegerse frente a todas las implementaciones posibles. Sin embargo, eXtreme Scale se protege frente a los tipos de internación de objetos utilizados más habitualmente. Para optimizar el uso de memoria con la internación de objetos, implemente la internación solo en objetos personalizados que se encuentren en las dos categorías siguientes para ampliar la precisión de las estadísticas de consumo de memoria:

- eXtreme Scale se ajusta automáticamente para las enumeraciones Java 5 y el patrón de enumeración Typesafe, tal como se describe en el documento Java 2 Platform Standard Edition 5.0 Overview: Enums.
- eXtreme Scale da cuenta automáticamente de la internación automática de tipos de derivador primitivos como, por ejemplo, un entero. La internación automática de tipos de derivador primitivos se introdujo en Java 5 mediante la utilización de los métodos valueOf estáticos.

Estadísticas de consumo de memoria

Utilice uno de estos métodos para acceder a las estadísticas de consumo de memoria.

API de estadísticas

Utilice el método MapStatsModule.getUsedBytes(), que proporciona estadísticas para una única correlación, incluido el número de entradas y la proporción de coincidencias.

Si desea detalles, consulte Módulos de estadísticas.

Beans gestionados (MBeans)

Utilice la estadística de MBean gestionado MapUsedBytes. Puede utilizar varios tipos distintos de MBeans JMX (Java Management Extensions) para administrar y supervisar despliegues. Cada MBean hace referencia a una entidad específica como, por ejemplo, una correlación, eXtreme Scale, servidor, grupo de réplicas o miembro del grupo de réplicas.

Si desea detalles, consulte Administración con beans gestionados (MBeans).

Módulos PMI (Performance Monitoring Infrastructure)

Puede supervisar el rendimiento de las aplicaciones con los módulos PMI. Especialmente, utilice el módulo PMI de correlación para los contenedores incorporados en WebSphere Application Server.

Si desea detalles, consulte Módulos PMI.

Consola de WebSphere eXtreme Scale

Con la consola, puede visualizar las estadísticas de consumo de memoria. Consulte Supervisión con la consola web.

Todos estos métodos acceden a la misma medición subyacente del consumo de memoria de una instancia de BaseMap determinada. El tiempo de ejecución de WebSphere eXtreme Scale intenta con un mejor esfuerzo calcular el número de bytes de memoria de almacenamiento dinámico que consumen los objetos de clave y valor almacenados en la correlación, así como la sobrecarga de la propia correlación. Puede ver cuánta memoria de almacenamiento dinámico consume cada correlación en toda la cuadrícula de datos distribuida.

En la mayoría de los casos, el valor notificado por WebSphere eXtreme Scale para una correlación determinada está muy próximo al valor notificado por el análisis de volcado de almacenamiento dinámico. WebSphere eXtreme Scale dimensiona de forma precisa su propia sobrecarga, pero no puede dar cuenta de todos los objetos posibles que podrían colocarse en una correlación. Si se siguen los métodos recomendados descritos en "Ajuste del agente de dimensionamiento de memoria caché para obtener estimaciones precisas del consumo de memoria" en la página 435 se podría mejorar la precisión del tamaño de las mediciones en bytes proporcionadas por WebSphere eXtreme Scale.

Tareas relacionadas:

"Ajuste del agente de dimensionamiento de memoria caché para obtener estimaciones precisas del consumo de memoria" en la página 435
WebSphere eXtreme Scale da soporte al dimensionamiento del consumo de memoria de instancias de BackingMap en cuadrículas de datos distribuidas. No se da soporte al dimensionamiento de consumo de memoria para instancias de cuadrícula de datos locales. El valor que notifica WebSphere eXtreme Scale para una cuadrícula determinada se aproxima mucho al valor notificado por los análisis de volcado de almacenamiento dinámico. Si el objeto de correlación es complejo, es posible que los dimensionamientos sean menos precisos. Se visualiza el mensaje CWOBJ4543 en el registro para cada objeto de entrada de memoria caché que no se pueda dimensionar de forma precisa debido a que es demasiado complejo. Puede obtener una medición más precisa evitando la complejidad innecesaria de la correlación.

Ajuste y rendimiento para el desarrollo de aplicaciones

Para mejorar el rendimiento de la cuadrícula de datos en memoria o del espacio de proceso de la base de datos, puede investigar varias consideraciones como, por ejemplo, utilizar los procedimientos recomendados para las características del producto como el bloqueo, la serialización y el rendimiento de la consulta.

Ajuste de la modalidad de copia

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

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

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

CopyMode para correlaciones de entidad

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

COPY_ON_READ_AND_COMMIT

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

COPY_ON_READ

La modalidad COPY_ON_READ mejora el rendimiento en comparación con la modalidad COPY_ON_READ_AND_COMMIT al eliminar la copia que se produce cuando se confirma una transacción. El argumento valueInterfaceClass se pasa por alto cuando se utiliza esta modalidad. Para conservar la integridad de los datos de BackingMap, la aplicación garantiza que todas las referencias que tiene de una entrada se destruyan una vez confirmada la transacción. Con esta modalidad, el método ObjectMap.get devuelve una copia del valor en lugar de una referencia al valor para garantizar que los cambios realizados por la aplicación en el valor no afecten al valor de BackingMap hasta que se confirme la transacción. No obstante, cuando se confirma, no se realiza una copia de los cambios. En su lugar, se almacena en BackingMap la referencia a la copia devuelta por el método ObjectMap.get. La aplicación destruye todas las referencias de la entrada de

correlación una vez que se confirma la transacción. Si la aplicación no las destruye, la aplicación podría dañar los datos almacenados en memoria caché de BackingMap. Si una aplicación que utiliza esta modalidad experimenta problemas, cambie a la modalidad COPY_ON_READ_AND_COMMIT para ver si se sigue produciendo el problema. Si desaparece, significa que la aplicación no está destruyendo todas las referencias después de la confirmación de la transacción.

COPY_ON_WRITE

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

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

```
class Person {
    String name;
    int age;
    public Person() {
    }
    public void setName(String n)  {
        name = n;
    }
    public String getName() {
```

```
        return name;
    }
    public void setAge(int a) {
        age = a;
    }
    public int getAge() {
        return age;
    }
}
```

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

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

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

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

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

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

Puede difundir el proxy para dos tipos. La última parte del fragmento de código anterior muestra lo que no se permite en la modalidad COPY_ON_WRITE. La

aplicación recupera el registro Bobby e intenta difundir el registro a un objeto Person. Esta acción produce una excepción de difusión de clase porque el proxy devuelto no es un objeto Person. El proxy devuelto implementa el objeto IPerson y ValueProxyInfo.

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

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

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

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

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

Otra consideración a tener en cuenta es cuando debe alterarse temporalmente el método equals. Como se ilustra en el fragmento de código siguiente, el método equals debe verificar que el argumento es un objeto que implementa la interfaz IPerson y difunde el argumento para ser IPerson. Como el argumento puede ser un proxy que implementa la interfaz IPerson, debe usar los métodos getAge y getName al comparar la igualdad de las variables de instancia.

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

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

NO_COPY

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

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

COPY_TO_BYTES

Puede almacenar los objetos en un formato serializado, en lugar del formato POJO. Mediante el uso del valor COPY_TO_BYTES, puede reducir la huella de la memoria que puede consumir un gráfico grande de objetos. Consulte “Mejora del rendimiento con correlaciones de matriz de bytes” en la página 446 si desea información adicional.

COPY_TO_BYTES_RAW

7.1.1+ Con COPY_TO_BYTES_RAW, puede acceder directamente el formato serializado de los datos. Esta modalidad de copia ofrece una manera eficaz de interactuar con bytes serializados, lo que le permite evitar el proceso de deserialización para acceder a objetos en la memoria.

En el archivo XML de descriptor de ObjectGrid, puede establecer la modalidad de copia COPY_TO_BYTES, y establecer programáticamente la modalidad de copia a COPY_TO_BYTES_RAW en las instancias donde desea acceder a los datos serializados en bruto. Establezca la modalidad de copia en COPY_TO_BYTES _RAW en el archivo XML de descriptor de ObjectGrid sólo cuando la aplicación utilice el formato de datos en bruto como parte de una aplicación de proceso principal.

Uso incorrecto de CopyMode

Los errores se producen cuando la aplicación intenta mejorar el rendimiento al usar las modalidades de copia COPY_ON_READ, COPY_ON_WRITE o NO_COPY,

como se ha descrito anteriormente. Los errores intermitentes no se producen al cambiar la modalidad de copia a la modalidad COPY_ON_READ_AND_COMMIT.

Problema

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

Solución

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

Referencia relacionada:

Archivo XML de descriptor ObjectGrid
Para configurar WebSphere eXtreme Scale, utilice el archivo XML de descriptor de ObjectGrid y la API ObjectGrid.

Mejora del rendimiento con correlaciones de matriz de bytes

Puede almacenar valores en las correlaciones en una matriz de bytes en lugar de hacerlo en formato POJO, lo que reduce la huella en la memoria que puede consumir un gráfico grande de objetos.

Ventajas

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

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

Si se utilizan las matrices de bytes, tenga en cuenta que tener un mecanismo de serialización optimizado es vital para ver una reducción en el consumo de memoria. Para obtener más información, consulte “Ajuste del rendimiento de serialización” en la página 453.

Configuración de correlaciones de matrices de bytes

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

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

Consideraciones

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

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

Tipo de objeto

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

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

Acceso local

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

Interacciones de plug-in

Con las correlaciones de matrices de bytes, los objetos no se inflan cuando se establece una comunicación entre un cliente y un servidor, a menos que el servidor

necesite un formato POJO. Los plug-ins que interactúan con el valor de correlación experimentará una reducción en el rendimiento debido a la necesidad de inflar el valor.

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

Índices y consultas

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

Bloqueo optimista

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

Cargador

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

ObjectGridEventListener

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

Referencia relacionada:

Archivo XML de descriptor ObjectGrid
Para configurar WebSphere eXtreme Scale, utilice el archivo XML de descriptor de ObjectGrid y la API ObjectGrid.

Ajuste de operaciones de copia con la interfaz ObjectTransformer

La interfaz ObjectTransformer utiliza devoluciones de llamadas a la aplicación para proporcionar implementaciones personalizadas de operaciones comunes y costosas, como la serialización y la copia exacta de objetos.

 La interfaz ObjectTransformer ha sido sustituida por los plug-ins DataSerializer, que puede utilizar para almacenar eficientemente datos arbitrarios en WebSphere eXtreme Scale de modo que las API existentes del producto puedan interactuar eficazmente con los datos.

Visión general

Siempre se realizan copias de los valores excepto cuando se utiliza la modalidad NO_COPY. El mecanismo de copia predeterminado que se emplea en eXtreme Scale es la serialización, que se sabe que es una operación costosa. La interfaz ObjectTransformer se utiliza en esta situación. La interfaz ObjectTransformer utiliza las devoluciones de llamada a la aplicación para proporcionar una implementación personalizada de las operaciones comunes y costosas como, por ejemplo, la serialización de objeto y la copia exacta de objetos.

Una aplicación puede proporcionar una implementación de la interfaz ObjectTransformer en una correlación y, a continuación, eXtreme Scale delega en los métodos de este objeto y se basa en la aplicación para proporcionar una versión optimizada de cada método de la interfaz. La interfaz ObjectTransformer actúa del modo siguiente:

```
public interface ObjectTransformer {
    void serializeKey(Object key, ObjectOutputStream stream) throws IOException;
    void serializeValue(Object value, ObjectOutputStream stream) throws IOException;
    Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object copyValue(Object value);
    Object copyKey(Object key);
}
```

Puede asociar una interfaz ObjectTransformer con una BackingMap utilizando el siguiente código de ejemplo:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ajuste de operaciones de copia exacta

Después de que una aplicación reciba un objeto de un ObjectMap, eXtreme Scale realiza una copia exacta del valor de objeto para garantizar que la copia de la correlación BaseMap mantiene la integridad de datos. La aplicación puede entonces modificar el valor de objeto sin riesgos. Cuando se confirma la transacción, se actualiza la copia del valor de objeto de la correlación BaseMap al nuevo valor modificado y se detiene la aplicación, que utilizará el valor de ahí en adelante. Podría haber vuelto a copiar el objeto en la fase de confirmación para realizar una copia privada. Sin embargo, en este caso, el coste del rendimiento de esta acción se ha compensado indicando al programador de aplicaciones que no

utilice el valor después de que se confirme la transacción. El ObjectTransformer predeterminado intenta utilizar un clon o un par de métodos serialize e inflate para generar una copia. El par de métodos serialize e inflate es el peor caso de rendimiento. Si la creación de perfiles revela que usar los métodos serialize e inflate es un problema para la aplicación, escriba un método clone apropiado para crear una copia exacta. Si no puede alterar la clase, cree un plug-in ObjectTransformer personalizado e implemente métodos copyValue y copyKey más eficaces.

Ajuste de desalojadores

Si utiliza desalojadores de plug-in, éstos no se activarán hasta que los cree y los asocie con una correlación de respaldo. Los siguiente métodos recomendados aumentan el rendimiento de desalojadores LFU (utilizados con menor frecuencia) y desalojadores LRU (menos utilizados recientemente).

Desalojador LFU (utilizados con menor frecuencia)

El concepto de un desalojador LFU es eliminar entradas de la correlación que menos se utilizan. Las entradas de la correlación se expanden en un volumen establecido de almacenamientos dinámicos. A medida que aumenta el uso de una entrada en memoria caché determinada, se ordena en una posición más alta en el almacenamiento dinámico. Cuando el desalojador intenta un conjunto de desalojos, elimina sólo las entradas de la memoria caché ubicadas por debajo de un punto específico del almacenamiento dinámico binario. Como resultado, se desalojan las entradas usadas con menor frecuencia.

Desalojador LRU (menos utilizados recientemente)

El desalojador LRU sigue los mismos conceptos que el desalojador LFU con unas pocas diferencias. La diferencia principal es que el LRU utiliza una cola FIFO (primero en entrar, primero en salir) en lugar de un conjunto de almacenamientos dinámicos binarios. Cada vez que se accede a una entrada de la memoria caché, ésta se mueve a la cabeza de la cola. En consecuencia, la parte inicial de la cola contiene las entradas de correlación utilizadas más recientemente y la parte final empieza con las entradas de correlación menos utilizadas recientemente. Por ejemplo, la entrada de la memoria caché A se utiliza 50 veces, y la entrada de la memoria caché B se utiliza sólo una después de la entrada de la memoria caché A. En esta situación, la entrada de la memoria caché B se coloca en la parte delantera de la cola porque se ha utilizado recientemente, mientras que la entrada de la memoria caché A se coloca al final de la cola. El desalojador LRU desaloja las entradas de la memoria caché que están en la parte final de la cola, que son las entradas de correlación menos usadas recientemente.

Propiedades de LFU y LRU y procedimientos recomendados para mejorar el rendimiento

Número de almacenamientos dinámicos

Al utilizar el desalojador LFU, todas las entradas de la memoria caché de una correlación determinada se ordenan según el número de almacenamientos dinámicos que se especifique, lo que mejora drásticamente el rendimiento e impide que se sincronicen los desalojos en un almacenamiento dinámico binario que contenga todas las ordenaciones de la correlación. A mayor número de almacenamientos dinámicos menor es el tiempo necesario para reordenarlos porque cada uno de ellos tiene menos entradas. Establezca el número de

almacenamientos dinámicos en 10% del número de entradas en BaseMap.

Número de colas

Al utilizar el desalojador LRU, todas las entradas de la memoria caché de una correlación determinada se ordenan según el número de colas LRU que se especifique, lo que mejora drásticamente el rendimiento e impide que se sincronicen los desalojos en una cola que contenga todas las ordenaciones de la correlación. Establezca el número de colas en 10% del número de entradas en BaseMap.

Propiedad MaxSize

Cuando un desalojador LFU o LRU empieza a desalojar entradas, utiliza la propiedad de desalojador MaxSize para determinar cuántos almacenamientos dinámicos binarios o elementos de cola LRU va a desalojar. Por ejemplo, presuponga que establece el número de almacenamientos dinámicos o colas en unas 10 entradas de correlación en cada cola de correlación. Si la propiedad MaxSize se establece en 7, el desalojador desaloja 3 entradas de cada almacenamiento dinámico u objeto de cola para que el tamaño del almacenamiento dinámico o de la cola se reduzca a 7. El desalojador sólo desaloja entradas de correlación de un almacenamiento dinámico o cola cuando en éstos hay un valor mayor que el de la propiedad MaxSize. Establezca MaxSize en 70% del tamaño de la cola o almacenamiento dinámico. En este ejemplo, el valor se estableció en 7. Puede obtener un tamaño aproximado de cada almacenamiento dinámico o cola si divide el número de entradas BaseMap entre el número de almacenamientos dinámicos o colas utilizado.

Propiedad SleepTime

Un desalojador no elimina constantemente entradas de la correlación. Está inactivo durante un tiempo determinado, y sólo comprueba la correlación cada n número de segundos, donde n equivale a la propiedad SleepTime. Esta propiedad afecta positivamente al rendimiento: ejecutar un barrido de desalojo suele disminuir el rendimiento debido a los recursos que se necesitan para procesarlos. Sin embargo, no utilizar el desalojador con frecuencia puede generar una correlación que tiene entradas que no son necesarias. Una correlación con muchas entradas que no necesita puede afectar negativamente a los requisitos de memoria y al proceso de los recursos de la correlación. Para la mayoría de las correlaciones se recomienda establecer el intervalo de barrido de desalojo en 15 segundos. Si la correlación se graba con frecuencia y se utiliza a una velocidad de transacción alta, conviene establecer el tiempo en un valor más bajo. Si se accede a la correlación con poca frecuencia, establezca el tiempo en un valor más alto.

Ejemplo

El ejemplo siguiente define una correlación, crea un desalojador LFU nuevo, establece las propiedades del desalojador y establece la correlación que va a utilizar el desalojador:

```
//Usar
ObjectGridManager para crear/obtener ObjectGrid. Consulte el
// apartado ObjectGridManger
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");

//Establecer propiedades presuponiendo 50.000 entradas de correlación
LFUEvictor someEvictor = new LFUEvictor();
```

```
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Usar el desalojador LRU es muy parecido a usar un desalojador LFU. A continuación se muestra un ejemplo:

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");

//Establecer propiedades presuponiendo 50.000 entradas de correlación
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Observe que sólo hay dos líneas distintas al del ejemplo del desalojador LFU.

Tareas relacionadas:

Habilitación de desalojadores mediante programación
Los desalojadores están asociados con instancias de BackingMap.

Habilitación de desalojadores con configuración XML
En lugar de utilizar la interfaz BackingMap para establecer mediante programación los atributos de BackingMap que pueden ser utilizados por el desalojador TTL, puede utilizar un archivo XML para configurar cada instancia de BackingMap. El siguiente código demuestra cómo establecer estos atributos para tres correlaciones de BackingMap distintas:

Referencia relacionada:

Archivo XML de descriptor ObjectGrid
Para configurar WebSphere eXtreme Scale, utilice el archivo XML de descriptor de ObjectGrid y la API ObjectGrid.

Ajuste del rendimiento de bloqueo

Los valores de las estrategias de bloqueo y del aislamiento de transacciones afectan el rendimiento de las aplicaciones.

Recuperar una instancia almacenada en memoria caché

Para obtener más información, consulte “Gestor de bloqueo” en la página 237:

Estrategia de bloqueo pesimista

Utilice la estrategia de bloqueo pesimista en operaciones de correlación de lectura y grabación donde las claves suelen colisionar. La estrategia de bloqueo pesimista tiene un gran impacto sobre el rendimiento.

Aislamiento de transacciones de lectura confirmada y no confirmada

Si utiliza la estrategia de bloqueo pesimista, establezca el nivel de aislamiento de transacción a través del método Session.setTransactionIsolation. Para el aislamiento confirmado de lectura o no confirmado de lectura, utilice los argumentos Session.TRANSACTION_READ_COMMITTED o Session.TRANSACTION_READ_UNCOMMITTED en función del aislamiento. Para restablecer el nivel de aislamiento en el comportamiento de bloqueo pesimista predeterminado, utilice el método Session.setTransactionIsolation con el argumento Session.REPEATABLE_READ.

El aislamiento de lectura confirmada reduce la duración de los bloqueos compartidos, que pueden mejorar la simultaneidad y reducir la probabilidad de puntos muertos. Este nivel de aislamiento debe utilizarse cuando una transacción no necesita la constatación de que los valores de lectura permanecen sin modificar durante la transacción.

Utilice una lectura no confirmada cuando la transacción no necesita ver los datos confirmados.

Estrategia de bloqueo optimista

El bloqueo optimista es la configuración predeterminada. Esta estrategia mejora el rendimiento y la escalabilidad en comparación con la estrategia pesimista. Utilice esta estrategia cuando las aplicaciones puedan tolerar anomalías de actualización optimista y el rendimiento siga siendo mejor que el de la estrategia pesimista. Esta estrategia es excelente en operaciones de lectura y aplicaciones con actualizaciones poco frecuentes.

Plug-in OptimisticCallback

La estrategia de bloqueo optimista realiza una copia de las entradas de la memoria caché y las compara como sea preciso. Esta operación puede resultar costosa porque la copia de la entrada podría implicar la clonación o serialización. Para conseguir el rendimiento más rápido posible, implemente el plug-in personalizado para las correlaciones que no son de entidad.

Si desea más información, consulte . Consulte la información sobre el plug-in OptimisticCallback en *Visión general del producto* si desea más información.

Uso de campos de versión para entidades

Cuando utilice el bloqueo optimista con entidades, utilice la anotación @Version o el atributo equivalente en el archivo de descriptor de metadatos de entidad. La anotación de versión proporciona a ObjectGrid una forma muy eficiente de realizar el seguimiento de la versión de un objeto. Si la entidad no tiene un campo de versión y se utiliza un bloqueo optimista para la entidad, debe copiarse y compararse toda la entidad.

Estrategia de ningún bloqueo

Utilice esta estrategia en aplicaciones de sólo lectura. La estrategia de ningún bloqueo no obtiene ningún bloqueo ni usa un gestor de bloqueos. Por lo tanto, esta estrategia ofrece el rendimiento y la escalabilidad con más concurrencia.

Ajuste del rendimiento de serialización

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

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

 La interfaz ObjectTransformer ha sido sustituida por los plug-ins DataSerializer, que puede utilizar para almacenar eficientemente datos arbitrarios en WebSphere eXtreme Scale de modo que las API existentes del producto puedan interactuar eficazmente con los datos.

Escribir un ObjectTransformer para cada BackingMap

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

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

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

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

Uso de entidades

Cuando se utiliza la API EntityManager con entidades, ObjectGrid no almacena los objetos de entidad en los objetos BackingMap. La API EntityManager convierte el objeto de entidad en objetos Tuple. Las correlaciones de entidad se asocian automáticamente con un objeto ObjectTransformer altamente optimizado. Siempre que se utiliza la API ObjectMap o EntityManager para interactuar con correlaciones de entidad, se invoca a la entidad ObjectTransformer.

Serialización personalizada

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

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

Matrices de bytes

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

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

Un plug-in MapSerializerPlugin puede estar asociado con un plug-in BackingMap cuando se utilizan las modalidades de copia COPY_TO_BYTES o COPY_TO_BYTES_RAW. Esta asociación permite que los datos se almacenen en formato serializado en la memoria, en lugar de hacerlo con el formato de objeto Java nativo. El almacenamiento de datos serializados conserva la memoria y mejora la réplica y el rendimiento en el cliente y el servidor. Puede utilizar un plug-in DataSerializer para desarrollar secuencias de serialización de alto rendimiento que se pueden comprimir, cifrar, desarrollar y consultar.

Ajuste de la serialización

Los plug-ins DataSerializer exponen metadatos que indican a WebSphere eXtreme Scale qué atributos puede o no utilizar directamente durante la serialización, la vía de acceso a los datos que se serializarán y el tipo de datos que se almacena en memoria. Puede optimizar la serialización de objetos y el rendimiento de inflado de forma que pueda interactuar eficazmente con la matriz de bytes.

Visión general

 La interfaz ObjectTransformer ha sido sustituida por los plug-ins DataSerializer, que puede utilizar para almacenar eficientemente datos arbitrarios en WebSphere eXtreme Scale de modo que las API existentes del producto puedan interactuar eficazmente con los datos.

Siempre se realizan copias de los valores excepto cuando se utiliza la modalidad NO_COPY. El mecanismo de copia predeterminado que se emplea en eXtreme Scale es la serialización, que se sabe que es una operación costosa. La interfaz ObjectTransformer se utiliza en esta situación. La interfaz ObjectTransformer utiliza las devoluciones de llamada a la aplicación para proporcionar una implementación personalizada de las operaciones comunes y costosas como, por ejemplo, la serialización de objeto y la copia exacta de objetos. Sin embargo, para obtener un rendimiento mejorado en la mayoría de los casos, puede utilizar los plug-ins DataSerializer para serializar objetos. Debe utilizar las modalidades de copia COPY_TO_BYTES o COPY_TO_BYTES_RAW para explotar los plug-ins DataSerializer. Para obtener más información, consulte Serialización mediante plug-ins DataSerializer.

Una aplicación puede proporcionar una implementación de la interfaz ObjectTransformer en una correlación y, a continuación, eXtreme Scale delega en los métodos de este objeto y se basa en la aplicación para proporcionar una versión optimizada de cada método de la interfaz. La interfaz ObjectTransformer actúa del modo siguiente:

```
public interface ObjectTransformer {
    void serializeKey(Object key, ObjectOutputStream stream) throws IOException;
    void serializeValue(Object value, ObjectOutputStream stream) throws IOException;
    Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException;
    Object copyValue(Object value);
    Object copyKey(Object key);
}
```

Puede asociar una interfaz ObjectTransformer con una BackingMap utilizando el siguiente código de ejemplo:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ajuste de la serialización e inflado de objetos

Normalmente la serialización de objetos es la consideración de rendimiento más importante con eXtreme Scale, que utiliza el mecanismo serializable predeterminado, si la aplicación no proporciona un plug-in ObjectTransformer. Una aplicación puede proporcionar implementaciones de readObject y writeObject Serializable o tener objetos que implementen la interfaz Externalizable, que es unas 10 veces más rápida. Si no se pueden modificar los objetos de la correlación, entonces una aplicación puede asociar una interfaz ObjectTransformer a la ObjectMap. Se proporcionan los métodos serialize e inflate para permitir que la aplicación proporcione código personalizado para optimizar estas operaciones, dado el gran impacto que tienen en el rendimiento del sistema. El método serialize serializa el objeto en la corriente de datos proporcionada. El método inflate proporciona la corriente de datos de entrada y espera que la aplicación cree el objeto, lo infle utilizando los datos de la corriente y devuelva el objeto. Las implementaciones de los métodos serialize e inflate deben duplicarse entre sí.

7.1.1+ Los plug-ins DataSerializer sustituyen a los plug-ins ObjectTransformer, que están en desuso. Para serializar los datos de la forma más eficaz, utilice los plug-ins DataSerializer para mejorar el rendimiento en la mayoría de los casos. Por ejemplo, si tiene la intención de utilizar funciones, como consulta e indexación, puede aprovechar inmediatamente la mejora de rendimiento que proporcionan los plug-ins DataSerializer sin realizar cambios de configuración o programación en el código de la aplicación.

Ajuste del rendimiento de consulta

Para ajustar el rendimiento de las consultas, utilice estas técnicas y sugerencias.

Uso de parámetros

Cuando se ejecuta una consulta, la serie de consulte se debe analizar y debe desarrollarse un plan para ejecutar la consulta, ambas operaciones pueden resultar costosas.WebSphere eXtreme Scale almacena en la memoria caché los planes de consulta por la serie de consulta. Puesto que la memoria caché tiene un tamaño finito, es importante reutilizar las series de consulta siempre que sea posible. El uso de parámetros posicionales o con nombre favorece el rendimiento al fomentar la reutilización de los planes de consulta.

```
Ejemplo de parámetro posicional  Query q = em.createQuery("select c from
Customer c where c.surname=?1");   q.setParameter(1, "Claus");
```

Uso de índices

El uso de índices adecuados en una correlación puede tener un impacto significativo en el rendimiento de las consultas, a pesar de que los índices puedan implicar una sobrecarga en el rendimiento global de la correlación. Si no se dispone de índices en los atributos de objeto de las consultas, el motor de consultas realiza una exploración de la tabla de cada atributo. La exploración de tabla es la operación más cara durante la ejecución de una consulta. El uso de índices en atributos de objetos de las consultas permite que el motor de consultas no tenga que realizar una exploración innecesaria de la tabla, lo cual mejora el rendimiento global de la consulta. Si se ha diseñado la aplicación para usar las consultas de forma intensiva en una correlación sobre todo de lectura, configure los índices para atributos de objeto involucrados en la consulta. Si la correlación se actualiza continuamente, debe sopesar entre mejorar el rendimiento de la consulta y la sobrecarga de índices en la correlación.

Cuando se almacenan objetos POJO (plain old Java Object) en una correlación, una indexación correcta puede evitar un reflejo Java. En la consulta del ejemplo siguiente, la cláusula WHERE se sustituye por la búsqueda de índices de intervalo, si el campo de presupuesto tiene un índice. De lo contrario, la consulta explora toda la correlación y evalúa la cláusula WHERE de la siguiente manera: primero obtiene el presupuesto mediante un reflejo de Java y después lo compara con el valor 50000:

```
SELECT d FROM DeptBean d WHERE d.budget=50000
```

Consulte el apartado “Plan de consulta” en la página 458 para obtener información sobre cómo ajustar consultas individuales y cómo pueden afectar sintaxis diferentes, modelos de objetos e índices al rendimiento de la consulta.

Uso de la paginación

En entornos cliente/servidor, el motor de consultas transporta toda la correlación de resultados al cliente. Los datos devueltos deben dividirse en partes razonables. Las interfaces EntityManager Query y ObjectMap ObjectQuery admiten los métodos setFirstResult y setMaxResults que permiten que la consulta devuelva un subconjunto de resultados.

Devolución de valores primitivos en lugar de entidades

Con la API de consulta EntityManager, las entidades se devuelven como parámetros de consulta. El motor de consultas devuelve actualmente las claves de estas entidades al cliente. Cuando el cliente itera en estas entidades mediante el uso de Iterator del método getResultIterator, cada entidad se infla automáticamente y se gestiona como si se creara con el método find de la interfaz EntityManager. Todo el gráfico de la entidad se crea a partir del objeto ObjectMap de entidad en el cliente. Los atributos del valor de entidad y las entidades relacionadas se resuelven rápidamente.

Para no tener que crear el tan costoso gráfico, modifique la consulta de modo que devuelva los atributos individuales con navegación de vías de acceso.

Por ejemplo:

```
//
Devuelve una entidad
SELECT p FROM Person p
// Devuelve atributos SELECT p.name, p.address.street, p.address.city, p.gender
FROM Person p
```

Plan de consulta

Todas las consultas de eXtreme Scale tienen un plan de consulta. El plan describe cómo el motor de consulta interactúa con ObjectMaps e índices. Consulte el plan de consulta para determinar si los índices o serie de consulta se están utilizando correctamente. El plan de consulta también se puede utilizar para explorar las diferencias que pequeños cambios en una serie de consulta realizan en la forma en la que eXtreme Scale ejecuta una consulta.

El plan de consulta puede verse de dos modos:

- Métodos EntityManager Query o ObjectQuery getPlan API
- Rastreo de diagnóstico de ObjectGrid

Método getPlan

El método getPlan en las interfaces ObjectQuery y Query devuelve una serie que describe el plan de consulta. Esta serie puede verse en una salida estándar o en un archivo de anotaciones cronológicas.

Nota: En un entorno distribuido, el método getPlan no se ejecuta en el servidor y no refleja los índices definidos. Para ver el plan, utilice un agente para visualizar el plan en el servidor.

Rastreo del plan de consulta

El plan de consulta puede verse mediante el rastreo de ObjectGrid. Para habilitar el rastreo del plan de consulta, utilice la especificación de rastreo siguiente:

```
QueryEnginePlan=debug=enabled
```

Consulte el apartado “Recopilación de rastreo” en la página 510 para obtener más información sobre cómo habilitar el rastreo y buscar los archivos de anotaciones cronológicas de rastreo.

Ejemplos de plan de consulta

El plan de consulta utiliza la palabra para indicar que la consulta itera por una colección de ObjectMap o por una colección derivada como por ejemplo: q2.getEmps(), q2.dept o una colección temporal devuelta por el bucle interno. Si la colección es de un ObjectMap, el plan de consulta muestras si se utiliza una exploración secuencial (indicada mediante `INDEX SCAN`), un índice exclusivo o un índice no exclusivo. El plan de consulta utiliza una serie de filtro para listar las expresiones de condición que se aplican a una colección.

En una consulta de objeto no se suele utilizar un producto cartesiano. La consulta siguiente explora toda la correlación EmpBean en el bucle externo y explora toda la correlación DeptBean en el bucle interno:

```
SELECT e, d FROM EmpBean e, DeptBean d

Rastreo de plan:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in DeptBean ObjectMap using INDEX SCAN
   returning new Tuple( q2,  q3  )
```

La consulta siguiente recupera todos los nombres de los empleados de un departamento determinado; para ello, explora secuencialmente la correlación EmpBean para obtener un objeto employee. En el objeto employee, la consulta navega a su objeto department y aplica el filtro d.no=1. En este ejemplo, cada empleado tiene solo una referencia de objeto de departamento, así que el bucle interno se ejecuta una sola vez:

```
SELECT e.name FROM EmpBean e JOIN e.dept d WHERE d.no=1

Rastreo de plan:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in  q2.dept
      filter ( q3.getNo() = 1 )
   returning new Tuple( q2.name  )
```

La consulta siguiente equivale a la anterior. Sin embargo, la consulta siguiente tiene un mejor rendimiento porque en primer lugar acota el resultado a un solo objeto de departamento utilizando el índice exclusivo definido sobre el número de campo de clave primaria DeptBean. A partir del objeto department, la consulta navega hasta los objetos employee para obtener sus nombres:

```
SELECT e.name FROM DeptBean d JOIN d.emps e WHERE d.no=1

Rastreo de plan:

for q2 in DeptBean ObjectMap using UNIQUE INDEX key=(1)
     for q3 in  q2.getEmps()
   returning new Tuple( q3.name  )
```

La consulta siguiente busca todos los empleados que trabajan en desarrollo o ventas. La consulta explora toda la correlación EmpBean y realiza un filtrado adicional al evaluar las expresiones: d.name = 'Sales' o d.name='Dev'

```
SELECT e FROM EmpBean e, in (e.dept) d WHERE d.name = 'Sales'
 or d.name='Dev'

Rastreo de plan:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in  q2.dept
      filter (( q3.getName() = Sales ) OR ( q3.getName() = Dev ) )
   returning new Tuple( q2  )
```

La consulta siguiente equivale a la anterior, pero esta consulta ejecuta un plan de consulta diferente y utiliza un índice de intervalo en el nombre de campo. En general, esta consulta tiene un mejor rendimiento porque el índice del campo de nombre se utiliza para limitar los objetos department, que se ejecuta rápidamente si sólo unos pocos departamentos son de desarrollo o ventas.

```
SELECT e FROM DeptBean d, in(d.emps) e WHERE d.name='Dev' or d.name='Sales'

Rastreo de plan:

IteratorUnionIndex of

     for q2 in DeptBean ObjectMap using INDEX on name = (Dev)
       for q3 in  q2.getEmps()


     for q2 in DeptBean ObjectMap using INDEX on name = (Sales)
       for q3 in  q2.getEmps()
```

La consulta siguiente busca departamentos que no tienen ningún empleado:

```
SELECT d FROM DeptBean d WHERE NOT EXISTS(select e from d.emps e)

Rastreo de plan:

for q2 in DeptBean ObjectMap using INDEX SCAN
    filter ( NOT  EXISTS (    correlated collection defined as

       for q3 in  q2.getEmps()
       returning new Tuple( q3      )

   returning new Tuple( q2  )
```

La consulta siguiente equivale a la anterior, pero utiliza la función escalar SIZE. Esta consulta tiene un rendimiento similar pero es más fácil de escribir.

```
SELECT d FROM DeptBean d WHERE SIZE(d.emps)=0
for q2 in DeptBean ObjectMap using INDEX SCAN
    filter (SIZE( q2.getEmps()) = 0 )
   returning new Tuple( q2  )
```

El ejemplo siguiente es otra manera de escribir la misma consulta que la anterior con un rendimiento similar, pero esta consulta es más fácil de escribir también:

```
SELECT d FROM DeptBean d WHERE d.emps is EMPTY

Rastreo de plan:

for q2 in DeptBean ObjectMap using INDEX SCAN
    filter ( q2.getEmps() IS EMPTY  )
   returning new Tuple( q2  )
```

La consulta siguiente busca empleados con un domicilio que coincida al menos con una de las direcciones del empleado cuyo nombre sea igual al valor del parámetro. El bucle interno no tiene dependencia del bucle externo. La consulta ejecuta el bucle interno una sola vez.

```
SELECT e FROM EmpBean e WHERE e.home =  any (SELECT e1.home FROM EmpBean e1
 WHERE e1.name=?1)
for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( q2.home =ANY     temp collection defined as

       for q3 in EmpBean ObjectMap using INDEX on name = ( ?1)
       returning new Tuple( q3.home      )
 )
   returning new Tuple( q2  )
```

La consulta siguiente es igual a la anterior, pero tiene una subconsulta correlacionada; además, el bucle interno se ejecuta repetidamente.

```
SELECT e FROM EmpBean e WHERE EXISTS(SELECT e1 FROM EmpBean e1 WHERE
 e.home=e1.home and e1.name=?1)

Rastreo de plan:

for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( EXISTS (    correlated collection defined as

       for q3 in EmpBean ObjectMap using INDEX on name = (?1)
        filter ( q2.home =  q3.home )
       returning new Tuple( q3      )

   returning new Tuple( q2  )
```

Optimización de consultas mediante el uso de índices

La definición y el uso correctos de índices puede mejorar significativamente el rendimiento de las consultas.

Las consultas de WebSphere eXtreme Scale pueden utilizar los plug-ins HashIndex incorporados para mejorar el rendimiento de las consultas. Los índices se pueden definir en atributos de entidad o de objeto. El motor de consultas utilizará automáticamente los índices definidos si su cláusula WHERE utiliza una de las series siguientes:

- Una expresión de comparación con estos operadores: =, <, >, <= o >= (cualquier expresión de comparación excepto el símbolo distinto <>).
- Una expresión con BETWEEN.
- Los operandos de las expresiones son constantes o términos simples.

Requisitos

Los índices tienen los siguientes requisitos cuando son utilizados por Query:

- Todos los índices deben utilizar el plug-in HashIndex incorporado.
- Todos los índices deben estar definidos estáticamente. Los índices dinámicos no están soportados.
- La anotación @Index se puede utilizar para crear automáticamente plug-ins HashIndex estáticos.
- Todos los índices de atributo único deben tener la propiedad RangeIndex establecida en true.
- Todos los índices compuestos deben tener la propiedad RangeIndex establecida en false.
- Todos los índices de asociación (relación) deben tener la propiedad RangeIndex establecida en false.

Para obtener información sobre cómo configurar HashIndex, consulte “Plug-ins para la indexación de datos” en la página 330.

Para obtener información acerca de la indexación, consulte “Índices” en la página 97.

Para obtener información sobre un modo eficaz de buscar objetos almacenados en memoria caché, consulte el apartado “Utilización de un índice compuesto” en la página 339.

Uso de sugerencias para elegir un índice

Se puede seleccionar manualmente un índice utilizando el método setHint en las interfaces Query y ObjectQuery con la constante HINT_USEINDEX. Esto puede ser útil al optimizar una consulta para utilizar el índice con mejor rendimiento.

Ejemplos de consulta que utilizan los índices de atributo

Los ejemplos siguientes utilizan términos simples: e.empid, e.name, e.salary, d.name, d.budget y e.isManager. En el ejemplo se da por supuesto que los índices se han definido en los campos de nombre, salario y presupuesto de un objeto de entidad o valor. El campo empid es una clave primaria e isManager no tiene ningún índice definido.

La consulta siguiente utiliza los índices en los campos de nombre y salario. Devuelve todos los empleados con nombres que son iguales al valor del primer parámetro o un salario que es igual al valor del segundo parámetro:

```
SELECT e FROM EmpBean e where e.name=?1 or e.salary=?2
```

La siguiente consulta utiliza ambos índices en los campos de nombre y presupuesto. La consulta devuelve todos los departamentos con nombre 'DEV' que tienen un presupuesto que es mayor que 2000.

```
SELECT d FROM DeptBean dwhere d.name='DEV' and d.budget>2000
```

La consulta siguiente devuelve todos los empleados con un salario mayor que 3000 y con un valor de distintivo isManager que es igual al valor del parámetro. La consulta utiliza el índice que se ha definido en el campo de salario y realiza un filtrado adicional al evaluar la expresión de comparación: `e.isManager=?1.`

```
SELECT e FROM EmpBean e where e.salary>3000 and e.isManager=?1
```

La consulta siguiente busca todos los empleados que ganan más que el primer parámetro o los empleados que son jefes. Aunque el campo de salario tiene un índice definido, la consulta explora el índice incorporado que se ha creado en las claves primarias del campo EmpBean y evalúa la expresión: `e.salary>?1` o `e.isManager=TRUE`.

```
SELECT e FROM EmpBean e WHERE e.salary>?1 or e.isManager=TRUE
```

La consulta siguiente devuelve los empleados con un nombre que contiene la letra a. Aunque el campo de nombre tiene un índice definido, la consulta no utiliza el índice porque el campo de nombre se utiliza en la expresión LIKE.

```
SELECT e FROM EmpBean e WHERE e.name LIKE '%a%'
```

La consulta siguiente busca todos los empleados con un nombre que no sea "Smith". Aunque el campo de nombre tiene un índice definido, la consulta no utiliza el índice porque la consulta utiliza el operador de comparación distinto (<>).

```
SELECT e FROM EmpBean e where e.name<>'Smith'
```

La consulta siguiente busca todos los departamentos con un presupuesto inferior al valor del parámetro, y con un salario de empleados superior a 3000. La consulta utiliza un índice para el salario, pero no utiliza un índice para el presupuesto porque dept.budget no es un término simple. Los objetos dept se derivan de la colección e. No es necesario utilizar el índice de presupuesto para buscar los objetos dept.

```
SELECT dept from EmpBean e, in (e.dept) dept where e.salary>3000 and
dept.budget<?
```

La consulta siguiente busca todos los empleados con un salario superior al salario de los empleados que tienen empid de 1, 2 y 3. No se utiliza el salario de índice porque la comparación tiene una subconsulta. empid es una clave primaria, y se utiliza para una búsqueda de índice único porque todas las claves primarias tienen un índice incorporado definido.

```
SELECT e FROM EmpBean e WHERE e.salary > ALL (SELECT e1.salary    FROM
EmpBean e1 WHERE e1.empid=1 or e1.empid =2 or e1.empid=99)
```

Para comprobar si la consulta utiliza el índice, puede consultar el apartado “Plan de consulta” en la página 458. A continuación se muestra un plan de consulta de ejemplo de la consulta anterior:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( q2.salary >ALL     temp collection defined as
       IteratorUnionIndex of

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(1)
        )

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(2)
        )

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(99)
        )
       returning new Tuple( q3.salary )
   returning new Tuple( q2  )

for q2 in EmpBean ObjectMap using RANGE INDEX on salary with range(3000,)
     for q3 in  q2.dept
      filter ( q3.budget <  ?1 )
   returning new Tuple( q3  )
```

Atributos de indexación

Los índices se pueden definir con un único tipo de atributo con las restricciones definidas previamente.

Definición de índices de entidad utilizando @Index

Para definir un índice en una entidad, simplemente defina una anotación:

Entidades que utilizan anotaciones

```
  @Entity
 public class Employee {
  @Id int empid;
  @Index String name
  @Index double salary
  @ManyToOne Department dept;
}
@Entity
  public class Department {
  @Id int deptid;
  @Index String name;
  @Index double budget;
  boolean isManager;
  @OneToMany Collection<Employee> employees;
  }
```

Con XML

Los índices también se pueden definir utilizando XML:

Entidades sin anotaciones

```
 public class Employee {
  int empid;
  String name
  double salary
  Department dept;
  }

  public class Department {
  int deptid;
 String name;
```

```
  double budget;
  boolean isManager;
  Collection employees;
  }
```

XML de ObjectGrid con índices de atributo

```
<?xml version="1.0" encoding="UTF-8"?>
 <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
  <objectGrid name="DepartmentGrid" entityMetadataXMLFile="entity.xml>
  <backingMap name="Employee" pluginCollectionRef="Emp"/>
  <backingMap name="Department" pluginCollectionRef="Dept"/>
  </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
  <backingMapPluginCollection id="Emp">
 <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Employee.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Se deben establecer los rangos en true para los atributos." />
  </bean>
 <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Employee.salary"/>
  <property name="AttributeName" type="java.lang.String" value="salary"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Se deben establecer los rangos en true para los atributos." />
  </bean>
  </backingMapPluginCollection>
  <backingMapPluginCollection id="Dept">
 <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Department.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Se deben establecer los rangos en true para los atributos." />
  </bean>
 <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Department.budget"/>
  <property name="AttributeName" type="java.lang.String" value="budget"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Se deben establecer los rangos en true para los atributos." />
  </bean>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
  </objectGridConfig>
```

XML de entidad

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

  <description>Department entities</description>
  <entity class-name="acme.Employee" name="Employee" access="FIELD">
  <attributes>
  <id name="empid" />
  <basic name="name" />
  <basic name="salary" />
  <many-to-one name="department"
  target-entity="acme.Department"
  fetch="EAGER">
  <cascade><cascade-persist/></cascade>
  </many-to-one>
  </attributes>
  </entity>
  <entity class-name="acme.Department" name="Department" access="FIELD">
  <attributes>
  <id name="deptid" />
  <basic name="name" />
  <basic name="budget" />
  <basic name="isManager" />
  <one-to-many name="employees"
  target-entity="acme.Employee"
  fetch="LAZY" mapped-by="parentNode">
  <cascade><cascade-persist/></cascade>
  </one-to-many>
  </attributes>
  </entity>
  </entity-mappings>
```

Definición de índices para no entidades utilizando XML

Los índices para tipos que no son entidad están definidos en XML. No hay ninguna diferencia al crear MapIndexPlugin para correlaciones con entidad y correlaciones sin entidad.

Bean de Java

```
public class Employee {
  int empid;
  String name
  double salary
  Department dept;

 public class Department {
  int deptid;
 String name;
  double budget;
  boolean isManager;
  Collection employees;
  }
```

XML de ObjectGrid con índices de atributo

```
<?xml version="1.0" encoding="UTF-8"?>
 <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
  <objectGrid name="DepartmentGrid">
  <backingMap name="Employee" pluginCollectionRef="Emp"/>
  <backingMap name="Department" pluginCollectionRef="Dept"/>
  <querySchema>
  <mapSchemas>
  <mapSchema mapName="Employee" valueClass="acme.Employee"
  primaryKeyField="empid" />
  <mapSchema mapName="Department" valueClass="acme.Department"
  primaryKeyField="deptid" />
  </mapSchemas>
  <relationships>
  <relationship source="acme.Employee"
  target="acme.Department"
  relationField="dept" invRelationField="employees" />
  </relationships>
  </querySchema>
  </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
  <backingMapPluginCollection id="Emp">
 <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Employee.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Se deben establecer los rangos en true para los atributos." />
  </bean>
 <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Employee.salary"/>
  <property name="AttributeName" type="java.lang.String" value="salary"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Se deben establecer los rangos en true para los atributos." />
  </bean>
  </backingMapPluginCollection>
  <backingMapPluginCollection id="Dept">
 <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Department.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Se deben establecer los rangos en true para los atributos." />
  </bean>
 <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Department.budget"/>
  <property name="AttributeName" type="java.lang.String" value="budget"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Se deben establecer los rangos en true para los atributos." />
  </bean>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
  </objectGridConfig>
```

Relaciones de índices

WebSphere eXtreme Scale almacena las claves foráneas para las entidades relacionadas dentro del objeto padre. En el caso de entidades, las claves se

almacenan en el tuple subyacente. En el caso de objetos que no son de entidad, las claves se almacenan explícitamente en el objeto padre.

Si se añade un índice a un atributo de relación, se pueden acelerar las consultas que utilizan referencias cíclicas o los filtros de consulta IS NULL, IS EMPTY, SIZE y MEMBER OF. Las asociaciones de un valor o de varios valores pueden tener la anotación @Index o una configuración de plug-in HashIndex en un archivo XML de descriptor de ObjectGrid.

Definición de los índices de relación de entidad utilizando @Index

El ejemplo siguiente define las entidades con las anotaciones @Index:

Entidad con anotación

```
@Entity
public class Node {
    @ManyToOne @Index
    Node parentNode;

    @OneToMany @Index
    List<Node> childrenNodes = new ArrayList();

    @OneToMany @Index
    List<BusinessUnitType> businessUnitTypes = new ArrayList();
}
```

Definición de índices de relación utilizando XML

El ejemplo siguiente define las mismas entidades e índices utilizando XML con los plug-ins HashIndex:

Entidad sin anotaciones

```
 public class Node {
  int nodeId;
 Node parentNode;
 List<Node> childrenNodes = new ArrayList();
 List<BusinessUnitType> businessUnitTypes = new ArrayList();
  }
```

XML de ObjectGrid

```
<?xml version="1.0" encoding="UTF-8"?>
 <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
  <objectGrid name="ObjectGrid_Entity" entityMetadataXMLFile="entity.xml>
 <backingMap name="Node" pluginCollectionRef="Node"/>
 <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
  </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
 <backingMapPluginCollection id="Node">
 <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
 <property name="Name" type="java.lang.String" value="parentNode"/>
  <property name="AttributeName" type="java.lang.String" value="parentNode"/>
<property name="RangeIndex" type="boolean" value="false"
  description="Los rangos no están soportados para los índices de asociación." />  </bean>
 <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
 <property name="Name" type="java.lang.String" value="businessUnitType"/>
  <property name="AttributeName" type="java.lang.String" value="businessUnitTypes"/>

<property name="RangeIndex" type="boolean" value="false"
  description="Los rangos no están soportados para los índices de asociación." />
 </bean>
 <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
 <property name="Name" type="java.lang.String" value="childrenNodes"/>
 <property name="AttributeName" type="java.lang.String" value="childrenNodes"/>
<property name="RangeIndex" type="boolean" value="false"
  description="Los rangos no están soportados para los índices de asociación." />
  </bean>
  </backingMapPluginCollection>
```

```
 </backingMapPluginCollections>
 </objectGridConfig>
```

XML de entidad

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

 <description>My entities</description>
 <entity class-name="acme.Node" name="Account" access="FIELD">
 <attributes>
 <id name="nodeId" />
 <one-to-many name="childrenNodes"
 target-entity="acme.Node"
 fetch="EAGER" mapped-by="parentNode">
 <cascade><cascade-all/></cascade>
 </one-to-many>
 <many-to-one name="parentNodes"
 target-entity="acme.Node"
 fetch="LAZY" mapped-by="childrenNodes">
 <cascade><cascade-none/></cascade>
 </one-to-many>
 <many-to-one name="businessUnitTypes"
 target-entity="acme.BusinessUnitType"
 fetch="EAGER">
 <cascade><cascade-persist/></cascade>
 </many-to-one>
</attributes>
 </entity>
 <entity class-name="acme.BusinessUnitType" name="BusinessUnitType" access="FIELD">
 <attributes>
 <id name="buId" />
 <basic name="TypeDescription" />
 </attributes>
 </entity>
 </entity-mappings>
```

A través de los índices definidos previamente, se optimizan los siguientes ejemplos de consulta de entidad:

```
SELECT n FROM Node n WHERE n.parentNode is null
SELECT n FROM Node n WHERE n.businessUnitTypes is EMPTY
  SELECT n FROM Node n WHERE size(n.businessUnitTypes)>=10
  SELECT n FROM BusinessUnitType b, Node n WHERE b member of n.businessUnitTypes
   and b.name='TELECOM'
```

Definición de índices de relación sin entidad

En el ejemplo siguiente se define un plug-in HashIndex para correlaciones que no son de entidad en un archivo XML de descriptor de ObjectGrid:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="ObjectGrid_POJO">
      <backingMap name="Node" pluginCollectionRef="Node"/>
      <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
      <querySchema>
        <mapSchemas>
          <mapSchema mapName="Node"
     valueClass="com.ibm.websphere.objectgrid.samples.entity.Node"
            primaryKeyField="id" />
          <mapSchema mapName="BusinessUnitType"
            valueClass="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
            primaryKeyField="id" />
        </mapSchemas>
        <relationships>
          <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
            target="com.ibm.websphere.objectgrid.samples.entity.Node"
            relationField="parentNodeId" invRelationField="childrenNodeIds" />
          <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
            target="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
            relationField="businessUnitTypeKeys" invRelationField="" />
        </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="Node">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="parentNode"/>
<property name="Name" type="java.lang.String" value="parentNodeId"/>
<property name="AttributeName" type="java.lang.String" value="parentNodeId"/>
```

```
<property name="RangeIndex" type="boolean" value="false"
  description="Los rangos no están soportados para los índices de asociación." />
      </bean>
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="businessUnitType"/>
        <property name="AttributeName" type="java.lang.String" value="businessUnitTypeKeys"/>

<property name="RangeIndex" type="boolean" value="false"
  description="Los rangos no están soportados para los índices de asociación." />
   </bean>
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="childrenNodeIds"/>
  <property name="AttributeName" type="java.lang.String" value="childrenNodeIds"/>
  <property name="RangeIndex" type="boolean" value="false"
  description="Los rangos no están soportados para los índices de asociación." />
</bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

Dadas las configuraciones de índice anteriores, se optimizan los ejemplos de consulta de objetos siguientes:

```
SELECT n FROM Node n WHERE n.parentNodeId is null
SELECT n FROM Node n WHERE n.businessUnitTypeKeys is EMPTY
SELECT n FROM Node n WHERE size(n.businessUnitTypeKeys)>=10
SELECT n FROM BusinessUnitType b, Node n WHERE
 b member of n.businessUnitTypeKeys and b.name='TELECOM'
```

Ajuste del rendimiento de la interfaz EntityManager

La interfaz EntityManager separa las aplicaciones del estado alojado en su almacén de datos de cuadrícula de servidores.

El coste de utilizar la interfaz EntityManager no es alto y depende del tipo de trabajo que se esté realizando. Utilice siempre la interfaz EntityManager y optimice la lógica empresarial crucial una vez que se complete la aplicación. Puede reacondicionar cualquier código que utilice interfaces EntityManager para utilizar correlaciones y tuples. Normalmente, es posible que sea necesario este reacondicionamiento del código para el 10 por ciento del código.

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

Las aplicaciones que utilizan la interfaz EntityManager no necesitan proporcionar una implementación ObjectTransformer. Las aplicaciones se optimizan automáticamente.

Reacondicionamiento del código de EntityManager para correlaciones

A continuación se muestra una entidad de ejemplo:

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

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

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

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

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

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

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

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

Tareas relacionadas:

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Referencia relacionada:

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

"Definición de un esquema de entidad" en la página 169
Un ObjectGrid puede tener varios un número ilimitado de esquemas de entidades lógicas. Las entidades se definen utilizando las clases Java anotadas, XML o una combinación de XML y clases Java. Las entidades definidas se registran con un servidor eXtreme Scale y se enlazan a BackingMaps, índices y otros plug-ins.

"Métodos de devolución de llamada y escuchas de entidad" en la página 186
Se puede notificar a las aplicaciones cuando el estado de una entidad pasa de un estado a otro. Existen dos mecanismos de devolución de llamada para los sucesos de cambio de estado: los métodos de devolución de llamada de ciclo de vida que están definidos en una clase de entidad y se invocan siempre que cambia el estado de la entidad, y los escuchas de entidad, que son más generales porque el escucha de entidad se puede registrar en varias entidades.

"Ejemplos de escucha de entidad" en la página 190
Puede escribir EntityListeners según sus necesidades. A continuación se ofrecen varios scripts de ejemplo.

"Interfaz EntityTransaction" en la página 203
Puede utilizar la interfaz EntityTransaction para delimitar transacciones.

Agente de instrumentación de rendimiento de entidades

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

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

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

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

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

Opciones del agente de eXtreme Scale

config
Altera temporalmente el nombre de archivo de configuración.

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

exclude
Especifica o altera temporalmente la definición @Exclude.

fieldAccessEntity
Especifica o altera temporalmente la definición @FieldAccessEntity.

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

trace.file
Especifica la ubicación del archivo de rastreo.

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

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

Archivo `ogagent.config`

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

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

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

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

especificada". El agente de eXtreme Scale determina que no se ha definido ninguna clase y paquete de entidad de acceso a campos. Si hay líneas después de la señal @FieldAccessEntity, estas representan las clases y paquetes de la entidad de acceso a campos especificada por el usuario. Por ejemplo, "dominio de entidad de acceso a campos". El dominio de entidad de acceso a campos es un subdominio del dominio de transformación. Los paquetes y clases que se listan en el dominio de entidad de acceso a campos forman parte del dominio de transformación, incluso cuando no se listan en el dominio de transformación. La señal @Exclude, que lista paquetes y clases que se excluyen de la transformación, no tiene ningún impacto en el dominio de entidad de acceso a campos. Cuando se especifica la señal @FieldAccessEntity, todas las entidades de acceso a campos deben estar en este dominio de entidad de acceso a campos. De lo contrario, puede producirse una excepción FieldAccessEntityNotInstrumentedException.

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

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

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

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

@Exclude
com.excludedPackage
com.excludedClass

@FieldAccessEntity
```

Consideración sobre el rendimiento

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

Conceptos relacionados:

"Ajuste del rendimiento de la interfaz EntityManager" en la página 468
La interfaz EntityManager separa las aplicaciones del estado alojado en su almacén de datos de cuadrícula de servidores.

"Almacenamiento en memoria caché de objetos y sus relaciones (API EntityManager)" en la página 166
La mayoría de los productos de memoria caché utilizan las API basadas en correlaciones para almacenar los datos como pares de clave-valor. La API ObjectMap y la memoria caché dinámica de WebSphere Application Server, entre otras, utilizan este enfoque. No obstante, las API basadas en correlaciones tienen limitaciones. La API EntityManager simplifica la interacción con la cuadrícula de datos proporcionando una forma fácil de declarar un gráfico complejo de objetos relacionados e interactuar con él.

"Gestor de entidades en un entorno distribuido" en la página 178
Puede utilizar la API EntityManager con un ObjectGrid local o en un entorno distribuido de eXtreme Scale. La diferencia principal es el modo de conectarse a este entorno remoto. Después de establecer una conexión, no hay ninguna diferencia entre el uso de un objeto Session o el uso de la API EntityManager.

"Interacción con EntityManager" en la página 183
Normalmente, las aplicaciones en primer lugar obtienen una referencia de ObjectGrid y, a continuación, una Session de dicha referencia para cada hebra. Las sesiones no pueden compartirse entre hebras. Hay disponible un método adicional en el elemento Session, el método getEntityManager. Este método devuelve una referencia a un gestor de entidades para utilizar para esta hebra. La interfaz EntityManager puede sustituir las interfaces Session y ObjectMap para todas las aplicaciones. Puede utilizar estas API EntityManager si el cliente tiene acceso a las clases de entidad definidas.

"Soporte de planes de captación de EntityManager" en la página 193
Un FetchPlan es la estrategia que el gestor de entidades utiliza para recuperar los objetos asociados si la aplicación tiene que acceder a las relaciones.

"Colas de consulta de entidades" en la página 198
Las colas de consulta permiten a las aplicaciones crear una cola calificada por una consulta en el servidor o en eXtreme Scale local para una entidad. Las entidades del resultado de la consulta se almacenan en esta cola. Actualmente, la cola de consulta sólo se admite en una correlación que utilice la estrategia de bloqueo pesimista.

Tareas relacionadas:

"Guía de aprendizaje: Almacenamiento de información de pedidos en entidades" en la página 7
La guía de aprendizaje para el gestor de entidades le muestra cómo utilizar WebSphere eXtreme Scale para almacenar la información de pedidos en un sitio web. Puede crear una aplicación Java Platform, Standard Edition 5 sencilla que utiliza un eXtreme Scale local en memoria local. Las entidades utilizan genéricos y anotaciones Java SE 5.

Capítulo 7. Seguridad

WebSphere eXtreme Scale puede proteger el acceso a los datos, incluida la posibilidad de integración con proveedores de datos externos. Entre los aspectos de seguridad se incluyen la autenticación, la autorización y la seguridad en el transporte, en la cuadrícula de datos, local y en JMX (MBean).

Configuración de perfiles de seguridad para el programa de utilidad `xscmd`

Mediante la creación de un perfil de seguridad, puede utilizar parámetros de seguridad guardados para utilizar el programa de utilidad **xscmd** con entornos seguros.

Antes de empezar

Para obtener más información sobre cómo configurar el programa de utilidad **xscmd**, consulte Administración con el programa de utilidad **xscmd**.

Acerca de esta tarea

Puede utilizar el parámetro **-ssp** *nombre_perfil* o **--saveSecProfile** *nombre_perfil* con el resto del mandato **xscmd** para guardar un perfil de seguridad. El perfil puede contener valores para los nombres de usuario y las contraseñas, los generadores de credenciales, los almacenes de claves, los almacenes de confianza y los tipos de transporte.

El grupo de mandatos **ProfileManagement** del programa de utilidad **xscmd** contiene mandatos para gestionar los perfiles de seguridad.

Procedimiento

- Guarde un perfil de seguridad.

  Para guardar un perfil de seguridad, utilice el parámetro **-ssp** *nombre_perfil* o **--saveSecProfile** *nombre_perfil* con el resto del mandato. Al añadir este parámetro al mandato se guardan los parámetros siguientes:

  ```
  -al,--alias <alias>
  -arc,--authRetryCount <entero>
  -ca,--credAuth <soporte>
  -cgc,--credGenClass <nombre_clase>
  -cgp,--credGenProps <propiedad>
  -cxpv,--contextProvider <proveedor>
  -ks,--keyStore <vía_acceso_archivo>
  -ksp,--keyStorePassword <contraseña>
  -kst,--keyStoreType <tipo>
  -prot,--protocol <protocolo>
  -pwd,--password <contraseña>
  -ts,--trustStore <vía_acceso_archivo>
  -tsp,--trustStorePassword <contraseña>
  -tst,--trustStoreType <tipo>
  -tt,--transportType <tipo>
  -user,--username <nombre_usuario>
  ```

  Los perfiles de seguridad se guardan en el directorio *inicio_usuario*`\.xscmd\profiles\security\`*<nombre_perfil>*`.properties`.

- Utilice un perfil de seguridad guardado.

  Para utilizar un perfil de seguridad guardado, añada el parámetro **-sp** *nombre_perfil* o **--securityProfile** *nombre_perfil* al mandato que está ejecutando. Ejemplo de mandato: `xscmd -c listHosts -cep myhost.mycompany.com -sp myprofile`
- Liste los mandatos del grupo de mandatos **ProfileManagement**.

  Ejecute el mandato siguiente: **xscmd -lc ProfileManagement**.
- Liste los perfiles de seguridad existentes.

  Ejecute el mandato siguiente: **xscmd -c listProfiles -v**.
- Visualice los valores que se han guardado en un perfil de seguridad.

  Ejecute el mandato siguiente: **xscmd -c showProfile -pn** *nombre_perfil*.
- Elimine un perfil de seguridad existente.

  Ejecute el mandato siguiente: **xscmd -c RemoveProfile -pn** *nombre_perfil*.

Referencia relacionada:

Migración de la herramienta **xsadmin** a la herramienta **xscmd**
En releases anteriores, la herramienta **xsadmin** era un programa de utilidad de línea de mandatos de ejemplo para supervisar el estado del entorno. La herramienta **xscmd** se ha presentado como una herramienta de línea de mandatos soportada oficialmente de supervisión y administración. Si utilizaba anteriormente la herramienta **xsadmin**, considere migrar los mandatos a la nueva herramienta **xscmd**.

Programación de la seguridad

Utilice las interfaces de programación para gestionar los distintos aspectos de seguridad de un entorno WebSphere eXtreme Scale.

API de seguridad

WebSphere eXtreme Scale adopta una arquitectura de seguridad abierta. Proporciona una infraestructura de seguridad básica para la autenticación, autorización y la seguridad de transporte y solicita a los clientes que implementen los plug-ins para completar la infraestructura de seguridad.

La siguiente imagen muestra el flujo básico de la autenticación y autorización de cliente para un servidor eXtreme Scale.

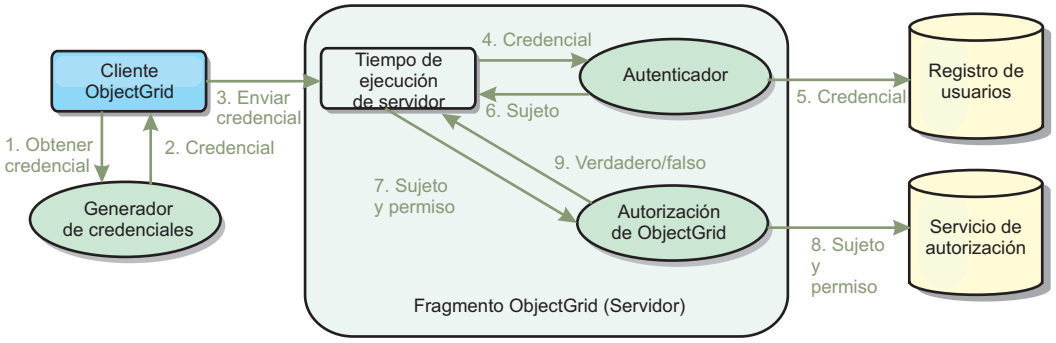


Figura 31. Flujo de autenticación y autorización de cliente

El flujo de autenticación y el flujo de autorización son los siguientes.

Flujo de autenticación

1. El flujo de autenticación se inicia con un cliente eXtreme Scale que obtiene una credencial. Esto se realiza a través del plug-in com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator.
2. Un objeto CredentialGenerator sabe como generar una credencial de cliente válida, por ejemplo, un par de ID de usuario y una contraseña, un ticket de Kerberos, etc. Esta credencial generada se vuelve a enviar al cliente.
3. Después de que el cliente recupere el objeto Credential utilizando el objeto CredentialGenerator, este objeto Credential se envía junto a la solicitud de eXtreme Scale al servidor eXtreme Scale.
4. El servidor eXtreme Scale autentica el objeto Credential antes de procesar la solicitud eXtreme Scale. A continuación, el servidor utiliza el plug-in Authenticator para autenticar el objeto Credential.
5. El plug-in Authenticator representa una interfaz para el registro de usuarios, por ejemplo, un servidor LDAP (Lightweight Directory Access Protocol) o un registro de usuarios del sistema operativo. El Authenticator consulta el registro de usuarios y toma decisiones de autenticación.
6. Si la autenticación se realiza correctamente, se devuelve un objeto Subject para representar este cliente.

   Flujo de autorización

   WebSphere eXtreme Scale adopta un mecanismo de autorización basado en los permisos y tiene distintas categorías de permiso representadas por diferentes clases de permiso. Por ejemplo, un objeto com.ibm.websphere.objectgrid.security.MapPermission representa los permisos para leer, escribir, insertar, invalidar y eliminar las entradas de datos en un ObjectMap. Puesto que WebSphere eXtreme Scale soporta la autorización JAAS (Java Authentication and Authorization Service) directamente, puede utilizar JAAS para manejar la autorización proporcionando políticas de autorización.

   Además, eXtreme Scale soporta las autorizaciones personalizadas. Las autorizaciones personalizadas se conectan mediante el com.ibm.websphere.objectgrid.security.plugins.ObjectGridAuthorization. El flujo de la autorización del cliente es la siguiente.
7. El tiempo de ejecución del servidor envía el objeto Subject y el permiso necesario al plug-in de autorización.
8. El plug-in de autorización consulta al servicio de autorización y toma una decisión sobre autorización. Si se otorga el permiso para este objeto Subject, se devuelve un valor de `true`, de lo contrario, se devuelve `false`.
9. Esta decisión de autorización, true o false, se devuelve al tiempo de ejecución del servidor.

Implementación de seguridad

Los temas de esta sección tratan cómo programar un despliegue seguro de WebSphere eXtreme Scale y cómo programar las implementaciones de plug-in. La sección se organiza basándose en las distintas características de seguridad. En cada subtema, obtendrá información sobre los plug-ins relevantes y cómo implementar los plug-ins. En la sección de autenticación, verá cómo conectarse a un entorno de despliegue seguro de WebSphere eXtreme Scale.

Autenticación de cliente: el tema de autenticación de cliente describe cómo un cliente WebSphere eXtreme Scale obtiene una credencial y cómo un servidor autentica el cliente. También tratará cómo un cliente de WebSphere eXtreme Scale se conecta a un servidor WebSphere eXtreme Scale seguro.

Autorización: el tema de autorización explica cómo utilizar ObjectGridAuthorization para realizar la autorización de cliente, además de la autorización JAAS.

Autenticación de cuadrícula: el tema de la autenticación de la cuadrícula de datos trata cómo puede utilizar SecureTokenManager para transportar de forma segura secretos de servidor.

Programación de Java Management Extensions (JMX): cuando el servidor WebSphere eXtreme Scale está protegido, el cliente JMX podría necesitar enviar una credencial JMX al servidor.

Programación de la autenticación de cliente

Para la autenticación, WebSphere eXtreme Scale proporciona un tiempo de ejecución para enviar la credencial del cliente al servidor y, a continuación, llama al plug-in autenticador para autenticar los usuarios.

WebSphere eXtreme Scale requiere que implemente los siguientes plug-ins para completar la autenticación.

- Credencial: una Credential representa una credencial de cliente como, por ejemplo, un par de ID de usuario y contraseña.
- CredentialGenerator: un CredentialGenerator representa una fábrica de credenciales para generar la credencial.
- Authenticator: un Authenticator autentica la credencial de cliente y recupera la información de cliente.

Plug-ins Credential y CredentialGenerator

Cuando un cliente de eXtreme Scale se conecta a un servidor que requiere la autenticación, es necesario que el cliente proporcione una credencial de cliente. Una credencial de cliente se representa mediante una interfaz com.ibm.websphere.objectgrid.security.plugins.Credential. Una credencial de cliente puede ser un par de nombre de usuario y contraseña, un ticket Kerberos, un certificado de cliente o datos en cualquier formato que hayan acordado el cliente y el servidor. Esta interfaz define explícitamente los métodos equals(Object) y hashCode. Estos dos métodos son importantes porque los objetos Subject autenticados se almacenan en memoria caché utilizando el objeto Credential como la clave en el lado del servidor. WebSphere eXtreme Scale también proporciona un plug-in para generar una credencial. Este plug-in se representa mediante la interfaz com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator y es práctico si la credencial puede caducar. En este caso, se llama al método getCredential para renovar una credencial.

La interfaz Credential define explícitamente los métodos equals(Object) y hashCode. Estos dos métodos son importantes porque los objetos Subject autenticados se almacenan en memoria caché utilizando el objeto Credential como la clave en el lado del servidor.

También puede utilizar el plug-in proporcionado para generar una credencial. Este plug-in se representa mediante la interfaz com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator, y es práctico si la credencial puede caducar. En este caso, se llama al método getCredential para renovar una credencial. Consulte la documentación de la API si desea más detalles.

Hay tres implementaciones predeterminadas proporcionadas para las interfaces Credential:

- La implementación com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential, que contiene un par de ID de usuario y contraseña.
- La implementación com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential, que contiene las señales de autenticación y autorización específicas de WebSphere Application Server. Estas señales pueden usarse para propagar los atributos de seguridad en los servidores de aplicaciones del mismo dominio de seguridad.

WebSphere eXtreme Scale también proporciona un plug-in para generar una credencial. Este plug-in se representa mediante la interfaz com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator.WebSphere eXtreme Scale proporciona dos implementaciones incorporadas predeterminadas:

- El constructor com.ibm.websphere.objectgrid.security.plugins.builtins. UserPasswordCredentialGenerator toma un ID de usuario y una contraseña. Cuando se llama al método getCredential, éste devuelve un objeto UserPasswordCredential que contiene el ID de usuario y una contraseña.
- com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator representa un generador de credenciales (señal de seguridad) al ejecutarse en WebSphere Application Server. Cuando se llama al método getCredential, se recupera el sujeto (Subject) asociado a la hebra actual. A continuación, la información de seguridad de este objeto Subject se convierte en un objeto WSTokenCredential. Puede especificar si va a recuperar un sujeto runAs o un sujeto caller de la hebra mediante la constante WSTokenCredentialGenerator.RUN_AS_SUBJECT o WSTokenCredentialGenerator.CALLER_SUBJECT.

UserPasswordCredential y UserPasswordCredentialGenerator

Con finalidades de pruebas, WebSphere eXtreme Scale proporciona las siguientes implementaciones de plug-in:

1. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential`
2. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator`

La credencial de contraseña de usuario almacena un ID de usuario y una contraseña. El generador de credenciales de contraseñas de usuarios contendrá este ID de usuario y contraseña.

El siguiente código de ejemplo muestra cómo implementar estos dos plug-ins.

UserPasswordCredential.java

```
// Este programa de ejemplo se proporciona TAL CUAL y se puede utilizar, ejecutar, copiar y modificar
// sin que el cliente tenga que pagar derechos
// (a) para su propia formación,
// (b) para desarrollar aplicaciones diseñadas para ejecutarse con un producto IBM WebSphere,
// para uso interno propio del cliente o para su redistribución por parte del cliente, como parte de una
// aplicación de ese tipo, en los productos propios del cliente.
// Material bajo licencia - Propiedad de IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import com.ibm.websphere.objectgrid.security.plugins.Credential;

/**
 * Esta clase representa una credencial que contiene un ID de usuario y una contraseña.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Credential
 * @see UserPasswordCredentialGenerator#getCredential()
 */
public class UserPasswordCredential implements Credential {
```

```
private static final long serialVersionUID = 1409044825541007228L;

private String ivUserName;

private String ivPassword;

/**
 * Crea una UserPasswordCredential con el nombre de usuario y contraseña
 * específicos.
 *
 * @param userName el nombre de usuario para esta credencial
 * @param password la contraseña para esta credencial
 *
 * @throws IllegalArgumentException si el userName o password es <code>null</code>
 */
public UserPasswordCredential(String userName, String password) {
    super();
    if (userName == null || password == null) {
        throw new IllegalArgumentException("El nombre y la contraseña no pueden ser nulos.");
    }
    this.ivUserName = userName;
    this.ivPassword = password;
}

/**
 * Obtiene el nombre de usuario para esta credencial.
 *
 * @return   el argumento del nombre de usuario que se ha pasado al constructor
 *           o <code>setUserName(String)</code>
 *           de esta clase
 *
 * @see #setUserName(String)
 */
public String getUserName() {
    return ivUserName;
}

/**
 * Establece el nombre de usuario para esta credencial.
 *
 * @param userName el nombre de usuario a establecer.
 *
 * @throws IllegalArgumentException si userName es <code>null</code>
 */
public void setUserName(String userName) {
    if (userName == null) {
        throw new IllegalArgumentException("El nombre de usuario no puede ser nulo.");
    }
    this.ivUserName = userName;
}

/**
 * Obtiene la contraseña para esta credencial.
 *
 * @return   el argumento de contraseña que se ha pasado al constructor
 *           o el método <code>setPassword(String)</code>
 *           de esta clase
 *
 * @see #setPassword(String)
 */
public String getPassword() {
    return ivPassword;
}

/**
 * Establece la contraseña para esta credencial.
 *
 * @param password la contraseña a establecer.
 *
 * @throws IllegalArgumentException si password es <code>null</code>
 */
public void setPassword(String password) {
    if (password == null) {
        throw new IllegalArgumentException("La contraseña no puede ser nula.");
    }
    this.ivPassword = password;
}

/**
 * Comprueba si dos objetos UserPasswordCredential son iguales.
 * <p>
 * Dos objetos UserPasswordCredential son iguales si y sólo si sus nombres de usuario
 * y contraseñas son iguales.
 *
 * @param o el objeto que se está comprobando que sea igual que este objeto.
 *
 * @return <code>true</code> si los dos objetos UserPasswordCredential son equivalentes.
 *
 * @see Credential#equals(Object)
 */
```

```
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o instanceof UserPasswordCredential) {
            UserPasswordCredential other = (UserPasswordCredential) o;
            return other.ivPassword.equals(ivPassword) && other.ivUserName.equals(ivUserName);
        }

        return false;
    }

    /**
     * Devuelve el código hash del objeto UserPasswordCredential.
     *
     * @return el código hash de este objeto
     *
     * @see Credential#hashCode()
     */
    public int hashCode() {
        return ivUserName.hashCode() + ivPassword.hashCode();
    }
}
```

UserPasswordCredentialGenerator.java

```
// Este programa de ejemplo se proporciona TAL CUAL y se puede utilizar, ejecutar, copiar y modificar
// sin que el cliente tenga que pagar derechos
// (a) para su propia formación,
// (b) para desarrollar aplicaciones diseñadas para ejecutarse con un producto IBM WebSphere,
// para uso interno propio del cliente o para su redistribución por parte del cliente, como parte de una
// aplicación de ese tipo, en los productos propios del cliente.
// Material bajo licencia - Propiedad de IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.util.StringTokenizer;

import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator;

/**
 * Este generador de credenciales crea objetos <code>UserPasswordCredential</code>.
 * <p>
 * UserPasswordCredentialGenerator tiene una relación de uno a uno con
 * UserPasswordCredential porque sólo puede crear una UserPasswordCredential
 * que represente una identidad.
 *
 * @since WAS XD 6.0.1
 * @ibm-api
 *
 * @see CredentialGenerator
 * @see UserPasswordCredential
 */
public class UserPasswordCredentialGenerator implements CredentialGenerator {

    private String ivUser;

    private String ivPwd;

    /**
     * Crea un UserPasswordCredentialGenerator sin nombre de usuario o contraseña.
     *
     * @see #setProperties(String)
     */
    public UserPasswordCredentialGenerator() {
        super();
    }

    /**
     * Crea un UserPasswordCredentialGenerator con un nombre de usuario y contraseña
     * específicos
     *
     * @param user el nombre de usuario
     * @param pwd la contraseña
     */
    public UserPasswordCredentialGenerator(String user, String pwd) {
        ivUser = user;
        ivPwd = pwd;
    }

    /**
     * Crea un nuevo objeto <code>UserPasswordCredential</code> utilizando el
     * nombre de usuario y la contraseña de este objeto.
     *
     * @return una nueva instancia de <code>UserPasswordCredential</code>
     *
     * @see CredentialGenerator#getCredential()
     * @see UserPasswordCredential
     */
    public Credential getCredential() {
        return new UserPasswordCredential(ivUser, ivPwd);
    }
```

```
/**
 * Obtiene la contraseña para este generador de credenciales.
 *
 * @return   el argumento de contraseña que se ha pasado al constructor
 */
public String getPassword() {
    return ivPwd;
}

/**
 * Obtiene el nombre de usuario para esta credencial.
 *
 * @return   el argumento de usuario que se ha pasado al constructor
 *           de esta clase
 */
public String getUserName() {
    return ivUser;
}
/**
 * Establece propiedades adicionales, en concreto, un nombre de usuario y una contraseña.
 *
 * @param properties una serie de propiedades con un nombre de usuario y
 *                   una contraseña separados por un blanco.
 *
 * @throws IllegalArgumentException si el formato no es válido
 */
public void setProperties(String properties) {
    StringTokenizer token = new StringTokenizer(properties, " ");
    if (token.countTokens() != 2) {
        throw new IllegalArgumentException(
            "Las propiedades deben tener un nombre de usuario y una contraseña separados por un blanco.");
    }

    ivUser = token.nextToken();
    ivPwd = token.nextToken();
}
/**
 * Comprueba si dos objetos UserPasswordCredentialGenerator son iguales.
 * <p>
 * Dos objetos UserPasswordCredentialGenerator son iguales si y sólo si
 * sus nombres de usuario y contraseñas son iguales.
 *
 * @param obj el objeto que se está comprobando que sea igual que este objeto.
 *
 * @return <code>true</code> si ambos objetos UserPasswordCredentialGenerator
 *         son equivalentes.
 */
public boolean equals(Object obj) {
    if (obj == this) {
        return true;
    }

    if (obj != null && obj instanceof UserPasswordCredentialGenerator) {
        UserPasswordCredentialGenerator other = (UserPasswordCredentialGenerator) obj;

        boolean bothUserNull = false;
        boolean bothPwdNull = false;

        if (ivUser == null) {
            if (other.ivUser == null) {
                bothUserNull = true;
            } else {
                return false;
            }
        }

        if (ivPwd == null) {
            if (other.ivPwd == null) {
                bothPwdNull = true;
            } else {
                return false;
            }
        }

        return (bothUserNull || ivUser.equals(other.ivUser)) && (bothPwdNull || ivPwd.equals(other.ivPwd));
    }

    return false;
}

/**
 * Devuelve el código hash del objeto UserPasswordCredentialGenerator.
 *
 * @return el código hash de este objeto
 */
public int hashCode() {
```

```
        return ivUser.hashCode() + ivPwd.hashCode();
    }

}
```

La clase UserPasswordCredential contiene dos atributos: nombre de usuario y contraseña. UserPasswordCredentialGenerator actúa como una fábrica que contiene los objetos UserPasswordCredential.

WSTokenCredential y WSTokenCredentialGenerator

Cuando los clientes y los servidores de WebSphere eXtreme Scale están desplegados en WebSphere Application Server, la aplicación cliente puede utilizar estas dos implementaciones incorporadas cuando se cumplen las siguientes condiciones:

1. La seguridad global de WebSphere Application Server está activa.
2. Todos los clientes y servidores WebSphere eXtreme Scale se ejecutan en WebSphere Application Server Mäquinas virtuales Java.
3. Los servidores de aplicaciones están en el mismo dominio de seguridad.
4. El cliente ya ha sido autenticado en WebSphere Application Server.

En esta situación, el cliente puede utilizar la clase com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator para generar una credencial. El servidor utiliza la clase de implementación WSAuthenticator para autenticar la credencial.

Este escenario se aprovecha del hecho de que el cliente eXtreme Scale ya se ha autenticado. Puesto que los servidores de aplicaciones que tienen los servidores en el mismo dominio de seguridad que los servidores de aplicaciones que alojan los clientes, las señales de seguridad se pueden propagar del cliente al servidor, de forma que no es necesario que se vuelva a autenticar el mismo registro de usuarios.

Nota: No dé por sentado que un CredentialGenerator siempre genera la misma credencial. Para una credencial que puede caducar y se puede renovar, CredentialGenerator debe poder generar la última credencial válida para garantizar el éxito de la autenticación. Un ejemplo es utilizar el ticket de Kerberos como un objeto Credential. Cuando se renueva el ticket Kerberos, CredentialGenerator deberá recuperar el ticket renovado cuando se llame a CredentialGenerator.getCredential.

Plug-in de autenticador

Después de que el cliente de eXtreme Scale recupere el objeto Credential mediante el objeto CredentialGenerator, el objeto Credential de este cliente se envía junto con la solicitud del cliente al servidor de eXtreme Scale. El servidor autentica el objeto Credential antes de procesar la solicitud. Si el objeto Credential se autentica correctamente, se devuelve un objeto Subject para representar este cliente.

A continuación, el objeto Subject se almacena en memoria caché y caduca después de que su vida útil alcance el valor de tiempo de espera de la sesión. El valor de tiempo de espera del inicio de sesión puede establecerse mediante la propiedad loginSessionExpirationTime del archivo XML del clúster. Por ejemplo, establecer `loginSessionExpirationTime="300"` hace que el objeto Subject caduque en 300 segundos.

Este objeto Subject se utilizará para autorizar la solicitud que se muestra más adelante. Un servidor de eXtreme Scale utiliza el plug-in Authenticator para autenticar el objeto Credential. Consulte la información sobre el Autenticador en la documentación de la API si desea más detalles.

El plug-in Authenticator es donde el tiempo de ejecución de eXtreme Scale autentica el objeto Credential del registro de usuarios del cliente, por ejemplo, un servidor LDAP (Lightweight Directory Access Protocol).

WebSphere eXtreme Scale no proporciona una configuración de registro de usuarios disponible inmediatamente. La configuración y la gestión del registro de usuarios se deja fuera de WebSphere eXtreme Scale con fines de simplicidad y flexibilidad. Este plug-in implementa la conexión y la autenticación al registro de usuarios. Por ejemplo, una implementación de autenticador extrae el ID de usuario y la contraseña de la credencial, los utiliza para conectarse a un servidor LDAP y validarlo, y crea un objeto Subject como resultado de la autenticación. La implementación podría utilizar los módulos de inicio de sesión JAAS. Como resultado de la autenticación, se devuelve un objeto Subject.

Tenga en cuenta que este método crea dos excepciones: InvalidCredentialException y ExpiredCredentialException. La excepción InvalidCredentialException indica que la credencial no es válida. La excepción ExpiredCredentialException indica que la credencial ha caducado. Si se genera una de estas dos excepciones del método de autenticación, las excepciones se envían de vuelta al cliente. Sin embargo, el tiempo de ejecución del cliente maneja estas dos excepciones de forma distinta:

- Si el error es una excepción InvalidCredentialException, el tiempo de ejecución del cliente visualiza esta excepción. La aplicación debe tratar la excepción. Pude corregir CredentialGenerator, por ejemplo, y volver a repetir la operación.
- Si el error es una excepción ExpiredCredentialException y el recuento de reintentos no es 0, el tiempo de ejecución del cliente vuelve a llamar la método CredentialGenerator.getCredential y envía el nuevo objeto Credential al servidor. Si la autenticación de la nueva credencial es correcta, el servidor procesa la solicitud. Si, en cambio, la autenticación falla, el excepción vuelve a enviarse al cliente. Si el número de intentos de autenticación alcanza el valor soportado y el cliente sigue obteniendo una excepción ExpiredCredentialException, se genera la excepción ExpiredCredentialException. La aplicación debe tratar el error.

La interfaz Authenticator proporciona una gran flexibilidad. Puede implementar la interfaz Authenticator de su propia manera específica. Por ejemplo, puede implementar esta interfaz para soportar dos registros de usuarios distintos.

WebSphere eXtreme Scale proporciona implementaciones de plug-in de autenticador de ejemplo. Excepto para el plug-in de autenticador de WebSphere Application Server, las otras implementaciones sólo son ejemplos con finalidades de prueba.

KeyStoreLoginAuthenticator

Este ejemplo utiliza una implementación incorporada de eXtreme Scale: KeyStoreLoginAuthenticator, que tiene finalidades de pruebas y de ejemplo (un almacén de claves es un registro de usuarios sencillo y no se debe utilizar para un entorno de producción). De nuevo, la clase se visualiza para demostrar de forma adicional cómo implementar un autenticador.

KeyStoreLoginAuthenticator.java

```
// Este programa de ejemplo se proporciona TAL CUAL y se puede utilizar, ejecutar, copiar y modificar
// sin que el cliente tenga que pagar derechos
// (a) para su propia formación,
```

```
// (b) para desarrollar aplicaciones diseñadas para ejecutarse con un producto IBM WebSphere,
// para uso interno propio del cliente o para su redistribución por parte del cliente, como parte de una
// aplicación de ese tipo, en los productos propios del cliente.
// Material bajo licencia - Propiedad de IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007

package com.ibm.websphere.objectgrid.security.plugins.builtins;

import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

import com.ibm.websphere.objectgrid.security.plugins.Authenticator;
import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.ExpiredCredentialException;
import com.ibm.websphere.objectgrid.security.plugins.InvalidCredentialException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.security.auth.callback.UserPasswordCallbackHandlerImpl;

/**
 * Esta clase es una implementación de la interfaz <code>Authenticator</code>
 * cuando un nombre de usuario y una contraseña se utilizan como credencial.
 * <p>
 * Cuando se utiliza la autenticación de ID de usuario y contraseña, la credencial pasada al
 * método <code>authenticate(Credential)</code> es un objeto UserPasswordCredential.
 * <p>
 * Esta implementación utilizará un <code>KeyStoreLoginModule</code> para autenticar
 * al usuario en el almacén de claves utilizando el módulo de inicio de sesión JAAS "KeyStoreLogin". El
 * almacén de claves se puede configurar como una opción para la clase
 * <code>KeyStoreLoginModule</code>. Consulte la clase
 * <code>KeyStoreLoginModule</code> para obtener más detalles sobre cómo configurar
 * el archivo de configuración de inicio de sesión JAAS.
 * <p>
 * Esta clase sólo sirve de ejemplo y de comprobación rápida. Los usuarios deben
 * crear su propia implementación de Authenticator que puede adaptarse mejor al
 * entorno.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Authenticator
 * @see KeyStoreLoginModule
 * @see UserPasswordCredential
 */
public class KeyStoreLoginAuthenticator implements Authenticator {

    /**
     * Crea un nuevo KeyStoreLoginAuthenticator.
     */
    public KeyStoreLoginAuthenticator() {
        super();
    }

    /**
     * Autentica un <code>UserPasswordCredential</code>.
     * <p>
     * Utiliza el nombre de usuario y la contraseña de la UserPasswordCredential especificada
     * para iniciar la sesión en el KeyStoreLoginModule llamado "KeyStoreLogin".
     *
     * @throws InvalidCredentialException si la credencial no es una
     *         UserPasswordCredential o se produce un error durante el proceso
     *         de la UserPasswordCredential proporcionada
     *
     * @throws ExpiredCredentialException si la credencial ha caducado. Esta excepción
     *         no la utiliza esta implementación
     *
     * @see Authenticator#authenticate(Credential)
     * @see KeyStoreLoginModule
     */
    public Subject authenticate(Credential credential) throws InvalidCredentialException,
    ExpiredCredentialException {

        if (credential == null) {
            throw new InvalidCredentialException("Supplied credential is null");
        }

        if (! (credential instanceof UserPasswordCredential) ) {
            throw new InvalidCredentialException("Supplied credential is not a UserPasswordCredential");
        }

        UserPasswordCredential cred = (UserPasswordCredential) credential;
        LoginContext lc = null;
        try {
            lc = new LoginContext("KeyStoreLogin",
                    new UserPasswordCallbackHandlerImpl(cred.getUserName(), cred.getPassword().toCharArray()));

            lc.login();

            Subject subject = lc.getSubject();
```

```
            return subject;
        }
        catch (LoginException le) {
            throw new InvalidCredentialException(le);
        }
        catch (IllegalArgumentException ile) {
            throw new InvalidCredentialException(ile);
        }
    }
}
```

KeyStoreLoginModule.java

```
// Este programa de ejemplo se proporciona TAL CUAL y se puede utilizar, ejecutar, copiar y modificar
// sin que el cliente tenga que pagar derechos
// (a) para su propia formación,
// (b) para desarrollar aplicaciones diseñadas para ejecutarse con un producto IBM WebSphere,
// para uso interno propio del cliente o para su redistribución por parte del cliente, como parte de una
// aplicación de ese tipo, en los productos propios del cliente.
// Material bajo licencia - Propiedad de IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.io.File;
import java.io.FileInputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.UnrecoverableKeyException;
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import javax.security.auth.x500.X500Principal;
import javax.security.auth.x500.X500PrivateCredential;

import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.util.ObjectGridUtil;

/**
 * Un KeyStoreLoginModule es un módulo de inicio de sesión de autenticación de almacén de claves
 * basado en la autenticación JAAS.
 * <p>
 * Una configuración de inicio de sesión debe proporcionar una opción
 * "<code>keyStoreFile</code>" para indicar donde está ubicado el archivo de almacén
 * de claves. Si el valor <code>keyStoreFile</code> contiene una propiedad del sistema
 * en el formato <code>${system.property}</code>, se expandirá hasta el valor de
 * la propiedad del sistema.
 * <p>
 * Si no se proporciona una opción "<code>keyStoreFile</code>", el nombre de archivo
 * del almacén de claves predeterminado es <code>"${java.home}${/}.keystore"</code>.
 * <p>
 * A continuación se muestra un ejemplo de configuración del módulo Login:
 * <pre><code>
 *    KeyStoreLogin {
 *        com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule required
 *            keyStoreFile="${user.dir}${/}security${/}.keystore";
 *    };
 * </code></pre>
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see LoginModule
 */
public class KeyStoreLoginModule implements LoginModule {

    private static final String CLASS_NAME = KeyStoreLoginModule.class.getName();

    /**
     * Nombre de propiedad del archivo de almacén de claves
     */
    public static final String KEY_STORE_FILE_PROPERTY_NAME = "keyStoreFile";

    /**
     * Tipo del almacén de claves. Sólo se da soporte a JKS
     */
    public static final String KEYSTORE_TYPE = "JKS";
```

```
/**
 * El nombre de archivo de almacén de claves predeterminado
 */
public static final String DEFAULT_KEY_STORE_FILE = "${java.home}${/}.keystore";

private CallbackHandler handler;

private Subject subject;

private boolean debug = false;

private Set principals = new HashSet();

private Set publicCreds = new HashSet();

private Set privateCreds = new HashSet();

protected KeyStore keyStore;

/**
 * Crea un nuevo KeyStoreLoginModule.
 */
public KeyStoreLoginModule() {
}

/**
 * Inicializa el módulo de inicio de sesión.
 *
 * @see LoginModule#initialize(Subject, CallbackHandler, Map, Map)
 */
public void initialize(Subject sub, CallbackHandler callbackHandler,
        Map mapSharedState, Map mapOptions) {

    // inicializar todas las opciones configuradas
    debug = "true".equalsIgnoreCase((String) mapOptions.get("debug"));

    if (sub == null)
        throw new IllegalArgumentException("Subject is not specified");

    if (callbackHandler == null)
        throw new IllegalArgumentException(
        "CallbackHander no especificado");

    // Obtener la vía de acceso del almacén de claves
    String sKeyStorePath = (String) mapOptions
        .get(KEY_STORE_FILE_PROPERTY_NAME);

    // Si no hay ninguna vía de acceso de almacén de claves, el valor predeterminado es
    // el archivo .keystore en el directorio inicial de java
    if (sKeyStorePath == null) {
        sKeyStorePath = DEFAULT_KEY_STORE_FILE;
    }

    // Sustituir la variable de entorno del sistema
    sKeyStorePath = ObjectGridUtil.replaceVar(sKeyStorePath);

    File fileKeyStore = new File(sKeyStorePath);

    try {
        KeyStore store = KeyStore.getInstance("JKS");
        store.load(new FileInputStream(fileKeyStore), null);

        // Guardar el almacén de claves
        keyStore = store;

        if (debug) {
            System.out.println("Inicializar [KeyStoreLoginModule]: almacén de claves cargado satisfactoriamente");
        }
    }
    catch (Exception e) {
        ObjectGridRuntimeException re = new ObjectGridRuntimeException(
                "Failed to load keystore: " + fileKeyStore.getAbsolutePath());
        re.initCause(e);
        if (debug) {
            System.out.println("[KeyStoreLoginModule] initialize: Key store loading failed with exception "
                    + e.getMessage());
        }
    }

    this.subject = sub;
    this.handler = callbackHandler;
}

/**
 * Autentica un usuario basándose en el archivo de almacén de claves.
 *
 * @see LoginModule#login()
 */
public boolean login() throws LoginException {
```

```
    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: entry");
    }

    String name = null;
    char pwd[] = null;

    if (keyStore == null || subject == null || handler == null) {
        throw new LoginException("Module initialization failed");
    }

    NameCallback nameCallback = new NameCallback("Username:");
    PasswordCallback pwdCallback = new PasswordCallback("Password:", false);

    try {
        handler.handle(new Callback[] { nameCallback, pwdCallback });
    }
    catch (Exception e) {
        throw new LoginException("Callback failed: " + e);
    }

    name = nameCallback.getName();
    char[] tempPwd = pwdCallback.getPassword();

    if (tempPwd == null) {
        // tratar una contraseña NULL como una contraseña vacía
        tempPwd = new char[0];
    }
    pwd = new char[tempPwd.length];
    System.arraycopy(tempPwd, 0, pwd, 0, tempPwd.length);

    pwdCallback.clearPassword();

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: "
                + "user entered user name: " + name);
    }

    // Validar el nombre de usuario y la contraseña
    try {
        validate(name, pwd);
    }
    catch (SecurityException se) {
        principals.clear();
        publicCreds.clear();
        privateCreds.clear();
        LoginException le = new LoginException(
        "Exception encountered during login");
        le.initCause(se);

        throw le;
    }

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: exit");
    }
    return true;
}

/**
 * Indica si el usuario se acepta.
 * <p>
 * Este método sólo se invoca si el usuario es autenticado por todos los módulos del archivo
 * de configuración de inicio de sesión. Los objetos principales se añadirán al asunto
 * almacenado.
 *
 * @return false si por alguna razón los principales no se pueden añadir; de lo contrario,
 *         true
 *
 * @exception LoginException
 *                Se genera una LoginException si el asunto es de sólo lectura o si se
 *                encuentran excepciones no recuperables.
 *
 * @see LoginModule#commit()
 */
public boolean commit() throws LoginException {
    if (debug) {
        System.out.println("[KeyStoreLoginModule] commit: entry");
    }

    if (principals.isEmpty()) {
        throw new IllegalStateException("Commit is called out of sequence");
    }

    if (subject.isReadOnly()) {
        throw new LoginException("Subject is Readonly");
    }

    subject.getPrincipals().addAll(principals);
    subject.getPublicCredentials().addAll(publicCreds);
    subject.getPrivateCredentials().addAll(privateCreds);
```

```
    principals.clear();
    publicCreds.clear();
    privateCreds.clear();

    if (debug) {
        System.out.println("[KeyStoreLoginModule] commit: exit");
    }
    return true;
}

/**
 * Indica que el usuario no se acepta
 *
 * @see LoginModule#abort()
 */
public boolean abort() throws LoginException {
    boolean b = logout();
    return b;
}

/**
 * Cierra la sesión de usuario. Borrar todas las correlaciones.
 *
 * @see LoginModule#logout()
 */
public boolean logout() throws LoginException {


    // Borrar las variables de instancia
    principals.clear();
    publicCreds.clear();
    privateCreds.clear();

    // borrar correlaciones en el asunto
    if (!subject.isReadOnly()) {
        if (subject.getPrincipals() != null) {
            subject.getPrincipals().clear();
        }

        if (subject.getPublicCredentials() != null) {
            subject.getPublicCredentials().clear();
        }

        if (subject.getPrivateCredentials() != null) {
            subject.getPrivateCredentials().clear();
        }
    }
    return true;
}

/**
 * Valida el nombre de usuario y la contraseña basándose en el almacén de claves.
 *
 * @param userName nombre de usuario
 * @param password contraseña
 * @throws SecurityException si se encuentran excepciones
 */
private void validate(String userName, char password[])
    throws SecurityException {

    PrivateKey privateKey = null;

    // Obtener la clave privada del almacén de claves
    try {
        privateKey = (PrivateKey) keyStore.getKey(userName, password);
    }
    catch (NoSuchAlgorithmException nsae) {
        SecurityException se = new SecurityException();
        se.initCause(nsae);
        throw se;
    }
    catch (KeyStoreException kse) {
        SecurityException se = new SecurityException();
        se.initCause(kse);
        throw se;
    }
    catch (UnrecoverableKeyException uke) {
        SecurityException se = new SecurityException();
        se.initCause(uke);
        throw se;
    }

    if (privateKey == null) {
        throw new SecurityException("Invalid name: " + userName);
    }

    // Comprobar certificados
    Certificate certs[] = null;
    try {
        certs = keyStore.getCertificateChain(userName);
```

```
        }
        catch (KeyStoreException kse) {
            SecurityException se = new SecurityException();
            se.initCause(kse);
            throw se;
        }

        if (debug) {
            System.out.println("  Print out the certificates:");
            for (int i = 0; i < certs.length; i++) {
                System.out.println("  certificate " + i);
                System.out.println("    " + certs[i]);
            }
        }

        if (certs != null && certs.length > 0) {

            // Si el primer certificado es un X509Certificate
            if (certs[0] instanceof X509Certificate) {
                try {
                    // Obtener el primer certificado que representa el usuario
                    X509Certificate certX509 = (X509Certificate) certs[0];

                    // Crear un principal
                    X500Principal principal = new X500Principal(certX509
                            .getIssuerDN()
                            .getName());
                    principals.add(principal);

                    if (debug) {
                        System.out.println("  Principal added: " + principal);
                    }
                    // Crear un objeto de vía de acceso de certificación y añadirlo al
                    // conjunto de credenciales públicas
                    CertificateFactory factory = CertificateFactory
                        .getInstance("X.509");
                    java.security.cert.CertPath certPath = factory
                        .generateCertPath(Arrays.asList(certs));
                    publicCreds.add(certPath);

                    // Añadir la credencial privada al conjunto de credenciales privadas
                    privateCreds.add(new X500PrivateCredential(certX509,
                            privateKey, userName));

                }
                catch (CertificateException ce) {
                    SecurityException se = new SecurityException();
                    se.initCause(ce);
                    throw se;
                }
            }
            else {
                // El primer certificado no es un X509Certificate
                // Sólo añadimos el certificado al conjunto de credenciales públicas
                // y la clave privada al conjunto de credenciales privadas.
                publicCreds.add(certs[0]);
                privateCreds.add(privateKey);
            }
        }
    }
}
```

Utilización del plug-in de autenticador LDAP

Se le proporciona la implementación predeterminada com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator para manejar la autenticación de nombre de usuario y contraseña en un servidor LDAP. Esta implementación utiliza el módulo de inicio de sesión LDAPLogin para conectar al usuario a un servidor LDAP (Lightweight Directory Access Protocol). El siguiente fragmento de código demuestra cómo se implementa el método authenticate:

```
/**
* @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);
    }
}
```

Asimismo, eXtreme Scale se entrega con un módulo de inicio de sesión com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule con esta finalidad. Debe proporcionar las siguientes dos opciones en el archivo de configuración del inicio de sesión JAAS.

- providerURL: URL del proveedor del servidor LDAP.
- factoryClass: clase de implementación de fábrica de contexto LDAP.

El módulo LDAPLoginModule llama al método com.ibm.websphere.objectgrid.security.plugins.builtins. Método LDAPAuthenticationHelper.authenticate. El siguiente fragmento de código muestra cómo implementar el método authenticate de LDAPAuthenticationHelper.

```
/**
* Autentica el usuario en el directorio LDAP.
* @param user El ID de usuario, por ejemplo 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 correcta, el ID de usuario y la contraseña se consideran válidos. El módulo de inicio de sesión obtiene la información de ID y de departamento a partir de este método authenticate. El módulo de inicio de sesión crea dos principales: SimpleUserPrincipal y SimpleDeptPrincipal. Puede usar el sujeto autenticado para autorización de grupos (en este caso, el departamento es un grupo) y para autorización individual.

El ejemplo siguiente muestra una configuración del módulo de inicio de sesión que se utiliza para iniciar sesión en el servidor LDAP:

```
LDAPLogin { com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule required
    providerURL="ldap://directory.acme.com:389/"
    factoryClass="com.sun.jndi.ldap.LdapCtxFactory";
};
```

En la configuración anterior, el servidor LDAP apunta al `ldap://directory.acme.com:389/server`. Cambie este valor por el de su servidor LDAP. Este módulo de inicio de sesión utiliza el ID y la contraseña proporcionados para conectarse al servidor LDAP. Esta implementación es sólo para realizar pruebas.

Uso del plug-in de autenticador de WebSphere Application Server

Además, eXtreme Scale proporciona la implementación incorporada com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator para utilizar la infraestructura de seguridad de WebSphere Application Server. Esta implementación incorporada se puede utilizar cuando las siguientes condiciones son verdaderas.

1. La seguridad global de WebSphere Application Server está activa.
2. Todos los clientes y servidores de eXtreme Scale se inician en las JVM de WebSphere Application Server.
3. Estos servidores de aplicaciones están en el mismo dominio de seguridad.
4. El cliente eXtreme Scale ya ha sido autenticado en WebSphere Application Server.

El cliente puede utilizar la clase com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator para generar una credencial. El servidor utiliza esta clase de implementación Authenticator para autenticar la credencial. Si la señal se autentica correctamente, se devuelve un objeto Subject.

Este escenario saca partido del hecho que el cliente ya ha sido autenticado. Puesto que los servidores de aplicaciones que tienen los servidores en el mismo dominio de seguridad que los servidores de aplicaciones que alojan los clientes, las señales de seguridad se pueden propagar del cliente al servidor, de forma que no es necesario que se vuelva a autenticar el mismo registro de usuarios.

Uso del plug-in de autenticador de Tivoli Access Manager

Tivoli Access Manager se utiliza ampliamente como un servidor de seguridad. También puede implementar el autenticador utilizando los módulos de inicio de sesión proporcionados de Tivoli Access Manager.

Para autenticar un usuario para Tivoli Access Manager, aplique el módulo de inicio de sesión com.tivoli.mts.PDLoginModule, que requiere que la aplicación que llama proporcione la siguiente información:

1. Un nombre principal, especificado como un nombre abreviado o un nombre X.500 (DN)
2. Una contraseña

El módulo de inicio de sesión autentica el principal y devuelve la credencial de Tivoli Access Manager. El módulo de inicio de sesión espera que la aplicación que llama proporcione la información siguiente:

1. El nombre de usuario, a través de un objeto javax.security.auth.callback.NameCallback.
2. La contraseña, a través de un objeto javax.security.auth.callback.PasswordCallback.

Cuando la credencial de Tivoli Access Manager se recupera correctamente, el módulo JAAS LoginModule crea un objeto Subject y PDPrincipal. No se proporciona ningún objeto incorporado para la autenticación de Tivoli Access Manager, porque sólo se proporciona con el módulo PDLoginModule. Consulte la publicación IBM Tivoli Access Manager Authorization Java Classes Developer Reference si desea más detalles.

Conexión a WebSphere eXtreme Scale de forma segura

Para conectar de forma segura un cliente de eXtreme Scale con un servidor, puede utilizar cualquier método connect en la interfaz ObjectGridManager que toma un objeto ClientSecurityConfiguration. A continuación, aparece un breve ejemplo:

```
public ClientClusterContext connect(String catalogServerAddresses,
   ClientSecurityConfiguration securityProps,
     URL overRideObjectGridXml) lanza ConnectException;
```

Este método toma un parámetro del tipo ClientSecurityConfiguration, que es una interfaz que representa una configuración de seguridad de cliente. Puede utilizar la API pública com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory para crear una instancia con valores predeterminados, o puede crear una instancia pasando el archivo de propiedades de cliente de WebSphere eXtreme Scale. Este archivo contiene las siguientes propiedades relacionadas con la autenticación. El valor marcado con un signo más (+) es el valor predeterminado.

- `securityEnabled (true, false+):` esta propiedad indica si la seguridad está habilitada. Cuando un cliente se conecta a un servidor, el valor securityEnabled en el lado del cliente y del servidor deben ser ambos `true` o `false`. Por ejemplo, si la seguridad del servidor conectado está habilitada, el cliente debe tener esta propiedad establecida en true para poder conectarse al servidor.
- authenticationRetryCount (un valor entero, 0+): esta propiedad determina cuántos reintentos se realizan para el inicio de sesión cuando caduca una credencial. Si el valor es 0, no se realiza ningún reintento. El reintento de autenticación sólo se aplica en el caso de que la credencial haya caducado. Si la credencial no es válida, no se produce ningún reintento. Su aplicación es responsable de reintentar la operación.

Después de crear un objeto com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration, establezca el objeto credentialGenerator en el cliente mediante el método siguiente:

```
/**
* Establezca el objeto {@link CredentialGenerator} para este cliente.
* @param generator El objeto CredentialGenerator asociado con este cliente
*/
void setCredentialGenerator(CredentialGenerator generator);
```

También, puede establecer el objeto CredentialGenerator en el archivo de propiedades de cliente de WebSphere eXtreme Scale, del modo siguiente:

- credentialGeneratorClass: nombre de la implementación de clase del objeto CredentialGenerator. Debe tener un constructor predeterminado.
- credentialGeneratorProps: propiedades de la clase CredentialGenerator. Si el valor es nulo, se establece en el objeto CredentialGenerator construido mediante el método setProperties(String).

En el ejemplo siguiente se crea una instancia de ClientSecurityConfiguration, que se utiliza para conectar con el servidor.

```
/**
* Obtiene un ClientClusterContext seguro
* @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 llama a la conexión, el cliente de WebSphere eXtreme Scale llama al método CredentialGenerator.getCredential para obtener la credencial del cliente. Esta credencial de autenticación se envía al servidor con la solicitud de conexión.

Uso de una instancia CredentialGenerator diferente por sesión

En algunos casos, un cliente de WebSphere eXtreme Scale representa sólo una identidad de cliente, pero en otros, podría representar varias identidades. Aquí, aparece un escenario del último caso: un cliente de WebSphere eXtreme Scale se crea y comparte en un servidor web. Todos los servlets de este servidor web utilizan este cliente de WebSphere eXtreme Scale. Puesto que cada uno de los servlets representa un cliente web diferente, utilice distintas credenciales al enviar solicitudes a servidores WebSphere eXtreme Scale.

WebSphere eXtreme Scale puede cambiar la credencial en el nivel de la sesión. Cada sesión puede utilizar un objeto CredentialGenerator diferente. Por lo tanto, los escenarios anteriores se pueden implementar dejando al servlet obtener una sesión con un objeto CredentialGenerator distinto. El ejemplo siguiente ilustra el método ObjectGrid.getSession(CredentialGenerator) en la interfaz ObjectGridManager.

```
/**
     * Obtener una sesión utilizando <code>CredentialGenerator</code>.
     * <p>
     * Este método sólo puede llamarlo el cliente ObjectGrid en un entorno de
     * cliente-servidor de ObjectGrid. Si se utiliza ObjectGrid en un modelo local, es decir,
     * dentro de la misma JVM donde no existe ningún cliente ni servidor, se debe utilizar el
     * plug-in <code>getSession(Subject)</code>
     * o <code>SubjectSource</code> para proteger el ObjectGrid.
     *
     * <p>Si el método <code>initialize()</code> no ha sido invocado antes de la
```

```
     * primera invocación de <code>getSession</code>, se producirá una inicialización
     * implícita. Esto garantiza que toda la configuración se completa
     * antes de que sea necesario cualquier uso del tiempo de ejecución.</p>
     *
     * @param credGen <code>CredentialGenerator</code> para generar una credencial
     *                para la sesión devuelta.
     *
     * @return Una instancia de <code>Session</code>
     *
     * @throws ObjectGridException si se produce un error durante el proceso
     * @throws TransactionCallbackException si <code>TransactionCallback</code>
     *         emite una excepción
     * @throws IllegalStateException si se llama a este método después de
     *         que se llame al método <code>destroy()</code>.
     *
     * @see #destroy()
     * @see #initialize()
     * @see CredentialGenerator
     * @see Session
     * @since WAS XD 6.0.1
 */
Session getSession(CredentialGenerator credGen) throws
ObjectGridException, TransactionCallbackException;
```

A continuación se muestra un ejemplo:

```
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();

CredentialGenerator credGenManager = new UserPasswordCredentialGenerator("manager", "xxxxxx");
CredentialGenerator credGenEmployee = new UserPasswordCredentialGenerator("employee", "xxxxxx");

ObjectGrid og = ogManager.getObjectGrid(ctx, "accounting");

// Obtiene una sesión con CredentialGenerator;
Session session = og.getSession(credGenManager );

// Obtiene la correlación de empleados
ObjectMap om = session.getMap("employee");

// inicia una transacción.
session.begin();

Object rec1 = map.get("xxxxxx");

session.commit();

// Obtiene otra sesión con otro CredentialGenerator;
session = og.getSession(credGenEmployee );

// Obtiene la correlación de empleados
om = session.getMap("employee");

// inicia una transacción.
session.begin();

Object rec2 = map.get("xxxxx");

session.commit();
```

Si utiliza el método ObjectGird.getSession para obtener un objeto Session, la sesión utiliza el objeto CredentialGenerator establecido en el objeto ClientConfigurationSecurity. El método ObjectGrid.getSession(CredentialGenerator) sustituye al objeto CredentialGenerator establecido en el objeto ClientSecurityConfiguration.

Si puede volver a utilizar el objeto Session, se favorece el rendimiento. Sin embargo, llamar al método ObjectGrid.getSession(CredentialGenerator) no es muy caro. La principal sobrecarga es el aumento del tiempo de recogida de basura del objeto. Asegúrese de que libera las referencias después de terminar con los objetos Session. Por lo general, si el objeto Session puede compartir la identidad, intente reutilizarlo. En caso contrario, utilice el método ObjectGrid.getSession(CredentialGenerator).

Información relacionada:
API de credencial

Programación de autorización de cliente

WebSphere eXtreme Scale soporta la autorización JAAS (Java Authentication and Authorization Service) que está preparada para utilizarse y también soporta la autorización personalizada utilizando la interfaz ObjectGridAuthorization.

El plug-in ObjectGridAuthorization se utiliza para autorizar los accesos de ObjectGrid, ObjectMap y JavaMap a los principales representados por un objeto Subject de manera personalizada. Una implementación típica de este plug-in consiste en recuperar los principales del objeto Subject y, a continuación, comprobar si se otorgan a los principales los permisos especificados.

Un permiso pasado en el método checkPermission(Subject, Permission) puede ser uno de los permisos siguientes:

- MapPermission
- ObjectGridPermission
- ServerMapPermission
- AgentPermission

Consulte la documentación de la API ObjectGridAuthorization para obtener más información.

MapPermission

La clase pública com.ibm.websphere.objectgrid.security.MapPermission representa los permisos de los recursos de ObjectGrid, específicamente los métodos de interfaces ObjectMap o JavaMap. WebSphere eXtreme Scale define las siguientes series de permiso para acceder a los métodos de ObjectMap y JavaMap:

- **leer**: permiso para leer los datos de la correlación. La constante entera se define como `MapPermission.READ`.
- **grabar**: permiso para actualizar los datos de la correlación. La constante entera se define como `MapPermission.WRITE`.
- **insertar**: permiso para insertar los datos en la correlación. La constante entera se define como `MapPermission.INSERT`.
- **eliminar**: permiso para eliminar los datos de la correlación. La constante entera se define como `MapPermission.REMOVE`.
- **invalidar**: permiso para invalidar los datos de la correlación. La constante entera se define como `MapPermission.INVALIDATE`.
- **todo**: todos los permisos anteriores: leer, grabar, insertar, eliminar e invalidar. La constante entera se define como `MapPermission.ALL`.

Consulte la documentación de la API MapPermission para obtener más información.

Puede construir un objeto MapPermission; para ello, pase el nombre de la correlación ObjectGrid totalmente calificado (con el formato [nombre_ObjectGrid].[nombre_ObjectMap]) y la serie de permiso o valor entero. Una serie de permiso puede ser una serie delimitada por comas de las series de permiso anteriores como leer, insertar, o todos ellos. Un valor entero de permiso puede ser cualquier constante entera de los permisos mencionados anteriormente o

un valor matemático de varias constantes enteras de permisos, como MapPermission.READ | MapPermission.WRITE.

La autorización se produce cuando se llama el método ObjectMap o JavaMap. El tiempo de ejecución de eXtreme Scale comprueba los distintos permisos para los métodos diferentes. Si los permisos requeridos no se conceden al cliente, se produce una excepción AccessControlException.

Tabla 11. Lista de métodos y MapPermission requeridos

Permiso	**ObjectMap/JavaMap**
leer	Boolean containsKey(Object)
	Boolean equals(Object)
	Object get(Object)
	Object get(Object, Serializable)
	List getAll(List)
	List getAll(List keyList, Serializable)
	List getAllForUpdate(List)
	List getAllForUpdate(List, Serializable)
	Object getForUpdate(Object)
	Object getForUpdate(Object, Serializable)
	public Object getNextKey(long)
grabar	Object put(Object key, Object value)
	void put(Object, Object, Serializable)
	void putAll(Map)
	void putAll(Map, Serializable)
	void update(Object, Object)
	void update(Object, Object, Serializable)
insertar	public void insert (Object, Object)
	void insert(Object, Object, Serializable)
eliminar	Object remove (Object)
	void removeAll(Collection)
	void clear()
invalidar	public void invalidate (Object, Boolean)
	void invalidateAll(Collection, Boolean)
	void invalidateUsingKeyword(Serializable)
	int setTimeToLive(int)

La autorización se basa únicamente en el método utilizado y no en lo que el método hace. Por ejemplo, un método put puede insertar o actualizar un registro en función de si el registro existe. No obstante, los casos de inserción o actualización no se distinguen.

Puede conseguirse un tipo de operación mediante la combinación de otros tipos. Por ejemplo, una actualización puede conseguirse mediante una operación de eliminación primero, y después una inserción. Tenga en cuenta estas combinaciones al diseñar las políticas de autorización.

ObjectGridPermission

com.ibm.websphere.objectgrid.security.ObjectGridPermission representa permisos para ObjectGrid:

- Consulta: permiso para crear una consulta de objeto o una consulta de entidad. La constante entera se define como ObjectGridPermission.QUERY.
- Correlación dinámica: permiso para crear una correlación dinámica basada en la plantilla de correlación. La constante entera se define como ObjectGridPermission.DYNAMIC_MAP.

Consulte la documentación de la API ObjectGridPermission para obtener más información.

En la siguiente tabla se resumen los métodos y los ObjectGridPermission requeridos:

Tabla 12. Lista de métodos y ObjectGridPermission requeridos

Acción de permiso	Métodos
consulta	com.ibm.websphere.objectgrid.Session.createObjectQuery(String)
consulta	com.ibm.websphere.objectgrid.em.EntityManager.createQuery(String)
dynamicmap	com.ibm.websphere.objectgrid.Session.getMap(String)

ServerMapPermission

ServerMapPermission representa permisos para ObjectMap alojado en un servidor. El nombre del permiso es el nombre completo del nombre de la correlación ObjectGrid. Tiene las acciones siguientes:

- replicar: permiso para replicar una correlación del servidor en una memoria caché cercana.
- dynamicIndex: permiso para que un cliente pueda crear o eliminar un índice dinámico en el servidor.

Consulte la documentación de la API ServerMapPermission para obtener más información. Los métodos detallados, que requieren diferentes ServerMapPermission, se listan en la siguiente tabla:

Tabla 13. Permisos para un ObjectMap alojado en un servidor

Acción de permiso	Métodos
replicar	com.ibm.websphere.objectgrid.ClientReplicableMap.enableClientReplication(Mode, int[], ReplicationMapListener)
dynamicIndex	com.ibm.websphere.objectgrid.BackingMap.createDynamicIndex(String, Boolean, String, DynamicIndexCallback)
dynamicIndex	com.ibm.websphere.objectgrid.BackingMap.removeDynamicIndex(String)

AgentPermission

Un AgentPermission representa permisos para los agentes de datagrid. El nombre del permiso es el nombre completo de la correlación ObjectGrid, y la acción es una serie delimitada por comas de los nombres de clase o nombres de paquete de la implementación del agente.

Consulte la documentación de la API AgentPermission si desea más información.

Los siguientes métodos de la clase com.ibm.websphere.objectgrid.datagrid.AgentManager requieren AgentPermission.

```
com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent, Collection)
```

```
com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent)
com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)
com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)
```

Mecanismos de autorización

WebSphere eXtreme Scale soporta dos tipos de mecanismos de autorización: la autorización JAAS (Java Authentication and Authorization Service) y la autorización personalizada. Estos mecanismos se aplican a todas las autorizaciones. La autorización JAAS aumenta las políticas de seguridad Java con controles de acceso centrados en el usuario. Los permisos se conceden no sólo en función del código que se ejecute, sino en función de quién lo ejecute. La autorización JAAS forma parte de SDK versión 5 y posterior.

De forma adicional, WebSphere eXtreme Scale también soporta la autorización personalizada con el siguiente plug-in:

- ObjectGridAuthorization: forma personalizada de autorizar el acceso a todos los artefactos.

Puede implementar su propio mecanismo de autorización si no desea utilizar la autorización JAAS. Mediante el uso de un mecanismo de autorización personalizado, puede utilizar la base de datos de políticas, o Tivoli Access Manager para gestionar las autorizaciones.

Puede configurar el mecanismo de autorización de dos maneras:

- Configuración XML

  Puede utilizar el archivo XML ObjectGrid para definir un ObjectGrid y establecer el mecanismo de autorización en AUTHORIZATION_MECHANISM_JAAS o AUTHORIZATION_MECHANISM_CUSTOM. A continuación se muestra el archivo secure-objectgrid-definition.xml que se utiliza en ObjectGridSample de aplicación de empresa:

  ```
  <objectGrids>
   <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
    <bean id="TransactionCallback"
  classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- Configuración mediante programación

  Si desea crear un ObjectGrid mediante un método ObjectGrid.setAuthorizationMechanism(int), puede llamar al método siguiente para establecer el mecanismo de autorización. La llamada a este método sólo se aplica al modelo de programación de WebSphere eXtreme Scale local cuando se crea directamente una instancia de ObjectGrid:

  ```
  /**
   * Establecer mecanismo de autorización. El valor predeterminado es
   * com.ibm.websphere.objectgrid.security.SecurityConstants.
   * AUTHORIZATION_MECHANISM_JAAS.
   * @param authMechanism El mecanismo de autorización de correlación
  */
  void setAuthorizationMechanism(int authMechanism);
  ```

Autorización JAAS

Un objeto javax.security.auth.Subject representa un usuario autenticado. Subject consta de un conjunto de principales y cada principal representa una identidad

para ese usuario. Por ejemplo, Subject puede tener un principal de nombre, por ejemplo, Cristina López, y un principal de grupo, por ejemplo, gestor.

Si usa la política de autorización JAAS, los permisos se pueden conceder a principales específicos. WebSphere eXtreme Scale asocia el Subject con el contexto de control de accesos actual. Para cada llamada al método ObjectMap o Javamap, el tiempo de ejecución de Java determina automáticamente si la política otorga el permiso necesario sólo a un Principal específico y, de esta forma, la operación sólo está permitida si el Subject asociado al contexto de control de accesos contiene el Principal designado.

Debe estar familiarizado con la sintaxis de la política del archivo de políticas. Si desea obtener una descripción detallada de la autorización JAAS, consulte la publicación JAAS Reference Guide.

WebSphere eXtreme Scale tiene una base de código especial que se utiliza para comprobar la autorización JAAS para las llamadas a los métodos ObjectMap y JavaMap. Esta base de código especial es `http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction`. Utilice esta base de código al conceder permisos ObjectMap o JavaMap a los principales. Este código especial se creó porque el archivo JAR (Java Archive) para eXtreme Scale se otorga con todos los permisos.

La plantilla de la política para conceder el permiso MapPermission es:

```
grant codeBase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
   <Principal field(s)>{
    permission com.ibm.websphere.objectgrid.security.MapPermission
               "[ObjectGrid_name].[ObjectMap_name]", "action";
    ....
    permission com.ibm.websphere.objectgrid.security.MapPermission
               "[ObjectGrid_name].[ObjectMap_name]", "action";
  };
```

Un campo de principal se parece al ejemplo siguiente:

```
principal Principal_class "principal_name"
```

En esta política, sólo se otorgan los permisos de inserción y lectura a estas cuatro correlaciones a un principal determinado. El otro archivo de políticas, `fullAccessAuth.policy`, otorga a un principal todos los permisos a estas correlaciones. Antes de ejecutar la aplicación, cambie el nombre del principal (principal_name) y la clase del principal por los valores correspondientes. El valor de principal_name depende del Registro de usuarios. Por ejemplo, si se utiliza el sistema operativo local como registro de usuarios, el nombre de máquina es MACH1, el ID de usuario es user1 y el nombre de principal es `MACH1/user1`.

La política de autorización JAAS se puede colocar directamente en el archivo de políticas Java, o se puede colocar en un archivo de autorización JAAS separado y, a continuación, establecerlo de cualquiera de estas dos maneras:

- Utilice el siguiente argumento de JVM :
  `-Djava.security.auth.policy=file:[JAAS_AUTH_POLICY_FILE]`
- Utilice la propiedad siguiente del archivo java.security:
  `-Dauth.policy.url.x=file:[JAAS_AUTH_POLICY_FILE]`

Autorización personalizada ObjectGrid

El plug-in ObjectGridAuthorization se utiliza para autorizar a ObjectGrid los accesos ObjectMap y JavaMap a los principales, representados por un objeto Subject de una manera personalizada. Una implementación típica de este plug-in es recuperar los principales del objeto Subject y después comprobar si se han concedido los permisos especificados a los principales.

Un permiso pasado al método checkPermission(Subject, Permission) puede ser uno de los siguientes:

- MapPermission
- ObjectGridPermission
- AgentPermission
- ServerMapPermission

Consulte la documentación de la API ObjectGridAuthorization para obtener más información.

El plug-in ObjectGridAuthorization puede configurarse de las siguientes maneras:

- Configuración XML

  Puede utilizar el archivo XML ObjectGrid para definir un plug-in ObjectAuthorization. Ejemplo:

```
<objectGrids>
 <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
...
  <bean id="ObjectGridAuthorization"
className="com.acme.ObjectGridAuthorizationImpl" />
</objectGrids>
```

- Configuración mediante programación

  Si desea crear un ObjectGrid mediante el método de API ObjectGrid.setObjectGridAuthorization(ObjectGridAuthorization), puede llamar al método siguiente para establecer el plug-in de autorización. Este método sólo se aplica al modelo de programación eXtreme Scale local cuando cree directamente la instancia de ObjectGrid.

```
/**
     * Establece <code>ObjectGridAuthorization</code> para esta
instancia de ObjectGrid.
     * <p>
     * Al pasar <code>null</code> a este método elimina un objeto
     * <code>ObjectGridAuthorization</code> establecido
anteriormente de una invocación anterior de este método
     * e indica que este <code>ObjectGrid</code> no está asociado a un
     * objeto <code>ObjectGridAuthorization</code>.
     * <p>
     * Este método sólo debe utilizarse cuando se ha habilitado la seguridad
ObjectGrid. Si
     * la seguridad ObjectGrid está inhabilitada, el objeto
<code>ObjectGridAuthorization</code> proporcionado
     * no se utilizará.
     * <p>
     * Puede utilizarse un plug-in <code>ObjectGridAuthorization</code> para autorizar
     * el acceso a ObjectGrid y correlaciones. Consulte
<code>ObjectGridAuthorization</code> para obtener más información.
     *
     * <p>
     * Desde XD 6.1, <code>setMapAuthorization</code> está en desuso
     * y se recomienda el uso de <code>setObjectGridAuthorization</code>.
No obstante,
     * si el plug-in <code>MapAuthorization</code> y el plug-in
<code>ObjectGridAuthorization</code>
     * se utilizan, ObjectGrid usará el
<code>MapAuthorization</code> proporcionado para autorizar los
accesos a las correlaciones,
     * aunque esté en desuso.
     * <p>
     * Para evitar una excepción <code>IllegalStateException</code>,
este método
     * debe llamarse antes que al método <code>initialize()</code>. Recuerde
también
     * que los métodos <code>getSession</code> llaman implícitamente
```

```
     * al método <code>initialize()</code> si debe llamarlo la
     * aplicación.
     *
     * @param ogAuthorization el plug-in <code>ObjectGridAuthorization</code>
     *
     * @throws IllegalStateException si se llama a este método después de
     * llamar al método <code>initialize()</code>.
     *
     * @see #initialize()
     * @see ObjectGridAuthorization
     * @since WAS XD 6.1
     */
    void setObjectGridAuthorization(ObjectGridAuthorization ogAuthorization);
```

Implementación de ObjectGridAuthorization

El tiempo de ejecución de WebSphere eXtreme Scale llama al método Boolean checkPermission(Subject subject, Permission permission) de la interfaz ObjectGridAuthorization para comprobar si el objeto de asunto pasado tiene el permiso pasado. La implementación de la interfaz ObjectGridAuthorization devuelve true, si el objeto tiene el permiso, y false, si no lo tiene.

Una implementación típica de este plug-in es recuperar los principales del objeto Subject y comprobar si los permisos especificados se han concedido a los principales mediante la consulta de políticas específicas. Estas políticas las definen los usuarios. Por ejemplo, las políticas se pueden definir en una base de datos, en un archivo plano, o un servidor de políticas de Tivoli Access Manager.

Por ejemplo, se puede utilizar el servidor de políticas Tivoli Access Manager para gestionar la política de autorización y utilizar su API para autorizar el acceso. Si desea saber cómo utilizar las API de Tivoli Access Manager Authorization, consulte IBM Tivoli Access Manager Authorization Java Classes Developer Reference para obtener más detalles.

Esta implementación de ejemplo tiene las siguientes presunciones:

- Sólo se comprueba la autorización de MapPermission. Para los demás permisos, siempre devuelve el valor `true`.
- El objeto Subject contiene un principal com.tivoli.mts.PDPrincipal.
- El servidor de políticas Tivoli Access Manager ha definido los siguientes permisos para el objeto de nombre ObjectMap o JavaMap. El objeto definido en el servidor de políticas debe tener el mismo nombre que el nombre de ObjectMap o JavaMap con el formato [nombre_ObjectGrid].[nombre_ObjectMap]. El permiso es el primer carácter de las series de permisos que se definen en el permiso MapPermission. Por ejemplo, el permiso "r" definido en el servidor de políticas representa el permiso read (lectura) para la correlación ObjectMap.

El siguiente fragmento de código muestra cómo implementar el método checkPermission:

```
/**
* @see com.ibm.websphere.objectgrid.security.plugins.
*   MapAuthorization#checkPermission
* (javax.security.auth.Subject, com.ibm.websphere.objectgrid.security.
*   MapPermission)
*/
public boolean checkPermission(final Subject subject,
 Permission p) {

  // Para non-MapPermission, siempre se da la autorización.
  if (!(p instanceof MapPermission)){
    return true;
  }
```

```
  MapPermission permission = (MapPermission) p;

  String[] str = permission.getParsedNames();

  StringBuffer pdPermissionStr = new StringBuffer(5);
  for (int i=0; i<str.length; i++) {
    pdPermissionStr.append(str[i].substring(0,1));
  }

  PDPermission pdPerm = new PDPermission(permission.getName(),
  pdPermissionStr.toString());

  Set principals = subject.getPrincipals();

  Iterator iter= principals.iterator();
  while(iter.hasNext()) {
    try {
      PDPrincipal principal = (PDPrincipal) iter.next();
      if (principal.implies(pdPerm)) {
        return true;
      }
    }
    catch (ClassCastException cce) {
      // Handle exception
    }
  }
  return false;
}
```

Información relacionada:

Módulo 4: Utilizar autorización JAAS (Java Authentication and Authorization Service) en WebSphere Application Server
Ahora que ha configurado la autenticación para clientes, puede configurar la autenticación para otorgar a distintos usuarios diversos permisos. Por ejemplo, es posible que un usuario operator solo pueda visualizar datos, mientras que un usuario administrador puede realizar todas las operaciones.

Autenticación de la cuadrícula de datos

Puede utilizar el plug-in de gestor de señales seguro para habilitar la autenticación de servidor a servidor, que requiere la implementación de la interfaz SecureTokenManager.

El método generateToken(Object) toma un objeto y, a continuación, genera una señal que los otros no pueden entender. El método verifyTokens(byte[]) realiza el proceso inverso: convierte la señal en el objeto original.

Una implementación sencilla de SecureTokenManager utiliza un algoritmo de codificación sencillo como, por ejemplo, un algoritmo XOR, para codificar el objeto en un formato serializado y, a continuación, utilizar el algoritmo de decodificación correspondiente para descifrar la señal. Esta implementación no es segura y es fácil quebrantarla.

Implementación predeterminada de WebSphere eXtreme Scale

WebSphere eXtreme Scale proporciona una implementación disponible de forma inmediata para esta interfaz. Esta implementación predeterminada utiliza un par de claves para firmar y verificar la firma y utiliza una clave secreta para cifrar el contenido. Cada servidor tiene un almacén de claves de tipo JCKES donde se almacena el par de claves, una clave privada y una clave pública, y un clave secreta. El almacén de claves tiene que ser de tipo JCKES para poder almacenar las claves secretas. Estas claves se utilizan para cifrar y firmar o verificar la serie

secreta en el envío. Además, la señal se asocia con un tiempo de caducidad. En el extremo receptor, los datos se verifican, se descifran y se comparan con la serie secreta del receptor. Los protocolos de comunicación SSL (Secure Sockets Layer) no son obligatorios para la autenticación entre un par de servidores porque las claves privadas y públicas sirven para ese mismo propósito. No obstante, si la comunicación del servidor no está cifrada, los datos podrían robarse con sólo observar la comunicación. Como la señal caduca pronto, la amenaza de ataque de reproducción se minimiza. Esta posibilidad disminuye en gran medida si todos los servidores se despliegan detrás de un cortafuegos.

La desventaja de este enfoque es que los administradores de WebSphere eXtreme Scale deben generar claves y transportarlas a todos los servidores, que pueden provocar una violación de seguridad durante el transporte.

Programación de la seguridad local

WebSphere eXtreme Scale proporciona varias puntos finales de seguridad que permiten integrar mecanismos personalizados. En el modelo de programación local, la principal función se seguridad es la autorización, que no tiene soporte de autenticación. Debe autenticar fuera de WebSphere Application Server. No obstante, se proporcionan plug-ins con el fin de obtener y validar objetos Subject.

Autenticación

En el modelo de programación local, eXtreme Scale no proporciona ningún mecanismo de autenticación, sino que se basa en el entorno, ya sean servidores de aplicación o aplicaciones, para realizar la autenticación. Cuando se utiliza eXtreme en WebSphere Application Server o WebSphere Extended Deployment, las aplicaciones pueden utilizar el mecanismo de autenticación de seguridad de WebSphere Application Server. Cuando eXtreme Scale se ejecuta en un entorno de Java 2 Platform, Standard Edition (J2SE), la aplicación debe gestionar las autenticaciones con la autenticación JAAS (Java Authentication and Authorization Service) u otro mecanismo de autenticación. Para obtener más información sobre cómo utilizar la autenticación JAAS, consulte la publicación JAAS Reference Guide. El contrato entre una aplicación y una instancia ObjectGrid es el objeto javax.security.auth.Subject. Una vez que el servidor de aplicaciones o la aplicación ha autenticado al cliente, la aplicación puede recuperar el objeto javax.security.auth.Subject autenticado y utilizar este objeto Subject para obtener una sesión de la instancia de ObjectGrid mediante la invocación del método ObjectGrid.getSession(Subject). Este objeto Subject se utiliza para autorizar los accesos a los datos de la correlación. Este contrato se denomina mecanismo de paso de sujetos. El ejemplo siguiente ilustra la API ObjectGrid.getSession(Subject).

```
/**
 * Esta API permite que la memoria caché utilice un sujeto específico en lugar del
 * configurado en ObjectGrid para obtener una sesión.
 * @param subject
 * @return Una intancia de sesión
 * @throws ObjectGridException
 * @throws TransactionCallbackException
 * @throws InvalidSubjectException - el asunto pasado no es válido según
 * el mecanismo SubjectValidation.
 */
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;
```

El método ObjectGrid.getSession() de la interfaz ObjectGrid también puede utilizarse para obtener un objeto Session:

```
/**
 * Este método devuelve un objeto Session que puede utilizar una única hebra cada vez.
 * No se puede compartir este objeto Session entre las hebras sin colocar una
 * sección crítica a su alrededor. Mientras que la infraestructura principal
 * permite que el objeto se mueva entre hebras, TransactionCallback y Loader
 * podrían impedir este uso, especialmente en entornos J2EE. Cuando la seguridad
 * está habilitada, este método utiliza el SubjectSource para obtener un
 * objeto Subject.
 *
 * Si el método initialize no se ha invocado antes de la primera
 * invocación de getSession, se producirá una inicialización implícita. Esta
 * inicialización garantiza que se completa toda la configuración antes
 * de que se necesite el uso de tiempo de ejecución.
 *
 * @see #initialize()
 * @return Una intancia de sesión
 * @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 se especifica en la documentación de la API, al habilitar la seguridad, este método utiliza el plug-in SubjectSource para obtener un objeto Subject. El plug-in SubjectSource es uno de los plug-ins de seguridad definidos en eXtreme Scale para dar soporte a la propagación de objetos Subject. Consulte los plug-ins relacionados con la seguridad para obtener más información. El método getSession(Subject) sólo puede llamarse en la instancia de ObjectGrid local. Si llama al método getSession(Subject) en un cliente en una configuración distribuida de eXtreme Scale, se genera una IllegalStateException.

Plug-ins de seguridad

WebSphere eXtreme Scale proporciona dos plug-ins de seguridad relacionados con el mecanismo de paso de asuntos: los plug-ins SubjectSource y SubjectValidation.

Plug-in SubjectSource

El plug-in SubjectSource, representado por la interfaz com.ibm.websphere.objectgrid.security.plugins.SubjectSource, es un plug-in que se utiliza para obtener un objeto Subject de un entorno de ejecución de eXtreme Scale. Este entorno puede ser una aplicación que utiliza el ObjectGrid o un servidor de aplicaciones que contiene la aplicación. Este plug-in SubjectSource es una alternativa al mecanismo de paso de sujetos. Si utiliza el mecanismo de paso de sujetos, la aplicación recupera el objeto Subject y lo utiliza para obtener el objeto de sesión ObjectGrid. Con el plug-in SubjectSource, la ejecución de eXtreme Scale recupera el objeto Subject y lo utiliza para obtener el objeto de sesión. El mecanismo de paso de sujetos otorga el control de objetos Subject a las aplicaciones, mientras que con el mecanismo de plug-in SubjectSource las aplicaciones no tienen que recuperar el objeto Subject. Puede utilizar el plug-in SubjectSource para obtener un objeto Subject que represente un cliente eXtreme Scale que se utilice para la autorización. Cuando se llama al método ObjectGrid.getSession, el Subject getSubject lanza una ObjectGridSecurityException si la seguridad está habilitada. WebSphere eXtreme Scale proporciona una implementación predeterminada de este plug-in: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl. Esta implementación se puede utilizar para recuperar un asunto llamante o un asunto RunAs de la hebra cuando una aplicación se ejecuta en WebSphere Application Server. Puede configurar esta clase en el archivo XML de descriptor de ObjectGrid como la clase de implementación de SubjectSource al utilizar eXtreme Scale en

WebSphere Application Server. El fragmento de código siguiente muestra el flujo principal del método WSSubjectSourceImpl.getSubject.

```
Subject s = null;
try {
  if (finalType == RUN_AS_SUBJECT) {
    // obtener el sujeto RunAs
    s = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
  }
  else if (finalType == CALLER_SUBJECT) {
    // obtener callersubject
    s = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
  }
}
catch (WSSecurityException wse) {
  throw new ObjectGridSecurityException(wse);
}

return s;
```

Si desea obtener más detalles, consulte la documentación de la API del plug-in SubjectSource y la implementación WSSubjectSourceImpl.

Plug-in SubjectValidation

El plug-in SubjectValidation, que está representado por la interfaz com.ibm.websphere.objectgrid.security.plugins.SubjectValidation, es otro plug-in de seguridad. El plug-in SubjectValidation puede utilizarse para validar que un objeto javax.security.auth.Subject, pasado a ObjectGrid o recuperado mediante el plug-in SubjectSource, es un Subject válido que no se ha manipulado de forma indebida.

El método SubjectValidation.validateSubject(Subject) de la interfaz SubjectValidation toma un objeto Subject y devuelve un objeto Subject. El que un objeto Subject se considere válido y qué objeto Subject se va a devolver dependerá de las implementaciones. Si el objeto Subject no es válido, se genera una InvalidSubjectException.

Puede utilizar este plug-in si no confía en el objeto Subject pasado a este método. Esto no es habitual si se tiene en cuenta que el usuario suele confiar en los desarrolladores de las aplicaciones que desarrollan el código que recupera el objeto Subject.

Una implementación de este plug-in necesita el soporte del creador del objeto Subject porque sólo el creador sabe si el objeto Subject ha sido manipulado indebidamente. Puede darse el caso, no obstante, de que el creador no sepa si el objeto Subject se ha manipulado de forma indebida. En un caso así, este plug-in no resulta de utilidad.

WebSphere eXtreme Scale proporciona una implementación predeterminada de SubjectValidation: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl. Puede utilizar esta implementación para validar el asunto autenticado por WebSphere Application Server. Puede configurar esta clase como la clase de implementación de SubjectValidation cuando se utiliza eXtreme Scale en WebSphere Application Server. La implementación WSSubjectValidationImpl considera válido un objeto Subject sólo si la señal de credencial asociada con este objeto Subject no se ha manipulado de forma indebida. Puede modificar otras partes del objeto Subject. La implementación de WSSubjectValidationImpl solicita a WebSphere Application Server el Subject original correspondiente a la señal de

credencial y devuelve el objeto Subject original como el objeto Subject validado. Por lo tanto, los cambios realizados en el contenido de Subject que no sea la señal de credencial, no tiene ningún efecto. El fragmento de código siguiente muestra el flujo básico de WSSubjectValidationImpl.validateSubject(Subject).

```
//
Crear un LoginContext con WSLogin de esquema y
// pasar un manejador de devolución de llamada.
LoginContext lc = new LoginContext("WSLogin",
new WSCredTokenCallbackHandlerImpl(subject));

// Cuando se llama a este método, los métodos del manejador de devolución de llamada
// se llamarán para iniciar la sesión de usuario.
lc.login();

// Obtener el objeto Subject de LoginContext
return lc.getSubject();
```

En el fragmento de código anterior, se crea un objeto de manejador de devolución de llamada de la señal de credencial, WSCredTokenCallbackHandlerImpl, con el objeto Subject para validar. Después, se crea un objeto LoginContext con el esquema de inicio de sesión WSLogin. Cuando se llama al método lc.login, la seguridad de WebSphere Application Server recupera la señal de credencial del objeto Subject y, a continuación, devuelve el Subject correspondiente como el objeto Subject validado.

Para ver otros detalles, consulte las API Java de la implementación SubjectValidation y WSSubjectValidationImpl.

Configuración de plug-in

Puede configurar el plug-in SubjectValidation y el plug-in SubjectSource de dos maneras:

- **Configuración del XML** Puede utilizar el archivo XML ObjectGrid para definir un ObjectGrid y establecer estos dos plug-ins. A continuación se muestra un ejemplo en el que la clase WSSubjectSourceImpl se configura como plug-in SubjectSource y la clase WSSubjectValidation se configura como plug-in SubjectValidation.

  ```
  <objectGrids>
   <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
          <bean id="SubjectSource"
  className="com.ibm.websphere.objectgrid.security.plugins.builtins.
   WSSubjectSourceImpl" />
      <bean id="SubjectValidation"
   className="com.ibm.websphere.objectgrid.security.plugins.builtins.
   WSSubjectValidationImpl" />
      <bean id="TransactionCallback"
  className="com.ibm.websphere.samples.objectgrid.
   HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- **Configuración mediante programa** Si desea crear un ObjectGrid mediante API, puede llamar a los métodos siguientes para establecer los plug-ins SubjectSource o SubjectValidation.

  ```
  **
  * Establecer el plug-in SubjectValidation para esta instancia de ObjectGrid. Un
  * plug-in SubjectValidation puede utilizarse para validar el objeto Subject
  * pasado como Subject válido. Consulte {@link SubjectValidation}
  * para obtener más información.
  * @param subjectValidation el plug-in SubjectValidation
  ```

```
*/
void setSubjectValidation(SubjectValidation subjectValidation);


/**
* Establecer el plug-in SubjectSource. Un plug-in SubjectSource puede utilizarse
* para obtener un objeto Subject del entorno para representar el
* cliente ObjectGrid.
*
* @param source el plug-in SubjectSource
*/
void setSubjectSource(SubjectSource source);
```

Escribir código de autenticación JAAS

Puede escribir su propio código de autenticación JAAS (Java Authentication and Authorization Service) para manejar la autenticación. Debe escribir los módulos de inicio de sesión y después configurarlos para el módulo de autenticación.

El módulo de inicio de sesión recibe información sobre un usuario y lo autentica. Esta información puede ser cualquier información que identifique al usuario. Por ejemplo, puede ser un ID de usuario y una contraseña, un certificado de cliente, etc. Después de recibir la información, el módulo de inicio de sesión comprueba que la información representa a un sujeto válido y crea un objeto Subject. Actualmente existen diversas implementaciones de módulos de inicio de sesión.

Una vez que se ha escrito un módulo de inicio de sesión, configure este módulo de inicio de sesión para su uso en el tiempo de ejecución. Debe configurar un módulo de inicio de sesión JAAS. Este módulo de inicio de sesión contiene el módulo de inicio de sesión y su esquema de autenticación. Por ejemplo:

```
FileLogin
{
    com.acme.auth.FileLoginModule required
};
```

El esquema de autenticación es FileLogin y el módulo de inicio de sesión es com.acme.auth.FileLoginModule. La señal requerida indica que el módulo FileLoginModule debe validar este inicio de sesión o se producirá una anomalía en todo el esquema.

Puede establecer el archivo de configuración del módulo de inicio de sesión JAAS de uno de los siguientes modos:

- Establezca el archivo de configuración del módulo de inicio de sesión JAAS en la propiedad login.config.url del archivo java.security, por ejemplo:
  ```
  login.config.url.1=file:${java.home}/lib/security/file.login
  ```
- Establezca el archivo de configuración del módulo de inicio de sesión JAAS desde la línea de mandatos utilizando los argumentos de la máquina virtual Java (JVM) **-Djava.security.auth.login.config**, por ejemplo,
  ```
  -Djava.security.auth.login.config ==$JAVA_HOME/lib/security/file.login
  ```

Si el código se está ejecutando en WebSphere Application Server, debe configurar el inicio de sesión JAAS en la consola de administración y almacenar esta configuración de inicio de sesión en la configuración del servidor de aplicaciones. Consulte la configuración del inicio de sesión para Java Authentication and Authorization Service si desea más detalles.

Capítulo 8. Resolución de problemas

Además de los registros y el rastreo, los mensajes y las notas de release que se describen en este apartado, puede utilizar herramientas de supervisión para descubrir cuestiones como, por ejemplo, la ubicación de los datos en el entorno, la disponibilidad de los servidores en la cuadrícula de datos, etc. Si está trabajando en un entorno WebSphere Application Server, podrá utilizar la infraestructura PMI (Performance Monitoring Infrastructure). Si está trabajando en un entorno autónomo, podrá utilizar una herramienta de supervisión de proveedor como, por ejemplo, CA Wily Introscope o Hyperic HQ. También puede utilizar y personalizar el programa de utilidad **xscmd** para visualizar información textual sobre el entorno.

Habilitación del registro

Puede utilizar los registros para supervisar y solucionar problemas del entorno.

Acerca de esta tarea

Los registros se guardan en distintas ubicaciones y formatos en función de la configuración.

Procedimiento

- **Habilite los registros en un entorno autónomo.**

  Con servidores de catálogo autónomos, los registros se encuentran en la ubicación en la que se ejecuta el mandato **startOgServer**. Para los servidores de contenedor, puede utilizar la ubicación predeterminada o establecer una ubicación de registro personalizada:

  – **Ubicación de registro predeterminada:** los registros se encuentran en el directorio donde se ha ejecutado el mandato del servidor. Si inicia los servidores en el directorio *inicio_wxs*/bin, los registros y los archivos de rastreo se encuentran en los directorios logs/<*nombre_servidor*> del directorio bin.

  – **Ubicación de registro personalizada:** para especificar una ubicación alternativa para los registros de servidor de contenedor, cree un archivo de propiedades como, por ejemplo, server.properties, con el contenido siguiente:

  ```
  workingDirectory=<directorio>
  traceSpec=
  systemStreamToFileEnabled=true
  ```

  La propiedad **workingDirectory** es el directorio raíz de los registros y del archivo de rastreo opcional. WebSphere eXtreme Scale crea un directorio con el nombre del servidor de contenedor con un archivo SystemOut.log, un archivo SystemErr.log y un archivo de rastreo. Para utilizar un archivo de propiedades durante el inicio del contenedor, utilice la opción **-serverProps** y proporcione la ubicación del archivo de propiedades de servidor.

- **Habilite los registros en WebSphere Application Server.**

  Consulte WebSphere Application Server: Habilitación e inhabilitación del registro para obtener más información.

- **Recupere los archivos FFDC.**

  Los archivos FFDC sirven para que el servicio de soporte de IBM ayude a realizar la depuración. Es posible que el servicio de soporte IBM solicite estos

archivos cuando se produzca un problema. Estos archivos están en un directorio denominado, `ffdc`, y contienen archivos que se parecen al siguiente:

```
server2_exception.log
server2_20802080_07.03.05_10.52.18_0.txt
```

Qué hacer a continuación

Visualice los archivos de registro en sus ubicaciones especificadas. Los mensajes comunes para buscar en el archivo `SystemOut.log` son mensajes de confirmación de inicio, como el ejemplo siguiente:

```
CWOBJ1001I: ObjectGrid Server catalogServer01 está listo para procesar
solicitudes.
```

Para obtener más información sobre un mensaje específico en los archivos de registro, consulte Mensajes.

Referencia relacionada:

"Opciones de rastreo" en la página 512
Puede habilitar el rastreo para proporcionar información sobre el entorno al servicio de soporte de IBM.

Mensajes
Cuando encuentre un mensaje en un registro u otras partes de la interfaz del producto, puede buscar el mensaje por su prefijo de componente para descubrir más información.

Recopilación de rastreo

Puede utilizar el rastreo para supervisar y resolver los problemas del entorno. Debe proporcionar rastreo para que un servidor funcione con el soporte de IBM.

Acerca de esta tarea

La recopilación de rastreo puede ayudar a supervisar y corregir problemas en el entorno de WebSphere eXtreme Scale. La forma de recopilar el rastreo dependerá de su configuración. Consulte "Opciones de rastreo" en la página 512 para ver una lista de las distintas especificaciones de rastreo que puede recopilar.

Procedimiento

- **Recopile el rastreo desde un entorno de WebSphere Application Server.**

  Si los servidores de catálogo y contenedor están en un entorno de WebSphere Application Server, consulte WebSphere Application Server: Cómo trabajar con el rastreo para obtener más información.

- **Recopile el rastreo con el mandato de inicio del servidor de catálogo o contenedor autónomo.**

  Puede establecer el rastreo en un servicio de catálogo o servidor de contenedor utilizando los parámetros **`-traceSpec`** y **`-traceFile`** con el mandato **`startOgServer`**. Por ejemplo:

  ```
  startOgServer.sh catalogServer -traceSpec ObjectGridPlacement=all=enabled -traceFile  /home/user1/logs/trace.log
  ```

  El parámetro **`-traceFile`** es opcional. Si no establece una ubicación **`-traceFile`**, el archivo de rastreo va a la misma ubicación que los archivos de registro de salida del sistema.Para obtener más información sobre estos parámetros, consulte la información sobre el script startOgServer en *Guía de administración*.

- **Recopile el rastreo desde el servidor de contenedor o catálogo autónomo con un archivo de propiedades.**

Para recopilar rastreo de un archivo de propiedades, cree un archivo como, por ejemplo, un archivo `server.properties`, con el contenido siguiente:

```
workingDirectory=<directorio>
traceSpec=<especificación_rastreo>
systemStreamToFileEnabled=true
```

La propiedad **workingDirectory** es el directorio raíz de los registros y del archivo de rastreo opcional. Si el valor **workingDirectory** no está establecido, el directorio de trabajo predeterminado es la ubicación utilizada para iniciar los servidores, por ejemplo, *inicio_wxs*`/bin`. Para utilizar un archivo de propiedades durante el inicio del servidor, utilice el parámetro **-serverProps** con el mandato **startOgServer** y proporcione la ubicación del archivo de propiedades del servidor.Para obtener más información sobre el archivo de propiedades del servidor y sobre cómo utilizar el archivo, consulte la información sobre el archivo de propiedades del servidor en la *Guía de administración*.

- **Recopile el rastreo de un cliente autónomo.**

  Puede iniciar la recopilación de rastreo en un cliente autónomo añadiendo propiedades del sistema al script de inicio de la aplicación cliente. En el ejemplo siguiente se especifican los valores de rastreo de la aplicación `com.ibm.samples.MyClientProgram`:

  ```
  java -DtraceSettingsFile=MyTraceSettings.properties
  -Djava.util.logging.manager=com.ibm.ws.bootstrap.WsLogManager
  -Djava.util.logging.configureByServer=true com.ibm.samples.MyClientProgram
  ```

  Consulte WebSphere Application Server: Habilitación del rastreo en aplicaciones cliente y autónomas para obtener más información.

- **Recopile el rastreo con la interfaz ObjectGridManager.**

  También puede establecer el rastreo durante el tiempo de ejecución en una interfaz ObjectGridManager. Si se establece el rastreo en una interfaz ObjectGridManager, se puede utilizar para obtener el rastreo en un cliente de eXtreme Scale, mientras se conecta a eXtreme Scale y confirma transacciones. Para establecer el rastreo en una interfaz ObjectGridManager, proporcione una especificación de rastreo y un registro de rastreo.

  ```
  ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
  ...
  manager.setTraceEnabled(true);
  manager.setTraceFileName("logs/myClient.log");
  manager.setTraceSpecification("ObjectGridReplication=all=enabled");
  ```

  Para obtener más información sobre la interfaz ObjectGridManager, consulte la información sobre cómo interactuar con el ObjectGrid mediante la interfaz ObjectGridManager en la *Guía de programación*.

- **Recopile el rastreo en los servidores de contenedor con el programa de utilidad xscmd.**

  Para recopilar el rastreo con el programa de utilidad **xscmd**, utilice el mandato **-c setTraceSpec**. Utilice el programa de utilidad **xscmd** para recopilar el rastreo en un entorno autónomo durante el tiempo de ejecución en lugar de hacerlo durante el arranque. Puede recopilar rastreo en todos los servidores y servicios de catálogo o bien puede filtrar los servidores en función del nombre del ObjectGrid y otras propiedades. Por ejemplo, para recopilar rastreo de ObjectGridReplication con acceso al servidor de servicio de catálogo, ejecute:

  ```
  xscmd -c setTraceSpec "ObjectGridReplication=all=enabled"
  ```

  También puede inhabilitar el rastreo estableciendo la especificación de rastreo en `*=all=disabled`.

Resultados

Los archivos de rastreo se graban en la ubicación especificada.

Referencia relacionada:

"Opciones de rastreo"
Puede habilitar el rastreo para proporcionar información sobre el entorno al servicio de soporte de IBM.

Mensajes
Cuando encuentre un mensaje en un registro u otras partes de la interfaz del producto, puede buscar el mensaje por su prefijo de componente para descubrir más información.

Opciones de rastreo

Puede habilitar el rastreo para proporcionar información sobre el entorno al servicio de soporte de IBM.

Sobre el rastreo

El rastreo de WebSphere eXtreme Scale se divide en varios componentes distintos. Puede especificar el nivel de rastreo que se debe utilizar. Los niveles comunes de rastreo son: all, debug, entryExit y event.

A continuación se muestra una serie de rastreo de ejemplo:

```
ObjectGridComponent=level=enabled
```

Puede concatenar valores de rastreo. Utilice el símbolo `*` (asterisco) para especificar un valor comodín como, por ejemplo, `ObjectGrid*=all=enabled`. Si necesita proporcionar un rastreo al servicio de soporte de IBM, se solicita una serie de rastreo específica. Por ejemplo, si hay un problema con la réplica, se puede solicitar la serie de rastreo `ObjectGridReplication=debug=enabled`.

Especificación de rastreo

ObjectGrid
Motor de memoria caché principal general.

ObjectGridCatalogServer
Servicio de catálogo general.

ObjectGridChannel
Comunicaciones de topología de despliegue estática.

ObjectGridClientInfo
Información del cliente DB2

ObjectGridClientInfoUser
Información del usuario de DB2.

ObjectgridCORBA
Comunicaciones de topología de despliegue dinámica.

ObjectGridDataGrid
API de AgentManager.

ObjectGridDynaCache
Proveedor de la memoria caché dinámica de WebSphere eXtreme Scale.

ObjectGridEntityManager
API de EntityManager. Utilícela con la opción Projector.

ObjectGridEvictors
Desalojadores incorporados de ObjectGrid.

ObjectGridJPA
Cargadores JPA (Java Persistence API).

ObjectGridJPACache
Plug-ins de memoria caché JPA.

ObjectGridLocking
Gestor de bloqueos de entradas de memoria caché de ObjectGrid.

ObjectGridMBean
Beans de gestión.

ObjectGridMonitor
Infraestructura de supervisión histórica.

7.1.1+ **ObjectGridNative**
Rastreo de código nativo de WebSphere eXtreme Scale, incluido el código nativo eXtremeMemory.

7.1.1+ **ObjectGridOSGi**
Componentes de integración OSGi de WebSphere eXtreme Scale.

ObjectGridPlacement
Servicio de colocación de fragmentos de servidor de catálogo.

ObjectGridQuery
Consulte de ObjectGrid.

ObjectGridReplication
Servicio de réplica.

ObjectGridRouting
Detalles de direccionamiento de cliente/servidor.

ObjectGridSecurity
Rastreo de seguridad.

7.1.1+ **ObjectGridSerializer**
Infraestructura de plug-in DataSerializer.

ObjectGridStats
Estadísticas de ObjectGrid.

ObjectGridStreamQuery
API de consulta de secuencia.

7.1.1+ **ObjectGridTransactionManager**
Gestor de transacciones de WebSphere eXtreme Scale.

ObjectGridWriteBehind
Escritura diferida de ObjectGrid

7.1.1+ **ObjectGridXM**
Rastreo general de IBM eXtremeMemory.

7.1.1+ **ObjectGridXMEviction**
Rastreo de desalojo de eXtremeMemory.

7.1.1+ **ObjectGridXMTransport**
Rastreo de transporte general de eXtremeMemory.

7.1.1+ **ObjectGridXMTransportInbound**
Rastreo de transporte específico de entrada de eXtremeMemory.

7.1.1+ **ObjectGridXMTransportOutbound**
Rastreo de transporte específico de entrada de eXtremeMemory.

Projector
Motor en la API EntityManager.

QueryEngine
Motor de consulta para la API de consulta de objetos y la API de consulta EntityManager.

QueryEnginePlan
Rastreo del plan de consulta.

7.1.1+ **TCPChannel**
Canal TCP/IP de IBM eXtremeIO.

7.1.1+ **XsByteBuffer**
Rastreo de almacenamiento intermedio de bytes de WebSphere eXtreme Scale.

Tareas relacionadas:

"Habilitación del registro" en la página 509
Puede utilizar los registros para supervisar y solucionar problemas del entorno.

"Recopilación de rastreo" en la página 510
Puede utilizar el rastreo para supervisar y resolver los problemas del entorno. Debe proporcionar rastreo para que un servidor funcione con el soporte de IBM.

Inicio de los servidores autónomos
Cuando se ejecuta una configuración autónoma, el entorno está formado por servidores de catálogo, servidores de contenedor y procesos de cliente. Los servidores WebSphere eXtreme Scale también pueden incorporarse en las aplicaciones Java existentes con la API de servidor incorporado. Debe configurar e iniciar manualmente estos procesos.

Administración con el programa de utilidad **xscmd**
Con **xscmd** puede completar tareas administrativas en el entorno como, por ejemplo: establecer enlaces de réplica multimaestro, sustituir quórum y detener grupos de servidores con el mandato teardown.

Análisis de datos de registro y rastreo

Puede utilizar las herramientas de análisis de registro para analizar cómo se realiza su ejecución y solucionar los problemas que se producen en el entorno.

Acerca de esta tarea

Puede generar informes a partir de los archivos de registro y de rastreo en el entorno. Estos informes visuales se pueden utilizar para los fines siguientes:

- **Para analizar el estado y el rendimiento del entorno de ejecución:**
  - Coherencia del entorno de despliegue
  - Frecuencia de registro
  - Topología en ejecución vs. topología configurada
  - Cambios de topología no planificados
  - Estado de quórum
  - Estado de réplica de partición
  - Estadísticas de memoria, rendimiento, uso de procesador, etc.
- **Para resolver problemas del entorno:**
  - Vistas de topología en puntos específicos en el tiempo

- Estadísticas de memoria, rendimiento, uso del procesador durante anomalías de cliente
- Niveles de fixpack actuales, valores de ajuste
- Estado de quórum

Visión general del análisis de registro

Puede utilizar la herramienta **xsLogAnalyzer** como ayuda para la resolución de problemas del entorno.

Todos los mensajes de migración tras error

Visualiza el número total de mensajes de migración tras error como un gráfico a lo largo del tiempo. También muestra una lista de los mensajes de migración tras error, incluidos los servidores que han resultado afectados

Todos los mensajes críticos de eXtreme Scale

Visualiza ID de mensaje junto con las explicaciones y acciones de usuario asociadas, que le pueden evitar perder tiempo buscando mensajes.

Todas las excepciones

Visualiza las cinco primeras excepciones, incluidos los mensajes, el número de veces que éstos se han producido y qué servidores se han visto afectados por la excepción.

Resumen de topología

Visualiza un diagrama de cómo se configura la topología según los archivos de registro. Puede utilizar este resumen para comparar con la configuración real, posiblemente identificando errores de configuración.

Coherencia de la topología: tabla de comparación de intermediario de solicitud de objetos (ORB)

Visualiza valores de ORB en el entorno. Puede utilizar esta tabla como ayuda para determinar si los valores son coherentes en todo el entorno.

Vista de franja horaria del suceso

Visualiza un diagrama de franja horaria de las distintas acciones que se han producido en la cuadrícula de datos, incluidos los sucesos de ciclo de vida, excepciones, mensajes críticos y sucesos de captura de datos en primer error (FFDC).

Ejecución de análisis de registro

Puede ejecutar la herramienta **xsLogAnalyzer** en un conjunto de archivos de registro y rastreo desde cualquier sistema.

Antes de empezar

- Habilite los registros y el rastreo. Consulte “Habilitación del registro” en la página 509 y “Recopilación de rastreo” en la página 510 para obtener más información.

- Recopile los archivos de registro. Los archivos de registro se pueden encontrar en diversas ubicaciones en función de cómo los haya configurado. Si está utilizando los valores de registro predeterminados, puede obtener los archivos de registro de las ubicaciones siguientes:
  - En una instalación autónoma: *raíz_intal_wxs*/bin/logs/<*nombre_servidor*>
  - En una instalación integrada con WebSphere Application Server: *raíz_was*/logs/<*nombre_servidor*>
- Recopile los archivos de rastreo. Los archivos de rastreo pueden estar en diversas ubicaciones en función de cómo los haya configurado. Si está utilizando los valores de rastreo predeterminados, puede obtener los archivos de rastreo en las ubicaciones siguientes:
  - En una instalación autónoma: si no se establece ningún valor de rastreo específico, los archivos de rastreo se graban en la misma ubicación que los archivos de registro de salida del sistema.
  - En una instalación integrada con WebSphere Application Server: *raíz_was*/profiles/*nombre_servidor*/logs.

  Copie los archivos de registro y rastreo en el sistema desde el que está planificando utilizar la herramienta Log Analyzer.
- Si desea crear exploraciones personalizadas en el informe generado, cree un archivo de propiedades de especificaciones de exploración y un archivo de configuración antes de ejecutar la herramienta. Para obtener más información, consulte "Creación de exploradores personalizados para el análisis de registro" en la página 517.

Procedimiento

1. Ejecute la herramienta **xsLogAnalyzer**.

   El script se encuentra en las ubicaciones siguientes:
   - En una instalación autónoma: *raíz_intal_wxs*/ObjectGrid/bin
   - En una instalación integrada con WebSphere Application Server: *raíz_was*/bin

   Consejo: Si los archivos de registro son grandes, tenga en cuenta la posibilidad de utilizar los parámetros **-startTime**, **-endTime** y **-maxRecords** al ejecutar el informe para restringir el número de entradas de registro que se exploran. Si utiliza estos parámetros al ejecutar el informe, será más fácil leer los informes y éstos se ejecutarán de forma más efectiva. Puede ejecutar varios informes en el mismo conjunto de archivos de registro.

   ```
   xsLogAnalyzer.sh|bat -logsRoot c:\myxslogs -outDir c:\myxslogs\out
   -startTime 11.09.27_15.10.56.089 -endTime 11.09.27_16.10.56.089 -maxRecords 100
   ```

   -logsRoot
   Especifica la vía de acceso absoluta al directorio de registro que desea evaluar (necesario).

   -outDir
   Especifica un directorio existente para grabar la salida de informe. Si no especifica un valor, el informe se graba en la ubicación raíz de la herramienta **xsLogAnalyzer**.

   -startTime
   Especifica la hora de inicio para realizar la evaluación en los registros. La fecha tiene el formato siguiente:
   año.mes.día.hora.minuto.segundo.milisegundo

   -endTime
   Especifica la hora de finalización para realizar la evaluación en los

registros. La fecha tiene el formato siguiente:
año.mes.día.hora.minuto.segundo.milisegundo

-trace Especifica una serie de rastreo, por ejemplo `ObjectGrid*=all=enabled`.

-maxRecords
Especifica el número máximo de registros a generar en el informe. El valor predeterminado es `100`. Si especifica el valor como 50, se generan los primeros 50 registros para el periodo de tiempo especificado.

2. Abra los archivos generados. Si no ha definido un directorio de salida, los informes se generan en una carpeta denominada `report_fecha_hora`. Para abrir la página principal de los informes, abra el archivo `index.html`.
3. Utilice los informes para analizar los datos de registro. Utilice las sugerencias siguientes para maximizar el rendimiento de las visualizaciones de informe:
   - Para maximizar el rendimiento de las consultas en los datos de registro, utilice la información más específica que sea posible. Por ejemplo, una consulta para `servidor` tarda mucho más tiempo en ejecutarse y devuelve más resultados que `nombre_host_servidor`.
   - Algunas vistas tienen un número limitado de puntos de datos que se visualizan a la vez. Puede ajustar el segmento de tiempo que se está viendo cambiando en la vista los datos actuales, por ejemplo hora de inicio y finalización.

Qué hacer a continuación

Para obtener más información acerca de la resolución de problemas de la herramienta **xsLogAnalyzer** y los informes generados, consulte "Resolución de problemas de análisis de registro" en la página 518.

Creación de exploradores personalizados para el análisis de registro

Puede crear exploradores personalizados para el análisis de registro. Después de configurar el explorador, se generan los resultados en los informes al ejecutar la herramienta **xsLogAnalyzer**. El explorador personalizado explora en las anotaciones cronológicas para obtener registros de sucesos basándose en las expresiones regulares que se han especificado.

Procedimiento

1. Cree un archivo de propiedades de especificaciones de explorador que especifique la expresión general a ejecutar para el explorador personalizado.
   a. Cree y guarde un archivo de propiedades. El archivo debe estar en el directorio *raíz_loganalyzer*`/config/custom`. Puede utilizar el nombre que desee para el archivo. Puesto que el nuevo explorador utilizará el archivo, resulta útil darle nombre al explorador en el archivo de propiedades, por ejemplo: *mi_especificación_explorador_servidor_nuevo*`.properties`.
   b. Incluya las propiedades siguientes en el archivo:
   mi_especificación_explorador_servidor_nuevo`.properties`

   ```
   include.regular_expression = EXPRESIÓN_REGULAR_PARA_EXPLORAR
   ```

   La variable *EXPRESIÓN_REGULAR_PARA_EXPLORAR* es una expresión regular en la que se deben filtrar los archivos de registro.

   Ejemplo: Para explorar en instancias de líneas que contienen las series `"xception"` y `"rror"` independientemente del orden, establezca la propiedad **include.regular_expression** en el valor siguiente:

```
include.regular_expression = (xception.+rror)|(rror.+xception)
```

Esta expresión regular hace que se registren sucesos si la serie "rror" va antes o después de la serie "xception".

Ejemplo:Para explorar en cada línea de los registros las instancias de líneas que contienen las series "xception" de frase o "rror" de frase independientemente del orden, establezca la propiedad **include.regular_expression** en el valor siguiente:

```
include.regular_expression = (xception)|(rror)
```

Esta expresión regular hace que se registren sucesos si la serie "rror" va antes o después de la serie "xception".

2. Cree un archivo de configuración que la herramienta **xsLogAnalyer** utilice para crear el explorador.
   a. Cree y guarde un archivo de propiedades. El archivo debe estar en el directorio *raíz_loganalyzer*/config/custom. Puede dar al archivo el nombre *nombre_explorador*Scanner.config, donde *nombre_explorador* es un nombre exclusivo para el nuevo explorador. Por ejemplo, puede dar al archivo el nombreserverScanner.config
   b. Incluya las propiedades siguientes en el archivo *nombre_explorador*Scanner.config:

      ```
      scannerSpecificationFiles = UBICACIÓN_DE_ARCHIVO_ESPECIFICACIÓN_EXPLORADOR
      ```

      La variable *UBICACIÓN_DE_ARCHIVO_ESPECIFICACIÓN_EXPLORADOR* es la vía de acceso y la ubicación del archivo de especificación que ha creado en el paso anterior. Por ejemplo: *raíz_loganalyzer*/config/custom/mi_especificación_escáner_nueva.properties. Puede también especificar varios archivos de especificación de explorador utilizando una lista separada por punto y coma:

```
scannerSpecificationFiles = UBICACIÓN_DE_ARCHIVO1_ESPECIFICACIÓN_EXPLORADOR;UBICACIÓN_DE_ARCHIVO2_ESPECIFICACIÓN_EXPLORADOR
```

3. Ejecute la herramienta **xsLogAnalyzer**. Para obtener más información, consulte "Ejecución de análisis de registro" en la página 515.

Resultados

Después de ejecutar la herramienta **xsLogAnalyzer**, el informe contiene separadores nuevos para los exploradores personalizados que ha configurado. Cada separador contiene las vistas siguientes:

Gráficos
Gráfico trazado que ilustra los sucesos registrados. Los sucesos se visualizan en el orden en el que se han encontrado.

Tablas Representación tabular de los sucesos registrados.

Informes de resumen

Resolución de problemas de análisis de registro

Utilice la siguiente información de resolución de problemas para diagnosticar y arreglar problemas con la herramienta **xsLogAnalyzer** y los informes generados.

Procedimiento

- **Problema:** Se producen condiciones de falta de memoria cuando se utiliza la herramienta **xsLogAnalyzer** para generar informes. A continuación se muestra un

ejemplo de un error que se puede producir: `java.lang.OutOfMemoryError: Se ha superado el límite de sobrecarga de GC`.

Solución: La herramienta **xsLogAnalyzer** se ejecuta en una máquina virtual Java (JVM). Puede configurar la JVM para aumentar el tamaño de almacenamiento dinámico antes de ejecutar la herramienta **xsLogAnalyzer** especificando algunos valores cuando ejecute la herramienta. Si aumenta el tamaño de almacenamiento dinámico se podrán almacenar más registros de sucesos en la memoria de JVM. Empiece con un valor de 2048M, suponiendo que el sistema operativo tenga suficiente memoria principal. En la misma instancia de línea de mandatos en la que piensa ejecutar la herramienta **xsLogAnalyzer**, establezca el tamaño de almacenamiento dinámico de JVM máximo:

```
java -Xmx TAMAÑO_ALMACENAMIENTO_DINÁMICO m
```

El valor de *TAMAÑO_ALMACENAMIENTO_DINÁMICO* puede ser cualquier cualquier entero y representa el número de megabytes que están asignados al almacenamiento dinámico de JVM. Por ejemplo, puede ejecutar `java -Xmx2048m`. Si continúan los mensajes que indican que falta memoria o no tiene los recursos para asignar 2048m o más memoria, limite el número de sucesos que se están manteniendo en el almacenamiento dinámico. Puede limitar el número de sucesos en el almacenamiento dinámico pasando el parámetro **-maxRecords** al mandato **xsLogAnalyzer**.

- **Problema:** Cuando se abre un informe generado desde la herramienta **xsLogAnalyzer**, el navegador se cuelga o no carga la página.

  Causa: Los archivos HTML generados son demasiado grandes y el navegador no los puede cargar. Estos archivos son grandes porque el ámbito de los archivos de registro que está analizando es demasiado amplio.

  Solución: Considere la posibilidad de utilizar los parámetros **-startTime**, **-endTime** y **-maxRecords** cuando ejecute la herramienta **xsLogAnalyzer** para restringir el número de entradas de registro que se exploran. Si utiliza estos parámetros al ejecutar el informe, será más fácil leer los informes y éstos se ejecutarán de forma más efectiva. Puede ejecutar varios informes en el mismo conjunto de archivos de registro.

Resolución de problemas de la conectividad de cliente

Existen varios problemas comunes específicos de los clientes y de la conectividad de cliente que puede resolver tal como se describe en las secciones siguientes.

Procedimiento

- **Problema:** si está utilizando la API EntityManager o una matriz de bytes se correlaciona con la modalidad de copia COPY_TO_BYTES, los métodos de acceso a datos de cliente generan diversas excepciones relacionadas con la serialización o una excepción NullPointerException.
  - Se produce el error siguiente al utilizar la modalidad de copia COPY_TO_BYTES:

```
java.lang.NullPointerException
        en com.ibm.ws.objectgrid.map.BaseMap$BaseMapObjectTransformer2.inflateObject(BaseMap.java:5278)
        en com.ibm.ws.objectgrid.map.BaseMap$BaseMapObjectTransformer.inflateValue(BaseMap.java:5155)
```

  - Se produce en error siguiente al utilizar la API EntityManager:

```
java.lang.NullPointerException
  en com.ibm.ws.objectgrid.em.GraphTraversalHelper.fluffFetchMD(GraphTraversalHelper.java:323)
  en com.ibm.ws.objectgrid.em.GraphTraversalHelper.fluffFetchMD(GraphTraversalHelper.java:343)
  en com.ibm.ws.objectgrid.em.GraphTraversalHelper.getObjectGraph(GraphTraversalHelper.java:102)
  en com.ibm.ws.objectgrid.ServerCoreEventProcessor.getFromMap(ServerCoreEventProcessor.java:709)
  en com.ibm.ws.objectgrid.ServerCoreEventProcessor.processGetRequest(ServerCoreEventProcessor.java:323)
```

Causa: la API EntityManager y la modalidad de copia COPY_TO_BYTES utilizan un repositorio de metadatos incorporado en la cuadrícula de datos. Cuando se conectan los clientes, la cuadrícula de datos almacena los identificadores de repositorio en el cliente y almacena en memoria caché los identificadores mientras dure la conexión de cliente. Si reinicia la cuadrícula de datos, perderá todos los metadatos y los identificadores regenerados no coincidirán con los identificadores almacenados en memoria caché en el cliente.

Solución: si está utilizando la API EntityManager o la modalidad de copia COPY_TO_BYTES, desconecte y vuelva a conectar todos los clientes si el ObjectGrid se detiene y reinicia. La desconexión y reconexión de los clientes renueva la memoria caché de identificadores de metadatos. Puede desconectar los clientes mediante el método ObjectGridManager.disconnect o el método ObjectGrid.destroy.

- **Problema:** El cliente se cuelga durante una llamada de método getObjectGrid.

  Podría parecer que un cliente se cuelga al llamar al método getObjectGrid en ObjectGridManager o que emite una excepción: com.ibm.websphere.projector.MetadataException. El repositorio EntityMetadata no está disponible y se ha alcanzado el umbral del tiempo de espera.

  Causa: la razón es que el cliente está esperando a que los metadatos de entidad del servidor ObjectGrid pasen a estar disponibles.

  Solución: Este error se puede producir cuando se ha iniciado un servidor de contenedor, pero la colocación aún no se ha iniciado. Realice las acciones siguientes:

  – Examine la política de despliegue de ObjectGrid y compruebe que el número de contenedores activos es mayor o igual que los atributos numInitialContainers y minSyncReplicas del archivo de descriptor de política de despliegue.

  – Examine el valor para la propiedad **placementDeferralInterval** en el archivo de propiedades de servidor de contenedor para ver cuánto tiempo debe transcurrir antes de que se produzcan las operaciones de colocación.

  – Si ha utilizado el mandato **xscmd -c suspendBalancing** para detener el equilibrio de fragmentos para una cuadrícula de datos y un conjunto de correlaciones específicos, utilice **xscmd -c resumeBalancing** para iniciar el equilibrio de nuevo.

Conceptos relacionados:

"Creación de instancia de ObjectGrid con la interfaz ObjectGridManager" en la página 136
Cada uno de estos métodos crea una instancia local de un ObjectGrid.

Resolución de problemas de la integración de la memoria caché

Utilice esta información para resolver problemas de la configuración de la integración de la memoria caché, incluidas las configuraciones de memoria caché dinámica y de sesión HTTP.

Procedimiento

- 7.1.1+ **Problema:** los ID de sesión HTTP no se están reutilizando.

  Causa: puede utilizar los ID de sesión. Si crea una cuadrícula de datos para la persistencia de sesión en la versión 7.1.1 o posterior, se habilita automáticamente la reutilización de ID de sesión. Sin embargo, si ha creado configuraciones anteriores, es posible que este valor ya se haya creado con un valor incorrecto.

  Solución: compruebe los valores siguientes para verificar que tiene habilitada la reutilización de ID de sesión HTTP:

– La propiedad `reuseSessionId` del archivo `splicer.properties` se debe establecer en `true`.
– El valor de la propiedad personalizada `HttpSessionIdReuse` se debe establecer en `true`. Esta propiedad personalizada podría establecerse en una de las siguientes vías de acceso en la consola administrativa de WebSphere Application Server:
  - **Servidores** > *nombre_servidor* > **Gestión de sesiones** > **Propiedades personalizadas**
  - **Clústeres dinámicos** > *nombre_clúster_dinámico* > **Plantilla de servidor** > **Gestión de sesiones** > **Propiedades personalizadas**
  - **Servidores** > **Tipos de servidor** > **Servidores de aplicaciones WebSphere** > *nombre_servidor* y, a continuación, en Infraestructura del servidor, pulse **Java y gestión de procesos** > **Definición de proceso** > **Máquina virtual Java** > **Propiedades personalizadas**
  - **Servidores** > **Tipos de servidor** > **Servidores de aplicaciones WebSphere** > *nombre_servidor* > **Valores de contenedor web** > **Contenedor web**

  Si actualiza cualquier valor de propiedad personalizada, vuelva a configurar la gestión de sesiones de eXtreme Scale de modo que el archivo `splicer.properties` tenga en cuenta el cambio.

- **Problema:** Cuando se utiliza una cuadrícula de datos para almacenar sesiones HTTP y la carga de transacciones es alta, se visualiza un mensaje CWOBJ0006W en el archivo `SystemOut.log`.

  ```
  CWOBJ0006W: Se ha producido una excepción:
  com.ibm.websphere.objectgrid.ObjectGridRuntimeException:
  java.util.ConcurrentModificationException
  ```

  Este mensaje sólo aparece cuando el parámetro **`replicationInterval`** del archivo `splicer.properties` está establecido en un valor mayor que cero y la aplicación web modifica un objeto de lista que se ha establecido como atributo en HTTPSession.

  Solución: Clone el atributo que contiene el objeto de lista modificado y ponga el atributo clonado en el objeto de sesión.

Referencia relacionada:

Archivos XML para la configuración del gestor de sesiones HTTP
Cuando inicia un servidor de contenedor que almacena datos de sesión HTTP, puede utilizar los archivos XML predeterminados o puede especificar archivos XML personalizados. Estos archivos crean nombres de ObjectGrid específicos, número de réplicas, etc.

Parámetros de inicialización del contexto del servlet
La siguiente lista de parámetros de inicialización de contexto de servlet se puede especificar en el archivo de propiedades de splicer como corresponda en el método de unión elegido.

Archivo `splicer.properties`
El archivo `splicer.properties` contiene todas las opciones de configuración para configurar un gestor de sesiones basado en filtro de servlets.

Resolución de problemas del plug-in de memoria caché JPA

Utilice esta información para resolver problemas de la configuración del plug-in de memoria caché JPA. Estos problemas se pueden producir tanto en configuraciones Hibernate como en configuraciones OpenJPA.

Procedimiento

- **Problema:** se visualiza la siguiente excepción: `CacheException: No se ha podido obtener el servidor ObjectGrid`.

  Con un valor de atributo de **`ObjectGridType`** `EMBEDDED` o `EMBEDDED_PARTITION`, la memoria caché de eXtreme Scale intenta obtener una instancia de servidor en el tiempo de ejecución. En un entorno Java Platform, Standard Edition, se inicia un servidor eXtreme Scale con el servicio de catálogo incorporado. El servicio de catálogo incorporado intenta estar a la escucha en el puerto 2809. Si ese puerto lo utiliza otro proceso, se produce el error.

  Solución: si se especifican puntos finales de servicio de catálogo externo, por ejemplo, con el archivo `objectGridServer.properties`, se produce este error si el nombre de host o puerto se especifica incorrectamente. Corrija el conflicto de puerto.
- **Problema:** se visualiza la siguiente excepción: `CacheException: No se ha podido obtener el ObjectGrid REMOTE para el ObjectGrid REMOTE configurado. objectGridName = [ObjectGridName], PU name = [persistenceUnitName]`

  Este error se produce cuando la memoria caché no puede obtener la instancia de ObjectGrid desde los puntos finales de servicio de catálogo proporcionados.

  Solución: este problema normalmente se produce debido a un nombre de host o puerto incorrecto.
- **Problema:** se visualiza la siguiente excepción: `CacheException: no se puede tener dos PU [nombreUnidadPersistencia_1, nombreUnidadPersistencia_2] configuradas con el mismo ObjectGridName [ObjectGridName] de ObjectGridType EMBEDDED`

  Esta excepción se produce si tiene muchas unidades de persistencia configuradas y las memorias caché de eXtreme Scale de estas unidades se configuran con el mismo nombre de ObjectGrid y valor de atributo de **`ObjectGridType`** `EMBEDDED`. Estas configuraciones de unidades de persistencia podrían estar en los mismos archivos `persistence.xml` o en archivos diferentes.

  Solución: debe verificar que el nombre de ObjectGrid sea exclusivo para cada unidad de persistencia cuando el valor de atributo **`ObjectGridType`** sea `EMBEDDED`.
- **Problema:** se visualiza la siguiente excepción: `CacheException: REMOTE ObjectGrid [ObjectGridName] no incluye las BackingMaps necesarias [nombreCorrelación_1, nombreCorrelación_2,...]`

  Con el tipo de ObjectGrid REMOTE, si el ObjectGrid del lado del cliente obtenido no tiene correlaciones de respaldo de entidad completas para dar soporte a la memoria caché de unidad de persistencia, se produce esta excepción. Por ejemplo, se listan cinco clases de entidad en la configuración de la unidad de persistencia, pero el ObjectGrid obtenido sólo tiene dos BackingMaps. Aunque el ObjectGrid obtenido podría tener 10 BackingMaps, si no se encuentra alguno de las cinco BackingMaps de entidad necesarias en las diez correlaciones de respaldo, aún se produce esta excepción.

  Solución: asegúrese de que la configuración de correlación de respaldo dé soporte a la memoria caché de unidad de persistencia.

Resolución de problemas de administración

Utilice la información siguiente para resolver problemas de administración, incluyendo el inicio de servidores, utilizando el programa de utilidad **`xscmd`**, etc.

Procedimiento

- **Problema:** Faltan los scripts de administración en el directorio *`raíz_perfil`*`/bin` de una instalación de WebSphere Application Server.

Causa: Cuando se actualiza la instalación, los nuevos archivos de script no se instalan automáticamente en los perfiles.

Solución: Si desea ejecutar un script desde el directorio *raíz_perfil*/bin, reduzca y vuelva a aumentar el perfil con el último release. Para obtener más información, consulte Reducción de un perfil utilizando el indicador de mandatos y Creación y aumento de perfiles para WebSphere eXtreme Scale.

- **Problema:** Cuando se ejecuta un mandato **xscmd**, se muestra el mensaje siguiente en la pantalla:

```
java.lang.IllegalStateException: Placement service MBean not available.
[]
        at
com.ibm.websphere.samples.objectgrid.admin.OGAdmin.main(OGAdmin.java:1449)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:60)
        at
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:37)
        at java.lang.reflect.Method.invoke(Method.java:611)
        at com.ibm.ws.bootstrap.WSLauncher.main(WSLauncher.java:267)
Ending at: 2011-11-10 18:13:00.000000484
```

  Causa: Se ha producido un problema de conexión con el servidor de catálogo.

  Solución: Verifique que los servidores de catálogo se están ejecutando y están disponibles a través de la red. Este mensaje también puede aparecer cuando se ha definido un dominio de servicio de catálogo, pero se están ejecutando menos de dos servidores de catálogo. El entorno no está disponible hasta que se inician dos servidores de catálogo.

Conceptos relacionados:

Procedimiento recomendado: Agrupación en clúster del servicio de catálogo con dominios de servicio de catálogo
Cuando se utiliza el servicio de catálogo, se requiere un mínimo de dos servidores de catálogo para evitar un punto único de anomalía. En función del número de nodos en el entorno, puede crear distintas configuraciones para garantizar que como mínimo haya dos servidores de catálogo siempre en ejecución.

Administración

Resolución de problemas de onfiguraciones de varios centros de datos

Utilice esta información para resolver los problemas de configuraciones de varios centros de datos, incluido el enlace entre los dominios de servicio de catálogo.

Procedimiento

Problema: faltan datos en uno o varios dominios de servicio de catálogo. Por ejemplo, puede ejecutar el mandato `xscmd -c establishLink`. Cuando comprueba los datos correspondientes a cada dominio de servicio de catálogo enlazado, parece que los datos son distintos, por ejemplo, desde el mandato `xscmd -c showMapSizes`.
Solución: puede solucionar este problema con el mandato `xscmd -c showLinkedPrimaries`. Este mandato imprime cada fragmento primario, incluidos qué primarios foráneos están enlazados.
En el escenario descrito, al ejecutar el mandato `xscmd -c showLinkedPrimaries` podría descubrir que los fragmentos primarios del primer dominio de servicio de catálogo están enlazados con los fragmentos primarios del segundo dominio de servicio de catálogo, pero que el segundo dominio de servicio de catálogo no tiene enlaces al primer dominio de servicio de catálogo. Puede considerar volver a ejecutar el mandato `xscmd -c establishLink` desde el segundo dominio de servicio de catálogo al primer dominio de servicio de catálogo.

Resolución de problemas de los cargadores

Utilice esta información para resolver problemas de los cargadores de base de datos.

Procedimiento

- **Problema:** cuando se utiliza un cargador OpenJPA con DB2 en WebSphere Application Server, se produce una excepción de cursor cerrado.

  Ls siguiente excepción procede de DB2 en el archivo de registro de org.apache.openjpa.persistence.PersistenceException:

  ```
  [jcc][t4][10120][10898][3.57.82] Operación no válida: el conjunto de
  resultados está cerrado.
  ```

  Solución: De forma predeterminada, el servidor de aplicaciones configura la propiedad personalizada resultSetHoldability con un valor de 2 (CLOSE_CURSORS_AT_COMMIT). Esta propiedad hace que DB2 cierre su conjunto de resultados/cursor en los límites de transacción. Para eliminar la excepción, cambie el valor de la propiedad personalizada a 1 (HOLD_CURSORS_OVER_COMMIT). Establezca la propiedad personalizada resultSetHoldability en la siguiente vía de acceso en la célula de WebSphere Application Server: **Recursos** > **Proveedor JDBC** > **Proveedor de controlador JDBC de DB2 Universal** > **Orígenes de datos** > *nombre_origen_datos* > **Propiedades personalizadas** > **Nueva**.

- **Problema** DB2 visualiza una excepción: `la transacción actual se ha retrotraído debido a un punto muerto o tiempo de espera excedido. Código de razón "2".. SQLCODE=-911, SQLSTATE=40001, DRIVER=3.50.152`

  Esta excepción se produce debido a un problema de contención de bloqueo cuando realiza la ejecución con OpenJPA con DB2 en WebSphere Application Server. El nivel de aislamiento predeterminado de WebSphere Application Server es Lectura repetitiva (RR), que obtiene bloqueos de larga duración con DB2.**Solución:**

  Establezca el nivel de aislamiento en Lectura confirmada para reducir la contención de bloqueo. Establezca la propiedad personalizada de origen de datos webSphereDefaultIsolationLevel para establecer el nivel de aislamiento en 2(TRANSACTION_READ_COMMITTED) en la siguiente vía de acceso en la célula de WebSphere Application Server: **Recursos** > **Proveedor JDBC** > *proveedor_JDBC* > **Orígenes de datos** > *nombre_origen_datos* > **Propiedades personalizadas** > **Nueva**. Para obtener más información sobre la propiedad personalizada webSphereDefaultIsolationLevel y los niveles de aislamiento de transacción, consulte Requisitos para establecer los niveles de aislamiento para el acceso a datos.

- **Problema:** al utilizar la función de precarga de JPALoader o JPAEntityLoader, el mensaje CWOBJ1511 siguiente no se visualiza para la partición en un servidor de contenedor: `CWOBJ1511I: GRID_NAME:MAPSET_NAME:PARTITION_ID (primario) está abierto para operaciones empresariales`.

  En su lugar, se produce una excepción TargetNotAvailableException en el servidor de contenedor, que activa la partición especificada por la propiedad preloadPartition.

  Solución: establezca el atributo preloadMode en `true` si utiliza un JPALoader o JPAEntityLoader para precargar los datos en la correlación. Si la propiedad preloadPartition de JPALoader y JPAEntityLoader se establece en un valor entre `0` y `número_total_de_particiones - 1`, JPALoader y JPAEntityLoader intentan

precargar los datos de la base de datos de respaldo con la correlación. El fragmento de código siguiente ilustra cómo se establece el atributo preloadMode para habilitar la precarga asíncrona:

```
BackingMap bm = og.defineMap( "map1" );
bm.setPreloadMode( true );
```

También puede establecer el atributo preloadMode mediante un archivo XML, tal como se muestra en el ejemplo siguiente:

```
 <backingMap name="map1" preloadMode="true" pluginCollectionRef="map1"
lockStrategy="OPTIMISTIC"/>
```

Conceptos relacionados:

"Programación de la integración JPA" en la página 399
Java Persistence API (JPA) es una especificación que permite la correlación de objetos Java con bases de datos relacionales. JPA contiene una especificación de correlación de objetos relacionales (ORM) completa que utiliza las anotaciones de metadatos de lenguaje Java, los descriptores XML, o ambos, para definir la correlación entre los objetos Java y una base de datos relacional. Hay diversas implementaciones de código abierto y comerciales disponibles.

Configuración de la integración de la memoria caché
WebSphere eXtreme Scale se puede integrar con otros productos relacionados con la memoria caché. También puede utilizar el proveedor de memoria caché dinámica de WebSphere eXtreme Scale para conectar WebSphere eXtreme Scale en el componente de la memoria caché dinámica en WebSphere Application Server. Otra ampliación para WebSphere Application Server es el gestor de sesiones HTTP de WebSphere eXtreme Scale, que puede ayudar a colocar en la memoria caché las sesiones HTTP.

Resolución de problemas de puntos muertos

Las siguientes secciones describen algunos de los escenarios más comunes de punto muerto y algunas sugerencias para evitarlos.

Antes de empezar

Implemente el manejo de excepciones en la aplicación. Si desea más información, consulte "Implementación de manejo de excepciones en escenarios de bloqueo" en la página 253.

Como resultado, se visualiza la siguiente excepción:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: Mensaje
```

Este mensaje representa la serie que se pasa como parámetro cuando se crea y se emite la excepción.

Procedimiento

- **Problema:** excepción LockTimeoutException.

  Descripción: cuando una transacción o un cliente solicita que se otorgue un bloqueo para una entrada de correlación específica, la solicitud a menudo espera a que el cliente actual libere el bloqueo antes de enviar la solicitud. Si la solicitud de bloqueo permanece desocupada durante un periodo largo de tiempo, y no se otorga nunca un bloqueo, se crea una excepción LockTimeoutException para evitar un punto muerto, lo que se describe más

detalladamente en la sección siguiente. Es más probable que vea esta excepción al utilizar una estrategia de bloqueo pesimista, ya que el bloqueo nunca se libera hasta que se confirma la transacción.

Recupere más detalles:

La excepción LockTimeoutException contiene el método getLockRequestQueueDetails, que devuelve una serie. Puede utilizar este método para ver una descripción detallada de la situación que desencadena la excepción. A continuación se muestra un ejemplo de código que detecta la excepción y visualiza un mensaje de error.

```
try {
    ...
}
catch (LockTimeoutException lte) {
    System.out.println(lte.getLockRequestQueueDetails());
}
```

La salida resultante 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 =
    Esperando 5348 milisegundos, mode = U]
->[TX:163C328C-0105-4000-E114-5B3B090A571D, state =
    Esperando 1402 milisegundos, mode = U]
```

Si recibe una excepción en un bloque de detección de excepciones de ObjectGridException, el código siguiente determina la excepción y visualiza los detalles de la cola. También utiliza el método de programa de utilidad findRootCause.

```
try {
...
}
catch (ObjectGridException oe) {
    Throwable Root = findRootCause( oe );
    if (Root instanceof LockTimeoutException) {
        LockTimeoutException lte = (LockTimeoutException)Root;
        System.out.println(lte.getLockRequestQueueDetails());
    }
}
```

Solución: una excepción LockTimeoutException evita posibles puntos muertos en la aplicación. Una excepción de este tipo se genera cuando la excepción espera un periodo de tiempo establecido. Puede establecer el periodo de tiempo que espera una excepción utilizando el método setLockTimeout(int), que está disponible para la BackingMap. Si no existe realmente un punto muerto en la aplicación, ajuste el tiempo de espera de bloqueo para evitar la LockTimeoutException.

El siguiente código muestra cómo crear un objeto ObjectGrid, definir una correlación y establecer su valor LockTimeout en 30 segundos:

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("MapName");
bMap.setLockTimeout(30);
```

Utilice el ejemplo codificado anterior para establecer las propiedades de ObjectGrid y de correlación. Si crea ObjectGrid a partir de un archivo XML, establezca el atributo **LockTimeout** en el elemento backingMap. A continuación se muestra un ejemplo de un valor LockTimeout de correlación de 30 segundos.

```
<backingMap name="MapName" lockStrategy="PESSIMISTIC" lockTimeout="30">
```

- **Problema:** puntos muertos de una sola clave.

  Descripción: los escenarios siguientes describen cómo se pueden producir puntos muertos cuando se accede a una sola clave utilizando un bloqueo S que

se actualiza posteriormente. Cuando esto se produce desde dos transacciones que se ejecutan simultáneamente, se produce un punto muerto.

Tabla 14. Escenario de puntos muertos de llave única

	Hebra 1	**Hebra 2**	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.get(key1)	map.get(key1)	Se otorga el bloqueo S a las dos transacciones para key1.
3	map.update(Key1,v)		No se produce un bloqueo U. La actualización se realiza en la memoria caché transaccional.
4		map.update(key1,v)	No se produce un bloqueo U. La actualización se realiza en la memoria caché transaccional.
5	session.commit()		Bloqueado: el bloqueo S de key1 no se puede actualizar a un bloqueo X porque la hebra 2 tiene un bloqueo S.
6		session.commit()	Punto muerto: el bloqueo S de key1 no se puede actualizar a un bloqueo X porque la hebra 1 tiene un bloqueo S.

Tabla 15. Puntos muertos de llave única, continuación

	Hebra 1	**Hebra 2**	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.get(key1)		Se otorga el bloqueo S para key1.
3	map.getForUpdate(key1,v)		El bloqueo S se actualiza a un bloqueo U para key1.
4		map.get(key1)	Se otorga el bloqueo S para key1.
5		map.getForUpdate(key1,v)	Bloqueado: la hebra 1 ya tiene el bloqueo U.
6	session.commit()		Punto muerto: el bloqueo U de key1 no se puede actualizar.
7		session.commit()	Punto muerto: no se puede actualizar el bloqueo S para la clave key1.

Tabla 16. Puntos muertos de llave única, continuación

	Hebra 1	**Hebra 2**	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.get(key1)		Se otorga el bloqueo S para key1.
3	map.getForUpdate(key1,v)		El bloqueo S se actualiza a un bloqueo U para key1.
4		map.get(key1)	Se otorga el bloqueo S para key1.
5		map.getForUpdate(key1,v)	Bloqueado: la hebra 1 ya tiene el bloqueo U.

Tabla 16. Puntos muertos de llave única, continuación (continuación)

	Hebra 1	Hebra 2	
6	session.commit()		Punto muerto: el bloqueo U de key1 no se puede actualizar a un bloqueo X porque la hebra 2 tiene un bloqueo S.

Si se utiliza ObjectMap.getForUpdate para evitar el bloqueo S, no tendrá lugar el punto muerto:

Tabla 17. Puntos muertos de llave única, continuación

	Hebra 1	Hebra 2	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.getForUpdate(key1)		Se otorga el bloqueo U a la hebra 1 para key1.
3		map.getForUpdate(key1)	Se bloquea la solicitud de bloqueo U.
4	map.update(key1,v)	<bloqueado>	
5	session.commit()	<bloqueado>	El bloqueo U de key1 puede actualizarse correctamente a un bloqueo X.
6		<liberado>	El bloqueo U se otorga finalmente a key1 para la hebra 2.
7		map.update(key2,v)	Se otorga el bloqueo U a la hebra 2 para key2.
8		session.commit()	El bloqueo U de key1 puede actualizarse correctamente a un bloqueo X.

Soluciones:

1. Utilice el método getForUpdate en lugar de get para obtener un bloqueo U en lugar de un bloqueo S.
2. Use un nivel de aislamiento de transacción de lectura confirmada para evitar mantener bloqueos S. Al reducir el nivel de aislamiento de la transacción, aumenta la posibilidad de lecturas no repetibles. Sin embargo, las lecturas no repetibles de un cliente son solo posibles si la memoria caché de transacciones es invalidada explícitamente por el mismo cliente.
3. Use la estrategia de bloqueo optimista. El uso de la estrategia de bloqueo optimista requiere el manejo de excepciones de colisión optimista.

- **Problema:** punto muerto de varias claves ordenadas

  Descripción: este escenario describe lo que sucede si dos transacciones intentan actualizar la misma entrada directamente y mantienen bloqueos S a otras entradas.

Tabla 18. Escenario de punto muerto de varias llaves ordenadas

	Hebra 1	Hebra 2	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.get(key1)	map.get(key1)	Se otorga el bloqueo S a las dos transacciones para key1.
3	map.get(key2)	map.get(key2)	Se otorga el bloqueo S a las dos transacciones para key2.

Tabla 18. Escenario de punto muerto de varias llaves ordenadas (continuación)

	Hebra 1	Hebra 2	
4	map.update(key1,v)		No se produce un bloqueo U. La actualización se realiza en la memoria caché transaccional.
5		map.update(key2,v)	No se produce un bloqueo U. La actualización se realiza en la memoria caché transaccional.
6.	session.commit()		Bloqueado: el bloqueo S de key1 no se puede actualizar a un bloqueo X porque la hebra 2 tiene un bloqueo S.
7		session.commit()	Punto muerto: el bloqueo S de key2 no se puede actualizar porque la hebra 1 tiene un bloqueo S.

Puede utilizar el método ObjectMap.getForUpdate para evitar el bloqueo S, después puede evitar el punto muerto:

Tabla 19. Escenario de punto muerto de varias llaves ordenadas, continuación

	Hebra 1	Hebra 2	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.getForUpdate(key1)		Se otorga el bloqueo U a la transacción de hebra 1 para key1.
3		map.getForUpdate(key1)	Se bloquea la solicitud de bloqueo U.
4	map.get(key2)	<bloqueado>	Se otorga el bloqueo S a la hebra 1 para key2.
5	map.update(key1,v)	<bloqueado>	
6	session.commit()	<bloqueado>	El bloqueo U de key1 puede actualizarse correctamente a un bloqueo X.
7		<liberado>	El bloqueo U se otorga finalmente a key1 para la hebra 2.
8		map.get(key2)	Se otorga el bloqueo S a la hebra 2 para key2.
9		map.update(key2,v)	Se otorga el bloqueo U a la hebra 2 para key2.
10		session.commit()	El bloqueo U de key1 puede actualizarse correctamente a un bloqueo X.

Soluciones:

1. Utilice el método getForUpdate en lugar del método get para adquirir un bloqueo U directamente para la primera clave. Esta estrategia sólo funciona si el orden de los métodos es determinista.
2. Use un nivel de aislamiento de transacción de lectura confirmada para evitar mantener bloqueos S. Esta solución es la más fácil de implementar si el orden de los métodos no es determinista. Al reducir el nivel de aislamiento de la transacción, aumenta la posibilidad de lecturas no repetibles. No obstante, las lecturas no repetibles sólo son posibles si la memoria caché de la transacción se invalida explícitamente.

3. Use la estrategia de bloqueo optimista. El uso de la estrategia de bloqueo optimista requiere el manejo de excepciones de colisión optimista.

- **Problema:** fuera de servicio con bloqueo U

  Descripción: si el orden en el que se solicitan las claves no se puede garantizar, aún se puede producir un punto muerto.

Tabla 20. Escenario de fuera de servicio con bloqueo U

	Hebra 1	**Hebra 2**	
1	session.begin()	session.begin()	Cada hebra establece una transacción independiente.
2	map.getforUpdate(key1)	map.getForUpdate(key2)	Se otorgan correctamente bloqueos U para key1 y key2.
3	map.get(key2)	map.get(key1)	Se otorga el bloqueos S para key1 y key2.
4	map.update(key1,v)	map.update(key2,v)	
5	session.commit()		El bloqueo U no se puede actualizar a un bloqueo X porque la hebra 2 tiene un bloqueo S.
6		session.commit()	El bloqueo U no se puede actualizar a un bloqueo X porque la hebra 1 tiene un bloqueo S.

Soluciones:

1. Envuelva todo el trabajo con un único bloqueo U global (mútex). Este método reduce la simultaneidad, pero maneja todos los escenarios cuando el acceso y el orden no son deterministas.
2. Use un nivel de aislamiento de transacción de lectura confirmada para evitar mantener bloqueos S. Esta solución es la más fácil de implementar si el orden de los métodos no es determinista y proporciona un alto nivel de simultaneidad. Al reducir el nivel de aislamiento de la transacción, aumenta la posibilidad de lecturas no repetibles. No obstante, las lecturas no repetibles sólo son posibles si la memoria caché de la transacción se invalida explícitamente.
3. Use la estrategia de bloqueo optimista. El uso de la estrategia de bloqueo optimista requiere el manejo de excepciones de colisión optimista.

Conceptos relacionados:

"Bloqueos" en la página 249
Los bloqueos tienen ciclos de vida y tipos de bloqueos diferentes son compatibles con otros de distintas formas. Los bloqueos deben manejarse en el orden correcto para evitar escenarios de punto muerto.

IBM Support Assistant para WebSphere eXtreme Scale

Puede utilizar IBM Support Assistant para recopilar los datos, analizar los síntomas y acceder a la información sobre el producto.

IBM Support Assistant Lite

IBM Support Assistant Lite para WebSphere eXtreme Scale proporciona una recopilación automática de los datos y soporte de análisis de síntomas para los casos de determinación de problemas.

IBM Support Assistant Lite reduce el tiempo que lleva reproducir un problema con los niveles de rastreo establecidos correctos de fiabilidad, disponibilidad y capacidad de servicio (la herramienta establece automáticamente los niveles de rastreo) para simplificar la determinación de problemas. Si necesita más asistencia, IBM Support Assistant Lite reduce también el esfuerzo necesario para enviar la información de registro adecuada a IBM Support.

IBM Support Assistant Lite se incluye en todas las instalaciones de WebSphere eXtreme Scale Versión 7.1.0

IBM Support Assistant

IBM® Support Assistant (ISA) proporciona un acceso rápido a los recursos del producto, formación y soporte que pueden ayudarle a contestar las preguntas y a resolver los problemas con los productos de software de IBM por sí solo, sin necesidad de ponerse en contacto con IBM Support. Distintos plug-ins específicos del producto le permiten personalizar IBM Support Assistant para los productos concretos que ha instalado. IBM Support Assistant recopila además los datos del sistema, los archivos de registro y otra información para ayudar a IBM Support a determinar la causa de un problema concreto.

IBM Support Assistant es un programa de utilidad para instalarlo en la estación de trabajo, no directamente en el sistema servidor WebSphere eXtreme Scale en sí. Los requisitos de memoria y de recursos para Assistant podrían afectar negativamente al rendimiento del sistema servidor WebSphere eXtreme Scale. Los componentes de diagnóstico portátiles incluidos están diseñados para un impacto mínimo en la operación normal de un servidor.

Puede utilizar IBM Support Assistant para que le ayude de estos modos:

- Para buscar en las fuentes de información y de conocimientos de IBM y no IBM entre varios productos de IBM para contestar una pregunta o solucionar un problema
- Para encontrar información adicional en los recursos web específicos del producto; incluidas las páginas iniciales del producto y de soporte, los foros y los grupos de noticias de clientes, las capacidades y los recursos de formación y la información sobre resolución de problemas y preguntas más frecuentes
- Para ampliar la capacidad para diagnosticar los problemas específicos del producto con herramientas de diagnóstico orientadas disponibles en Support Assistant
- Para simplificar la recopilación de datos de diagnóstico para ayudarle a usted y a IBM a resolver los problemas (recopilando datos generales o específicos del síntoma o producto)
- Para ayudarle a informar de las incidencias de problemas a IBM Support mediante una interfaz en línea personalizada para adjuntar los datos de diagnóstico mencionados anteriormente o cualquier otra información a las incidencias nuevas o existentes.

Finalmente, puede utilizar el recurso actualizador incorporado para obtener soporte de los productos y las capacidades de software adicionales a medida que están disponibles. Para configurar IBM Support Assistant para utilizarlo con WebSphere eXtreme Scale, instale en primer lugar IBM Support Assistant con los archivos proporcionados en la imagen descargada de la página web Visión general de soporte de IBM en: http://www-947.ibm.com/support/entry/portal/Overview/Software/Other_Software/IBM_Support_Assistant. A continuación, utilice IBM Support Assistant para ubicar e instalar las actualizaciones del

producto. Puede elegir también instalar los plug-ins disponibles para otro software de IBM en el entorno. Hay disponible más información y la última versión de IBM Support Assistant desde la página web de IBM Support Assistant en la dirección: http://www.ibm.com/software/support/isa/.

Avisos

Las referencias en esta publicación a productos, programas o servicios de IBM no implica que IBM tenga previsto ponerlos a la venta en todos los países en los que IBM opera. Cualquier referencia a un producto, programa o servicio de IBM no pretende indicar ni implica que sólo se pueda utilizar este producto, programa o servicio de IBM. En su lugar, se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ningún derecho de propiedad intelectual de IBM. La evaluación y la verificación del funcionamiento con otros productos, excepto aquellos expresamente designados por IBM, es responsabilidad del usuario.

IBM puede tener patentes o solicitudes de patentes pendientes que conciernan al tema de este documento. La posesión de este documento no le da ninguna licencia sobre estas patentes. Puede enviar preguntas acerca de licencias por escrito a:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 Estados Unidos

Los propietarios de licencias de este programa que deseen obtener información sobre el mismo con el fin de habilitar: (i) el intercambio de información entre programas creados de forma independiente y otros programas (incluido este) y (ii) el uso mutuo de la información intercambiada, se deben poner en contacto con:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
Estados Unidos
Attention: Information Requests

Esta información puede estar disponible, bajo las condiciones y los términos adecuados, incluyendo en algunos casos, el pago de una cuota.

Marcas registradas

Los siguientes términos son marcas registradas de IBM Corporation en Estados Unidos yen otros países.

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java y todas las marcas registradas basadas en Java son marcas registradas de Sun Microsystems, Inc. en Estados Unidos y/o en otros países.

LINUX es una marca registrada de Linus Torvalds en Estados Unidos y/o en otros países.

Microsoft, Windows, Windows NT y el logotipo de Windows son marcas registradas de Microsoft Corporation en Estados Unidos y/o en otros países.

UNIX es una marca registrada de The Open Group en Estados Unidos y en otros países.

Otros nombres de compañías, productos y servicios pueden ser marcas registradas o de servicio de terceros.

Índice

D

E

F

G

H

I

J

L

M

O

P

R

S

T

V

X

Impreso en España