IBM WebSphere eXtreme Scale Versão 7.1.1
Versão 7 Release 1

Guia de Programação

21 de Novembro de 2011

Esta edição aplica-se à versão 7, release 1, modificação 1 do WebSphere eXtreme Scale e a todos os releases e modificações subsequentes até que seja indicado de outra forma em novas edições.

Índice

Figuras

Tabelas

Sobre o *Guia de Programação*

O conjunto da documentação do WebSphere eXtreme Scale inclui três volumes que fornecem as informações necessárias para utilizar, programar e administrar o produto WebSphere eXtreme Scale.

Biblioteca do WebSphere eXtreme Scale

A biblioteca do WebSphere eXtreme Scale contém os seguintes livros:

- O *Visão Geral do Produto* contém um visualização de alto nível dos conceitos do WebSphere eXtreme Scale, incluindo cenários de caso de uso e tutoriais.
- O *Guia de Instalação* descreve como instalar topologias comuns do WebSphere eXtreme Scale.
- O *Guia de Administração* contém as informações necessárias para os administradores de sistema, incluindo como planejar implementações do aplicativo, planejar capacidade, instalar e configurar o produto, iniciar e parar servidores, monitorar o ambiente e proteger o ambiente.
- O *Guia de Programação* contém informações para desenvolvedores de aplicativos sobre como desenvolver aplicativos para o WebSphere eXtreme Scale utilizando as informações da API incluídas.

Para fazer download dos manuais, vá para a Página da Biblioteca do WebSphere eXtreme Scale.

Também é possível acessar as mesmas informações nesta biblioteca no Centro de Informações do WebSphere eXtreme Scale Versão 7.1.1.

Usando Manuais Off-line

Todos os manuais na biblioteca do WebSphere eXtreme Scale contêm links para o centro de informações com a seguinte URL raiz: `http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r1m1`. Esses links levam diretamente para as informações relacionadas. No entanto, se estiver trabalhando off-line e encontrar um desses links, será possível procurar pelo título do link nos outros manuais na biblioteca. A documentação da API, o glossário e a referência de mensagens não estão disponíveis em manuais PDF.

Quem Deve Utilizar este Manual

Este manual é destinado principalmente a desenvolvedores de aplicativos.

Obtendo Atualizações para este Manual

É possível obter as atualizações para esse manual ao fazer download da versão mais recente da Página da Biblioteca do WebSphere eXtreme Scale.

Como Enviar Seus Comentários

Entre em contato com a equipe de documentação. Você localizou o que precisava? O conteúdo era exato e completo? Envie seus comentários sobre esta documentação por e-mail para wasdoc@us.ibm.com.

Capítulo 1. Tutoriais

Os tutoriais podem ser usados para ajudar a entender os cenários de uso do produto, incluindo o gerenciador, consultas e segurança da entidade.

Tutorial: Consultando uma Grade de Dados na Memória Local

É possível desenvolver um ObjectGrid na memória local que pode armazenar informações de pedido para um website e usar a API do ObjectQuery para consultar a grade de dados.

Antes de Iniciar

Certifique-se de ter o arquivo `objectgrid.jar` em seu caminho de classe.

Sobre Esta Tarefa

Cada etapa no tutorial é construída na etapa anterior. Siga cada uma das etapas para construir um aplicativo Java Platform, Standard Edition Versão 5 ou posterior simples que usa uma grade de dados locais na memória.

Tutorial do ObjectQuery - Etapa 1

Com as seguintes etapas, você poderá continuar desenvolvendo um ObjectGrid local de memória que armazena informações de pedidos para uma loja varejista on-line usando as APIs do ObjectMap. Defina um esquema para o mapa e execute uma consulta em relação ao mapa.

Procedimento

1. Crie um ObjectGrid com um esquema de mapa.

   Crie um ObjectGrid com um esquema de mapa para o mapa; em seguida, insira um objeto no cache e recupere-o posteriormente, utilizando uma consulta simples.

   OrderBean.java

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

2. Defina a chave principal.

   O código anterior mostra um objeto OrderBean. Este objeto implementa a interface java.io.Serializable porque todos os objetos no cache devem (por padrão) ser Serializáveis.

   O atributo orderNumber é a chave principal do objeto. O programa de exemplo a seguir pode ser executado no modo independente. É necessário seguir esse tutorial em um projeto Eclipse Java que tenha o arquivo `objectgrid.jar` incluído no caminho de classe.

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 o 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();
    }
}
```

Esse aplicativo eXtreme Scale primeiro inicializa um ObjectGrid local com um nome gerado automaticamente. Em seguida, o aplicativo cria um BackingMap e um QueryConfig que definem qual tipo Java está associado ao mapa, o nome do campo que é a chave principal para o mapa e como acessar os dados no objeto. A seguir, você obtém uma Sessão para adquirir a instância do ObjectMap e inserir um objeto OrderBean no mapa em uma transação.

Depois que os dados forem confirmados no cache, será possível usar o ObjectQuery para localizar o OrderBean usando qualquer um dos campos persistentes na classe. Campos persistentes são aqueles que não possuem o modificador temporário. Como nenhum índice foi definido no BackingMap, o ObjectQuery deverá varrer cada objeto no mapa usando o reflexo Java.

O que Fazer Depois

O “Tutorial do ObjectQuery - Etapa 2” demonstra como um índice pode ser usado para otimizar a consulta.

Tutorial do ObjectQuery - Etapa 2

Nas seguintes etapas, você continuará criando um ObjectGrid com um mapa e um índice, junto com um esquema para o mapa. Em seguida, você poderá inserir um objeto no cache e, mais tarde, recuperá-lo utilizando uma consulta simples.

Antes de Iniciar

Certifique-se de ter concluído o "Tutorial do ObjectQuery - Etapa 1" na página 1 antes de continuar com esta etapa do tutorial.

Procedimento

Esquema e índice

Application.java

```
// Create an index
    HashIndex idx= new HashIndex();
    idx.setName("theItemName");
    idx.setAttributeName("itemName");
    idx.setRangeIndex(true);
    idx.setFieldAccessAttribute(true);
    orderBMap.addMapIndexPlugin(idx);
}
```

O índice deve ser uma instância com.ibm.websphere.objectgrid.plugins.index.HashIndex com as seguintes configurações:

- O Nome é arbitrário, mas deve ser exclusivo para um BackingMap fornecido.
- O AttributeName é o nome do campo ou propriedade do bean que o mecanismo de indexação utiliza para examinar a classe. Neste caso, este é o nome do campo para o qual você criará um índice.
- RangeIndex deve ser sempre verdadeiro.
- FieldAccessAttribute deve corresponder ao conjunto de valores no objeto QueryMapping quando o esquema de consulta foi criado. Nesse caso, o objeto Java é acessado usando os campos diretamente.

Quando uma consulta executa esses filtros no campo itemName, o mecanismo de consulta automaticamente usa o índice definido. Usar o índice permite que a consulta seja executada muito mais rapidamente e uma varredura de mapa não é necessária. A próxima etapa demonstra como um índice pode ser utilizado para otimizar a consulta.
Próxima etapa

Tutorial do ObjectQuery - Etapa 3

Na etapa a seguir, você criará um ObjectGrid com dois mapas e um esquema para os mapas com um relacionamento e, em seguida, inserirá os objetos no cache e posteriormente irá recuperá-los utilizando uma consulta simples.

Antes de Iniciar

Certifique-se de ter concluído o "Tutorial do ObjectQuery - Etapa 2" na página 2 antes de continuar com esta etapa.

Sobre Esta Tarefa

Neste exemplo, há dois mapas, cada um com um tipo único Java mapeado para ele. O mapa Order possui objetos OrderBean e o mapa Customer contém objetos CustomerBean.

Procedimento

Defina mapas com um relacionamento.

OrderBean.java

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

O OrderBean não contém mais o customerName. Ao invés disso, ele contém o customerId, que é a chave principal para o objeto CustomerBean e o mapa Customer.

CustomerBean.java

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

O relacionamento entre os dois tipos ou Mapas é:

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 o 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();
    }
}
```

O XML equivalente no descritor de implementação do ObjectGrid é:

```
<?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>
```

O que Fazer Depois

O "Tutorial do ObjectQuery - Etapa 4" expande a etapa atual ao incluir um campo, objetos de acesso da propriedade e relacionamentos adicionais.

Tutorial do ObjectQuery - Etapa 4

A etapa a seguir mostra como criar um ObjectGrid com quatro mapas e um esquema para os mapas com vários relacionamentos unidirecionais e bidirecionais. Em seguida, você poderá inserir objetos no cache e, mais tarde, recuperá-los utilizando várias consultas.

Antes de Iniciar

Certifique-se de ter concluído o "Tutorial do ObjectQuery - Etapa 3" na página 3 antes de continuar com a etapa atual.

Procedimento

Relacionamentos de mapas múltiplos

OrderBean.java

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

Na etapa anterior, o OrderBean não possuía mais o customerName nele. Ao invés disso, ele contém o customerId, que é a chave principal para o objeto CustomerBean e o mapa Customer.

CustomerBean.java

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

Ao ter criado as classes especificadas acima, é possível executar o aplicativo abaixo.

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 o 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();
    }
}
```

Utilizar a configuração XML abaixo (no descritor de implementação ObjectGrid) é equivalente à abordagem programática acima.

```
<?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>
```

Tutorial: Armazenando Informações de Pedido nas Entidades

Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Antes de Iniciar

Certifique-se de atender aos seguintes requisitos antes de começar o tutorial:

- É necessário ter o Java SE 5.
- É necessário ter o arquivo `objectgrid.jar` em seu caminho de classe.

Conceitos relacionados

"Objetos de Armazenamento em Cache sem Relacionamentos Envolvidos (API ObjectMap)" na página 155
Os ObjectMaps são como Mapas Java que permitem que os dados sejam armazenados como pares chave-valor. Os ObjectMaps apresentam uma abordagem simples e intuitiva para o aplicativo que armazenará os dados. Um ObjectMap é ideal para o armazenamento em cache de objetos que não tenham nenhum relacionamento envolvido. Se os relacionamentos de objetos estiverem envolvidos, então você deve usar a API EntityManager.

"Ajustando o Desempenho da Interface EntityManager" na página 463
A interface EntityManager separa aplicativos do estado de suspensão no armazenamento de dados da grade do servidor.

"Objetos de Armazenamento em Cache e seus Relacionamentos (API EntityManager)" na página 166
A maioria dos produtos de cache utiliza APIs baseadas em mapa para armazenar dados como pares de chave-valor. A API ObjectMap e o cache dinâmico no WebSphere Application Server, entre outros, usam essa abordagem. Entretanto, APIs baseadas em mapas têm limitações. A API EntityManager simplifica a interação com a grade de dados ao fornecer uma maneira fácil de declarar e interagir com um gráfico complexo de objetos relacionados.

"Entity Manager em um Ambiente Distribuído" na página 177
É possível usar a API EntityManager com um ObjectGrid local ou em um ambiente distribuído do eXtreme Scale . A principal diferença é como você se conecta a esse ambiente remoto. Após você estabelecer uma conexão, não existe diferença entre o uso de um objeto Session ou uma API do EntityManager.

"Interagindo com EntityManager" na página 182
Geralmente os aplicativos primeiro obtêm uma referência do ObjectGrid e, depois, uma Sessão dessa referência para cada encadeamento. As sessões não podem ser compartilhadas entre encadeamentos. Um método extra em Session, o método getEntityManager, está disponível. Este método retorna uma referência para um gerenciador de entidades para uso para este encadeamento. A interface de EntityManager pode substituir as interfaces de Session e ObjectMap para todos os aplicativos. É possível utilizar essas APIs de EntityManager se o cliente tiver acesso às classes de entidade definidas.

"Suporte ao Plano de Carregamento do EntityManager" na página 191
Um FetchPlan é a estratégia que o gerenciador de entidade usa para recuperar objetos associados se o aplicativo precisar acessar relacionamentos.

"Filas de Consulta da Entidade" na página 196
Filas de consulte permitem que aplicativos criem uma fila qualificada por uma consulta no lado do servidor ou eXtreme Scale local sobre uma entidade. As entidades do resultado da consulta são armazenadas nesta fila. Atualmente, a fila de consulta é suportada apenas em um mapa que está utilizando a estratégia de bloqueio pessimista.

Referências relacionadas

"Agente de Instrumentação de Desempenho da Entidade" na página 465
É possível melhorar o desempenho de entidades de acesso a campo ativando o agente de instrumentação do WebSphere eXtreme Scale ao utilizar o Java Development Kit (JDK) Versão 1.5 ou posterior.

"Definindo um Esquema de Entidade" na página 169
Um ObjectGrid pode ter inúmeros esquemas de entidade lógicos. As entidades são definidas usando as classes Java anotadas, o XML ou uma combinação de classes XML e Java. Entidades definidas são registradas com um servidor eXtreme Scale e ligadas a BackingMaps, índices e outros plug-ins.

"Listeners de Entidade e Métodos de Retorno de Chamada" na página 185
Os aplicativos podem ser notificados quando o estado de uma entidade é alterado de estado para estado. Dois mecanismos de retorno de chamada existem para os eventos de alteração de estado: os métodos de retorno de chamada do ciclo de vida, que são definidos em uma classe de entidade e são chamados sempre que o estado da entidade for alterado, e os listeners de entidade que são mais genéricos porque o listener da entidade pode ser registrado em várias entidades.

"Exemplos do Listener de Entidade" na página 189
É possível gravar EntityListeners com base em seus requisitos. Veja a seguir vários scripts de exemplo.

"Interface EntityTransaction" na página 201
É possível utilizar a interface EntityTransaction para demarcar transações.

Informações relacionadas

"Lição 2 do Tutorial de Introdução: Criando um Aplicativo Cliente" na página 63
Para inserir, excluir, atualizar e recuperar dados de sua grade de dados, você deverá gravar um aplicativo cliente. A introdução de amostra inclui um aplicativo cliente que pode ser usado para saber mais sobre a criação de seu próprio aplicativo cliente.

Tutorial do Entity Manager: Criando uma Classe de Entidade

Crie um ObjectGrid local com uma entidade criando uma classe Entity, registrando o tipo de entidade e armazenando uma instância da entidade no cache.

Procedimento

1. Crie o objeto Order. Para identificar o objeto como uma entidade ObjectGrid, inclua a anotação @Entity. Ao incluir esta anotação, todos os atributos serializáveis no objeto são automaticamente persistidos no eXtreme Scale, a menos que você utilize anotações nos atributos para substituí-los. O atributo **orderNumber** é anotado com `@Id` para indicar que este atributo é a chave primária. A seguir, está um exemplo de um objeto Order:

   Order.java

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

2. Execute o aplicativo eXtreme Scale Hello World para demonstrar as operações entity. O programa de exemplo a seguir pode ser emitido no modo independente para demonstrar as operações entity. Use esse programa em um projeto Eclipse Java que tenha o arquivo `objectgrid.jar` incluído no caminho de classe. A seguir, está um exemplo de um aplicativo Hello world simples que utiliza o 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();
    }
}
```

Este aplicativo de exemplo executa as seguintes operações:

a. Inicializa um eXtreme Scale local com um nome gerado automaticamente.
b. Registra as classes entity com o aplicativo utilizando a API do registerEntities, embora utilizar a API do registerEntities não seja sempre necessário.
c. Recupera um objeto Session e uma referência para o entity manager para Session.
d. Associa cada objeto Session do eXtreme Scale com um EntityManager e EntityTransaction únicos. O EntityManager agora é utilizado.
e. O método registerEntities cria um objeto BackingMap que é chamado Order e associa os metadados para o objeto Order com o objeto BackingMap. Esses metadados incluem os atributos chave e não-chave, juntamente com os tipos e nomes de atributo.
f. Uma transação inicia e cria uma instância Order. A transação é preenchida com alguns valores. A transação é, então, persistida usando o método EntityManager.persist, que identifica a entidade como aguardando para ser incluída no mapa associado.
g. A transação é, então, confirmada, e a entidade é incluída na instância de ObjectMap.
h. Uma outra transação é feita e o objeto Order é recuperado usando a chave 1. O cast de tipo no método EntityManager.find é necessário. A capacidade do Java SE 5 não é usada para assegurar que o arquivo `objectgrid.jar` funciona em um Java SE Versão 5 e Java Virtual Machine posterior.

Tutorial do Entity Manager: Formando Relacionamentos de Entidades

Crie um relacionamento simples entre entidades criando duas classes de entidades com um relacionamento, registrando as entidades com o ObjectGrid e armazenamento as instâncias da entidade no cache.

Procedimento

1. Crie a entidade customer, que é usada para armazenar detalhes do cliente independentemente do objeto Order. Um exemplo da entidade customer é apresentado a seguir:

Customer.java

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

Esta classe inclui informações sobre o cliente, tais como nome, endereço e número de telefone.

2. Crie o objeto Order, que é semelhante ao objeto Order no tópico do "Tutorial do Entity Manager: Criando uma Classe de Entidade" na página 9. A seguir, está um exemplo do 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;
}
```

Neste exemplo, uma referência para um objeto Customer substitui o atributo customerName. A referência possui uma anotação que indica uma relação muitos-para-um. Um relacionamento muitos-para-um indica que cada pedido possui um cliente, mas vários pedidos podem fazer referência ao mesmo cliente. O modificador de anotação em cascata indica que, se o gerenciador de entidade persistir o objeto Order, ele também deverá persistir o objeto Customer. Se você decidir não definir a opção de persistência em cascata, que é a opção padrão, deve persistir manualmente o objeto Customer com o objeto Order.

3. Utilizando as entidades, defina os mapas para a instância do ObjectGrid. Cada mapa é definido para uma entidade específica e uma entidade é denominada Order e a outra é denominada Customer. O aplicativo de exemplo a seguir ilustra como armazenar e recuperar um pedido do 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();
    }
}
```

Este aplicativo é semelhante ao aplicativo de exemplo que está na etapa anterior. No exemplo anterior, apenas uma única classe Order é registrada. O WebSphere eXtreme Scale detecta e automaticamente inclui a referência na entidade Customer e uma instância Customer para `John Smith` é criada e referenciada a partir do novo objeto Order. Como resultado, o novo cliente é persistido automaticamente, porque o relacionamento entre duas ordens inclui o modificador em cascata, que requer que cada objeto seja persistido. Quando o objeto Order é localizado, o entity manager automaticamente localiza o objeto Customer associado e insere uma referência no objeto.

Tutorial do Entity Manager: Esquema da Entidade Order

Crie quatro classes de entidade utilizando relacionamentos únicos e bidirecionais, listas ordenadas e relacionamentos de chave estrangeira. As APIs do EntityManager são utilizadas para persistir e localizar as entidades. Com base nas entidades Order e Customer que estão nas partes anteriores do tutorial, esta etapa do tutorial inclui mais duas entidades: as entidades Item e OrderLine.

Sobre Esta Tarefa

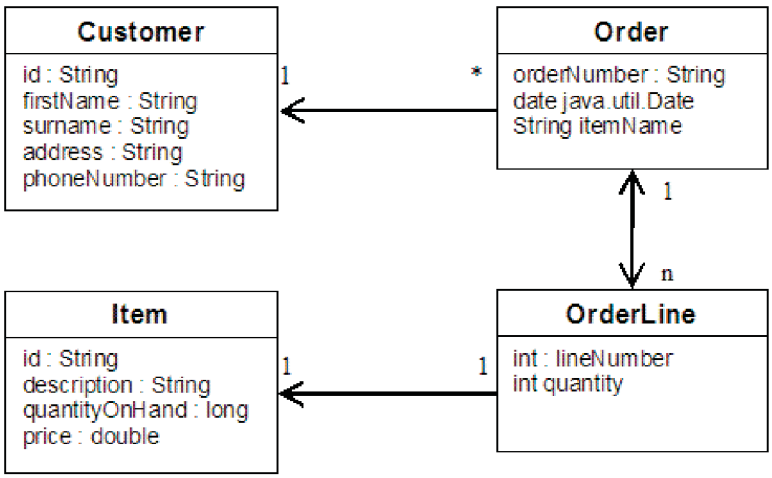


Figura 1. Esquema da Entidade Order. Uma entidade Order possui uma referência para um cliente e zero ou mais OrderLines. Cada entidade OrderLine possui uma referência para um único item e inclui a quantidade solicitada.

Procedimento

1. Crie a entidade customer, que é semelhante aos exemplos anteriores.

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

```
    @Id String id;
    String firstName;
    String surname;
    String address;
    String phoneNumber;
}
```

2. Crie a entidade Item, que contém informações sobre um produto que está incluído no inventário da loja, como a descrição do produto, a quantidade e o preço.

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

3. Crie a entidade OrderLine. Cada Order possui zero ou mais OrderLines, que identificam a quantidade de cada item no pedido. A chave para a OrderLine é uma chave composta que consiste no Order que possui o OrderLine e um número inteiro que designa um número para a linha do pedido. Inclua o modificador de persistência em cascata em cada relacionamento em suas 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. Criar o Order Object final, que possui uma referência ao Customer para a ordem e uma coleta 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;  }
   ```

   ALL em cascata é utilizado como o modificador para as linhas. Esse modificador sinaliza o EntityManager para exibir em cascata a operação PERSIST e a operação REMOVE. Por exemplo, se a entidade Order for persistida ou removida, então, todas as entidades OrderLine também são persistidas ou removidas.

   Se uma entidade OrderLine for removida da lista de linhas no objeto de Pedido, a referência então será quebrada. No entanto, a entidade OrderLine não será removida do cache. Você deve utilizar a API de remoção do EntityManager para remover entidades do cache. A operação REMOVE não é utilizada na entidade do cliente ou na entidade de item de OrderLine. Como resultado, a entidade do cliente permanece mesmo que o item ou o item seja removido quando a OrderLine for removida.

   O modificador mappedBy indica um relacionamento inverso com a entidade de destino. O modificador identifica qual atributo na entidade de destino refere-se à entidade de origem, e o lado pertencente de um relacionamento um para um

ou muitos para muitos. Geralmente, é possível omitir o modificador. Entretanto, um erro é exibido para indicar que ele deve ser especificado se WebSphere eXtreme Scale não puder descobrí-lo automaticamente. Uma entidade OrderLine que contém dois tipos de atributos Order em uma relacionamento muitos para um normalmente causa o erro.

A anotação @OrderBy especifica a ordem na qual cada entidade OrderLine deve estar na lista de linhas. Se a anotação não for especificada, então, as linhas são exibida em uma ordem arbitrária. Embora as linhas sejam incluídas na entidade Order emitindo ArrayList, o que preserva o pedido, o EntityManager não necessariamente reconhecerá a pedido. Quando você emite o método de localização para recuperar o objeto Order do cache, o objeto de lista não é um objeto ArrayList.

5. Crie o aplicativo. O exemplo a seguir ilustra o objeto Order final, que possui uma referência para o Customer para o pedido e uma coleta de objetos OrderLine.
   a. Encontre os Itens a serem ordenados, que podem se tornar entidades Gerenciadas.
   b. Crie a OrderLine e anexe-a a cada Item.
   c. Crie o Pedido e associe-o a cada OrderLine e ao cliente.
   d. Persista o pedido, que persiste automaticamente cada OrderLine.
   e. Confirme a transação, que desconecta cada entidade e sincroniza o estado das entidades com o cache.
   f. Imprima as informações do pedido. As entidades OrderLine são armazenadas automaticamente pelo ID da OrderLine.

```
Application.java

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

        // Add some items to our inventory.
        em.getTransaction().begin();
        createItems(em);
        em.getTransaction().commit();

        // Create a new customer with the items in his cart.
        em.getTransaction().begin();
        Customer cust = createCustomer();
        em.persist(cust);

        // Create a new order and add an order line for each item.
        // Each line item is automatically persisted since the
    // Cascade=ALL option is set.
        Order order = createOrderFromItems(em, cust, "ORDER_1",
     new String[]{"1", "2"}, new int[]{1,3});
        em.persist(order);
        em.getTransaction().commit();

        // Print the order summary
        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;
    }
```

A próxima etapa é excluir uma entidade. A interface EntityManager possui um método de remoção que marca um objeto como excluído. O aplicativo deve remover a entidade de qualquer coleta de relacionamento antes de chamar o método de remoção. Edite as referências e emita o método de remoção ou em.remove(object), como uma etapa final.

Tutorial do Entity Manager: Atualizando Entradas

Se você desejar alterar uma entidade, é possível localizar a instância, atualizar a instância e quaisquer entidades referenciadas, além de executar o commit da transação.

Procedimento

Entradas de atualização. O exemplo a seguir demonstra como localizar a instância Order, alterá-la e qualquer entidade mencionada, e confirmar a transação.

```
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);
    }
}
```

Executar o flushing da transação sincroniza todas as entidades gerenciadas com o cache. Quando ocorre o commit de uma transação, automaticamente ocorre um flush. Neste caso, Order se torna uma entidade gerenciada. Quaisquer entidades referenciadas de Order, Customer e OrderLine também se tornam entidades gerenciadas. No flush da transação, cada entidade é verificada para determinar se foi modificada. As que foram modificadas são atualizadas no cache. Após a conclusão da transação, através de commit ou rollback, as entidades se separam e quaisquer alterações feitas nas entidades não são refletidas no cache.

Tutorial do Entity Manager: Atualizando e Removendo Entradas com um Índice

É possível utilizar um índice para localizar, atualizar e remover entidades.

Procedimento

Atualize e remova entidades utilizando um índice. Utilize um índice para localizar, atualizar e remover entidades. Nos exemplos anteriores, a classe de entidade Order é atualizada para utilizar a anotação @Index. A anotação @Index sinaliza ao WebSphere eXtreme Scale para criar um índice de intervalo para um atributo. O nome do índice é o mesmo nome do atributo e é sempre um 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;  }
```

O exemplo a seguir demonstra como cancelar todas os pedidos enviados no último minuto. Encontrar o pedido utilizando um índice, incluir os itens no pedido de volta no inventário e remover o pedido e os itens da linha associados do sistema.

```
public static void cancelOrdersUsingIndex(Session s)
  throws ObjectGridException {
    // Cancel all orders that were submitted 1 minute ago
    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();
   // Localizar o Pedido para que possamos removê-lo.
   Order curOrder = (Order) em.find(Order.class, orderKey);
   // Verificar se o pedido não foi atualizado por outra pessoa.
   if(curOrder != null && curOrder.date.getTime() >= cancelTime.getTime()) {
    for(OrderLine line : curOrder.lines) {
     // Incluir o item novamente no inventário.
     line.item.quantityOnHand += line.quantity;
     line.quantity = 0;
    }
   em.remove(curOrder);
   }
  }
 em.getTransaction().commit();
}
```

Tutorial do Entity Manager: Atualizando e Removendo Entradas Utilizando uma Consulta

É possível atualizar e remover entidades utilizando uma consulta.

Procedimento

Atualize e remova entidades utilizando uma 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;  }
```

A classe de entidade order é a mesma que a do exemplo anterior. A classe ainda fornece a anotação @Index, porque a cadeia de consultas utiliza a data para localizar a entidade. O mecanismo de consulta utiliza índices quando eles podem ser utilizados.

```
public static void cancelOrdersUsingQuery(Session s) {
        // Cancel all orders that were submitted 1 minute ago
        java.util.Date cancelTime =
     new java.util.Date(System.currentTimeMillis() - 60000);
        EntityManager em = s.getEntityManager();
        em.getTransaction().begin();

        // Create a query that will find the order based on date.  Since
        // we have an index defined on the order date, the query
    // will automatically use it.
        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 se o pedido não foi atualizado por outra pessoa.
     // Since the query used an index, there was no lock on the row.
     if(order != null && order.date.getTime() >= cancelTime.getTime()) {
       for(OrderLine line : order.lines) {
        // Incluir o item novamente no inventário.
        line.item.quantityOnHand += line.quantity;
        line.quantity = 0;
       }
       em.remove(order);
     }
   }
  em.getTransaction().commit();
}
```

Como o exemplo anterior, o método cancelOrdersUsingQuery é destinado a cancelar todos os pedidos que foram enviados no último minuto. Para cancelar o pedido, você o localiza utilizando uma consulta, inclui os itens no pedido de volta

no inventário e remove o pedido e os itens de linha associados do sistema.

Tutorial: Executando Pacotes Configuráveis do eXtreme Scale na Estrutura do OSGi

A amostra OSGi é construída sobre as amostras do serializador do Google Protocol Buffers. Quando esse conjunto de lições for concluído, os plug-ins de amostra do serializador terão sido executados na estrutura OSGi.

Objetivos do aprendizado

Esta amostra demonstra os pacotes configuráveis OSGi. O plug-in serializador é incidental e não é necessário. A amostra de OSGi está disponível na galeria de amostras do WebSphere eXtreme Scale. Você deve fazer o download da amostra e extraí-la no diretório *wxs_home*/samples. O diretório raiz para a amostra OSGi é wxs_home/samples/OSGiProto.

A amostra do serializador do Google Protocol Buffers está localizada no diretório *wxs_home*/samples/SerializerProto.

A amostra do serializador Binary JSON (BSON) está localizada no diretório *wxs_home*/samples/SerializerBSON.

Os exemplos de comando neste tutorial assumem que você está executando no sistema operacional UNIX. Você deve ajustar o exemplo de comando para ser executado em um sistema operacional Windows.

Depois de concluir as lições neste tutorial, você entenderá os conceitos de amostra do OSGi e saberá como concluir os seguintes objetivos:

- Instalar o pacote configurável do servidor WebSphere eXtreme Scale no contêiner OSGi para iniciar o servidor eXtreme Scale.
- Configurar seu ambiente de desenvolvimento do eXtreme Scale para executar o cliente de amostra.
- Usar o comando xscmd para consultar a classificação do serviço do pacote configurável da amostra, fazer upgrade dele para uma nova classificação de serviço e verificar a nova classificação de serviço.

Tempo necessário

Esse módulo demora aproximadamente 60 minutos para ser concluído.

Pré-requisitos

Além de fazer download e extrair as amostras do serializador, este tutorial também possui os pré-requisitos a seguir:

- Instalar e extrair o produto eXtreme Scale
- Configurar o Eclipse Equinox Environment

Introdução: Iniciando e Configurando o Servidor e o Contêiner do eXtreme Scale para Executar Plug-ins na Estrutura do OSGi

Neste tutorial, inicie um servidor eXtreme Scale na estrutura do OSGi, inicie um contêiner do eXtreme Scale e ligue os plug-ins de amostra com o ambiente de tempo de execução do eXtreme Scale.

Objetivos do aprendizado

Depois de concluir as lições neste tutorial, você entenderá os conceitos de amostra do OSGi e saberá como concluir os seguintes objetivos:

- Instalar o pacote configurável do servidor WebSphere eXtreme Scale no contêiner OSGi para iniciar o servidor eXtreme Scale.
- Configurar seu ambiente de desenvolvimento do eXtreme Scale para executar o cliente de amostra.
- Usar o comando xscmd para consultar a classificação do serviço do pacote configurável da amostra, fazer upgrade dele para uma nova classificação de serviço e verificar a nova classificação de serviço.

Tempo necessário

Esse tutorial demora aproximadamente 60 minutos para ser concluído. Se você explorar outros conceitos relacionados a este tutorial, poderá demorar mais tempo para ele ser concluído.

Nível de qualificação

Intermediário.

Público

Desenvolvedores e administradores que desejam construir, instalar e executar os pacotes configuráveis do eXtreme Scale na estrutura do OSGi.

Requisitos do Sistema

- Luminis OSGi Configuration Admin Command Line Client, versão 0.2.5
- Apache Felix File Install, versão 3.0.2
- Quando usar o Eclipse Gemini como o provedor de contêiner blueprint, os seguintes itens são necessários:
  - Eclipse Gemini Blueprint, versão 1.0.0
  - Spring Framework, versão 3.0.5
  - SpringSource AOP Alliance API, versão 1.0.0
  - SpringSource Apache Commons Logging, versão 1.1.1
- Quando usar o Aries Apache como o provedor do Blueprint Container, você deve ter os seguintes requisitos:
  - Apache Aries, captura instantânea mais recente
  - Biblioteca ASM
  - Criação de Log PAX

Pré-requisitos

Para concluir este tutorial, você deve fazer o download da amostra e extraí-la no diretório `wxs_home/samples`. O diretório raiz para a amostra OSGi é `wxs_home/samples/OSGiProto`.

Resultados Esperados:

Ao concluir este tutorial, você terá instalado os pacotes configuráveis de amostra e executado um cliente do eXtreme Scale para inserir dados na grade. Também espera-se que esses pacotes configuráveis de amostra sejam consultados e atualizados usando os recursos dinâmicos fornecidos pelo contêiner OSGi.

Conceitos relacionados

"Visão Geral da Estrutura do OSGi" na página 37
O OSGi define um sistema módulo dinâmico para Java. A plataforma de serviço OSGi possui uma arquitetura em camadas e é projetada para ser executada em vários perfis padrão Java. É possível iniciar servidores e clientes do WebSphere eXtreme Scale em um contêiner OSGi.

Tarefas relacionadas

"Instalando a Estrutura do Eclipse Equinox OSGi com o Eclipse Gemini para Clientes e Servidores" na página 39
Se desejar implementar o WebSphere eXtreme Scale na estrutura do OSGi, você deverá configurar o Ambiente do Eclipse Equinox.

Referências relacionadas

Arquivo de Propriedades do Servidor
O arquivo de propriedades do servidor contém várias propriedades que definem configurações diferentes para o servidor, como configurações de rastreio, criação de log e configuração de segurança. O arquivo de propriedades do servidor é usado pelo serviço de catálogo e pelos servidores de contêiner em servidores independentes e também em servidores hospedados no WebSphere Application Server.

Módulo 1: Preparando para Instalar e Configurar os Pacotes Configuráveis do Servidor eXtreme Scale

Conclua este módulo para explorar os pacotes configuráveis de amostra OSGi e examine os arquivos de configuração que você usa para configurar o servidor eXtreme Scale.

Objetivos do aprendizado

Depois de concluir as lições neste módulo, você entenderá os conceitos e saberá como concluir os seguintes objetivos:

- Localizar e explorar os pacotes configuráveis que estão incluídos na amostra OSGi.
- Examine os arquivos de configuração que são usados para configurar a grade e o servidor do eXtreme Scale.

Lição 1.1: Entendendo os Pacotes Configuráveis OSGi de Amostra

Conclua esta lição para localizar e explorar os pacotes configuráveis que são fornecidos na amostra do OSGi.

Pacotes Configuráveis OSGi de Amostra:

Diferente dos pacotes configuráveis que estão configurados no arquivo `config.ini`, que é mostrado no tópico sobre como configurar o ambiente do Eclipse Equinox, os seguintes pacotes configuráveis adicionais são usados na amostra OSGi:

objectgrid.jar
O pacote configurável do tempo de execução do servidor WebSphere eXtreme Scale. Este pacote configurável está localizado no diretório *wxs_home*`/lib directory`.

com.google.protobuf_2.4.0a.jar
O pacote configurável Google Protocol Buffers, versão 2.4.0a. Este pacote configurável está localizado no diretório *wxs_sample_osgi_root*`/lib`.

ProtoBufSamplePlugins-1.0.0.jar
Versão 1.0.0 do pacote configurável de plug-in do usuário com as implementações de plug-in ObjectGridEventListener e MapSerializerPlugin. Este pacote configurável está localizado no diretório *wxs_sample_osgi_root*`/lib`. Os serviços são configurados com a classificação de serviço 1.

Esta versão usa o XML Blueprint padrão para configurar os serviços de plug-in do eXtreme Scale. A classe de serviço é uma classe implementada pelo usuário na interface do WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory. A classe implementada pelo usuário cria um bean para cada solicitação e funciona de modo semelhante a um bean de protótipo com escopo definido.

ProtoBufSamplePlugins-2.0.0.jar
Versão 2.0.0 do pacote configurável de plug-in do usuário com as implementações de plug-in ObjectGridEventListener e MapSerializerPlugin de amostra. Este pacote configurável está localizado no diretório *wxs_sample_osgi_root*`/lib`. Os serviços são configurados com a classificação de serviço 2.

Esta versão usa o XML Blueprint padrão para configurar os serviços de plug-in do eXtreme Scale. A classe de serviço está usando uma classe integrada do WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, que usa o serviço do BlueprintContainer. Usando a configuração XML blueprint padrão, os beans podem ser configurados como um escopo de protótipo ou escopo singleton. O bean não é configurado como escopo de shard.

ProtoBufSamplePlugins-Gemini-3.0.0.jar
Versão 3.0.0 do pacote configurável de plug-in do usuário com as implementações de plug-in ObjectGridEventListener e MapSerializerPlugin. Este pacote configurável está localizado no diretório *wxs_sample_osgi_root*`/lib`. Os serviços são configurados com o serviço de classificação 3.

Esta versão usa o XML blueprint específico do Eclipse Gemini para configurar os serviços de plug-in do eXtreme Scale. A classe de serviço está usando uma classe integrada do WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, que usa o serviço do BlueprintContainer. A maneira de configurar um bean do escopo de shard é usar uma abordagem específica do Gemini. Esta versão configura o bean myShardListener como um bean de escopo de shard ao fornecer `{http://www.ibm.com/schema/objectgrid}shard` como o valor do escopo e configurar um atributo fictício para que o escopo customizado seja reconhecido pelo Gemini. Isso ocorre devido ao seguinte problema do Eclipse: https://bugs.eclipse.org/bugs/show_bug.cgi?id=348776

ProtoBufSamplePlugins-Aries-4.0.0.jar
Versão 4.0.0 do pacote configurável de plug-in do usuário com as implementações de plug-in ObjectGridEventListener e MapSerializerPlugin de amostra. Este pacote configurável está localizado no diretório *wxs_sample_osgi_root*/lib. Os serviços são configurados com a classificação de serviço 4.

Esta versão usa o XML blueprint padrão para configurar os serviços de plug-in do eXtreme Scale. A classe de serviço está usando uma classe integrada do WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactoryImpl, que usa o serviço do BlueprintContainer. Ao usar a configuração XML blueprint padrão, os beans podem ser configurados usando um escopo customizado. Esta versão configura o myShardListenerbean como bean de shard com escopo definido ao fornecer {http://www.ibm.com/schema/objectgrid}shard como o valor do escopo.

ProtoBufSamplePlugins-Activator-5.0.0.jar
Versão 5.0.0 do pacote configurável de plug-in do usuário com as implementações de plug-in ObjectGridEventListener e MapSerializerPlugin de amostra. Este pacote configurável está localizado no diretório *wxs_sample_osgi_root*/lib. Os serviços são configurados com o serviço de classificação 5.

Esta versão não é usada em todo o contêiner blueprint. Nesta versão, os serviços são registrados usando o registro de serviço OSGi. A classe de serviço é uma classe implementada pelo usuário para a interface do WebSphere eXtreme Scale, com.ibm.websphere.objectgrid.plugins.osgi.PluginServiceFactory. A classe implementada pelo usuário cria um bean para cada solicitação. Ela funciona de modo semelhante a um bean de protótipo com escopo definido.

Ponto de verificação de lições:

Ao explorar os pacotes configuráveis que são fornecidos com a amostra OSGi, é possível entender melhor como desenvolver suas próprias implementações que serão executadas no contêiner OSGi.

Você aprendeu:

- Sobre os pacotes configuráveis que estão incluídos com a amostra OSGi
- O local desses pacotes configuráveis
- A classificação de serviço com a qual cada pacote configurável foi configurado

Lição 1.2: Entender os Arquivos de Configuração do OSGi

A amostra OSGi inclui três arquivos de configuração. Esses arquivos podem ser usados para iniciar e configurar a grade e o servidor do WebSphere eXtreme Scale.

Arquivos de Configuração OSGi:

Nesta lição, os seguintes arquivos de configuração serão explorados:

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

collocated.server.properties

Uma configuração do servidor é necessária para iniciar um servidor. Quando o pacote configurável do servidor eXtreme Scale é iniciado, ele não inicia um servidor. Ele aguarda o PID de configuração, `com.ibm.websphere.xs.server`, ser criado com um arquivo de propriedades do servidor. Esse arquivo de propriedades do servidor especifica o nome do servidor, o número da porta e outras propriedades do servidor.

Na maioria dos casos, uma configuração é criada para configurar o arquivo de propriedades do servidor. Raramente você pode querer iniciar um servidor apenas com cada propriedade configurada para um valor padrão. Nesse caso, é possível criar uma configuração denominada `com.ibm.websphere.xs.server` com o valor configurado para `padrão`.

Para obter mais detalhes sobre o arquivo de propriedades de servidor, consulte o tópico Arquivo de Propriedades do Servidor.

A amostra OSGi inclui o arquivo de propriedades do servidor de amostra, *wxs_sample_osgi_root*`/server/properties/collocated.server.properties`. Esse arquivo de propriedades de amostra inicia um serviço de catálogo e um servidor de contêiner únicos no processo da estrutura do OSGi. Os clientes do eXtreme Scale se conectam à porta 2809 e os clientes do JMX se conectam à porta 1099. O conteúdo do arquivo de propriedades do servidor de amostra é:

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

protoBufObjectGrid.xml

O arquivo XML do descritor do ObjectGrid `protoBufObjectGrid.xml` de amostra contém o conteúdo a seguir, com comentários removidos.

```
<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>
```

Há dois plug-ins configurados neste arquivo XML do descritor do ObjectGrid:

ObjectGridEventListener

O plug-in de nível de shard. Para cada instância do ObjectGrid, há uma instância de ObjectGridEventListener. Ela é configurada para usar o myShardListener de serviço do OSGi. Isso significa que quando a grade é criada, o plug-in ObjectGridEventListener usa o serviço OSGi myShardListener com a classificação deserviço mais alta disponível.

MapSerializerPlugin

O plug-in de nível de mapa. Para o mapa de apoio denominado `Map`, há um plug-in MapSerializerPlugin configurado. Ele é configurado para usar o serviço OSGimyProtoBufSerializer. Isso significa que quando o mapa é criado, o plug-in MapSerializerPlugin usa o serviço myProtoBufSerializer com a maior classificação de serviço obtida disponível.

protoBufDeployment.xml

O arquivo XML do descritor de implementação descreve a política de implementação para a grade denominada `Grid`, que usa cinco partições. Consulte o seguinte código de exemplo desse arquivo 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 para o uso do arquivo `collocated.server.properties` em conjunto com PID de configuração, `com.ibm.websphere.xs.server`, o XML do ObjectGrid e os arquivos XML de implementação podem ser incluídos em um pacote configurável OSGi, junto com um arquivo XML blueprint conforme mostrado no exemplo a seguir :

```
<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>
```

Ponto de verificação de lições:

Nesta lição, você aprendeu sobre os arquivos de configuração que são usados na amostra OSGi. Agora, quando iniciar e configurar a grade e o servidor eXtreme Scale, você entenderá quais arquivos estão sendo usados nestes processos e como esses arquivos interagem com seus plug-ins na estrutura do OSGi.

Módulo 2: Instalando e Iniciando Pacotes Configuráveis do eXtreme Scale na Estrutura do OSGi

Utilize os módulos nessas lições para instalar o pacote configurável do servidor eXtreme Scale no contêiner OSGi e para iniciar o servidor WebSphere eXtreme Scale.

Iniciar o servidor na estrutura do OSGi não significa que seus pacotes configuráveis OSGi estão prontos para execução. Você deve configurar as propriedades do servidor e os contêineres para que os pacotes configuráveis OSGi que forem instalados sejam reconhecidos e executados corretamente.

Objetivos do aprendizado

Depois de concluir as lições neste módulo, você entenderá os conceitos e saberá como concluir as seguintes tarefas:

- Instalar os pacotes configuráveis do eXtreme Scale usando o console do Equinox OSGi.
- Configurar o servidor eXtreme Scale.
- Configure o contêiner do eXtreme Scale.
- Iniciar pacotes configuráveis de amostra do eXtreme Scale.

Pré-requisitos

Para concluir este módulo, as tarefas a seguir são necessárias antes de iniciar:

- Instalar e extrair o produto eXtreme Scale
- Configurar o Eclipse Equinox Environment

Você também deve preparar para acessar os seguintes arquivos para concluir as lições neste módulo:

- Pacote Configurável `objectgrid.jar`. Instale esse pacote configurável do eXtreme Scale.
- Arquivo `collocated.server.properties`. Inclua as propriedades do servidor nesse arquivo de configuração.
- Espera-se que os seguintes pacotes configuráveis sejam instalados e iniciados:
- Pacote configurável `protobuf-java-2.4.0a-bundle.jar`
- Pacote configurável `ProtoBufSamplePlugins-1.0.0.jar`
- Pacote configurável `ProtoBufSamplePlugins-2.0.0.jar`

Lição 2.1: Iniciar o Console e Instalar o Pacote Configurável do Servidor eXtreme Scale

Nesta lição, use o console do Equinox OSGi para iniciar e instalar um WebSphere eXtreme Scale.

1. Execute o seguinte comando para iniciar o console Equinox OSGi:

```
cd equinox_root

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

2. Após iniciar o console OSGI, emita o comando `ss` no console e os seguintes pacotes configuráveis serão iniciados:

Saída do 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
```

Saída do 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 o pacote configurável `objectgrid.jar`. Para iniciar um servidor na Java virtual machine (JVM), é necessário instalar um pacote configurável do servidor eXtreme Scale. Este pacote configurável do servidor eXtreme Scale pode iniciar um servidor e criar contêineres. Use o comando a seguir para instalar o arquivo `objectgrid.jar`:

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

   Consulte o seguinte exemplo:

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

   O Equinox exibe o ID do pacote configurável, por exemplo:

   ```
   Bundle id is 19
   ```

   Lembre-se: Seu ID do pacote configurável pode ser diferente. O caminho do arquivo deve ser uma URL absoluta para o caminho do pacote configurável. Caminhos relativos não são suportados.

Ponto de verificação de lições:

Nesta lição, você usou o console do Equinox OSGi para instalar o pacote configurável `objectgrid.jar`, o qual você usará para iniciar um servidor e criar um contêiner posteriormente neste tutorial.

Lição 2.2: Customizar e Configurar o Servidor eXtreme Scale

Use essa lição para customizar e incluir as propriedades do servidor para o servidor WebSphere eXtreme Scale.

1. Edite o arquivo `wxs_sample_osgi_root/server/properties/collocated.server.properties`.
   a. Altere a propriedade do workingDirectory para `equinox_root`.
   b. Altere a propriedade do traceFile para `equinox_roo/logs/trace.log`.
2. Salve o arquivo.
3. Insira as seguintes linhas de código no console OSGI para criar a configuração do servidor a partir do arquivo:

```
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 a configuração, execute o seguinte comando:

```
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
```

Ponto de verificação de lições:

Nesta lição, você editou o arquivo `wxs_sample_osgi_root/server/properties/collocated.server.properties` para especificar as configurações do servidor, como o diretório de trabalho e o local para os arquivos de log de rastreio.

Lição 2.3: Configurar o Contêiner do eXtreme Scale

Conclua esta lição para configurar um contêiner, que inclui o arquivo descritor XML e o arquivo XML de implementação do ObjectGrid do WebSphere eXtreme Scale. Esses arquivos incluem a configuração para a grade e sua topologia.

Para criar um contêiner, primeiro crie um serviço de configuração usando o factory de serviço gerenciado pelo número de identificação do processo (PID), com.ibm.websphere.xs.container. A configuração do serviço é um factory de serviço gerenciado para que seja possível criar vários PIDs de serviço a partir do PID do factory. Em seguida, para iniciar o serviço do contêiner, configure os PIDs objectgridFile e deploymentPolicyFile para cada PID de serviço.

Conclua as seguintes etapas para customizar e incluir as propriedades do servidor para a estrutura do OSGi:

1. No console OSGI, insira o seguinte comando para criar o contêiner a partir do arquivo:

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

2. Insira o seguinte comando para ligar o PID recém-criado aos arquivos XML do ObjectGrid.

   Lembre-se: O número de PID será diferente do número que está incluído neste exemplo.

```
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. Use o seguinte comando para exibir a configuração:

```
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
```

Ponto de verificação de lições:

Nesta lição, você criou um serviço de configuração, que foi usado para criar um contêiner do eXtreme Scale. Como os arquivos XML do ObjectGrid contêm a configuração para a grade e sua topologia, foi necessário vincular o contêiner criado para esses arquivos XML do ObjectGrid. Com essa configuração, o contêiner do eXtreme Scale pode reconhecer os pacotes configuráveis do OSGi que serão executados posteriormente neste tutorial.

Lição 2.4: Instalar o Google Protocol Buffers e os Pacotes Configuráveis do Plug-in de Amostra

Conclua este tutorial para instalar o pacote configurável `protobuf-java-2.4.0a-bundle.jar` e o pacote configurável de plug-in `ProtoBufSamplePlugins-1.0.0.jar` usando o console do Equinox OSGi.

Conclua as seguintes etapas para instalar o pacote configurável Google Protocol Buffers.

No console OSGI, insira o seguinte comando para instalar o pacote configurável:

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

A saída a seguir é exibida:

```
Bundle ID is 21
```

Visão Geral dos Pacotes Configuráveis do Plug-in de Amostra:

A amostra do OSGi inclui cinco pacotes configuráveis que incluem os plug-ins do eXtreme Scale, dentre eles um plug-in ObjectGridEventListener e MapSerializerPlugin customizado. O plug-in MapSerializerPlugin usa a amostra do Google Protocol Buffers e as mensagens fornecidas pela amostra do MapSerializerPlugin.

Os seguintes pacotes configuráveis estão localizados no diretório *wxs_sample_osgi_root*/lib: `ProtoBufSamplePlugins-1.0.0.jar` e o `ProtoBufSamplePlugins-2.0.0.jar`.

O arquivo `blueprint.xml` possui o seguinte conteúdo com comentários removidos:

```
<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>
```

O arquivo XML Blueprint exporta dois serviços, myShardListener e myProtoBufSerializer. Esses dois serviços são referenciados no arquivo `protoBufObjectgrid.xml`.

Instalar o Pacote Configurável do Plug-in de Amostra:

Conclua as seguintes etapas para instalar o pacote configurável `ProtoBufSamplePlugins-1.0.0.jar`.

Execute o seguinte comando no console do Equinox OSGi para instalar o pacote configurável do plug-in `ProtoBufSamplePlugins-1.0.0.jar`:

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

A saída a seguir é exibida:

```
Bundle ID is 22
```

Ponto de verificação de lições:

Nesta lição, você instalou o pacote configurável `protobuf-java-2.4.0a-bundle.jar` e o pacote configurável de plug-in `ProtoBufSamplePlugins-1.0.0.jar`.

Lição 2.5: Iniciar os Pacotes Configuráveis do OSGi

O servidor WebSphere eXtreme Scale é incluído em um pacote configurável do OSGi. Conclua esta lição para instalar o pacote configurável do servidor eXtreme Scale e também outros pacotes configuráveis OSGi que forem instalados.

1. Inicie o pacote configurável do plug-in de amostra. Execute o seguinte comando no console do Equinox OSGi para iniciar o pacote configurável. Neste exemplo, o ID do pacote configurável do plug-in de amostra é 22.

   ```
   osgi> start 22
   ```

2. Inicie o pacote configurável do Google Protocol Buffers. Execute o seguinte comando no console do Equinox OSGi para iniciar o pacote configurável. Neste exemplo, o ID do pacote configurável do plug-in Google Protocol Buffers é 21.

   ```
   osgi> start 21
   ```

3. Inicie o pacote configurável do servidor. Execute o seguinte comando no console OSGi para iniciar o servidor. Neste exemplo, o ID do pacote configurável do servidor eXtreme Scale é 19.

   ```
   osgi> start 19
   ```

Depois de iniciar o servidor, o listener de event MyShardListener é iniciado e pronto para inserir ou atualizar registros. A seguinte saída pode ser visualizada no console OSGI para confirmar que o pacote configurável do plug-in foi iniciado com êxito:

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

Ponto de verificação de lições:

Nesta lição, você iniciou dois pacotes configuráveis de plug-in e o pacote configurável do servidor no contêiner do eXtreme Scale configurado para a estrutura do OSGi.

Módulo 3: Executando o Cliente de Amostra do eXtreme Scale

O servidor WebSphere eXtreme Scale agora está em execução em um ambiente do OSGi. Conclua as etapas neste módulo para executar um cliente do WebSphere eXtreme Scale que insere dados na grade.

Objetivos do aprendizado

Depois de concluir as lições neste módulo, você saberá como concluir as seguintes tarefas:

- Executar um aplicativo cliente que se conecta à grade e insere e recupera dados a partir dele.
- Inicie uma ordem usando um aplicativo cliente não OSGi.

Pré-requisitos

Conclua Módulo 2: Instalando e Iniciando Pacotes Configuráveis do eXtreme Scale na Estrutura OSGi.

Lição 3.1: Configurar o Eclipse para Executar o Cliente e Construir as Amostras

Conclua esta lição para importar o projeto Eclipse que será usado para executar o cliente e construir os plug-ins de amostra.

A amostra inclui um programa cliente Java SE que se conecta à grade e insere e recupera dados a partir dele. Ele também inclui projetos que podem ser usados para construir e reimplementar os pacotes configuráveis do OSGi.

O projeto fornecido foi testado com o Eclipse 3.x e posterior e requer somente a perspectiva de projetos de desenvolvimento Java padrão. Conclua as etapas a seguir para configurar o seu ambiente de desenvolvimento do WebSphere eXtreme Scale.

1. Abra o Eclipse para uma área de trabalho nova ou existente.
2. No menu Arquivo, selecione **Importar**.
3. Expanda a pasta Geral. Selecione **Pacotes Existentes na Área de Trabalho** e clique em **Avançar**.
4. No campo **Selecionar Diretório-Raiz**, digite ou navegue até o diretório *wxs_sample_osgi_root*. Clique em **Concluir**. Vários novos projetos são exibidos na sua área de trabalho. Você deve corrigir diversos erros de construção ao definir a biblioteca de usuário do eXtreme Scale. Conclua as próximas etapas para definir a biblioteca de usuário.
5. No menu Janela, selecione **Preferências**.
6. Expanda a ramificação **Java** > **Caminho de Construção** e selecione **Bibliotecas de Usuário**.
7. Clique em **Novo**.
8. Digite `eXtremeScale` no campo **Nome da Biblioteca de Usuário** e clique em **OK**.
9. Selecione a nova biblioteca de usuário e clique em **Incluir JARs**.
   a. Procure por e selecione o arquivo `objectgrid.jar` a partir do diretório *wxs_install_root*`/lib`. Clique em **OK**.

b. Para incluir a documentação da API para as APIs do ObjectGrid, selecione o local da documentação da API para o arquivo `objectgrid.jar` que você incluiu na etapa anterior. Clique em **Editar**.
c. Na caixa do caminho do local para a documentação da API, selecione o arquivo `Javadoc.zip` que está incluído no seguinte diretório: *wxs_install_root*`/docs/javadoc.zip`.

Ponto de verificação de lições:

Nesta lição, você importou o projeto Eclipse de amostra, definiu a biblioteca de usuário do eXtreme Scale e incluiu uma documentação da API de suporte no projeto de amostra. Agora o aplicativo cliente de amostra está pronto para ser iniciado.

Lição 3.2: Iniciar um Cliente e Inserir Dados na Grade

Conclua esta lição para iniciar um cliente não OSGi e executar um aplicativo cliente.

O aplicativo de cliente Java é `com.ibm.websphere.samples.xs.proto.client.Client`.

Este cliente usa uma substituição do cliente, o arquivo descritor XML do ObjectGrid para substituir a configuração OSGi, para que o cliente possa ser executado em um ambiente não OSGi. Consulte o conteúdo a seguir do arquivo com comentários e cabeçalhos removidos. Algumas linhas de código são exibidas em diversas linhas para propósitos de formatação.

```
<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>
```

Conclua as seguintes etapas para iniciar o aplicativo cliente.

1. Use o exemplo de código a seguir para modificar os atributos da classe Cliente para refletir seu ambiente.

   ```
   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. Execute o aplicativo cliente.

Quando você executa o aplicativo, a seguinte mensagem é exibida. A mensagem indica que uma solicitação foi inserida:

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

Ponto de verificação de lições:

Nesta lição, você iniciou o aplicativo `com.ibm.websphere.samples.xs.proto.client.Client`, que produziu uma solicitação.

Módulo 4: Consultando e Fazendo Upgrade do Pacote Configurável de Amostra

Conclua as lições neste módulo para usar o comando **xscmd** para consultar a classificação de serviço do pacote configurável da amostra, fazer upgrade dela para uma nova classificação de serviço e verificar o novo serviço de classificação.

Um projeto eclipse é fornecido como uma maneira conveniente de executar os aplicativos de amostra.

Objetivos do aprendizado

Depois de concluir as lições deste módulo, você saberá como concluir as tarefas:

- Consultar a classificação de serviço atual para um serviço.
- Consultar a classificação atual de todos os serviços.
- Consultar todas as classificações disponíveis para um serviço.
- Consultar todas as classificações de serviço disponíveis.
- Use a ferramenta xscmd para verificar se as classificações específicas do serviço estão disponíveis.
- Atualizar as classificações de serviço dos serviços OSGi de amostra.

Pré-requisitos

Executar o Módulo 3: Executando o Cliente de Amostra do eXtreme Scale.

Lição 4.1: Consultar Classificações de Serviço

Conclua esta lição para consultar as classificações de serviço atuais, bem como as classificações de serviço que estão disponíveis para upgrade.

- Consultar a classificação de serviço atual para um serviço. Insira o seguinte comando para consultar a classificação de serviço atual usada para serviço, myShardListener, que é usado pelo ObjectGrid denominado `Grid` e pelo conjunto de mapas denominado `MapSet`.
  1. Alterne para o diretório a seguir:

     ```
     cd wxs_home/bin
     ```

  2. Insira o seguinte comando para consultar a classificação de serviço atual para o serviço myShardListener.

     ```
     ./xscmd.sh -c osgiCurrent -g Grid -ms MapSet -sn myShardListener
     ```

     A saída a seguir é exibida:

     ```
     OSGi Service Name: myShardListener
     ObjectGrid Name MapSet Name Server Name        Current Ranking
     --------------- ----------- -----------        ---------------
     ```

```
Grid            MapSet      collocatedServer   1

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

- Consultar a classificação atual de todos os serviços. Insira o seguinte comando para consultar as classificações de serviço atuais para todos os serviços usados pelo ObjectGrid denominado `Grid` e pelo conjunto de mapas denominado `MapSet`:
  1. Alterne para o diretório a seguir:
     ```
     cd wxs_home/bin
     ```
  2. Insira o seguinte comando para consultar a classificação de serviço atual para todos os serviços.
     ```
     ./xscmd.sh -c osgiCurrent -g Grid -ms MapSet
     ```

     A saída a seguir é exibida:

```
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.
```

- Consultar todas as classificações disponíveis para um serviço. Insira o seguinte comando para consultar todas as classificações de serviço disponíveis para o serviço denominado myShardListener.
  1. Alterne para o diretório a seguir:
     ```
     cd wxs_home/bin
     ```
  2. Insira o seguinte comando para consultar todas as classificações disponíveis para um serviço.
     ```
     ./xscmd.sh -c osgiAll -sn myShardListener
     ```

     A saída a seguir é exibida:

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

        Summary - All servers have the same service rankings.

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

     A saída é agrupada pelo servidor. Neste exemplo, apenas o seguinte servidor existe: collocatedServer.
- Consultar todas as classificações de serviço disponíveis. Insira o comando a seguir para consultar todas as classificações de serviço disponíveis para todos os serviços.
  1. Alterne para o diretório a seguir:
     ```
     cd wxs_home/bin
     ```
  2. Insira o seguinte comando para consultar todas as classificações de serviço disponíveis.
     ```
     ./xscmd.sh -c osgiAll
     ```

     A saída a seguir é exibida:

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

```
myShardListener 1

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

- Instale e inicie a Versão 2 do pacote configurável do plug-in. No console OSGi servidor, instale um novo pacote configurável que contém uma nova versão da classe Order e o plug-in MapSerializerPlugin. Consulte Lição 2.4: Instalar o Google Buffers Protocol e os pacotes configuráveis do plug-in de amostra para obter detalhes sobre como instalar o pacote configurável `ProtoBufSamplePlugins-2.0.0.jar`.
  1. Após a instalação, inicie o novo pacote configurável. Os serviços para seu novo pacote configurável estão disponíveis, mas eles ainda não são usados pelo servidor eXtreme Scale. Você deve executar uma solicitação de atualização de serviço para usar um serviço com uma versão específica.
- Agora, quando você consultar todas as classificações de serviço disponíveis novamente, o serviço de classificação 2 é incluído na saída.
  1. Alterne para o diretório a seguir:
     ```
     cd wxs_home/bin
     ```
  2. Insira o seguinte comando para consultar todas as classificações de serviço disponíveis.
     ```
     ./xscmd.sh -c osgiAll
     ```

     A saída a seguir é exibida:
     ```
     Server: collocatedServer
        OSGi Service Name    Available Rankings
        -----------------    ------------------
        myProtoBufSerializer 1, 2
        myShardListener      1, 2

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

Ponto de verificação de lições:

Neste tutorial, você consultou atualmente todas as classificações de serviço, e classificações de serviço especificadas, disponíveis. Você também exibiu a classificação de serviço para um novo pacote configurável que você instalou e iniciou.

Lição 4.2: Determinar Se Classificações de um Serviço Específico Estão Disponíveis

Conclua esta lição para determinar se as classificações de um serviço específico estão disponíveis para os nomes de serviço que forem especificados.

1. Insira o seguinte comando para determinar se o serviço myShardListener, com a classificação de serviço 2, e o serviço myProtoBufSerializer, com a classificação de serviço 2, estão disponíveis. A lista de classificação do serviço é passada usando a opção `-sr`.
   a. Alterne para o diretório a seguir:
      ```
      cd wxs_home/bin
      ```
   b. Insira o seguinte comando para determinar se os serviços estão disponíveis:
      ```
      ./xscmd.sh -c osgiCheck -g Grid -ms MapSet -sr
      "myShardListener;2,myProtoBufSerializer;2"
      ```

      A saída a seguir é exibida:
      ```
      CWXSI0040I: The command osgiCheck has completed successfully.
      ```

2. Insira o seguinte comando para determinar se o serviço myShardListener, com a classificação de serviço 2, e o serviço myProtoBufSerializer, com a classificação de serviço 3 estão disponíveis.
   a. Alterne para o diretório a seguir:
      ```
      cd wxs_home/bin
      ```
   b. Insira o seguinte comando para determinar se os serviços estão disponíveis:
      ```
      ./xsadmin.sh -c osgiCheck -g Grid -ms MapSet -sr
      "myShardListener;2,myProtoBufSerializer;3"
      ```

      A saída a seguir é exibida:
      ```
      Server  OSGi Service Unavailable Rankings
      ------ ------------ --------------------
      collocatedServer myProtoBufSerializer 3
      ```

Ponto de verificação de lições:

Nesta lição, você especificou os serviços myShardListener e myProtoBufSerializer, junto com classificações de serviço específicas para determinar se essas classificações estavam disponíveis.

Lição 4.3: Atualizar as Classificações do Serviço

Conclua esta lição para atualizar as classificações do serviço atual que você consultou.

1. Insira o comando a seguir atualiza as classificações dos serviços denominados myShardListener e myProtoBufSerializer o serviço de classificação 2. A lista de classificação do serviço é passada usando a opção `-sr`.
   a. Alterne para o diretório a seguir:
      ```
      cd wxs_home/bin
      ```
   b. Insira o seguinte comando para atualizar as classificações de serviço:
      ```
      ./xscmd.sh -c osgiUpdate -g Grid -ms MapSet
      -sr "myShardListener;2,myProtoBufSerializer;2"
      ```

      A saída a seguir é exibida:
      ```
      Update succeeded for the following service rankings:
      Service Ranking
      ------- -------
      myProtoBufSerializer 2
      myShardListener 2

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

      A seguinte saída é exibida no console do OSGi:
      ```
      SystemOut O MyShardListener@326505334(version=2.0.0) order
      com.ibm.websphere.samples.xs.serializer.proto.DataObjects2$Order$Builder@
      22342234(34) updated
      ```

      Observe que o serviço MyShardListener está agora na versão 2.0.0, que possui um serviço de classificação 2.
2. Se você executar o comando **xscmd** para consultar a classificação de serviço atual que está sendo usada para todos os serviços usados pelo ObjectGrid denominado `Grid` e o conjunto de mapa denominado `MapSet`.
   a. Alterne para o diretório a seguir:
      ```
      cd wxs_home/bin
      ```
   b. Insira o seguinte comando para consultar as classificações de serviço para todos os serviços usados por `Grid` e `MapSet`:

```
./xscmd.sh -c osgiCurrent -g Grid -ms MapSet
```

A saída a seguir é exibida:

```
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.
```

Ponto de verificação de lições:

Nesta lição você atualizou as classificações de serviço para os serviços myShardListener e myProtoBufSerializer.

.

Capítulo 2. Cenários

Um cenário apresenta exemplos para ajudar o usuário a entender um conceito ou realizar uma tarefa. O cenário usa as informações do mundo real para construir uma imagem completa.

Usando um Ambiente OSGi para Desenvolver e Executar Plug-ins do eXtreme Scale

Use estes cenários para concluir tarefas comuns em um ambiente OSGi. Por exemplo, a estrutura do OSGi é ideal para iniciar os servidores e clientes em um contêiner OSGi, que permite incluir e atualizar dinamicamente plug-ins do WebSphere eXtreme Scale para o ambiente de tempo de execução.

Os cenários a seguir são sobre construir e executar dinamicamente os plug-ins, que permite instalar, iniciar, parar, modificar e desinstalar os plug-ins de modo dinâmico. Você também pode concluir um outro cenário provável, que permite usar a estrutura do OSGi sem os recursos dinâmicos. Também é possível incluir os aplicativos em pacotes configuráveis, que são definidos e comunicados por meio de serviços. Esses pacotes configuráveis baseados em serviço oferecem vários benefícios, incluindo capacidades de desenvolvimento e de implementação mais eficientes.

Objetivos do Cenário

Depois de concluir as lições deste módulo, você saberá como concluir as tarefas:

- Construa plug-ins dinâmicos do eXtreme Scale para uso em um ambiente OSGi.
- Execute os contêineres do eXtreme Scale em um ambiente OSGi sem os recursos dinâmicos.

Pré-requisitos

Leia o tópico “Visão Geral da Estrutura do OSGi” para aprender mais sobre o suporte de OSGi e os benefícios que ele pode oferecer.

Visão Geral da Estrutura do OSGi

O OSGi define um sistema módulo dinâmico para Java. A plataforma de serviço OSGi possui uma arquitetura em camadas e é projetada para ser executada em vários perfis padrão Java. É possível iniciar servidores e clientes do WebSphere eXtreme Scale em um contêiner OSGi.

Benefícios de Executar Aplicativos no Contêiner OSGi

O suporte do WebSphere eXtreme Scale OSGi permite implementar o produto na estrutura do Eclipse Equinox OSGi. Anteriormente, se você desejava atualizar os plug-ins usados pelo eXtreme Scale, era necessário reiniciar a Java Virtual Machine (JVM) para aplicar as novas versões dos plug-ins. Com a capacidade de atualização dinâmica que a estrutura OSGi fornece, agora é possível atualizar as classes de plug-in sem reiniciar a JVM. Esses plug-ins são exportados pelos pacotes configuráveis do usuário como serviços. O WebSphere eXtreme Scale acessa o serviço ou serviços consultando-os no registro OSGi.

Os contêineres do eXtreme Scale podem ser configurados para iniciar mais fácil e dinamicamente usando o serviço administrativo de configuração do OSGi ou com o OSGi Blueprint. Se desejar implementar uma nova grade de dados com sua estratégia de posicionamento, será possível fazer isso criando uma configuração de OSGi ou implementando um pacote configurável com arquivos XML do descritor eXtreme Scale. Com o suporte do OSGi, os pacotes configuráveis de aplicativo que contém dados de configuração do eXtreme Scale podem ser instalados, iniciados, interrompidos, atualizados e desinstalados sem reiniciar o sistema inteiro. Com esta capacidade, é possível fazer upgrade do aplicativo sem interromper a grade de dados.

Beans de plug-in e serviços podem ser configurados com escopos de shard customizados, permitindo opções de integração sofisticadas com outros serviços em execução na grade de dados. Cada plug-in pode usar classificações do OSGi Blueprint para verificar se cada instância do plug-in ativada está na versão correta. Um bean gerenciado por OSGi (MBean) e o utilitário **xscmd** são fornecidos, o que permite que você consulte os serviços OSGi de plug-in do eXtreme Scale e suas classificações.

Este recurso permite que os administradores reconheçam rapidamente erros de configuração e administração em potencial e atualize as classificações de serviço de plug-in em uso pelo eXtreme Scale.

Pacotes Configuráveis OSGi

Para interagir com, e implementar, plug-ins na estrutura do OSGi, você deve usar os *pacotes configuráveis*. Na plataforma de serviço OSGi, um pacote configurável é um arquivo Java archive (JAR) que contém código Java, recursos e um manifesto que descrevem o pacote configurável e suas dependências. O pacote configurável é a unidade de implementação para um aplicativo. O produto eXtreme Scale suporta os seguintes tipos de pacotes configuráveis:

Pacote configurável do servidor
O pacote configurável do servidor é o arquivo `objectgrid.jar` e é instalado com a instalação do servidor independente do eXtreme Scale e é necessário para executar servidores do eXtreme Scale e também pode ser usado para executar clientes do eXtreme Scale ou caches na memória locais. O ID do pacote configurável para o arquivo `objectgrid.jar` é com.ibm.websphere.xs.server_`<version>`, em que a versão está no formato: `<Version>.<Release>.<Modification>`. Por exemplo, o pacote configurável do servidor para o eXtreme Scale versão 7.1.1 é com.ibm.websphere.xs.server_7.1.1.

Pacote configurável do cliente
O pacote configurável do cliente é o arquivo `ogclient.jar` e é instalado com instalações independentes e do cliente do eXtreme Scale e é usado para executar os clientes do eXtreme Scale ou caches na memória locais. O ID do pacote configurável para o arquivo `ogclient.jar` é com.ibm.websphere.xs.client_`version`, em que a versão está no formato: `<Version>.<Release>.<Modification>`. Por exemplo, o pacote configurável do cliente para o eXtreme Scale versão 7.1.1 é com.ibm.websphere.xs.client_7.1.1.

Limitações

Não é possível reiniciar o pacote configurável do eXtreme Scale porque você não pode reiniciar o object request broker (ORB). Para reiniciar o servidor do eXtreme

Scale, você deve reiniciar a estrutura do OSGi.

Tarefas relacionadas

"Instalando a Estrutura do Eclipse Equinox OSGi com o Eclipse Gemini para Clientes e Servidores"
Se desejar implementar o WebSphere eXtreme Scale na estrutura do OSGi, você deverá configurar o Ambiente do Eclipse Equinox.

"Gerenciando Ciclos de Vida de Plug-in" na página 299
É possível gerenciar ciclos de vida de plug-in com métodos especializados de cada plug-in, que estão disponíveis para serem chamados em pontos funcionais designados. Ambos os métodos initialize e destroy definem o ciclo de vida de plug-ins, que são controlados pelos seus objetos *proprietário*. Um objeto proprietário é o objeto que realmente usa o plug-in fornecido. Um proprietário pode ser um cliente de grade, um servidor ou um mapa de apoio.

Referências relacionadas

Arquivo de Propriedades do Servidor
O arquivo de propriedades do servidor contém várias propriedades que definem configurações diferentes para o servidor, como configurações de rastreio, criação de log e configuração de segurança. O arquivo de propriedades do servidor é usado pelo serviço de catálogo e pelos servidores de contêiner em servidores independentes e também em servidores hospedados no WebSphere Application Server.

Informações relacionadas

"Introdução: Iniciando e Configurando o Servidor e o Contêiner do eXtreme Scale para Executar Plug-ins na Estrutura do OSGi" na página 19
Neste tutorial, inicie um servidor eXtreme Scale na estrutura do OSGi, inicie um contêiner do eXtreme Scale e ligue os plug-ins de amostra com o ambiente de tempo de execução do eXtreme Scale.

Documentação da API

Instalando a Estrutura do Eclipse Equinox OSGi com o Eclipse Gemini para Clientes e Servidores

Se desejar implementar o WebSphere eXtreme Scale na estrutura do OSGi, você deverá configurar o Ambiente do Eclipse Equinox.

Sobre Esta Tarefa

A tarefa requer fazer o download e instalar a estrutura do blueprint, que permite configurar posteriormente o JavaBeans e expô-los como serviços. O uso de serviços é importante porque é possível expor plug-ins como serviços OSGi, de modo que eles possam ser usados pelo ambiente de tempo de execução do eXtreme Scale. O produto suporta dois contêineres blueprint dentro da estrutura OSGi principal do Eclipse Equinox: Eclipse Gemini e Aries Apache. Use este procedimento para configurar o contêiner do Eclipse Gemini.

Procedimento

1. Faça download do Eclipse Equinox SDK Versão 3.6.1 ou posterior a partir do website do Eclipse. Crie um diretório para a estrutura do Equinox, por exemplo: `/opt/equinox`. Essas instruções referenciam esse diretório como `equinox_root`. Extraia o arquivo compactado no diretório `equinox_root`.
2. Faça download do arquivo compactado gemini-blueprint incubation 1.0.0 a partir do Website Eclipse. Extraia o conteúdo do arquivo em um diretório temporário e copie os seguintes arquivos extraídos para o diretório `equinox_root/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. Faça download do Spring Framework Versão 3.0.5 a partir da página da web SpringSource a seguir: http://www.springsource.com/download/community. Extraia-o em um diretório temporário e copie os seguintes arquivos extraídos para o diretório `equinox_root/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. Faça download do arquivo Java archive (JAR) AOP Alliance a partir da página da web do SpringSource. Copie o arquivo com.springsource.org.aopalliance-1.0.0.jar para o diretório `equinox_root/plugins` .
5. Faça download do arquivo JAR Apache Commons Logging 1.1.1 a partir da página da web do SpringSource. Copie o arquivo `com.springsource.org.apache.commons.logging-1.1.1.jar` para o diretório `equinox_root/plugins`.
6. Faça download do cliente de linha de comandos Luminis OSGi Configuration Admin. Use esse pacote configurável para gerenciar as configurações administrativas do OSGi. É possível fazer o download do arquivo JAR a partir da seguinte página da web: https://opensource.luminis.net/wiki/display/SITE/OSGi+Configuration+Admin+command+line+client. Copie o arquivo `net.luminis.cmc-0.2.5.jar` para o diretório `equinox_root/plugins`.
7. Faça download do pacote configurável do arquivo de instalação Apache Felix Versão 3.0.2 a partir da seguinte página da web: http://felix.apache.org/site/index.html. Copie o arquivo `org.apache.felix.fileinstall-3.0.2.jar` para o diretório `equinox_root/plugins`.
8. Crie um diretório de configuração dentro do diretório `equinox_root/plugins`, por exemplo:

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

9. Crie o arquivo `config.ini` a seguir no diretório `equinox_root/plugins/configuration`, substituindo *equinox_root* com o caminho absoluto para seu diretório `equinox_root` e removendo todos os espaçoss à direita após a barra invertida em cada linha. Você deve incluir uma linha em branco no final do arquivo; por exemplo:

   ```
   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
   ```

   Se já tiver configurado o ambiente, será possível limpar o repositório de plug-in do Equinox ao remover o seguinte diretório: `equinox_root\plugins\configuration\org.eclipse.osgi`.
10. Execute os seguintes comandos para iniciar o console do equinox.

Se você estiver executando uma versão diferente do Equinox, o nome do seu arquivo JAR será diferente do nome no exemplo a seguir:

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

Conceitos relacionados

"Visão Geral da Estrutura do OSGi" na página 37
O OSGi define um sistema módulo dinâmico para Java. A plataforma de serviço OSGi possui uma arquitetura em camadas e é projetada para ser executada em vários perfis padrão Java. É possível iniciar servidores e clientes do WebSphere eXtreme Scale em um contêiner OSGi.

Referências relacionadas

Arquivo de Propriedades do Servidor
O arquivo de propriedades do servidor contém várias propriedades que definem configurações diferentes para o servidor, como configurações de rastreio, criação de log e configuração de segurança. O arquivo de propriedades do servidor é usado pelo serviço de catálogo e pelos servidores de contêiner em servidores independentes e também em servidores hospedados no WebSphere Application Server.

Informações relacionadas

"Introdução: Iniciando e Configurando o Servidor e o Contêiner do eXtreme Scale para Executar Plug-ins na Estrutura do OSGi" na página 19
Neste tutorial, inicie um servidor eXtreme Scale na estrutura do OSGi, inicie um contêiner do eXtreme Scale e ligue os plug-ins de amostra com o ambiente de tempo de execução do eXtreme Scale.

Instalando Pacotes Configuráveis do eXtreme Scale

O WebSphere eXtreme Scale inclui pacotes configuráveis que podem ser instalados em uma estrutura do Eclipse Equinox OSGi. Esses pacotes configuráveis são necessários para iniciar servidores do eXtreme Scale ou usar clientes do eXtreme Scale no OSGi.

Antes de Iniciar

Esta tarefa assume que os produtos a seguir foram instalados:

- Estrutura do Eclipse Equinox OSGi
- Cliente ou servidor independente do eXtreme Scale

Sobre Esta Tarefa

O eXtreme Scale inclui dois pacotes configuráveis. Apenas um dos pacotes configuráveis a seguir é requerido em uma estrutura do OSGi:

objectgrid.jar
O pacote configurável do servidor é o arquivo `objectgrid.jar` e é instalado com a instalação do servidor independente do eXtreme Scale e é requerido para executar servidores do eXtreme Scale e também pode ser usado para executar clientes do eXtreme Scale ou caches na memória locais. O ID do pacote configurável para o arquivo `objectgrid.jar` é com.ibm.websphere.xs.server_`<version>`, em que a versão está no formato: `<Version>.<Release>.<Modification>`. Por exemplo, o pacote configurável do servidor para o eXtreme Scale versão 7.1.1 é com.ibm.websphere.xs.server_7.1.1.

ogclient.jar
O pacote configurável do `ogclient.jar` é instalado com as instalações independentes e de cliente do eXtreme Scale e é usado para executar

clientes do eXtreme Scale ou caches na memória locais. O ID do pacote configurável para o arquivo `ogclient.jar` é com.ibm.websphere.xs.client_<version>, em que a versão está no formato: <Version>_<Release>_<Modification. Por exemplo, o pacote configurável do cliente para o eXtreme Scale Versão 7.1.1 é com.ibm.websphere.xs.client_7.1.1.

Para obter mais informações sobre como desenvolver plug-ins do eXtreme Scale, consulte o tópico APIs e Plug-ins do Sistema.

Procedimento

Para instalar o pacote configurável do cliente ou servidor do eXtreme Scale na estrutura do Eclipse Equinox OSGi usando o console do OSGi:

1. Inicie a estrutura do Eclipse Equinox com o console ativado; por exemplo:
   ```
   java_home/bin/java -jar <equinox_root>/plugins/
   org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
2. Instale o pacote configurável do cliente ou servidor do eXtreme Scale no console do Equinox:
   ```
   osgi> install file:///<path to bundle>
   ```
3. O Equinox exibe o ID do pacote configurável para o pacote configurável recém-instalado:
   ```
   Bundle id is 25
   ```
4. Inicie o pacote configurável no console do Equinox, em que <id> é o ID do pacote configurável designado quando o pacote configurável foi instalado:
   ```
   osgi> start <id>
   ```
5. Recupere o status de serviço no console do Equinox para verificar se o pacote configurável foi iniciado; por exemplo:
   ```
   osgi> ss
   ```

   Quando o pacote configurável foi iniciado com êxito, o pacote configurável exibe o estado ACTIVE; por exemplo:
   ```
   25      ACTIVE      com.ibm.websphere.xs.server_7.1.1
   ```

Instale o pacote configurável do cliente ou servidor do eXtreme Scale na estrutura do Eclipse Equonix OSGi usando o arquivo `config.ini`:

6. Copie o pacote configurável do cliente ou servidor do eXtreme Scale (objectgrid.jar ou ogclient.jar) do <wxs_install_root>/ObjectGrid/lib para o diretório de plug-ins do Eclipse Equinox; por exemplo: <equinox_root>/plugins
7. Edite o arquivo de configuração `config.ini` do Eclipse Equinox e inclua o pacote configurável na propriedade osgi.bundles; por exemplo:
   ```
   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 se existe uma linha em branco após o nome do último pacote configurável. Cada pacote configurável é separado por uma vírgula.
8. Inicie a estrutura do Eclipse Equinox com o console ativado; por exemplo:
   ```
   java_home/bin/java -jar <equinox_root>/plugins/
   org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```

9. Recupere o status do serviço no console do Equinox para verificar se o pacote configurável foi iniciado:

   ```
   osgi> ss
   ```

   Quando o pacote configurável tiver iniciado com sucesso, o pacote configurável exibirá o estado ACTIVE; por exemplo:

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

Resultados

O pacote configurável do servidor ou cliente do eXtreme Scale é instalado e iniciado em sua estrutura do Eclipse Equinox OSGi.

Construindo e Executando Plug-ins Dinâmicos do eXtreme Scale para Uso em um Ambiente OSGi

Todos os plug-ins do eXtreme Scale podem ser configurados para um ambiente OSGi. O benefício principal dos plug-ins dinâmicos é que o upgrade deles pode ser feito sem encerrar a grade. Isso permite desenvolver um aplicativo sem reiniciar os processos do contêiner de grade.

Sobre Esta Tarefa

O suporte do WebSphere eXtreme Scale OSGi permite implementar o produto em uma estrutura OSGi, como o Eclipse Equinox. Anteriormente, se você desejava atualizar os plug-ins usados pelo eXtreme Scale, era necessário reiniciar a Java Virtual Machine (JVM) para aplicar as novas versões dos plug-ins. Com o suporte de plug-in dinâmico fornecido pelo eXtreme Scale e a possibilidade de atualizar pacotes configuráveis que a estrutura do OSGi fornece, agora é possível atualizar as classes de plug-in sem reiniciar a JVM. Esses plug-ins são exportados pelo *pacotes configuráveis* como serviços. O WebSphere eXtreme Scale acessa o serviço ao consultar o registro do OSGi. Na plataforma de serviço OSGi, um pacote configurável é um arquivo Java archive (JAR) que contém código Java, recursos e um manifesto que descrevem o pacote configurável e suas dependências. O pacote configurável é a unidade de implementação para um aplicativo.

Procedimento

1. Construa plug-ins dinâmicos do eXtreme Scale.
2. Configure plug-ins do eXtreme Scale with OSGi Blueprint.
3. Instale e inicie plug-ins ativados por OSGi.

Construindo Plug-ins Dinâmicos do eXtreme Scale

O WebSphere eXtreme Scale inclui plug-ins ObjectGrid e BackingMap. Estes plug-ins são implementados em Java e são configurados usando o arquivo XML do descritor do ObjectGrid. Para criar um plug-in dinâmico que pode ser dinamicamente atualizado, eles precisam estar cientes dos eventos de ciclo de vida de ObjectGrid e BackingMap porque eles podem precisar concluir algumas ações durante uma atualização. Aprimorar um pacote configurável de plug-in com métodos de retorno de chamada, listeners de eventos, ou ambos, do ciclo de vida permite que o plug-in conclua essas ações em momentos apropriados.

Antes de Iniciar

Este tópico supõe que você construiu o plug-in apropriado. Para obter informações adicionais sobre como desenvolver plug-ins do eXtreme Scale, consulte o tópico APIs e Plug-ins do Sistema.

Sobre Esta Tarefa

Todos os plug-ins do eXtreme Scale se aplicam a uma instância de BackingMap ou de ObjectGrid. Muitos plug-ins também interagem com outros plug-ins. Por exemplo, um plug-in Loader e TransactionCallback trabalham juntos para interagir corretamente com uma transação do banco de dados e as várias chamadas de banco de dados JDBC. Alguns plug-ins também pode precisar armazenar em cache dados de configuração a partir de outros plug-ins para melhorar o desempenho.

Os plug-ins BackingMapLifecycleListener e ObjectGridLifecycleListener fornecem operações de ciclo de vida para as respectivas instâncias de BackingMap e ObjectGrid. Este processo permite que plug-ins sejam notificados quando o BackingMap ou ObjectGrid pai e seus respectivos plug-ins podem ser alterados. Os plug-ins BackingMap implementam a interface de BackingMapLifecyleListener e os plug-ins ObjectGrid implementam a interface de ObjectGridLifecycleListener. Estes plug-ins são chamados automaticamente quando o ciclo de vida do BackingMap ou ObjectGrid pai é alterado. Para obter mais informações sobre os plug-ins de ciclo de vida, consulte o tópico “Gerenciando Ciclos de Vida de Plug-in” na página 299.

É possível aprimorar os pacotes configuráveis usando os métodos ou os listeners de evento do ciclo de vida nas seguintes tarefas comuns:

- Iniciar e parar recursos, como encadeamentos ou assinantes de sistema de mensagens.
- Especificar que uma notificação ocorra quando os plug-ins equivalentes forem atualizados, permitindo o acesso direto ao plug-in e a detecção de quaisquer mudanças.

Sempre que outro plug-in for acessado diretamente, acesse esse plug-in por meio do contêiner OSGi para assegurar que todas as partes do sistema referenciem o plug-in correto. Se, por exemplo, algum componente no aplicativo referenciar diretamente, ou armazenar em cache, uma instância de um plug-in, ele manterá sua referência para essa versão do plug-in, mesmo depois que o plug-in tiver sido atualizado dinamicamente. Esse comportamento pode causar problemas relacionados ao aplicativo, bem como fugas de memória. Portanto, grave o código que depende dos plug-ins dinâmicos que obtêm sua referência usando semânticas getService() do OSGi. Se o aplicativo precisar armazenar em cache um ou mais plug-ins, ele atenderá eventos de ciclo de vida usando interfaces ObjectGridLifecycleListener e BackingMapLifecycleListener. O aplicativo também deve poder atualizar seu cache quando necessário, de modo thread safe.

Todos os plug-ins do eXtreme Scale usados com o OSGi também devem implementar as respectivas interfaces BackingMapPlugin ou ObjectGridPlugin. Novos plug-ins, tal como a interface MapSerializerPlugin impingem essa prática. Essas interfaces fornecem ao ambiente de tempo de execução do eXtreme Scale e ao OSGi uma interface consistente para injeção de estado no plug-in e controle de seu ciclo de vida.

Ao usar esta tarefa para especificar que uma notificação ocorre quando os plug-ins equivalentes são atualizados, é possível criar um factory de listener que produz uma instância do listener.

Procedimento

- Atualize a classe de plug-in ObjectGrid para implementar a interface ObjectGridPlugin. Esta interface inclui métodos que permitem que o eXtreme Scale inicialize, configure a instância do ObjectGrid e destrua o plug-in. Consulte o exemplo de código a seguir:

```
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() {
        // Handle any plug-in initialization here.  This is called by
        // eXtreme Scale, and not the OSGi bean manager.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destroy the plug-in and release any resources.  This
        // can be callsed by the OSGi Bean Manager or by eXtreme Scale.
        state = State.DESTROYED;
    }

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

- Atualize a classe de plug-in do ObjectGrid para implementar a interface ObjectGridLifecycleListener. Consulte o exemplo de código a seguir:

```
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:
                  // Lookup a Loader or MapSerializerPlugin using
                  // OSGi or directly from the ObjectGrid instance.
                  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;
            }
        }
        ...
    }
```

- Atualize um plug-in do BackingMap. Atualize a classe de plug-in do BackingMap para implementar a interface de plug-in do BackingMap. Esta interface inclui métodos que permitem que o eXtreme Scale inicialize, configure a instância do BackingMap e destrua o plug-in. Consulte o exemplo de código a seguir:

```
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() {
        // Handle any plug-in initialization here.  This is called by
        // eXtreme Scale, and not the OSGi bean manager.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destroy the plug-in and release any resources.  This
        // can be callsed by the OSGi Bean Manager or by eXtreme Scale.
        state = State.DESTROYED;
    }

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

- Atualize a classe de plug-in do BackingMap para implementar a interface BackingMapLifecycleListener. Consulte o exemplo de código a seguir:

```
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:
                // Lookup a MapSerializerPlugin using
                // OSGi or directly from the ObjectGrid instance.
                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

Implementando a interface ObjectGridPlugin ou BackingMapPlugin, o eXtreme Scale pode controlar o ciclo de vida de seu plug-in nos momentos certos.

Implementando a interface ObjectGridLifecycleListener ou BackingMapLifecycleListener, o plug-in é automaticamente registrado como um listener dos eventos de ciclo de vida do ObjectGrid ou do BackingMap associados. O evento INITIALIZING é usado para sinalizar que todos os plug-ins do ObjectGrid e do BackingMap foram inicializados e estão disponíveis para consultar e usar. O evento ONLINE é usado para sinalizar que o ObjectGrid está on-line e pronto para iniciar eventos de processamento.

Configurando os Plug-ins do eXtreme Scale com o OSGi Blueprint

Todos os plug-ins do ObjectGrid e do BackingMap do eXtreme Scale podem ser definidos como beans e serviços OSGi usando o Serviço OSGi Blueprint disponível com o Eclipse Gemini ou o Aries Apache.

Antes de Iniciar

Antes de poder configurar seus plug-ins como serviços OSGi, você deve primeiro empacotar seus plug-ins em um pacote configurável OSGi e entender os princípios fundamentais dos plug-ins necessários. O pacote configurável deve importar os pacotes do cliente ou servidor do WebSphere eXtreme Scale e outros pacotes dependentes requeridos pelos plug-ins ou criar uma dependência do pacote configurável nos pacotes configuráveis do servidor ou cliente doeXtreme Scale. Este tópico descreve como configurar o XML Blueprint para criar beans de plug-in e expô-los como serviços OSGi para o eXtreme Scale usar.

Sobre Esta Tarefa

Beans e serviços são definidos em um arquivo XML do Blueprint e o contêiner do Blueprint descobre, cria e liga os beans juntos e os expõe como serviços. O processo torna os beans disponíveis para outros pacotes configuráveis OSGi, incluindo os pacotes configuráveis de servidor e cliente do eXtreme Scale.

Ao criar serviços de plug-in customizados para uso com o eXtreme Scale, o pacote configurável que deve hospedar os plug-ins deve ser configurado para usar Blueprint. Além disso, um arquivo XML do Blueprint deve ser criado e armazenado dentro do pacote configurável. Leia sobre construção de aplicativos OSGi com a especificação do Contêiner do Blueprint para obter um entendimento geral da especificação.

Procedimento

1. Crie um arquivo XML do Blueprint. É possível nomear o arquivo de qualquer jeito. No entanto, você deve incluir o namespace do projeto:

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

2. Crie definições de bean no arquivo XML do Blueprint para cada plug-in do eXtreme Scale.

   Beans são definidos usando o elemento `<bean>` e podem ser ligados a outras referências de bean e podem incluir parâmetros de inicialização.

   Importante: Ao definir um bean, você deve usar o escopo correto. O Blueprint suporta os escopos singleton e de protótipo. O eXtreme Scale também suporta um escopo de shard customizado.

   Defina a maioria dos plug-ins do eXtreme Scale como protótipo ou beans com escopo definido em shard, uma vez que todos os beans devem ser exclusivos para cada shard do ObjectGrid ou instância do BackingMap à qual eles estão associados. Beans com escopo definido em shard podem ser úteis ao usar osbeans em outros contextos para permitir a recuperação da instância correta.

   Para definir um bean com escopo definido em protótipo, use o atributo `scope="prototype"` no bean:

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

   Para definir um bean com escopo definido em shard, você deve incluir o namespace `objectgrid` no esquema XML e usar o atributo `scope="objectgrid:shard"` no 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. Crie definições de bean PluginServiceFactory para cada bean de plug-in. Todos os beans do eXtreme Scale devem ter um bean do PluginServiceFactory definido para que o escopo do bean correto possa ser aplicado. O eXtreme Scale inclui um BlueprintServiceFactory que você pode usar. Ele inclui duas propriedades que devem ser configuradas. Você deve configurar a propriedade blueprintContainer com a referência `blueprintContainer` e a propriedade beanId deve ser configurada com o nome do identificador de bean. Quando o eXtreme Scale consulta o serviço para instanciar os beans apropriados, o servidor procura a instância do componente do bean usando o contêiner Blueprint.

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

4. Crie um gerenciador de serviços para cada bean PluginServiceFactory. Cada gerenciador de serviços expõe o bean PluginServiceFactory, usando o elemento `<service>`. O elemento de serviço identifica o nome a ser exposto para o OSGi,

a referência para o bean PluginServiceFactory, a interface a ser exposta e a classificação do serviço. O eXtreme Scale usa a classificação do gerenciador de serviços para executar upgrades de serviço quando a grade do eXtreme Scale está ativa. Se a classificação não for especificada, a estrutura do OSGi assumirá uma classificação igual a 0. Leia sobre como atualizar as classificações de serviço para obter mais informações.

Blueprint inclui várias opções para configurar gerenciadores de serviços. Para definir um gerenciador de serviços simples para um bean PluginServiceFactory, crie um elemento `<service>` para cada bean PluginServiceFactory:

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

5. Armazene o arquivo XML do Blueprint no pacote configurável de plug-ins. O arquivo XML do Blueprint deve ser armazenado no diretório `OSGI-INF/blueprint` para o contêiner do Blueprint a ser descoberto.

   Para armazenar o arquivo XML do Blueprint em um diretório diferente, você deve especificar o seguinte cabeçalho de manifesto Bundle-Blueprint:

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

Resultados

Os plug-ins do eXtreme Scale agora são configurados para serem expostos em um contêiner OSGi Blueprint, além disso, o arquivo XML do descritor do ObjectGrid é configurado para referenciar os plug-ins usando o serviço OSGi Blueprint.

Instalando e Iniciando Plug-ins Ativados pelo OSGi

Nesta tarefa, você instala o pacote configurável de plug-in dinâmico na estrutura do OSGi. Em seguida, inicia o plug-in.

Antes de Iniciar

Este tópico assume que as seguintes tarefas foram concluídas:

- O pacote configurável do servidor ou cliente do eXtreme Scale foi instalado na estrutura do Eclipse Equinox OSGi. Consulte “Instalando Pacotes Configuráveis do eXtreme Scale” na página 41.
- Um ou mais plug-ins BackingMap ou ObjectGrid dinâmicos foram implementados. Consulte “Construindo Plug-ins Dinâmicos do eXtreme Scale” na página 43.
- Os plug-ins dinâmicos foram empacotados como serviços OSGi nos pacotes configuráveis OSGi.

Sobre Esta Tarefa

Esta tarefa descreve como instalar o pacote configurável usando o console do Eclipse Equinox. O pacote configurável pode ser instalado usando vários métodos diferentes, incluindo a modificação do arquivo de configuração `config.ini`. Os produtos que incorporam o Eclipse Equinox incluem métodos alternativos para gerenciar pacotes configuráveis. Para obter mais informações sobre como incluir pacotes configuráveis no arquivo `config.ini` no Eclipse Equinox, consulte as opções de tempo de execução do Eclipse.

O OSGi permite que pacotes configuráveis que possuem serviços duplicados sejam iniciados. O WebSphere eXtreme Scale usa a classificação de serviço mais recente. Ao iniciar diversas estruturas OSGi em uma grade de dados do eXtreme Scale,

você deve se certificar de que as classificações de serviço corretas sejam iniciados em cada servidor. Não fazer isso faz com que a grade seja iniciada com uma mistura de diferentes versões.

Para ver quais versões estão em uso pela grade de dados, use o utilitário xscmd para verificar as classificações atuais e disponíveis. Para obter mais informações sobre classificações de serviço disponíveis, consulte Atualizando Serviços OSGi para Plug-ins do eXtreme Scale com **xscmd**.

Procedimento

Instale o pacote configurável de plug-in na estrutura do Eclipse Equinox OSGi usando o console do OSGi.

1. Inicie a estrutura do Eclipse Equinox com o console ativado; por exemplo:
   ```
   <java_home>/bin/java -jar <equinox_root>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
2. Instale o pacote configurável de plug-in no console do Equinox.
   ```
   osgi> install file:///<path to bundle>
   ```

   O Equinox exibe o ID do pacote configurável para o pacote configurável recém-instalado:
   ```
   Bundle id is 17
   ```
3. Insira a linha a seguir para iniciar o pacote configurável no console do Equinox, em que `<id>` é o ID do pacote configurável designado quando o pacote configurável foi instalado:
   ```
   osgi>  install <id>
   ```
4. Recupere o status do serviço no console do Equinox para verificar se o pacote configurável foi iniciado:
   ```
   osgi> ss
   ```

   Quando o pacote configurável foi iniciado com êxito, o pacote configurável exibe o estado ACTIVE; por exemplo:
   ```
   17      ACTIVE      com.mycompany.plugin.bundle_VRM
   ```

Instale o pacote configurável de plug-in na estrutura do Eclipse Equonix OSGi usando o arquivo config.ini.

5. Copie o pacote configurável de plug-in no diretório de plug-ins do Eclipse Equinox; por exemplo:
   ```
   <equinox_root>/plugins
   ```
6. Edite o arquivo de configuração `config.ini` do Eclipse Equinox e inclua o pacote configurável na propriedade osgi.bundles; por exemplo:
   ```
   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 se existe uma linha em branco após o nome do último pacote configurável. Cada pacote configurável é separado por uma vírgula.
7. Inicie a estrutura do Eclipse Equinox com o console ativado; por exemplo:
   ```
   <java_home>/bin/java -jar <equinox_root>/plugins/org.eclipse.osgi_3.6.1.R36x_v20100806.jar -console
   ```
8. Recupere o status de serviço no console do Equinox para verificar se o pacote configurável foi iniciado; por exemplo:
   ```
   osgi> ss
   ```

Quando o pacote configurável foi iniciado com êxito, o pacote configurável exibe o estado ACTIVE; por exemplo:

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

Resultados

O pacote configurável de plug-in agora está instalado e iniciado. O contêiner ou cliente do eXtreme Scale agora pode ser iniciado. Para obter mais informações sobre como desenvolver plug-ins do eXtreme Scale, consulte o tópico APIs e Plug-ins do Sistema "APIs e Plug-ins do Sistema" na página 299.

Executando os Contêineres do eXtreme Scale com Plug-ins Dinâmicos em um Ambiente do OSGi

Se seu aplicativo estiver hospedado na estrutura do Eclipse Equinox OSGi com o Eclipse Gemini ou o Apache Aries, será possível usar esta tarefa para ajudá-lo a instalar e configurar seu aplicativo WebSphere eXtreme Scale no OSGi.

Antes de Iniciar

Antes de iniciar esta tarefa, certifique-se de concluir as tarefas a seguir:

- Instale a estrutura do Eclipse Equinox OSGi com o Eclipse Gemini
- Construa e execute plug-ins dinâmicos do eXtreme Scale para usar em um ambiente OSGi

Sobre Esta Tarefa

Com plug-ins dinâmicos, é possível atualizar dinamicamente o plug-in enquanto a grade ainda está ativa. Isso permite atualizar um aplicativo sem reiniciar os processos do contêiner de grade. Para obter informações adicionais sobre como desenvolver plug-ins do eXtreme Scale, consulte APIs e Plug-ins do Sistema.

Procedimento

1. Configure plug-ins ativados por OSGi usando o arquivo XML do descritor do ObjectGrid.
2. Inicie os servidores de contêiner do eXtreme Scale usando a estrutura do Eclipse Equinox OSGi.
3. Administre serviços OSGi para plug-ins do eXtreme Scale com o utilitário xscmd.
4. Configure servidores com o OSGi Blueprint.

Configurando Plug-ins Ativados pelo OSGi Usando o Arquivo Descritor XML do ObjectGrid

Nesta tarefa, você inclui serviços OSGi existentes em um arquivo XML descritor de forma que os contêineres do WebSphere eXtreme Scale possam reconhecer e carregar os plug-ins ativados pelo OSGi corretamente.

Antes de Iniciar

Para configurar seus plug-ins, certifique-se de:

- Criar seu pacote e ativar plug-ins dinâmicos para implementação do OSGi.
- Ter os nomes dos serviços OSGi que representam seus plug-ins disponíveis.

Sobre Esta Tarefa

Você criou um serviço OSGi para agrupar seu plug-in. Agora, esses serviços devem ser definidos no arquivo `objectgrid.xml` de modo que os contêineres do eXtreme Scale possam carregar e configurar o plug-in ou plug-ins com êxito.

Procedimento

1. Quaisquer plug-ins específicos da grade, tal como TransactionCallback, devem ser especificados sob o elemento objectGrid. Consulte o exemplo a seguir a partir do arquivo `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: O valor de atributo `osgiService` deve corresponder ao valor de atributo `ref` que é especificado no arquivo XML `blueprint`, no qual o serviço foi definido para myTranCallback PluginServiceFactory.

2. Quaisquer plug-ins específicos do mapa, como carregadores ou serializadores, por exemplo, devem ser especificados no elemento backingMapPluginCollections e referenciados a partir do elemento backingMap. Consulte o exemplo a seguir a partir do arquivo `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

O arquivo `objectgrid.xml` neste exemplo informa ao eXtreme Scale para criar uma grade denominada `MyGrid` com dois mapas, `MyMap1` e `MyMap2`. O mapa `MyMap1` usa o serializador agrupado pelo serviço OSGi, mySerializerFactory. O mapa `MyMap2` usa

um serializador do serviço OSGi, myOtherSerializerFactory, e um carregador a partir do serviço OSGi, myLoader.

Iniciando Servidores do eXtreme Scale Usando a Estrutura do Eclipse Equinox OSGi

Os servidores de contêiner do WebSphere eXtreme Scale podem ser iniciados em uma estrutura do Eclipse Equinox OSGi usando vários métodos.

Antes de Iniciar

Antes de poder iniciar um contêiner do eXtreme Scale, você deve ter concluído as tarefas a seguir:

1. O pacote configurável do servidor do WebSphere eXtreme Scale deve estar instalado no Eclipse Equinox.
2. Seu aplicativo deve ser empacotado como um pacote configurável OSGi.
3. Seus plug-ins do WebSphere eXtreme Scale (se houver) devem ser empacotados como um pacote configurável OSGi. Eles podem ser empacotados no mesmo pacote configurável que seu aplicativo ou como pacotes configuráveis separados.

Sobre Esta Tarefa

Esta tarefa descreve como iniciar um servidor de contêiner do eXtreme Scale em uma estrutura do Eclipse Equinox OSGi. É possível usar qualquer um dos seguintes métodos para iniciar os servidores de contêiner usando a implementação do Eclipse Equinox:

- Serviço do OSGi Blueprint

  É possível incluir toda a configuração e os metadados em um pacote configurável OSGi. Consulte a imagem a seguir para compreender o processo do Eclipse Equinox para este método:

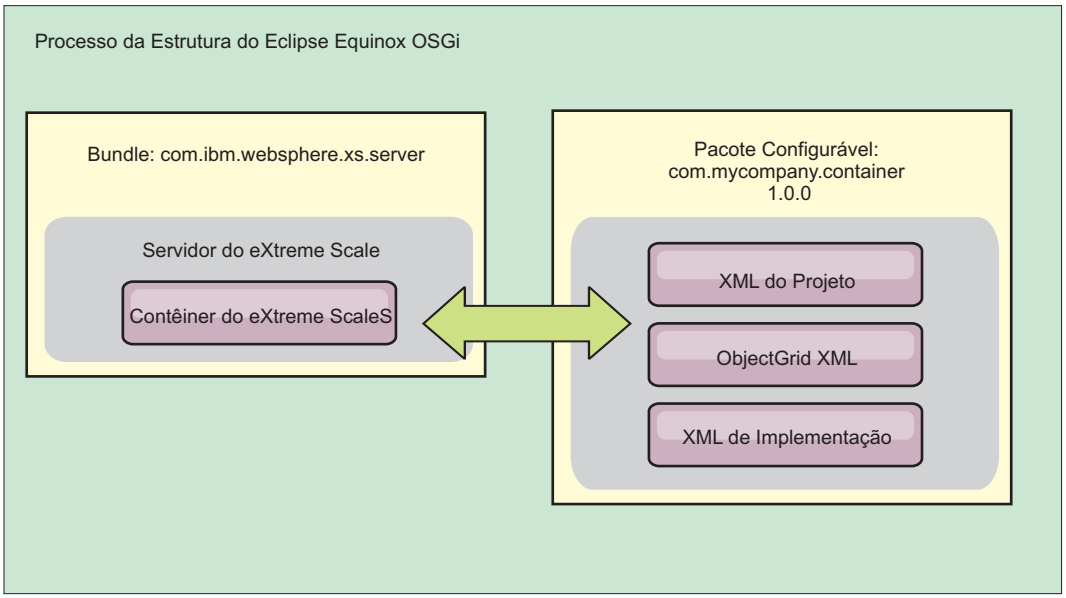


Figura 2. Processo do Eclipse Equinox para Incluir Toda a Configuração e Todos os Metadados em um Pacote Configurável OSGi

- Serviço de Administração de Configuração do OSGi

É possível especificar a configuração e os metadados fora de um pacote configurável OSGi. Consulte a imagem a seguir para compreender o processo do Eclipse Equinox para este método:

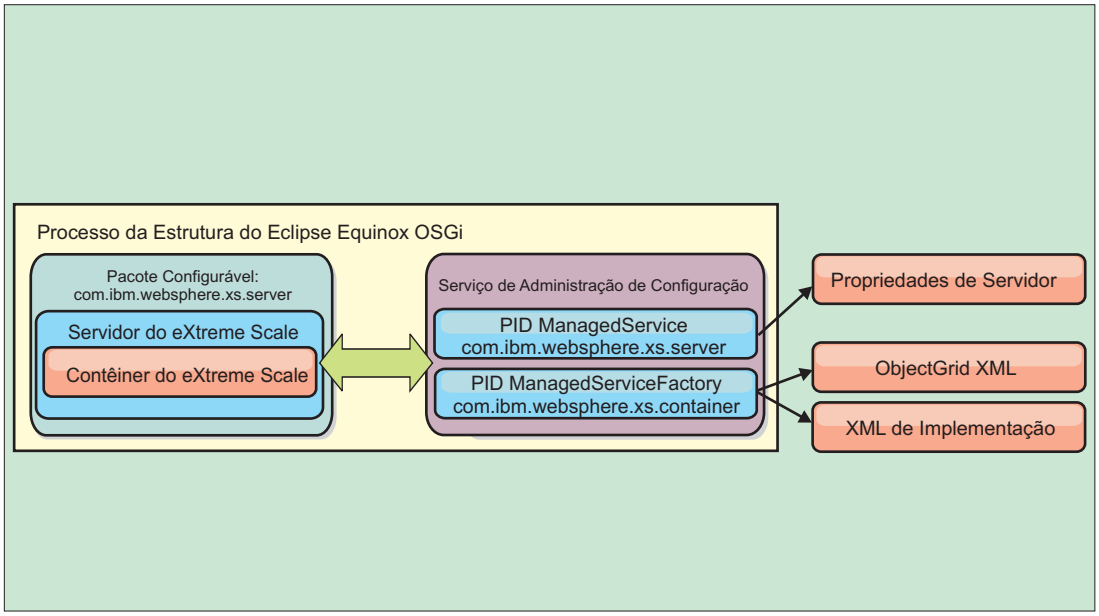


Figura 3. Processo do Eclipse Equinox para Especificar a Configuração e os Metadados Fora de um Pacote Configurável OSGi

- Programaticamente
  Suporta soluções de configuração customizadas.

Em cada caso, um singleton do servidor do eXtreme Scale é configurado e um ou mais contêineres são configurados.

O pacote configurável do servidor do eXtreme Scale, `objectgrid.jar`, inclui todas as bibliotecas necessárias para iniciar e executar um contêiner de grade do eXtreme Scale em uma estrutura OSGi. O ambiente de tempo de execução do servidor se comunica com os plug-ins e objetos de dados fornecidos pelo usuário usando o gerenciador de serviço OSGi.

Importante: Depois de um pacote configurável do servidor do eXtreme Scale ser iniciado e o servidor do eXtreme Scale ser inicializado, ele não poderá ser reiniciado. O processo do Eclipse Equinox deve ser reiniciado para reiniciar um servidor do eXtreme Scale.

É possível usar o suporte do eXtreme Scale para o namespace do Spring para configurar os servidores de contêiner do eXtreme Scale em um arquivo XML do Blueprint. Quando os elemento XML do servidor e do contêiner são incluídos no arquivo XML do Blueprint, o manipulador de namespace do eXtreme Scale inicia automaticamente um servidor de contêiner usando os parâmetros que são definidos no arquivo XML do Blueprint quando o pacote configurável é iniciado. A manipulação para o contêiner quando o pacote configurável é interrompido.

Para configurar servidores de contêiner do eXtreme Scale com o XML do Blueprint, conclua as etapas a seguir:

Procedimento

- Inicie um servidor de contêiner do eXtreme Scale usando o OSGi Blueprint.
  1. Crie um pacote configurável de contêiner.
  2. Instale o pacote configurável de contêiner na estrutura do Eclipse Equinox OSGi. Consulte “Instalando e Iniciando Plug-ins Ativados pelo OSGi” na página 49.
  3. Inicie o pacote configurável do contêiner.
- Inicie uma servidor de contêiner do eXtreme Scale usando a administração de configuração do OSGi.
  1. Configure o servidor e o contêiner usando a administração de configuração.
  2. Quando o pacote configurável do servidor do eXtreme Scale é iniciado ou os identificadores persistentes são criados com config admin, o servidor e o contêiner iniciam automaticamente.
- Inicie um servidor de contêiner do eXtreme Scale usando a API do ServerFactory. Consulte a documentação da API do servidor.
  1. Crie uma classe de ativador do pacote configurável OSGi e use a API do ServerFactory do eXtreme Scale para iniciar um servidor.

Administrando Serviços Ativados pelo OSGi Usando o Utilitário `xscmd`

É possível usar o utilitário **xscmd** para concluir as tarefas de administrador, como visualizar serviços e suas classificações que estão sendo usados por cada contêiner e atualizar o ambiente de tempo de execução para utilizar novas versões dos pacotes configuráveis.

Sobre Esta Tarefa

Com a estrutura do Eclipse Equinox OSGi, é possível instalar diversas versões do mesmo pacote configurável e você pode atualizar esses pacotes configuráveis durante o tempo de execução. O WebSphere eXtreme Scale é um ambiente distribuído que executa os servidores de contêiner em muitas instâncias da estrutura do OSGi.

Os administradores são responsáveis por copiar, instalar e iniciar manualmente pacotes configuráveis na estrutura do OSGi. O eXtreme Scale inclui um ServiceTrackerCustomizer OSGi para controlar quaisquer serviços que foram identificados como plug-ins do eXtreme Scale no arquivo XML do descritor do ObjectGrid. Use o utilitário **xscmd** para validar qual versão do plug-in é usado, quais versões estão disponíveis para serem usadas e para executar upgrades do pacote configurável.

O eXtreme Scale usa o número de classificação do serviço para identificar a versão de cada serviço. Quando dois ou mais serviços são carregados com a mesma referência, o eXtreme Scale usa automaticamente o serviço com a classificação mais alta.

Procedimento

- Execute o comando **osgiCurrent** e verifique se cada servidor eXtreme Scale está usando a classificação do serviço de plug-in correta.

  Como o eXtreme Scale escolhe automaticamente a referência de serviço com a classificação mais alta, é possível que a grade de dados possa iniciar com diversas classificações de um serviço de plug-in.

Se o comando detecta uma incompatibilidade de classificações ou se ele é incapaz de localizar um serviço, um nível de erro diferente de zero é configurado. Se o comando foi concluído com êxito, o nível de erro é configurado como 0.

O exemplo a seguir mostra a saída do comando **osgiCurrent** quando dois plug-ins estão instalados na mesma grade em quatro servidores. O plug-in loaderPlugin está usando classificação de 1 e txCallbackPlugin está usando classificação 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
```

O exemplo a seguir mostra a saída do comando **osgiCurrent** quando server2 foi iniciado com uma classificação mais nova do 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
```

- Execute o comando **osgiAll** para verificar se os serviços de plug-in foram iniciados corretamente em cada servidor de contêiner do eXtreme Scale.

  Quando pacotes configuráveis que contêm serviços que uma configuração do ObjectGrid está referenciando são iniciados, o ambiente de tempo de execução do eXtreme Scale controla automaticamente o plug-in, mas não o usa imediatamente. O comando **osgiAll** mostra quais plug-ins estão disponíveis para cada servidor.

  Quando executados sem quaisquer parâmetros, todos os serviços são mostrados para todas as grades e todos os servidores. Filtros adicionais, incluindo o filtro **-serviceName** <service_name>, podem ser especificados para limitar a saída para um único serviço ou um subconjunto da grade de dados.

  O exemplo a seguir mostra a saída do comando **osgiAll** quando dois plug-ins são iniciados em dois servidores. O loaderPlugin possui ambas as classificações, 1 e 2, iniciadas e o txCallbackPlugin tem a classificação 1 iniciada. A mensagem de resumo no final da saída confirma que ambos os servidores consultam as mesmas classificações de serviço:

```
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.
```

O exemplo a seguir mostra a saída do comando **osgiAll** quando o pacote configurável que inclui o loaderPlugin com classificação 1 é interrompido no server1. A mensagem de resumo na parte inferior da saída confirma que o server1 agora está ausente no loaderPlugin com classificação 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
```

O exemplo a seguir mostra a saída se o nome do serviço é especificado com o argumento **-sn**, mas o serviço não 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.
```

- Execute o comando **osgiCheck** para verificar conjuntos de serviços de plug-in e classificações para ver se eles estão disponíveis.

  O comando **osgiCheck** aceita um ou mais conjuntos de classificações de serviço no formato: `-serviceRankings <service name>;<ranking>[,<serviceName>;<ranking>]`

  Quando as classificações estão todas disponíveis, o método retorna com um nível de erro igual a 0. Se uma ou mais classificações estão indisponíveis, um erro de nível diferente de zero é configurado e uma tabela de todos os servidores que não incluem as classificações de serviço especificadas. Filtros adicionais podem ser usados para limitar a verificação de serviço para um subconjunto dos servidores disponíveis no domínio do eXtreme Scale.

  Por exemplo, se a classificação ou o serviço especificado estiver ausente, a seguinte mensagem será exibida:

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

- Execute o comando **osgiUpdate** para atualizar a classificação de um ou mais plug-ins para todos os servidores em um único ObjectGrid e MapSet em uma única operação.

  O comando aceita um ou mais conjuntos de classificações de serviço no formato: `-serviceRankings <service name>;<ranking>[,<serviceName>;<ranking>] -g <grid name> -ms <mapset name>`

  Com este comando, é possível concluir as operações a seguir:

- Verifique se os serviços especificados estão disponíveis para atualização em cada um dos servidores.
- Altere o estado da grade para offline usando a interface StateManager. Consulte Gerenciando a Disponibilidade do ObjectGrid para obter mais informações. Este processo coloca a grade em modo quiesce e aguarda até que qualquer transação em execução tenha concluído e impede o início de qualquer nova transação. Este processo também sinaliza quaisquer plug-ins ObjectGridLifecycleListener e BackingMapLifecycleListener para descontinuar qualquer atividade transacional. Consulte "Plug-ins para Fornecer Listeners de Eventos" na página 317 para obter informações sobre plug-ins do listener de eventos.
- Atualize cada contêiner do eXtreme Scale em execução em uma estrutura OSGi para usar as novas versões de serviço.
- Altere o estado da grade para online, permitindo que as transações continuem.

O processo de atualização é idempotente, de forma que, se um cliente falhar ao concluir qualquer tarefa, isto resultará na operação sendo recuperada. Se um cliente for incapaz de executar a recuperação ou for interrompido durante o processo de atualização, o mesmo comando poderá ser emitido novamente e ele continuará na etapa apropriada.

Se o cliente for incapaz de continuar, e o processo for reiniciado a partir de um outro cliente, use a opção `-force` para permitir que o cliente execute a atualização. O comando **xscmd.bat/xscmd.sh** impede que diversos clientes atualizem o mesmo conjunto de mapas simultaneamente. Para obter mais detalhes sobre o comando **osgiUpdate**, consulte Atualizando Serviços OSGi para Plug-ins do eXtreme Scale com **xscmd**.

Configurando Servidores com o OSGi Blueprint

É possível configurar os servidores de contêiner do WebSphere eXtreme Scale usando um arquivo XML do OSGi Blueprint, permitindo o empacotamento e o desenvolvimento simplificados de pacotes configuráveis do servidor autocontidos.

Antes de Iniciar

Este tópico assume que as seguintes tarefas foram concluídas:

- A estrutura do Eclipse Equinox OSGi foi instalada e iniciada com o contêiner de projeto do Eclipse Gemini ou do Apache Aries.
- O pacote configurável do servidor eXtreme Scale foi instalado e iniciado.
- O pacote configurável de plug-ins dinâmicos do eXtreme Scale foi criado.
- O arquivo XML do descritor do ObjectGrid do eXtreme Scale e o arquivo XML da política de implementação foram criados.

Sobre Esta Tarefa

Esta tarefa descreve como configurar um servidor do eXtreme Scale com um contêiner usando um arquivo XML do projeto. O resultado do procedimento é um pacote configurável do contêiner. Quando o pacote configurável do contêiner for iniciado, o pacote configurável do servidor eXtreme Scale controlará o pacote configurável, analisará o XML do servidor e iniciará um servidor e um contêiner.

Um pacote configurável do contêiner pode ser, opcionalmente, combinado com o aplicativo e os plug-ins do eXtreme Scale quando atualizações do plug-in dinâmico não são necessárias ou os plug-ins não suportam a atualização dinâmica.

Procedimento

1. Crie um arquivo XML do Blueprint com o namespace `objectgrid` incluído. É possível nomear o arquivo de qualquer jeito. No entanto, ele deve incluir o namespace do projeto:

```
<?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. Inclua a definição de XML para o servidor eXtreme Scale com as propriedades de servidor apropriadas. Consulte o arquivo XML do descritor do Spring para obter detalhes sobre todas as propriedades de configuração disponíveis. Consulte o exemplo a seguir da definição de 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. Inclua a definição de XML para o contêiner do eXtreme Scale com a referência para a definição de servidor e os arquivos XML do descritor do ObjectGrid e de implementação do ObjectGrid integrados no pacote configurável; por exemplo:

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

4. Armazene o arquivo XML do Blueprint no pacote configurável do contêiner. O XML do Blueprint deve ser armazenado no diretório `OSGI-INF/blueprint` para que o contêiner do Blueprint seja localizado.

   Para armazenar o XML do Blueprint em um diretório diferente, você deve especificar o cabeçalho de manifesto Bundle-Blueprint; por exemplo:

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

5. Empacote os arquivos em um arquivo JAR do pacote configurável único. Consulte o exemplo a seguir de uma hierarquia do diretório do pacote configurável:

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

Resultados

Um pacote configurável do contêiner do eXtreme Scale agora está criado e pode ser instalado no Eclipse Equinox. Quando o pacote configurável do contêiner é iniciado, o ambiente de tempo de execução do servidor eXtreme Scale no pacote configurável do servidor eXtreme Scale irá iniciar automaticamente servidor do eXtreme Scale de singleton usando os parâmetros definidos no pacote configurável e um servidor de contêiner é iniciado. O pacote configurável pode ser interrompido e iniciado, o que resulta no contêiner parando e iniciando. O servidor

é um singleton e não para quando o pacote configurável é iniciado pela primeira vez.

Capítulo 3. Introdução

Depois de instalar o produto, será possível usar a introdução de amostra para testar a instalação e usar o produto pela primeira vez.

Tutorial: Introdução ao WebSphere eXtreme Scale

Depois de instalar o WebSphere eXtreme Scale em um ambiente independente, é possível usar o aplicativo de amostra de introdução como uma introdução simples para sua capacidade como uma grade de dados em memória.

Objetivos do aprendizado

- Aprenda sobre o arquivo descritor XML do ObjectGrid e os arquivos do descritor XML da política de implementação que são usados para configurar seu ambiente.
- Inicie os servidores de catálogos e de contêiner usando os arquivos de configuração.
- Aprenda sobre como desenvolver um aplicativo cliente.
- Execute o aplicativo cliente para inserir dados na grade de dados.
- Monitore as grades de dados com o console da web

Tempo necessário

60 minutos

Introduzindo a Lição 1 do Tutorial: Definindo Grades de Dados com Arquivos de Configuração

Para configurar grades de dados simples, use os arquivos `objectgrid.xml` e `deployment.xml` que são fornecidos na amostra Introdução.

A amostra usa os arquivos `objectgrid.xml` e `deployment.xml` que estão no diretório *wxs_install_root*`/ObjectGrid/gettingstarted/xml`. Estes arquivos são transmitidos para os comandos iniciais para iniciar servidores de contêiner e um servidor de catálogos. O arquivo `objectgrid.xml` é o arquivo descritor XML do ObjectGrid. O arquivo `objectgrid.xml` é o arquivo descritor XML da política de implementação do ObjectGrid. Esses arquivos definem uma topologia distribuída.

Referências relacionadas

Arquivo XML descritor do ObjectGrid
Para configurar o WebSphere eXtreme Scale, utilize um arquivo XML descritor do ObjectGrid e a API do ObjectGrid.

Arquivo Descritor XML de Política de Implementação
Para configurar uma política de implementação, utilize um arquivo XML do descritor da política de implementação.

Arquivo XML descritor do ObjectGrid

Um arquivo XML descritor de ObjectGrid é usado para definir a estrutura do ObjectGrid que será usada pelo aplicativo. Ele inclui uma lista de configurações de mapa de apoio. Esses mapas de apoio armazenam os dados de cache. O exemplo a

seguir é um arquivo objectgrid.xml de amostra. As primeiras linhas do arquivo incluem o cabeçalho obrigatório de cada arquivo XML do ObjectGrid. O arquivo de exemplo a seguir define Grid ObjectGrid com os mapas de apoio Map1 e 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>
```

Arquivo Descritor XML de Política de Implementação

Um arquivo descritor XML de política de implementação é passado para o servidor de contêiner durante a inicialização. Uma política de implementação deve ser usada com o arquivo XML de ObjectGrid e deve ser compatível com o XML de ObjectGrid que é usado com ele. Para cada elemento objectgridDeployment na política de implementação, você deve ter um elemento ObjectGrid correspondente no arquivo XML do ObjectGrid. Os elementos de backingMap que são definidos dentro do elemento objectgridDeployment devem ser consistentes com os backingMaps localizados no XML de ObjectGrid. Cada backingMap deve ser referido dentro de um e apenas um mapSet.

O arquivo XML descritor de política de implementação deve igualar-se com o arquivo XML ObjectGrid correspondente, o arquivo objectgrid.xml. No seguinte exemplo, as primeiras linhas do arquivo deployment.xml incluem o cabeçalho obrigatório de cada arquivo XML de política de implementação. O arquivo define o elemento objectgridDeployment para o ObjectGrid da Grade que está definido no arquivo objectgrid.xml. Ambos os BackingMaps, Map1 e Map2, que são definidos dentro do ObjectGrid da Grade estão incluídos no mapSet que tem os atributos numberOfPartitions, minSyncReplicas e 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>
```

O atributo numberOfPartitions do elemento mapSet especifica o número de partições para o mapSet. Ele é um atributo opcional e o padrão é 1. O número deve ser apropriado para a capacidade antecipada da grade de dados.

O atributo minSyncReplicas de mapSet especifica o número mínimo de réplicas síncronas para cada partição no mapSet. Ele é um atributo opcional e o padrão é 0. Primária e réplica não são colocadas até que o domínio possa suportar o número mínimo de réplicas síncronas. Para suporte do valor minSyncReplicas, é preciso de

mais um contêiner do que o valor de minSyncReplicas. Se o número de réplicas síncronas ficar abaixo do valor de minSyncReplicas, grave transações que não são mais permitidas àquela partição.

O atributo maxSyncReplicas de mapSet especifica o número máximo de réplicas síncronas para cada partição no mapSet. Ele é um atributo opcional e o padrão é 0. Nenhuma outra réplica síncrona é posicionada para uma partição após um domínio alcançar este número de réplicas síncronas para esta partição específica. A inclusão de contêineres que podem suportar esse ObjectGrid pode resultar em um aumento no número de réplicas síncronas se seu valor de maxSyncReplicas ainda não tiver sido atingido. A amostra define o maxSyncReplicas para 1, que significa que o domínio posicionará, no máximo, uma réplica síncrona. Se você iniciar mais de uma instância do servidor de contêineres, haverá somente uma réplica síncrona posicionada em uma das instâncias do servidor de contêineres.

Ponto de verificação de lições

Nesta lição, você aprendeu:

- Como definir os mapas que armazenam os dados no arquivo descritor XML do ObjectGrid.
- Como usar o arquivo descritor XML de implementação para definir o número de partições e réplicas para a grade de dados.

Lição 2 do Tutorial de Introdução: Criando um Aplicativo Cliente

Para inserir, excluir, atualizar e recuperar dados de sua grade de dados, você deverá gravar um aplicativo cliente. A introdução de amostra inclui um aplicativo cliente que pode ser usado para saber mais sobre a criação de seu próprio aplicativo cliente.

O arquivo `Client.java` no diretório *wxs_install_root*`/ObjectGrid/gettingstarted/client/src/` é o programa cliente que demonstra como se conectar a um servidor de catálogos, obter a instância do ObjectGrid e usar a API ObjectMap. A API ObjectMap armazena dados como pares de valores de chave e é ideal para armazenar em cache os objetos que não possuírem nenhum relacionamento envolvido.

Se você precisar armazenar em cache objetos que possuem relacionamentos, use a API EntityManager .

1. Conexão com o serviço de catálogo por meio da obtenção de uma instância de ClientClusterContext

   Para se conectar a um servidor de catálogos, use o método connect da API ObjectGridManager. O método connect que é usado requer apenas o terminal do servidor de catálogos no formato de *`hostname:port`*. É possível indicar diversos terminais do servidor de catálogos separando a lista de valores de *`hostname:port`* com vírgulas. O fragmento de código a seguir demonstra como se conectar a um servidor de catálogos e obter uma instância de ClientClusterContext:

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

   Se as conexões com os servidores de catálogos forem bem-sucedidas, o método connect retorna uma instância de ClientClusterContext. A instância do ClientClusterContext é necessária para a obtenção do ObjectGrid a partir da API do ObjectGridManager.

2. Obter uma instância do ObjectGrid.

   Para obter uma instância do ObjectGrid, use o método getObjectGrid da API do ObjectGridManager. O método getObjectGrid requer a instância de ClientClusterContext e o nome da instância da grade de dados. A instância do ClientClusterContext é obtida durante a conexão com o servidor de catálogos. O nome da instância de ObjectGrid é `Grid` que é especificado no arquivo `objectgrid.xml`. O fragmento de código a seguir demonstra como obter a grade de dados chamando o método getObjectGrid da API ObjectGridManager.

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

3. Obter uma instância de Sessão.

   É possível obter uma Sessão da instância do ObjectGrid obtida. Uma instância da Sessão é necessária para obter a instância do ObjectMap e executar a demarcação da transação. O fragmento de código a seguir demonstra como obter uma instância da Sessão chamando o método getSession da API ObjectGrid.

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

4. Obter uma instância do ObjectGrid.

   Após obter uma Sessão, é possível obter uma instância do ObjectMap a partir de uma instância da Sessão chamando o método getMap da API de Sessão. Você deve transmitir o nome do mapa como parâmetro para o método getMap para obter a instância de ObjectMap. O fragmento de código a seguir demonstra como obter o ObjectMap chamando o método getMap da API de Sessão.

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

5. Use os métodos ObjectMap.

   Depois que uma instância ObjectMap for obtida, será possível usar a API ObjectMap. Lembre-se de que a interface ObjectMap é um mapa transacional e requer demarcação de transação usando os métodos begin e commit da API Session. Se não houver demarcação de transação explícita no aplicativo, as operações do ObjectMap serão executadas com transações de confirmação automática.

   O fragmento de código a seguir demonstra como usar a API ObjectMap com uma transação de confirmação automática.

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

   O fragmento de código a seguir demonstra como usar a API ObjectMap com demarcação de transação explícita.

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

Conceitos relacionados

"Objetos de Armazenamento em Cache sem Relacionamentos Envolvidos (API ObjectMap)" na página 155
Os ObjectMaps são como Mapas Java que permitem que os dados sejam armazenados como pares chave-valor. Os ObjectMaps apresentam uma abordagem simples e intuitiva para o aplicativo que armazenará os dados. Um ObjectMap é ideal para o armazenamento em cache de objetos que não tenham nenhum relacionamento envolvido. Se os relacionamentos de objetos estiverem envolvidos, então você deve usar a API EntityManager.

Tarefas relacionadas

"Introdução ao Desenvolvimento de Aplicativos" na página 70
Para começar a desenvolver aplicativos WebSphere eXtreme Scale, configure um ambiente de desenvolvimento no Eclipse.

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Ponto de verificação de lições

Nesta lição, você aprendeu como criar um aplicativo cliente simples para executar operações da grade de dados.

Lição 3 do Tutorial Introdução: Executando o Aplicativo Cliente de Amostra Introdução

Use as seguintes etapas para iniciar sua primeira grade de dados e executar um cliente para interagir com a grade de dados.

O script `env.sh|bat` é chamado pelos outros scripts para configurar as variáveis necessárias do ambiente. Normalmente não é necessário alterar esse script.

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

Para executar o aplicativo, primeiro você deve iniciar o processo do serviço de catálogo. O serviço de catálogo é o centro de controle da grade de dados. Ele controla os locais dos servidores de contêiner e também o posicionamento dos dados para hospedar servidores de contêiner. Após o serviço de catálogo ser iniciado, é possível iniciar os servidores de contêiner, que armazenam os dados do aplicativo para a grade de dados. Para armazenar diversas cópias dos dados, diversos servidores de contêiner podem ser iniciados. Quando todos os servidores forem iniciados, será possível executar o aplicativo cliente para inserir, atualizar, remover e obter dados da grade de dados.

1. Abra uma janela de sessão do terminal ou linha de comandos.
2. Utilize o seguinte comando para navegar para o diretório `gettingstarted`:
   `cd wxs_install_root/ObjectGrid/gettingstarted`
   Substitua *wxs_install_root* pelo caminho para o diretório-raiz da instalação do eXtreme Scale ou pelo caminho de arquivo raiz de teste do eXtreme Scale extraído *wxs_install_root*.
3. Execute o script a seguir para iniciar um processo de serviço de catálogo no host local:

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

O processo do serviço de catálogo executa na janela do terminal atual.

Também é possível iniciar o serviço de catálogo com o comando **startOgServer**. Execute **startOgServer** a partir do diretório *wxs_install_root*/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 outra janela de sessão de terminal ou de linha de comandos e execute o seguinte comando para iniciar uma instância do servidor de contêiner:

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

   O servidor de contêiner será executado na janela do terminal atual. É possível repetir esta etapa com um nome de servidor diferente se desejar iniciar mais instâncias do servidor de contêiner para suportar a replicação.

   Os servidores de contêiner também podem ser iniciados com o comando **startOgServer**. Execute **startOgServer** a partir do diretório *wxs_install_root*/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 outra janela de sessão de terminal ou linha de comandos para executar comandos do cliente.

   O script `runclient.sh|bat` executa o cliente CRUD simples e inicia a operação especificada. O script `runclient.sh|bat` é executado com os seguintes parâmetros:

   - UNIX Linux `./runclient.sh` *`command value1 value2`*
   - Windows `runclient.bat` *`command value1 value2`*

   Para *command*, use uma das seguintes opções:
   - Especifique `i` para inserir *value2* na grade de dados com a chave *value1*
   - Especifique `u` para atualizar o objeto com chave pelo *value1* para o *value2*
   - Especifique `d` para excluir o objeto com chave pelo *value1*
   - Especifique `g` para recuperar e exibir o objeto com chave pelo *value1*

   a. Inclua dados na grade de dados:
      - UNIX Linux `./runclient.sh i key1 helloWorld`
      - Windows `runclient.bat i key1 helloWorld`

   b. Procurar e exibir o valor:
      - UNIX Linux `./runclient.sh g key1`
      - Windows `runclient.bat g key1`

   c. Atualizar o valor:
      - UNIX Linux `./runclient.sh u key1 goodbyeWorld`

- Windows `runclient.bat u key1 goodbyeWorld`

d. Excluir o valor:

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

Tarefas relacionadas

Iniciando e Parando Servidores Independentes
É possível iniciar e parar servidores de catálogos e de contêiner independentes com os scripts **startOgServer** e **stopOgServer** ou com a API do servidor integrado.

Referências relacionadas

Script**startOgServer**
O script **stopOgServer** inicia servidores de catálogos e de contêiner. É possível utilizar uma variedade de parâmetros quando você inicia seu servidores para ativar o rastreio, especificar números de porta e assim por diante.

Ponto de verificação de lições

Nesta lição, você aprendeu:

- Como iniciar os servidores de catálogos e servidores de contêiner
- Como executar o aplicativo cliente de amostra

Lição 4 do Tutorial de Introdução: Monitore seu Ambiente

É possível usar o utilitário **xscmd** e as ferramentas do console da web para monitorar o ambiente da grade de dados.

Tarefas relacionadas

Visualizando Estatísticas com o Console da Web
É possível monitorar estatísticas e outras informações de desempenho com o console da web.

Monitorando com o Console da Web
Com o console da Web, é possível registrar no gráfico as estatísticas atuais e de histórico. Este console fornece alguns gráficos pré-configurados para visões gerais de alto nível e tem uma página de relatórios customizada que pode ser usada para construir gráficos a partir das estatísticas disponíveis. É possível usar os recursos gráficos no console de monitoramento do WebSphere eXtreme Scale para visualizar o desempenho geral das grades de dados em seu ambiente.

Iniciando e Efetuando Login no Console da Web
Inicie o servidor de console executando o comando **startConsoleServer** e efetuando login no servidor com o ID de usuário e a senha padrão.

Conectando o Console da Web nos Servidores de Catálogos
Para visualizar as estatísticas no console da Web, primeiro você deve se conectar com os servidores de catálogo que deseja monitorar. Etapas adicionais serão necessárias se a segurança estiver ativada nos seus servidores de catálogos.

Monitorando com o Utilitário **xscmd**
O utilitário **xscmd** substitui o utilitário **xsadmin** de amostra como uma ferramenta de monitoramento e de administração totalmente suportadas. Com o utilitário **xscmd**, é possível exibir informações sobre sua topologia do WebSphere eXtreme Scale.

Administrando com o Utilitário **xscmd**
Com o **xscmd**, é possível concluir tarefas administrativas no ambiente como: estabelecer links de replicação multimestre, substituir o quorum e parar grupos de servidores com o comando teardown.

Referências relacionadas

Estatísticas do Console da Web
Dependendo da visualização que você está utilizando no console da Web, é possível visualizar estatísticas diferentes sobre sua configuração. Essas estatísticas incluem a memória usada, as principais grades de dados usadas e o número de entradas de cache.

Script stopOgServer
O script **stopOgServer** para servidores de catálogos e de contêiner.

Monitorando com o Console da Web

Com o console da Web, é possível registrar no gráfico as estatísticas atuais e de histórico. Este console fornece alguns gráficos pré-configurados para visões gerais de alto nível e tem uma página de relatórios customizada que pode ser usada para construir gráficos a partir das estatísticas disponíveis. É possível usar os recursos gráficos no console de monitoramento do WebSphere eXtreme Scale para visualizar o desempenho geral das grades de dados em seu ambiente.

Instale o console da web como um recurso opcional no qual você executa o assistente de instalação.

1. Inicie o servidor do console. O script **startConsoleServer.bat|sh** para iniciar o servidor do console está no diretório *wxs_install_root*/ObjectGrid/bin de sua instalação.
2. Efetue logon no console.
   a. No seu navegador da web, acesse `https://your.console.host:7443`, substituindo `your.console.host` pelo nome do host do servidor na qual o console foi instalado.

b. Efetue logon no console.
   - **ID do usuário:** `admin`
   - **Senha:** `admin`

   A página de boas-vindas do console é exibida.

3. Edite a configuração do console. Clique em **Definições** > **Configuração** para revisar a configuração do console. A configuração do console inclui informações como:
   - Cadeia de rastreio para o cliente do WebSphere eXtreme Scale, como `*=all=disabled`
   - O nome e a senha do Administrador
   - O endereço de e-mail do Administrador
4. Estabeleça e mantenha conexões com servidores de catálogos que você deseja monitorar. Repita as seguintes etapas para incluir cada servidor de catálogos na configuração.
   a. Clique em **Configurações** > **Servidores de Catálogos do eXtreme Scale** .
   b. Inclua um novo servidor de catálogos.

      1) Clique no ícone Incluir ( ) para registrar um servidor de catálogos existente.
      2) Forneça informações, como o nome do host e a porta do listener. Consulte Planejamento para Portas de Rede para obter mais informações sobre a configuração da porta e os padrões.
      3) Clique em **OK**.
      4) Verifique se o servidor de catálogos foi incluído na árvore de navegação.
5. Visualize o status da conexão. O campo **Domínio Atual** indica o nome do domínio do serviço de catálogo que está atualmente sendo usado para exibir informações no console da web. O status da conexão é exibido ao lado do nome do domínio de serviço de catálogo.
6. Visualize as estatísticas para as grades de dados e servidores ou crie um relatório customizado.

Monitorando com o Utilitário xscmd

1. Abra uma janela de linha de comandos. Na linha de comandos, configure as variáveis de ambiente apropriadas.
   a. Configure a variável de ambiente *CLIENT_AUTH_LIB*:
      - Windows `set CLIENT_AUTH_LIB=<path_to_security_JAR_or_classes>`
      - UNIX `set CLIENT_AUTH_LIB=<path_to_security_JAR_or_classes>`
        `export CLIENT_AUTH_LIB`
2. Acesse o diretório *wxs_home*`/bin`.

   `cd wxs_home/bin`
3. Execute vários comandos para exibir informações sobre seu ambiente.
   - Mostre todos os servidores de contêiner on-line para a grade de dados Grid e o conjunto de mapas mapSet:

     `xscmd -c showPlacement -g Grid -ms mapSet`
   - Exiba as informações de roteamento para a grade de dados.

     `xscmd -c routetable -g Grid`
   - Exiba o número de entradas do mapa na grade de dados.

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

Parando os Servidores

Depois de usar o aplicativo cliente e o monitorar a amostra do ambiente de introdução, os servidores poderão ser interrompidos.

- Se os arquivos de script forem usados para iniciar os servidores, utilize <ctrl+c> para parar o processo de serviço de catálogo e os servidores de contêiner em suas respectivas janelas.
- Se o comando **startOgServer** foi usados para iniciar seus servidores, use o comando **stopOgServer** para parar os servidores.

  Pare o servidor de contêiner:

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

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

  Pare o servidor de catálogos:

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

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

Ponto de verificação de lições

Nesta lição, você aprendeu:

- Como iniciar o console da web e conectá-lo ao servidor de catálogos
- Como monitorar a grade de dados e as estatísticas do servidor
- Como parar os servidores

Introdução ao Desenvolvimento de Aplicativos

Para começar a desenvolver aplicativos WebSphere eXtreme Scale, configure um ambiente de desenvolvimento no Eclipse.

Sobre Esta Tarefa

Quando estiver desenvolvendo aplicativos WebSphere eXtreme Scale, as APIs do servidor integrado podem ser usadas para criar e iniciar servidores, instâncias do ObjectGrid e para inserir dados na grade de dados. É possível realizar um teste de unidade de seu aplicativo e da configuração associada diretamente no ambiente Eclipse.

Quando estiver pronto para mover seu aplicativo para um ambiente maior, arquivos XML de configuração poderão ser criados e, em seguida, importados para criar sua implementação.

Procedimento

1. Configure um ambiente de desenvolvimento no Eclipse.

   Ao incluindo os arquivos Java archive (JAR) do WebSphere eXtreme Scale noambiente de desenvolvimento, as APIs já poderão ser usadas para desenvolver seus aplicativos.

Mais informações: “Configurando um Ambiente de Desenvolvimento Independente” na página 125

2. Crie um aplicativo simples que inicia os servidores, cria uma instância do ObjectGrid e insere dados na grade de dados.
   a. Use a API ServerFactory para iniciar e parar servidores.

      Mais informações: Usando a API do Servidor Integrado para Iniciar e Parar Servidores
   b. Use a API do ObjectGridManager para recuperar a instância do ObjectGrid que foi criada.

      Mais informações: “Interagindo com um ObjectGrid Usando a Interface ObjectGridManager” na página 136
   c. Use a API ObjectMap para inserir dados na grade de dados.

      Mais informações: “Objetos de Armazenamento em Cache sem Relacionamentos Envolvidos (API ObjectMap)” na página 155A API ObjectMap é a forma mais simples de acessar e gravar dados na grade de dados. Se seus objetos possuírem relacionamentos complexos, as seguintes APIs poderão ser usadas para ler, gravar e atualizar os dados:
      - “Acessando Dados com Índices (API de Índice)” na página 144
      - “Objetos de Armazenamento em Cache e seus Relacionamentos (API EntityManager)” na página 166
      - “Recuperando Entidades e Objetos (API de Consulta)” na página 202
      - “Acessando Dados com o Serviço de Dados REST” na página 268

      Para obter mais informações sobre como escolher entre diferentes APIs, consulte Capítulo 5, “Desenvolvendo Aplicativos”, na página 131.
3. Execute um teste de unidade de seu aplicativo.

   Também é possível usar o utilitário **xscmd** para exibir informações sobre os servidores em execução, réplicas, e assim por diante. Consulte o Administrando com o Utilitário **xscmd** para obter informações adicionais.
4. Quando estiver satisfeito com seu aplicativo no ambiente de desenvolvimento, crie arquivos de configuração XML e atualize seu aplicativo para usar a configuração. O aplicativo de amostra Introdução fornece exemplos desses arquivos de configuração em um aplicativo Java simples que usa esses arquivos de configuração.

   Mais informações: “Tutorial: Introdução ao WebSphere eXtreme Scale” na página 61
5. Execute seu aplicativo usando os arquivos de configuração XML. O modo com você inicia seus servidores depende do ambiente que estiver sendo usado.

   É possível executar seu aplicativo em um dos seguintes contêineres:
   - Java virtual machine (JVM) independente
   - Tomcat
   - WebSphere Application Server
   - OSGi

Conceitos relacionados

"Objetos de Armazenamento em Cache sem Relacionamentos Envolvidos (API ObjectMap)" na página 155
Os ObjectMaps são como Mapas Java que permitem que os dados sejam armazenados como pares chave-valor. Os ObjectMaps apresentam uma abordagem simples e intuitiva para o aplicativo que armazenará os dados. Um ObjectMap é ideal para o armazenamento em cache de objetos que não tenham nenhum relacionamento envolvido. Se os relacionamentos de objetos estiverem envolvidos, então você deve usar a API EntityManager.

Informações relacionadas

"Lição 2 do Tutorial de Introdução: Criando um Aplicativo Cliente" na página 63
Para inserir, excluir, atualizar e recuperar dados de sua grade de dados, você deverá gravar um aplicativo cliente. A introdução de amostra inclui um aplicativo cliente que pode ser usado para saber mais sobre a criação de seu próprio aplicativo cliente.

Capítulo 4. Planejamento

Antes de instalar o WebSphere eXtreme Scale e de implementar seus aplicativos de grade de dados, você deve decidir sobre sua topologia de armazenamento em cache, concluir o planejamento da capacidade, revisar os requisitos de hardware e de software, as configurações de rede e ajustes, e assim por diante. Também é possível usar a lista de verificação operacional para garantir que seu ambiente esteja pronto para ter um aplicativo implementado.

Para uma discussão das boas práticas que é possível usar ao projetar seus aplicativos WebSphere eXtreme Scale, leia o seguinte artigo em developerWorks: Princípios e boas práticas para construir aplicativos WebSphere eXtreme Scale de alta execução e alta resiliência.

Planejando a Topologia

Com WebSphere eXtreme Scale, sua arquitetura pode utilizar armazenamento em cache de dados em memória local ou armazenamento em cache de dados de cliente/servidor distribuídos. A arquitetura pode ter relacionamentos variados com seus bancos de dados. Também é possível configurar a topologia para estender diversos datacenters.

O WebSphere eXtreme Scale requer infraestrutura adicional mínima para operar. A infraestrutura consiste em scripts para instalar, iniciar e parar um aplicativo Java Platform, Enterprise Edition em um servidor. Os dados em cache são armazenados nos servidores de contêiner e os clientes se conectam remotamente com o servidor.

Ambientes em Memória

Quando implementar em um local, um ambiente em memória, o WebSphere eXtreme Scale é executado em uma única Java Virtual Machine e não é replicado. Para configurar um ambiente local, é possível usar um arquivo XML do ObjectGrid ou as APIs do ObjectGrid.

Ambientes Distribuídos

Quando você implementa em um ambiente distribuído, o WebSphere eXtreme Scale é executado em um conjunto de Java Virtual Machines, aumentando o desempenho, a disponibilidade e a escalabilidade. Com essa configuração, poderá utilizar a replicação de dados e criação de partições. Também é possível incluir servidores adicionais sem restaurar seus servidores eXtreme Scale existentes. Assim como ocorre com um ambiente local, um arquivo XML do ObjectGrid, ou uma configuração programática equivalente, é necessário em um ambiente distribuído. Você também deve fornecer um arquivo XML de política de implementação com detalhes de configuração.

É possível criar implementações simples ou grandes a nível de terabytes, em que milhares de servidores são necessários.

Cache de Memória Local

Em um caso mais simples, o WebSphere eXtreme Scale pode ser utilizado como um cache de grade de dados em memória local (não distribuído). O caso local

pode beneficiar especialmente aplicativos de alta simultaneidade nos quais vários encadeamentos precisam acessar e modificar dados transientes. Os dados mantidos em uma grade de dados local podem ser indexados e recuperados utilizando consultas. As consultas ajudam você a trabalhar com grandes em conjuntos de dados de memória. O suporte fornecido com o Java Virtual Machine (JVM), embora esteja pronto para uso, tem uma estrutura de dados limitada.

A topologia de cache em memória local para WebSphere eXtreme Scale é usado para oferecer acesso transacional e consistente aos dados temporários dentro de uma única Java virtual machine.

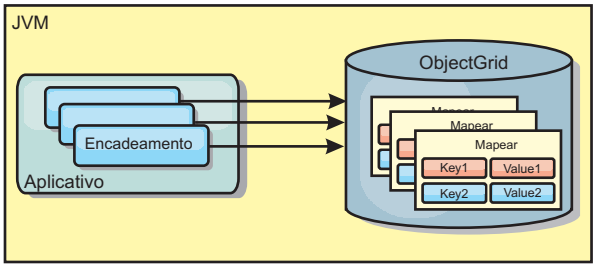


Figura 4. Cenário de Cache em Memória Local

Vantagens

- Configuração simples: Um ObjectGrid pode ser criado programaticamente ou declarativamente com o arquivo XML do descritor de implementação ObjectGrid ou com outras estruturas como Spring.
- Rápido: Cada BackingMap pode ser ajustado de maneira independente para utilização de memória e simultaneidade ideais.
- Ideal para topologias de uma única Java virtual machine com pequenos conjuntos de dados ou para armazenamento em cache de dados frequentemente acessados.
- Transacional. As atualizações BackingMap podem ser agrupadas em uma única unidade de trabalho e podem ser integradas como um último participante nas transações de duas fases como transações JTA (Java Transaction Architecture).

Desvantagens

- Não tolerante a falhas.
- Os dados não são replicados. Caches em memória são melhores para dados de referência somente para leitura.
- Não escalável. A quantidade de memória necessária pelo banco de dados pode ultrapassar a capacidade da Java virtual machine.
- Ocorrem problemas na inclusão de Java virtual machines:
  - Os dados não podem ser facilmente particionados
  - Você deve replicar manualmente o estado entre as Java virtual machines ou cada instância do cache poderá ter diferentes versões dos mesmos dados.
  - A invalidação é custosa.
  - Cada cache deve ser aquecido de maneira independente. O aquecimento é o período de carregamento de um conjunto de dados para que o cache seja preenchido com dados válidos.

Quando Utilizar

A topologia de implementação de cache em memória local deve ser usada somente quando a quantidade de dados a serem armazenados em cache for pequena (puder ser colocada em uma única Java virtual machine) e for relativamente estável. Dados antigos devem ser tolerados com esta abordagem. A utilização de evictors para manter os dados mais frequentemente ou recentemente usados no cache pode ajudar a diminuir o tamanho do cache e a aumentar a relevância dos dados.

Cache Local Replicado pelo Peer

Você deverá assegurar-se de que o cache esteja sincronizado se vários processos com instâncias de cache independentes existirem. Para assegurar-se de que as instâncias de cache estejam sincronizadas, ative um cache replicado por peer com o Java Message Service (JMS).

WebSphere eXtreme Scale inclui dois plug-ins que propagam automaticamente mudanças de transação entre instâncias do ObjectGrid peer. O plug-in JMSObjectGridEventListener propaga automaticamente as mudanças do eXtreme Scale usando JMS.

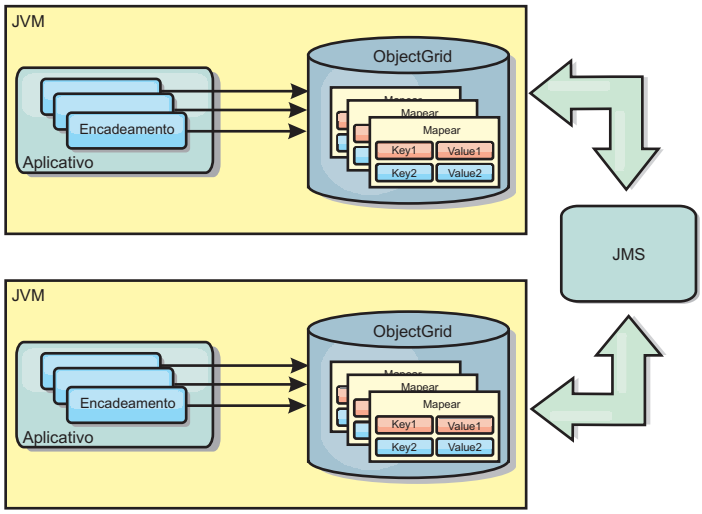


Figura 5. Cache Replicado pelo Peer com Alterações que são Propagadas com JMS

Se você estiver executando um ambiente doWebSphere Application Server, o plug-in TranPropListener também estará disponível. O plug-in TranPropListener usa o gerenciador de alta disponibilidade (HA) para propagar as mudanças em cada instância do cache de peer.

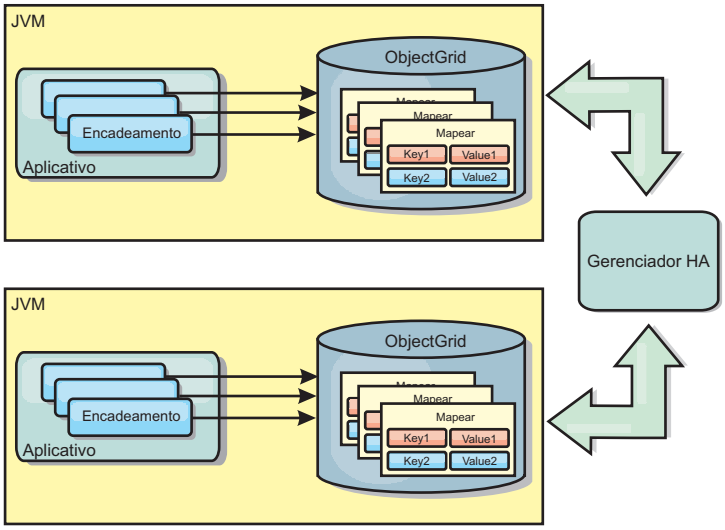


Figura 6. Cache Replicado pelo Peer com Alterações que são Propagadas com o Gerenciador de Alta Disponibilidade

Vantagens

- Os dados são mais válidos porque os dados são atualizados mais frequentemente.
- Com o plug-in TranPropListener, como no ambiente local, o eXtreme Scale pode ser criado programaticamente ou declarativamente com o arquivo XML descritor de implementação do eXtreme Scale ou com outras estruturas como Spring. A integração com o gerenciador de alta disponibilidade é feita automaticamente.
- Cada BackingMap pode ser independentemente ajustado para melhor utilização da memória e concorrência.
- As atualizações BackingMap podem ser agrupadas em uma única unidade de trabalho e podem ser integradas como um último participante nas transações de duas fases como transações JTA (Java Transaction Architecture).
- Ideal para poucas topologias JVM com um conjunto de dados razoavelmente pequeno ou para armazenamento em cache de dados frequentemente acessados.
- As atualizações em cada eXtreme Scale são replicadas para todas as instâncias do eXtreme Scale do peer. As alterações são consistentes desde que uma assinatura durável seja utilizada.

Desvantagens

- A configuração e a manutenção para o JMSObjectGridEventListener podem ser complexas. O eXtreme Scale pode ser criado programaticamente ou declarativamente com o arquivo XML descritor de implementação do eXtreme Scale ou com outras estruturas tais como Spring.
- Não escalável: A quantidade de memória necessária para que o banco de dados possa dominar a JVM.
- Funciona inadequadamente ao incluir Java Virtual Machines:
  - Os dados não podem ser facilmente particionados
  - A invalidação é custosa.
  - Cada cache deve ser aquecido de maneira independente

Quando Utilizar

Use a topologia de implementação apenas quando a quantia de dados a ser armazenada em cache for pequena, podendo ajustar-se a uma única JVM e se for relativamente estável.

Cache Integrado

As grades do WebSphere eXtreme Scale podem ser executadas nos processos existentes como servidores eXtreme Scale integrados ou podem ser gerenciadas como processos externos.

As grades integradas são úteis quando você está executando em um servidor de aplicativos, como o WebSphere Application Server. É possível iniciar servidores eXtreme Scale que não são integrados usando scripts da linha de comandos e executar em um processo Java.

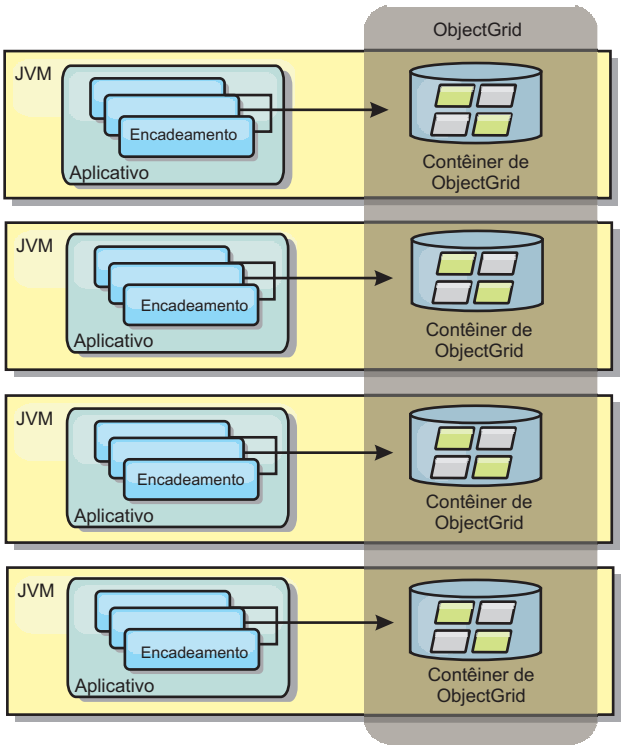


Figura 7. Cache Integrado

Vantagens

- Administração simplificada já que há menos processos para gerenciar.
- Implementação do aplicativo simplificada, já que a grade usa o carregador de classe do aplicativo cliente.
- Suporta particionamento e alta disponibilidade.

Desvantagens

- Aumento da área de cobertura da memória no processo do cliente já que todos os dados são colocados no processo.
- Aumento da utilização da CPU para atender pedidos de clientes.

- Mais difícil para manipular atualizações de aplicativo, pois os clientes estão usando os mesmos arquivos archive de Java do aplicativo que os servidores.
- Menos flexível. A escala dos clientes e servidores de grade não pode aumentar na mesma proporção. Quando os servidores são externamente definidos, é possível ter mais flexibilidade no gerenciamento do número de processos.

Quando Utilizar

Utilize grades integradas quando há grande quantidade de memória livre no processo do cliente para dados da grade e dados de failover potenciais.

Para obter mais informações, consulte o tópico sobre como ativar o mecanismo de invalidação do cliente no *Guia de Administração*.

Cache Distribuído

O WebSphere eXtreme Scale é mais frequentemente usado como um cache compartilhado, para fornecer acesso transacional a dados para múltiplos componentes onde, caso contrário, um banco de dados tradicional seria usado. O cache compartilhado elimina a necessidade de configurar um banco de dados.

Coerência do Cache

O cache é coerente porque todos os clientes veem os mesmos dados no cache. Cada pedaço de dado é armazenado em exatamente um servidor no cache, evitando cópias de registros desperdiçadas que poderiam potencialmente conter diferentes versões dos dados. Um cache coerente também pode conter mais dados, à medida que mais servidores são incluídos na grade de dados, e escalado linearmente à medida que a grade cresce em tamanho. Como os clientes acessam dados a partir desta grade de dados com chamadas de processo remotas, ela também é conhecida como um cache remoto, ou cache distante. Através do particionamento de dados, cada processo contém um subconjunto exclusivo do conjunto de dados total. As grades de dados maiores podem conter mais dados e atender mais solicitações para esses dados. A coerência também elimina a necessidade de enviar dados de invalidação ao redor da grade de dados porque não há dados antigos. O cache coerente retém somente a cópia mais recente de cada pedaço de dados.

Se você estiver executando um ambiente doWebSphere Application Server, o plug-in TranPropListener também estará disponível. O plug-in TranPropListener usa o componente de alta disponibilidade (Gerenciador HA) do WebSphere Application Server para propagar as alterações para cada instância de cache ObjectGrid de peer.

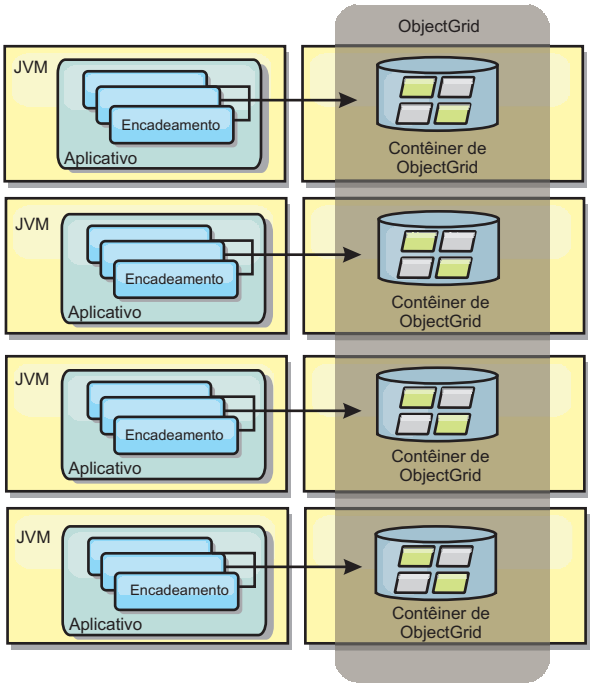


Figura 8. Cache Distribuído

Cache Local

Opcionalmente, os clientes têm um cache local, sequencial quando o eXtreme Scale é usado em uma topologia distribuída. Este cache opcional é chamado de cache local, um ObjectGrid independente em cada cliente, servindo como um cache para o cache remoto, do lado do cliente. O cache local é ativado por padrão quando o bloqueio é configurado como otimista ou nenhum e não pode ser utilizado quando é configurado como pessimista.

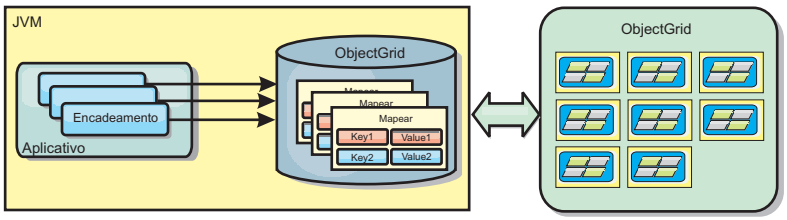


Figura 9. Cache Local

Um cache local é muito rápido porque fornece acesso em memória a um subconjunto do conjunto inteiro de dados em cache que é armazenado remotamente nos servidores do eXtreme Scale. O cache local não é particionado e contém dados de qualquer uma das partições eXtreme Scale remotas. O WebSphere eXtreme Scale pode ter até três camadas de cache, como a seguir.

1. O cache da camada da transação contém todas as alterações para uma única transação. O cache da transação contém uma cópia de trabalho dos dados até que a transação seja confirmada. Quando uma transação do cliente solicita dados de um ObjectMap, a transação é verificada primeiro

2. O cache local na camada do cliente contém um subconjunto de dados da camada do servidor. Quando a camada da transação não possui os dados, eles são buscados em uma camada do cliente, se disponíveis, e inseridos no cache da transação.
3. A grade de dados na camada do servidor contém a maioria dos dados e é compartilhada entre todos os clientes. A camada do servidor pode ser particionada, o que permite que uma grande quantidade de dados seja armazenada em cache. Quando o cache local do cliente não possui os dados, eles são buscados na camada do servidor e inseridos no cache cliente. A camada do servidor também pode ter um plug-in do Utilitário de Carga. Quando a grade não tem os dados necessários, o Utilitário de Carga é chamado e os dados resultantes são inseridos do armazém de dados de backend para a grade.

Para desativar o cache local, configure o atributo numberOfBuckets como 0 no arquivo descritor do ObjectGrid de substituição do cliente. Consulte o tópico sobre bloqueio de entrada de mapa para obter detalhes sobre estratégias de bloqueio do eXtreme Scale. O cache local também pode ser configurado para ter uma política de despejo configurada e diferentes plug-ins usando uma configuração do descritor do eXtreme Scale de substituição.

Vantagem

- Tempo de resposta rápido porque todos os acessos aos dados é local. Procurar pelos dados no cache próximo primeiro economiza acesso à grade de servidores, o que torna até mesmo os dados remotos acessíveis localmente.

Desvantagens

- Aumenta a duração dos dados antigos porque o cache próximo em cada camada pode estar fora de sincronização com os dados atuais na grade de dados.
- Depende de um evictor para invalidar dados a fim de evitar a falta de memória.

Quando Utilizar

Utilize quando o tempo de resposta for importante e dados antigos puderem ser tolerados.

Integração com o Banco de Dados: Armazenamento em Cache Write-behind, Sequencial e Lateral

O WebSphere eXtreme Scale é usado para colocar um banco de dados tradicional na frente e eliminar a atividade de leitura que normalmente é armazenada no banco de dados. Um cache coerente pode ser utilizado com um aplicativo direta ou indiretamente, utilizando um mapeador relacional de objeto. O cache coerente pode transferir o banco de dados ou o backend a partir das leituras. Em um cenário levemente mais complexo, tal como o acesso transacional a um conjunto de dados no qual apenas parte dos dados requer garantias de persistência tradicional, a filtragem pode ser utilizada para transferir até mesmo transações de gravação.

É possível configurar o WebSphere eXtreme Scale para funcionar como um espaço de processamento de banco de dados em memória altamente flexível. Entretanto, o WebSphere eXtreme Scale não é um object relational mapper (ORM). Ele não reconhece de onde vieram os dados na grade de dados. Um aplicativo ou um ORM pode colocar dados em um servidor eXtreme Scale. É responsabilidade da origem dos dados certificar-se de que eles permaneçam consistentes com o banco de dados no qual os dados se originaram. Isto significa que o eXtreme Scale não

pode invalidar dados que são extraídos de um banco de dados automaticamente. O aplicativo ou mapeador deve fornecer esta função e gerenciar os dados armazenados no eXtreme Scale.

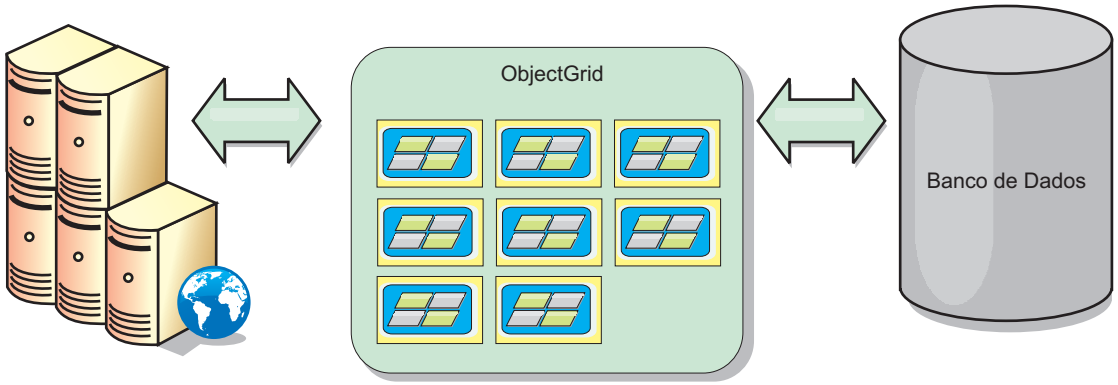


Figura 10. ObjectGrid como um Buffer de Banco de Dados

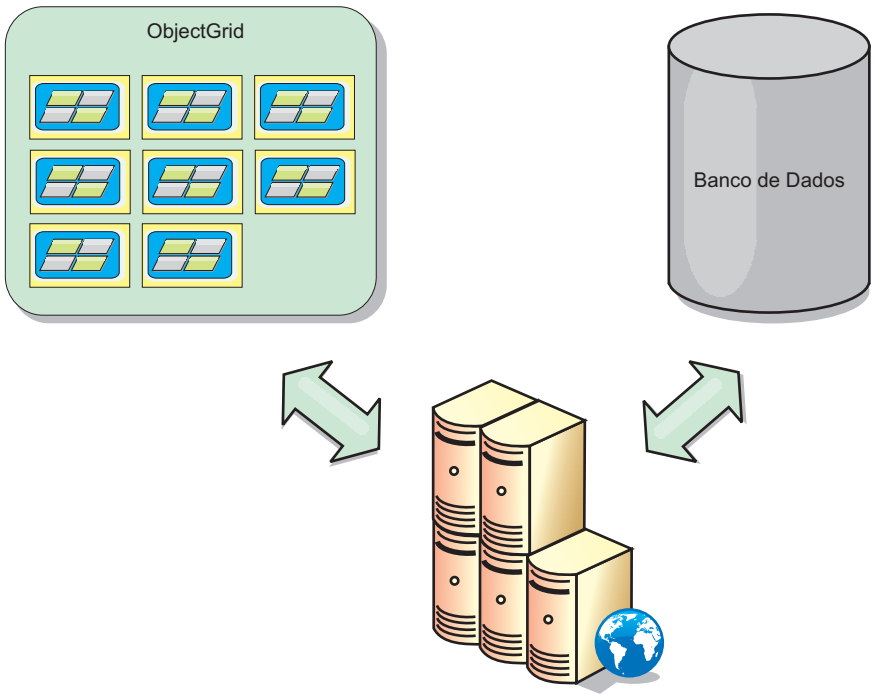


Figura 11. ObjectGrid como um Cache Secundário

Cache Disperso e Completo

O WebSphere eXtreme Scale pode ser utilizado como um cache disperso ou um cache completo. Um cache disperso mantém apenas um subconjunto do total de dados, enquanto que um cache completo mantém todos os dados. Ele também pode ser preenchido gradualmente, conforme os dados são necessários. Os caches dispersos normalmente são acessados usando chaves (ao invés de índices ou consultas) porque os dados estão disponíveis apenas parcialmente.

Cache Disperso

Quando uma chave não está presente em um cache disperso, ou os dados não estão disponíveis e uma falta de cache ocorre, a próxima camada é chamada. Os

dados são buscados, a partir de um banco de dados, por exemplo, e inseridos na camada de cache da grade de dados. Se estiver usando uma consulta ou um índice, apenas os valores atualmente carregados serão acessados e as solicitações não serão encaminhadas para as outras camadas.

Cache Completo

Um cache completo contém todos os dados necessários e pode ser acessado usando atributos não-chaves com índices ou consultas. Um cache completo é pré-carregado com dados a partir do banco de dados antes que o aplicativo tente acessar os dados. Um cache completo pode funcionar como uma substituição do banco de dados após os dados serem carregados. Como todos os dados estão disponíveis, as consultas e índices podem ser usados para localizar e agregar dados.

Cache Secundário

Quando o WebSphere eXtreme Scale é usado como um cache secundário, o backend é usado com a grade de dados.

Cache Secundário

É possível configurar o produto como um cache secundário para a camada de acesso a dados de um aplicativo. Neste cenário, o WebSphere eXtreme Scale é utilizado para armazenar temporariamente objetos que normalmente poderiam ser recuperados de um banco de dados de backend. Aplicativos verificam se a grade de dados contém os dados. Se os dados estiverem na grade de dados, eles serão retornados para o responsável pela chamada. Se os dados não existirem, eles serão recuperados a partir do banco de dados de backend. Os dados são então inseridos na grade de dados para que a próxima solicitação possa usar a cópia em cache. O diagrama a seguir ilustra como o WebSphere eXtreme Scale pode ser usado como um cache secundário com uma camada de acesso a dados arbitrários, como OpenJPA ou Hibernate.

Plug-ins do cache secundário para Hibernate e OpenJPA

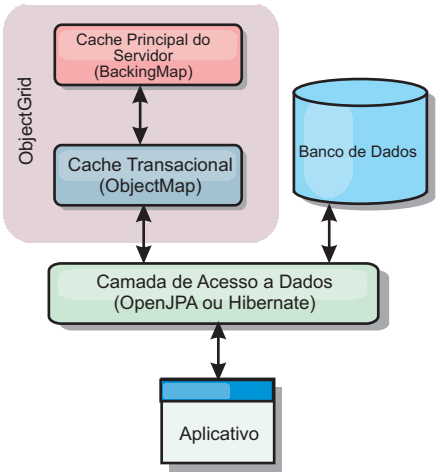


Figura 12. Cache Secundário

Os plug-ins de cache para ambos OpenJPA e Hibernate são incluídos no WebSphere eXtreme Scale, o que permite usar o produto como um cache secundário automático. Usar o WebSphere eXtreme Scale como um provedor de cache aumenta o desempenho ao ler e enfileirar dados e reduz a carga para o

banco de dados. Existem vantagens que o WebSphere eXtreme Scale tem sobre as implementações de cache integrado porque o cache é automaticamente replicado entre todos os processos. Quando um cliente armazena em cache um valor, todos os outros clientes podem usar o valor em cache.

Cache Sequencial

É possível configurar em cache sequencial para um backend de banco de dados ou como um cache secundário para um banco de dados. O armazenamento em cache sequencial utiliza o eXtreme Scale como o meio principal de interação com os dados. Quando o eXtreme Scale é usado como um cache sequencial, o aplicativo interage com o backend usando um plug-in Loader.

Cache Sequencial

Quando usado como um cache sequencial, o WebSphere eXtreme Scale interage com o backend usando um plug-in Loader. Este cenário pode simplificar o acesso a dados porque os aplicativos podem acessar as APIs do eXtreme Scale diretamente. Vários cenários de armazenamento em cache diferentes são suportados no eXtreme Scale para garantir que os dados no cache e os dados no backend sejam sincronizados. O diagrama a seguir ilustra como um cache sequencial interage com o aplicativo e o back end.

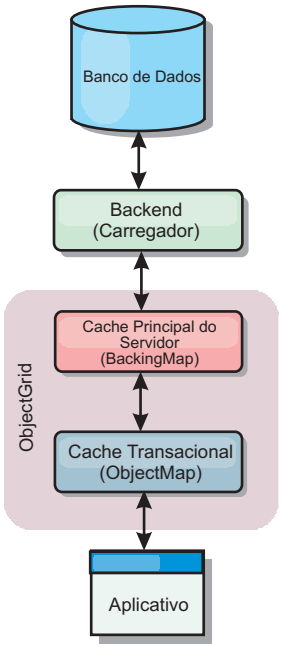


Figura 13. Cache Sequencial

A opção de armazenamento em cache em linha simplifica o acesso aos dados pois ela permite que os aplicativos acessem diretamente as APIs do eXtreme Scale. O WebSphere eXtreme Scale suporta diversos cenários de armazenamento em cache em linha, como os seguintes:

- Read-through
- Write-through
- Write-behind

Cenário de Armazenamento em Cache Read-through

Um cache read-through é um cache disperso que lentamente carrega entradas de dados por chave à medida que elas são solicitadas. Isto é feito sem exigir que o responsável pela chamada saiba quais entradas estão preenchidas. Se os dados não puderem ser localizados no cache do eXtreme Scale, o eXtreme Scale irá recuperar os dados ausentes do plug-in do utilitário de carga, que carrega os dados do banco de dados backend e insere os dados no cache. Pedidos subsequentes para a mesma chave de dados serão localizados no cache até que ele possa ser removido, invalidado ou despejado.

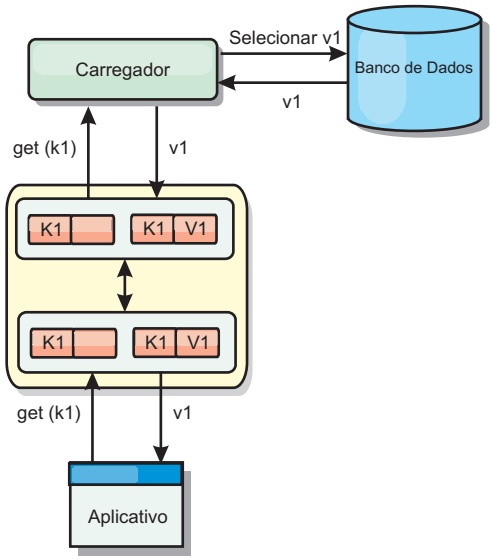


Figura 14. Armazenamento em Cache Read-through

Cenário de Armazenamento em Cache Write-through

Em um cache write-through, cada gravação no cache é gravada de maneira síncrona no banco de dados utilizando o Utilitário de Carga. Este método fornece consistência com o backend, mas diminui o desempenho de gravação pois a operação do banco de dados é síncrona. Como o cache e o banco de dados são ambos atualizados, as leituras subsequentes para os mesmos dados serão localizadas no cache, evitando a chamada do banco de dados. Um cache write-through sempre é utilizado em conjunto com um cache read-through.

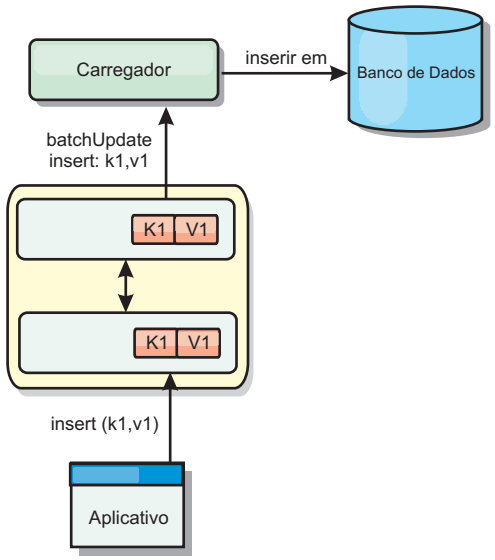


Figura 15. Armazenamento em Cache Write-through

Cenário de Armazenamento em Cache Write-behind

A sincronização do banco de dados pode ser aprimorada pela gravação de alterações de maneira assíncrona. Isto é conhecido como um cache write-behind ou write-back. Alterações que normalmente poderiam ser gravadas de maneira síncrona no utilitário de carga são, ao invés disso, armazenadas em buffer no eXtreme Scale e gravadas no banco de dados utilizando um encadeamento secundário. O desempenho de gravação é significativamente aumentado pois a operação do banco de dados é removida da transação do cliente e as gravações do banco de dados podem ser compactadas.

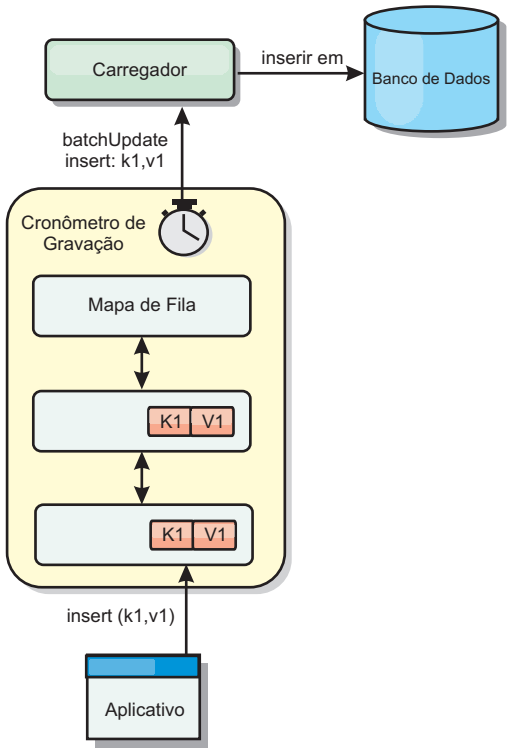


Figura 16. Armazenamento em Cache Write-behind

Armazenamento em Cache Write-behind

É possível utilizar armazenamento em cache write-behind para reduzir o gasto adicional que ocorre durante a atualização de um banco de dados que você está utilizando como back end.

Visão Geral do Armazenamento em Cache Write-Behind

O armazenamento em cache write-behind enfileira assincronamente as atualizações no plug-in do Utilitário de Carga. É possível melhorar o desempenho desconectando atualizações, inserções e remoções para um mapa, a sobrecarga de atualização do banco de dados de backend. A atualização assíncrona é executada após um atraso baseado em tempo (por exemplo, cinco minutos) ou um atraso baseado em entradas (1000 entradas).

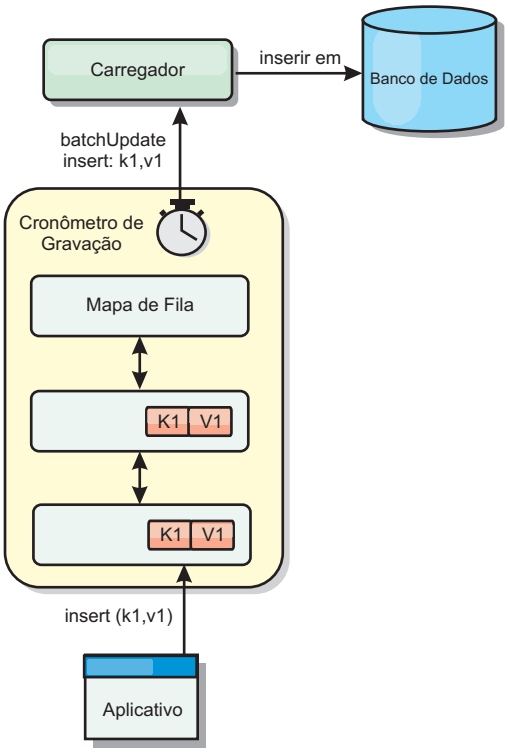


Figura 17. Armazenamento em Cache Write-behind

A configuração write-behind em um BackingMap cria um encadeamento entre o utilitário de carga e o mapa. O utilitário de carga então delega pedidos de dados através do encadeamento de acordo com as definições da configuração no método BackingMap.setWriteBehind. Quando uma transação do eXtreme Scale insere, atualiza ou remove uma entrada de um mapa, um objeto LogElement é criado para cada um destes registros. Estes elementos são enviados para o utilitário de carga write-behind e enfileirados em um ObjectMap especial denominado mapa de fila. Cada mapa de apoio com a configuração write-behind ativada possui seus próprios mapas de fila. Um encadeamento write-behind remove periodicamente os dados enfileirados dos mapas de fila e executa o push deles para o utilitário de carga de backend real.

O utilitário de carga write-behind enviará apenas os tipos insert, update e delete dos objetos LogElement para o utilitário de carga real. Todos os outros tipos de objetos LogElement, por exemplo, o tipo EVICT, são ignorados.

O suporte write-behind *é* uma extensão do plug-in do Carregador, que você usa para integrar o eXtreme Scale ao banco de dados. Por exemplo, consulte as informações do Configurando Utilitários de Carga do JPA sobre como configurar um carregador JPA.

Benefícios

Ativar o suporte write-behind possui os seguintes benefícios:

- **Isolamento de falha de backend:** O armazenamento em cache write-behind fornece uma camada de isolamento das falhas de backend. Quando o banco de dados de backend falha, as atualizações são enfileiradas no mapa de fila. Os

aplicativos podem continuar a conduzir transações para o eXtreme Scale. Quando o backend se recupera, os dados no mapa de fila são enviados para o backend.

- **Carga de backend reduzida:** O utilitário de carga write-behind mescla as atualizações em uma base de chave, portanto, apenas uma atualização mesclada por chave existe no mapa de fila. Esta mesclagem diminui o número de atualizações no backend.
- **Desempenho de transação aprimorado:** Tempos de transação do eXtreme Scale individuais são reduzidos porque a transação não precisa aguardar até que os dados sejam sincronizados com o backend.

Considerações de Design do Aplicativo

Ativar o suporte write-behind é simples, mas o design de um aplicativo para trabalhar com o suporte write-behind precisa de consideração cuidadosa. Sem o suporte de write-behind, a transação de ObjectGrid engloba a transação de backend. A transação do ObjectGrid inicia antes da transação de backend iniciar e termina após a transação de backend terminar.

Com suporte write-behind ativado, a transação do ObjectGrid é concluída antes que a transação de backend inicie. A transação do ObjectGrid e a transação de backend não estão acopladas.

Limitadores de Integridade Referencial

Cada mapa de apoio que é configurado com suporte write-behind possui seu próprio encadeamento write-behind para enviar os dados para o backend. Portanto, os dados que são atualizados em diferentes mapas em uma transação do ObjectGrid são atualizadas no backend em diferentes transações de backend. Por exemplo, a transação T1 atualiza a chave key1 no mapa Map1 e a chave key2 no mapa Map2. A atualização da key1 para o mapa Map1 é atualizada no backend em uma transação de backend e a key2 atualizada para o mapa Map2 é atualizada no backend em outra transação de backend por encadeamentos write-behind diferentes. Se os dados armazenados no Map1 e Map2 possuírem relações, tais como limitadores de chave estrangeira no backend, as atualizações podem falhar.

Ao projetar os limitadores de integridade referencial em seu banco de dados de backend, certifique-se de que atualizações fora de ordem sejam permitidas.

Comportamento do Bloqueio de Mapa de Fila

Outra grande diferença de comportamento da transação é o comportamento do bloqueio. O ObjectGrid suporta três diferentes estratégias de bloqueio: PESSIMISTIC, OPTIMISITIC e NONE. Os mapas de fila write-behind utilizam a estratégia de bloqueio pessimista não importando qual estratégia de bloqueio está configurada para seu mapa de apoio. Existem dois diferentes tipos de operações que adquirem um bloqueio no mapa de fila:

- Quando uma transação do ObjectGrid é confirmada ou um flush (flush de mapa ou flush de sessão) acontece, a transação lê a chave no mapa de fila e coloca um bloqueio S na chave.
- Quando uma transação do ObjectGrid é confirmada, a transação tenta atualizar o bloqueio S para o bloqueio X na chave.

Devido a este comportamento do mapa de fila extra, é possível visualizar algumas diferenças de comportamento de bloqueio.

- Se o mapa do usuário for configurado como a estratégia de bloqueio PESSIMISTIC, não há muita diferença no comportamento de bloqueio. Sempre que um flush ou commit é chamado, um bloqueio S é colocado na mesma chave no mapa de fila. Durante o momento do commit, um bloqueio X não é adquirido apenas para a chave no mapa do usuário, ele também é adquirido para a chave no mapa de fila.
- Se o mapa do usuário for configurado com a estratégia de bloqueio OPTIMISTIC ou NONE, a transação do usuário seguirá o padrão de estratégia de bloqueio PESSIMISTIC. Sempre que um flush ou commit é chamado, um bloqueio S é adquirido para a mesma chave no mapa de fila. Durante o momento do commit, um bloqueio X é adquirido para a chave no mapa de fila utilizando a mesma transação.

Novas Tentativas de Transações do Utilitário de Carga

O ObjectGrid não suporta transações 2-phase ou XA. O encadeamento write-behind remove registros do mapa de fila e atualiza os registros no backend. Se o servidor falhar no meio da transação, algumas atualizações de backend podem ser perdidas.

O utilitário de carga write-behind automaticamente tentará gravar novamente transações falhas e enviará uma LogSequence duvidosa para o backend para evitar a perda de dados. Esta ação requer que o utilitário de carga seja idempotente, o que significa que o Loader.batchUpdate(TxId, LogSequence) é chamado duas vezes com o mesmo valor, ele fornece o mesmo resultado como se tivesse sido aplicado uma vez. As implementações do utilitário de carga devem implementar a interface RetryableLoader para ativar este recurso. Consulte a documentação da API para obter mais detalhes.

Falha do Utilitário de Carga

O plug-in do utilitário de carga pode falhar quando não consegue se comunicar com o back end do banco de dados. Isto pode acontecer se o servidor de banco de dados ou a conexão de rede estiver inativa. O utilitário de carga write-behind irá enfileirar as atualizações e tentará executar o push das alterações de dados para o utilitário de carga periodicamente. O utilitário de carga deve notificar o tempo de execução do ObjectGrid que há um problema de conectividade do banco de dados lançando uma exceção LoaderNotAvailableException.

Portanto, a implementação do Utilitário de Carga deve poder distinguir uma falha de dados ou uma falha de utilitário de carga físico. A falha de dados deve ser lançada ou relançada como uma LoaderException ou uma OptimisticCollisionException, mas uma falha de utilitário de carga físico deve ser lançada ou relançada como uma LoaderNotAvailableException. O ObjectGrid manipula estas exceções de maneira diferente:

- Se uma LoaderException for capturada pelo utilitário de carga write-behind, o utilitário de carga write-behind a considerará falha devido a alguma falha de dados, tal como uma falha de chave duplicada. O utilitário de carga write-behind irá remover a atualização do lote e tentará atualizar um registro em um momento para isolar a falha de dados. Se uma {{LoaderException}} for capturada durante uma atualização de registro, um registro de atualização falho é criado e registrado no mapa de atualização falho.
- Se uma LoaderNotAvailableException for capturada pelo utilitário de carga write-behind, o utilitário de carga write-behind a considerará falha porque não pode se conectar ao final do banco de dados, por exemplo, porque o backend do

banco de dados estiver inativo, uma conexão com o banco de dados não estiver disponível ou a rede estiver inativa. O utilitário de carga write-behind aguardará por 15 segundo e, em seguida, tentará novamente executar uma atualização de lote no banco de dados.

O erro comum é lançar uma LoaderException enquanto uma LoaderNotAvailableException deve ser lançada. Todos os registros enfileirados no utilitário de carga write-behind se tornarão atualizações de registro falhas, o que frustra o propósito do isolamento de falha do backend.

Considerações sobre Desempenho

O suporte ao armazenamento em cache write-behind aumenta o tempo de resposta removendo a atualização do utilitário de carga da transação. Ele também aumenta o rendimento do banco de dados porque as atualizações de banco de dados são combinadas. É importante compreender o gasto adicional introduzido pelo encadeamento write-behind, que executa o pull dos dados da mapa de fila e executa o push para o utilitário de carga.

A contagem máxima de atualização ou o tempo máximo de atualização necessário a ser ajustado com base nos padrões de uso e no ambiente esperados. Se o valor da contagem máxima de atualização ou o tempo máximo de atualização for muito pequeno, o gasto adicional do encadeamento write-behind pode exceder os benefícios. Configurar um valor maior para estes dois parâmetros também pode aumentar o uso da memória para enfileirar os dados e aumentar o tempo de envelhecimento dos registros do banco de dados.

Para obter um melhor desempenho, ajuste os parâmetros write-behind com base nos seguintes fatores:

- Proporção de transações de leitura e gravação
- Mesma frequência de atualização de registro
- Latência de atualização de banco de dados.

Referências relacionadas

"Exemplo: Gravando uma Classe Dumper no Modo write-behind" na página 363
Essa amostra de código de origem mostra como gravar um watcher (dumper) para manipular atualizações write-behind com falhas.

Utilitários de Carga

Com um plug-in Carregador, uma grade de dados pode se comportar como um cache de memória para dados que normalmente são mantidos em um armazenamento persistente no mesmo sistema ou em outro sistema. Geralmente, um banco de dados ou sistema de arquivos é utilizado como o armazenamento persistente. Uma JVM (Java Virtual Machine) também pode ser usada como a origem de dados, permitindo que caches baseados em hub seja construído usando o eXtreme Scale. Um utilitário de carga possui a lógica para leitura e gravação de dados para um armazenamento persistente e a partir dele.

Visão Geral

Os utilitários de carga são plug-ins de mapa de apoio que são chamados quando são feitas alterações no mapa de apoio ou quando o mapa de apoio não pode atender a um pedido de dados (um erro de cache). O utilitário de carga é chamado quando o cache não pode satisfazer uma solicitação para uma chave, fornecendo capacidade read-through e lazy-population do cache. Um utilitário de carga também permite atualizações no banco de dados quando os valores do cache

mudam. Todas as mudanças dentro de uma transação são agrupadas para permitir que o número de interações do banco de dados seja minimizado. Um plug-in TransactionCallback é usado em conjunto com o utilitário de carga para acionar a demarcação da transação backend. O uso deste plug-in é importante quando múltiplos mapas são incluídos em uma única transação ou quando os dados da transação forem enviados para o cache sem consolidação.

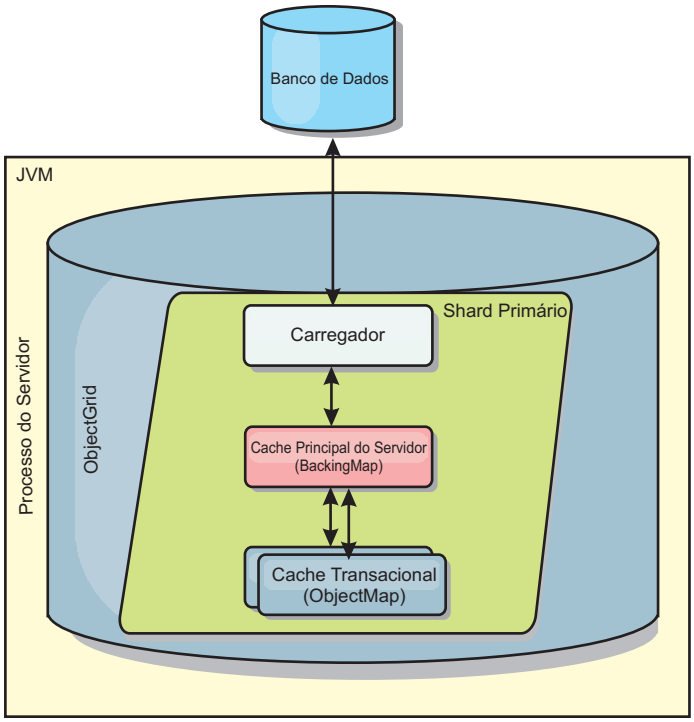


Figura 18. Utilitário de Carga

O utilitário de carga também pode usar atualizações super qualificadas para evitar manter bloqueios do banco de dados. Armazenar um atributo de versão no valor do cache, permite ao utilitário de carga ver a imagem antes e depois do valor quando ele for atualizado no cache. Este valor pode assim ser usado ao atualizar o banco de dados ou backend para verificar se os dados foram atualizados. Um Loader também pode ser configurado para pré-carregar a grade de dados quando for iniciado. Quando particionado, uma instância do utilitário de carga é associada a cada partição. Se o Mapa "Company" tiver dez partições, haverá dez instâncias do utilitário de carga, uma por partição primária. Quando shard primário para o Mapa é ativado, o método preloadMap para o utilitário de carga é chamado síncrona ou assincronamente, o qual permite o carregamento da partição do mapa com dados a partir do backend ocorra automaticamente. Quando chamado sincronamente, todas as transações do cliente são bloqueadas, evitando o acesso inconsistente à grade de dados. Como alternativa, um pré-utilitário de cliente pode ser usado para carregar a grade de dados inteira.

Dois utilitários de carga integrados podem simplificar muito a integração com back ends de banco de dados relacional. Os utilitários de carga JPA utilizam os recursos ORM (Object-Relational Mapping) de ambas as implementações OpenJPA e Hibernate da especificação JPA (Java Persistence API). Consulte “Carregadores JPA” na página 397 para obter mais informações.

Se estiver usando carregadores em uma configuração de diversos datacenters, você deverá considerar como dados de revisão e a consistência de cache são mantidos entre as grades de dados. Para obter informações adicionais, consulte “Considerações Sobre o Carregador em uma Topologia Multimestre” na página 106.

Configuração do Utilitário de Carga

Para incluir um Utilitário de Carga na configuração do BackingMap, é possível utilizar a configuração programática ou a configuração do XML. Um utilitário de carga possui o seguinte relacionamento com um mapa de apoio.

- Um mapa de apoio pode ter apenas um utilitário de carga.
- Um mapa de apoio de cliente (cache local) não pode ter um utilitário de carga.
- Uma definição de utilitário de carga pode ser aplicado a múltiplos mapas de apoio, mas cada mapa de apoio possui sua própria instância do utilitário de carga.

Referências relacionadas

“Considerações sobre a Programação do Utilitário de Carga do JPA” na página 366
Um Utilitário de Carga do Java Persistence API (JPA) é uma implementação do plug-in do utilitário de carga que usa o JPA para interagir com o banco de dados. Use as seguintes considerações ao desenvolver um aplicativo que usa um utilitário de carga do JPA.

Pré-carregamento de Dados e Aquecimento

Em vários cenários que incorporam o uso de um carregador, é possível preparar sua grade de dados ao pré-carregá-lo com dados.

Quando usado como um cache completo, a grade de dados deve manter todos os dados e deve ser carregada antes que quaisquer clientes possam se conectar a ele. Quando estiver usando um cache esparso, é possível efetuar um warm-up do cache com dados para que os clientes possam ter acesso imediato aos dados quando eles se conectarem.

Existem duas abordagens para o pré-carregamento de dados na grade de dados: Usando um plug-in do Carregador ou usando um carregador do cliente, conforme descrito nas seguintes seções.

Plug-in do Utilitário de Carga

O plug-in do carregador é associado a cada mapa e é responsável pela sincronização de um único shard de partição primário com o banco de dados. O método preloadMap do plug-in do utilitário de carga é chamado automaticamente quando um shard é ativado. Por exemplo, se você tiver 100 partições, existem 100 instâncias do carregador, cada um carregando os dados para sua partição. Quando executado de modo síncrono, todos os clientes serão bloqueados até que o pré-carregamento seja concluído.

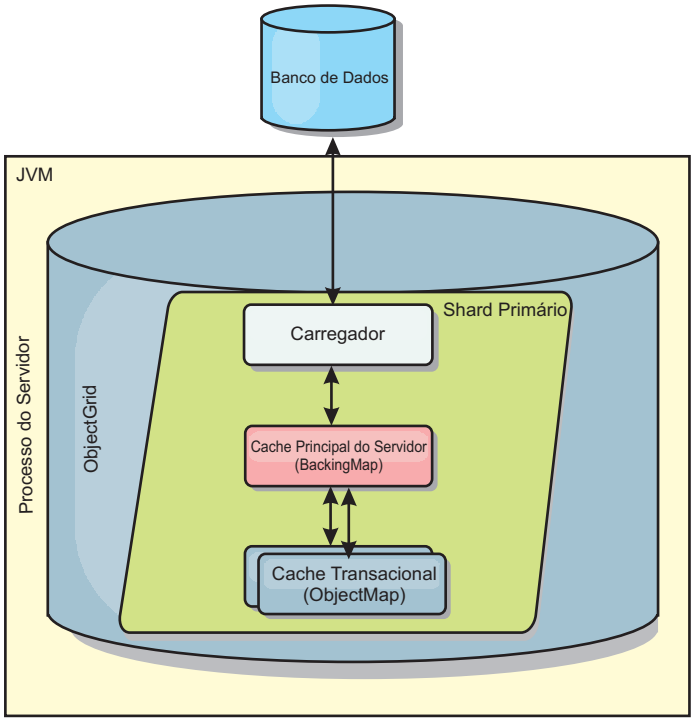


Figura 19. Plug-in do Utilitário de Carga

Utilitário de Carga do Cliente

Um utilitário de carga do cliente é um padrão para uso de um ou mais clientes para carregar a grade com dados. O uso de múltiplos clientes para carregamento de dados da grade pode ser efetivo quando o esquema de partições não está armazenado no banco de dados. É possível chamar os carregadores de cliente manual ou automaticamente quando a grade de dados é iniciada. Os carregadores do cliente podem usar, opcionalmente, o StateManager para configurar o estado da grade de dados no modo de pré-carregamento, para que os clientes não possam acessar a grade enquanto ela estiver pré-carregando os dados. WebSphere eXtreme Scale inclui um carregador baseado em Java Persistence API (JPA)pode ser usado para carregar automaticamente a grade de dados com os provedores JPA OpenJPA ou Hibernate. Para obter mais informações sobre os provedores de cache, consulte Plug-in do Cache JPA Nível 2 (L2).

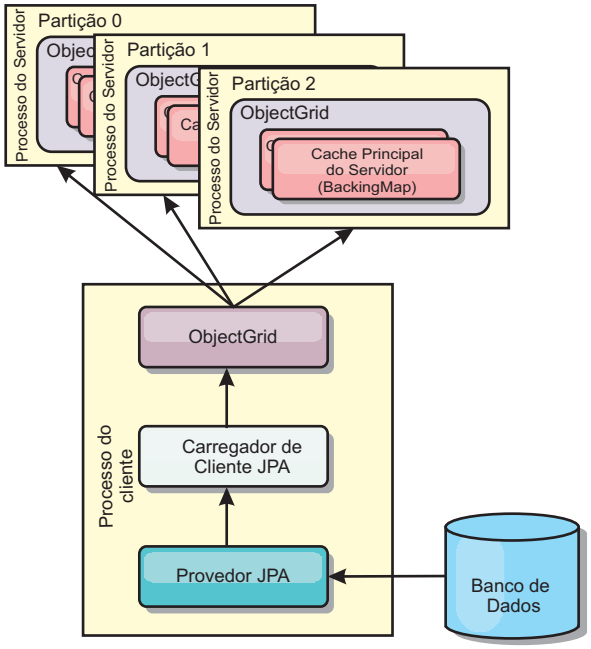


Figura 20. Utilitário de Carga do Cliente

Técnicas de Sincronização de Banco de Dados

Quando o WebSphere eXtreme Scale é utilizado como um cache, os aplicativos devem ser criados para tolerar dados antigos se o banco de dados puder ser atualizado de maneira independente de uma transação do eXtreme Scale. Para atuar como um espaço de processamento de banco de dados de memória sincronizado, o eXtreme Scale fornece várias maneiras de manter o cache atualizado.

Técnicas de Sincronização de Banco de Dados

Atualização periódica

O cache pode ser automaticamente invalidado ou atualizado automaticamente usando o atualizador de banco de dados baseado em tempo JPA (Java Persistence API).O atualizador periodicamente consulta o banco de dados usando um provedor JPA para todas as atualizações ou inserções que ocorreram desde a atualização anterior. Quaisquer alterações identificadas são automaticamente invalidadas ou atualizadas quando utilizadas com um cache disperso. Se utilizadas com um cache completo, as entradas podem ser descobertas e inseridas no cache. As entradas nunca são removidas do cache.

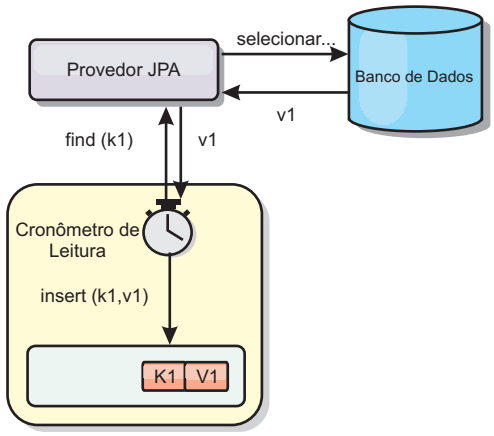


Figura 21. Atualização Periódica

Despejo

Os caches dispersos podem utilizar políticas de despejo para automaticamente remover dados do cache sem afetar o banco de dados. Há três políticas integradas incluídas no eXtreme Scale: time-to-live, least-recently-used e least-frequently-used. Todas as três políticas podem opcionalmente despejar dados mais agressivamente à medida que a memória torna-se restrita ao ativar a opção de despejo baseada em memória.

Invalidação Baseada em Eventos

Caches dispersos e completos podem ser invalidados ou atualizados usando um gerador de eventos como JMS (Java Message Service). A invalidação utilizando JMS pode ser manualmente vinculada a qualquer processo que atualiza o backend utilizando um acionador do banco de dados. Um plug-in ObjectGridEventListener do JMS é fornecido no eXtreme Scale que pode notificar quando o cache do servidor tiver qualquer alteração. Isto pode diminuir a quantidade de tempo que o cliente pode visualizar dados antigos.

Invalidação programática

As APIs do eXtreme Scale permitem interação manual do cache local e do servidor usando os métodos de API Session.beginNoWriteThrough(), ObjectMap.invalidate() e EntityManager.invalidate(). Se um processo do cliente ou servidor não precisar mais de uma parte dos dados, os métodos de invalidação podem ser utilizados para remover dados do cache local ou do servidor. O método beginNoWriteThrough aplica qualquer operação ObjectMap ou EntityManager para o cache local sem chamar o utilitário de carga. Se chamada a partir de um cliente, a operação é aplicável apenas para o cache local (o utilitário de carga remoto não é chamado). Se chamada no servidor, a operação é aplicável apenas ao cache principal do servidor sem chamar o utilitário de carga.

Invalidação de Dados

Para remover os dados de cache de escala, é possível usar um mecanismo de invalidação programático ou baseado em evento.

Invalidação Baseada em Evento

Caches dispersos e completos podem ser invalidados ou atualizados usando um gerador de eventos como JMS (Java Message Service). A invalidação utilizando JMS pode ser manualmente vinculada a qualquer processo que atualiza o backend utilizando um acionador do banco de dados. É fornecido um plug-in JMS ObjectGridEventListener no eXtreme Scale que pode notificar os clientes quando o cache do servidor é alterado. Esse tipo de notificação diminui a quantidade de tempo em que o cliente pode ver dados antigos.

A invalidação baseada em evento normalmente consiste nos três componentes a seguir.

- **Fila de eventos:** Uma fila de eventos armazena os eventos de mudança de dados. Ela pode ser uma fila JMS, um banco de dados, uma fila FIFO em memória ou qualquer tipo de manifesto, contanto que possa gerenciar os eventos de mudança de dados.
- **Publicador de evento:** Um publicador de evento publica os eventos de mudança de dados na fila de eventos. Um publicador de evento geralmente é um aplicativo que você cria ou uma implementação de plug-in do eXtreme Scale. O publicador de evento sabe quando os dados são alterados ou quando ele mesmo altera os dados. Quando uma transação é confirmada, eventos são gerados para os dados alterados e o publicador de eventos publica esses eventos na fila de eventos.
- **Consumidor de evento:** Um consumidor de evento consome eventos de mudança de dados. O consumidor de evento geralmente é um aplicativo para garantir que os dados da grade de destino sejam atualizados com a mudança mais recente das outras grades. Esse consumidor de evento interage com a fila de eventos para obter a mudança de dados mais recente, além de aplicar as mudanças de dados na grade de destino. Os consumidores de evento podem utilizar APIs do eXtreme Scale para invalidar dados antigos ou atualizar a grade com os dados mais recentes.

Por exemplo, JMSObjectGridEventListener tem uma opção para um modelo de cliente/servidor, no qual a fila de eventos é um destino JMS designado. Todos os processos do servidor são publicadores de eventos. Quando uma transação é confirmada, o servidor obtém as mudanças de dados e as publica no destino JMS designado. Todos os processos do cliente são consumidores de evento. Eles recebem as mudanças de dados do destino JMS designado e aplicam as mudanças no cache perto do cliente.

Consulte o tópico sobre a ativação do mecanismo de invalidação de cliente no *Guia de Administração* para obter mais informações.

Invalidação Programática

As APIs do WebSphere eXtreme Scale permitem interação manual do cache local e do servidor usando os métodos de API Session.beginNoWriteThrough(), ObjectMap.invalidate() e EntityManager.invalidate(). Se um processo do cliente ou servidor não precisar mais de uma parte dos dados, os métodos de invalidação podem ser utilizados para remover dados do cache local ou do servidor. O método beginNoWriteThrough aplica qualquer operação ObjectMap ou EntityManager para o cache local sem chamar o utilitário de carga. Se chamada a partir de um cliente, a operação é aplicável apenas para o cache local (o utilitário de carga remoto não é chamado). Se chamada no servidor, a operação é aplicável apenas ao cache principal do servidor sem chamar o utilitário de carga.

É possível utilizar invalidação programática com outras técnicas para determinar quando invalidar os dados. Por exemplo, esse método de invalidação utiliza mecanismos de invalidação baseados em evento para receber eventos de mudança de dados e depois utiliza as APIs para invalidar os dados antigos.

Indexação

Use o plug-in MapIndexPlugin para construir um índice ou vários índices em um BackingMap para suportar acesso a dados sem chave.

Tipos e Configuração de Índice

O recurso de indexação é representado pelo plug-in MapIndexPlugin ou Index, para abreviar. O Índice é um plug-in BackingMap. Um BackingMap pode ter múltiplos plug-ins de Índice configurados, enquanto cada um seguir as regras de configuração de Índice.

O recurso de indexação pode ser usado para construir um ou mais índices em um BackingMap. Um índice é construído a partir de um atributo ou uma lista de atributos de um objeto no BackingMap. Este recurso fornece uma maneira para os aplicativos localizarem determinados objetos mais rapidamente. Com o recurso de indexação, os aplicativos podem localizar objetos com um valor específico ou em um intervalo com os valores de atributos indexados.

Dois tipos de indexação são possíveis: estática e dinâmica. Com a indexação estática, é necessário configurar o plug-in de índice no BackingMap antes de inicializar a instância do ObjectGrid. É possível fazer esta configuração com a configuração XML ou programática do BackingMap. A indexação estática inicia a construção de um índice durante a inicialização do ObjectGrid. O índice é sempre sincronizado com o BackingMap e está pronto para utilização. Depois de o processo de indexação estático iniciar, a manutenção do índice é parte do processo de gerenciamento de transação do eXtreme Scale. Quando as consolidações de transações mudam, estas alterações também atualizam o índice estático, e as alterações de índice são recuperadas se a transação for recuperada.

Com a indexação dinâmica, é possível criar um índice num BackingMap antes ou depois da inicialização da instância do ObjectGrid que o contém. Os aplicativos possuem controle de ciclo de vida sobre o processo de indexação dinâmica para que você possa remover um índice dinâmico quando ele não for mais necessário. Quando um aplicativo cria um índice dinâmico, o índice pode não estar pronto para utilização imediata devido ao tempo gasto na conclusão do processo de construção do índice. Como a quantidade de tempo depende da quantidade de dados indexados, a interface DynamicIndexCallback é fornecido para aplicativos que desejam receber notificações quando ocorrem determinados eventos de indexação. Estes eventos incluem ready, error e destroy. Os aplicativos podem implementar esta interface de retorno de chamada e registrar-se no processo de indexação dinâmica.

Se um BackingMap tiver um plug-in de índice configurado, você pode obter o objeto de proxy do índice do aplicativo a partir do ObjectMap correspondente. Chamar o método getIndex no ObjectMap e passar o nome do plug-in de índice retornam o objeto de proxy do índice. Você deve efetuar o cast do objeto de proxy do índice para uma interface de índice adequada do aplicativo, como MapIndex, MapRangeIndex ou uma interface de índice customizada. Após obter o objeto de proxy do índice, é possível utilizar métodos definidos na interface do índice do aplicativo para localizar objetos armazenados em cache.

As etapas para utilizar a indexação estão resumidas na lista a seguir:

- Incluir plug-ins de indexação, estáticos ou dinâmicos, no BackingMap;
- Obter um objeto de proxy de indexação do aplicativo, emitindo o método getIndex do ObjectMap.
- Direcione o objeto de proxy de índice a uma interface de índice de aplicativo apropriado, como MapIndex, MapRangeIndex, ou uma interface de índice customizada.
- Utilizar os métodos definidos na interface Index do aplicativo para localizar objetos no cache.

A classe HashIndex é a implementação de plug-in de índice integrada que pode suportar ambas as interfaces integradas de índice do aplicativo: MapIndex e MapRangeIndex. Também é possível criar seus próprios índices. É possível incluir o HashIndex como um índice estático ou dinâmico no BackingMap, obter o objeto proxy do índice MapIndex ou MapRangeIndex e usar o objeto proxy do índice para localizar objetos em cache.

Índice Padrão

Se desejar iterar por meio das chaves em um mapa local, o índice padrão poderá ser usado. Este índice não requer nenhuma configuração, porém ele deve ser usado com relação ao shard, usando um agente ou uma instância do ObjectGrid recuperados a partir do método ShardEvents.shardActivated(shard do ObjectGrid).

Consideração sobre a Qualidade dos Dados

Os resultados dos métodos de consulta de índice somente representar uma captura instantânea de dados em um determinado ponto no tempo. Nenhum bloqueio contra as entradas de dados é obtido depois do retorno dos resultados para o aplicativo. O aplicativo deve estar ciente de que podem ocorrer atualizações de dados em um conjunto de dados retornado. Por exemplo, o aplicativo obtém a chave de um objeto armazenado em cache executando o método findAll de MapIndex. Este objeto de chave retornado está associado a uma entrada de dados no cache. O aplicativo deve poder executar o método get no ObjectMap para localizar um objeto fornecendo o objeto chave. Se outra transação remover o objeto de dados do cache imediatamente antes de o método get ser chamado, o resultado retornado será nulo.

Considerações sobre Desempenho de Indexação

Um dos principais objetivos do recurso de indexação é melhorar o desempenho geral do BackingMap. Se a indexação não for utilizada corretamente, o desempenho do aplicativo poderá ficar comprometido. Considere os seguintes fatores antes de usar este retorno.

- **A quantidade de transações de gravação concorrentes:** O processamento de índices pode ocorrer todas as vezes que uma transação gravar dados em um BackingMap. O desempenho será afetado se muitas transações gravarem dados no mapa simultaneamente quando um aplicativo tentar operações de consulta ao índice.
- **O tamanho do conjunto de resultados que é retornado por uma operação de consulta:** Como o tamanho do conjunto de resultados aumenta, o desempenho da consulta cai. O desempenho tente a degradas quando o tamanho do conjunto de resultados é de 15% (ou mais) do BackingMap.

- **A quantidade de índices construídos sobre o mesmo BackingMap:** Cada índice consome os recursos do sistema. Conforme a quantidade dos índices construídos sobre o BackingMap aumenta, o desempenho diminui.

A função de indexação pode aumentar significativamente o desempenho do BackingMap. Os casos ideais ocorrem quando o BackingMap possui a maioria de operações de leitura, o conjunto de resultados da consulta é de uma porcentagem pequena das entradas do BackingMap, e somente poucos índices são construídos sobre o BackingMap.

Tarefas relacionadas

"Configurando o Plug-in HashIndex" na página 327
É possível configurar o HashIndex integrado, a classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, com um arquivo XML, programaticamente ou com uma anotação de entidade em um mapa de entidade.

"Acessando Dados com Índices (API de Índice)" na página 144
Use indexação para acesso a dados mais eficiente.

Referências relacionadas

"Atributos do Plug-in HashIndex" na página 330
É possível usar os seguintes atributos para configurar o plug-in HashIndex. Esses atributos definem propriedades, como se você estiver usando um atributo ou HashIndex composto ou se a indexação do intervalo estiver ativada.

Planejando Diversas Topologias do Datacenter

Ao usar a replicação assíncrona multimestre, duas ou mais grades de dados podem se tornar cópias exatas de uns dos outros. Cada grade de dados é hospedada em um domínio do serviço de catálogo independente, com seu próprio serviço de catálogo, servidores de contêiner e um nome exclusivo. Com a replicação assíncrona multimestre, é possível usar links para conectar uma coleção de domínios do serviço de catálogo. Os domínios do serviço de catálogo são então sincronizados usando a replicação sobre os links. É possível construir quase qualquer topologia por meio da definição de links entre os domínios de serviço de catálogo.

Tarefas relacionadas

Configurando Diversas Topologias do Datacenter
Com a replicação assíncrona multimestre, um conjunto de domínios de serviço de catálogo é vinculado. Os domínios de serviço de catálogo conectados são então sincronizado usando a replicação sobre os links. É possível definir os links usando arquivos de propriedades, no tempo de execução com programas Java Management Extensions (JMX) ou com utilitários de linha de comandos. O conjunto de links atuais para um domínio é armazenado no serviço de catálogo. É possível incluir e remover links sem reiniciar o domínio de serviço de catálogo que hospeda a grade de dados.

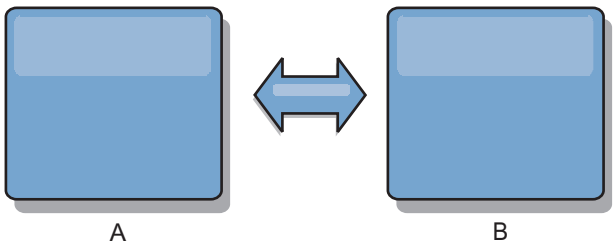
"Desenvolvendo Árbitros Customizados para a Replicação Multimestre" na página 304
Poderão ocorrer colisões de mudanças se os mesmos registros puderem ser alterados simultaneamente em dois locais. Em uma topologia de replicação multimestre, os domínios do serviço de catálogo detectam colisões automaticamente. Quando um domínio de serviço de catálogo detecta uma colisão, ele chama um árbitro. Geralmente, as colisões são resolvidas com o árbitro de colisão padrão. No entanto, um aplicativo pode fornecer um árbitro de colisão customizado.

Topologias de Replicação Multimestre

Há várias opções diferentes quando escolher a topologia para sua implementação que incorpora a replicação multimestre.

Links Conectando Domínios de Serviço de Catálogos

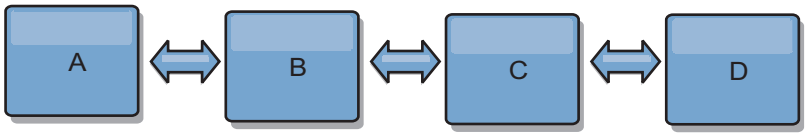
Uma infraestrutura de grade de dados de replicação é um gráfico conectado dos domínios do serviço de catálogo com links bidirecionais entre eles. Com um link, dois domínios de serviço de catálogo podem comunicar as mudanças de dados. Por exemplo, a topologia mais simples é um par de domínios do serviço de catálogo com um único link entre eles. Os domínios do serviço de catálogo são denominados em ordem alfabética: A, B, C e assim por diante, a partir da esquerda. Um link pode cruzar uma rede de longa distância (WAN), expandindo-se para grandes distâncias. Mesmo se o link for interrompido, os dados ainda poderão ser alterados em qualquer um dos domínios de serviço de catálogo. A topologia reconcilia as mudanças quando o link reconecta os domínios do serviço de catálogo. Os links tentarão automaticamente reconectar se a conexão com a rede for interrompida.



Após configurar os links, o eXtreme Scale primeiro tentará tornar idêntico cada domínio do serviço de catálogo. Em seguida, o eXtreme Scale tenta manter as condições idênticas conforme as mudanças ocorrerem em qualquer domínio de serviço de catálogo. O objetivo é que cada domínio de serviço de catálogo seja um espelho exato de qualquer outro domínio de serviço de catálogo conectado pelos links. Os links de replicação entre os domínios de serviço de catálogo ajudam a assegurar que quaisquer mudanças feitas em um domínio sejam copiadas para os outros domínios.

Topologias em Linha

Embora esta seja uma implementação muito simples, uma topologia em linha demonstra algumas qualidades dos links. Primeiro, um domínio de serviço de catálogo não precisa estar conectado diretamente a outro domínio de serviço de catálogo para receber as mudanças. O Domínio B pega as mudanças de Domínio A. O Domínio C recebe mudanças do Domínio A por meio do Domínio B, que se conecta os Domínios A e C. Da mesma forma, o Domínio D recebe as mudanças dos outros domínios por meio do Domínio C. Esta habilidade envia o carregamento das mudanças de distribuição para longe da origem das mudanças.



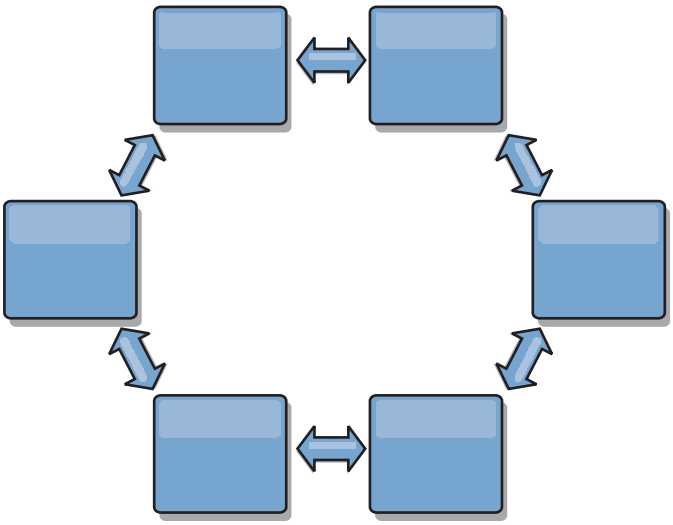
Observe que, se o Domínio C falhar, as seguintes ações ocorrerão:

1. O Domínio D pode ficar órfão até o Domínio C ser reiniciado
2. O Domínio C pode se sincronizar com o Domínio B, que é uma cópia do Domínio A
3. O Domínio D pode usar o Domínio C para se sincronizar com as mudanças nos Domínios A e B. Essas mudanças inicialmente ocorreram enquanto o Domínio D estava órfão (enquanto o Domínio C estava inativo).

Por fim, os Domínios A, B, C e D podem se tornar todos idênticos entre si novamente.

Topologias em Anel

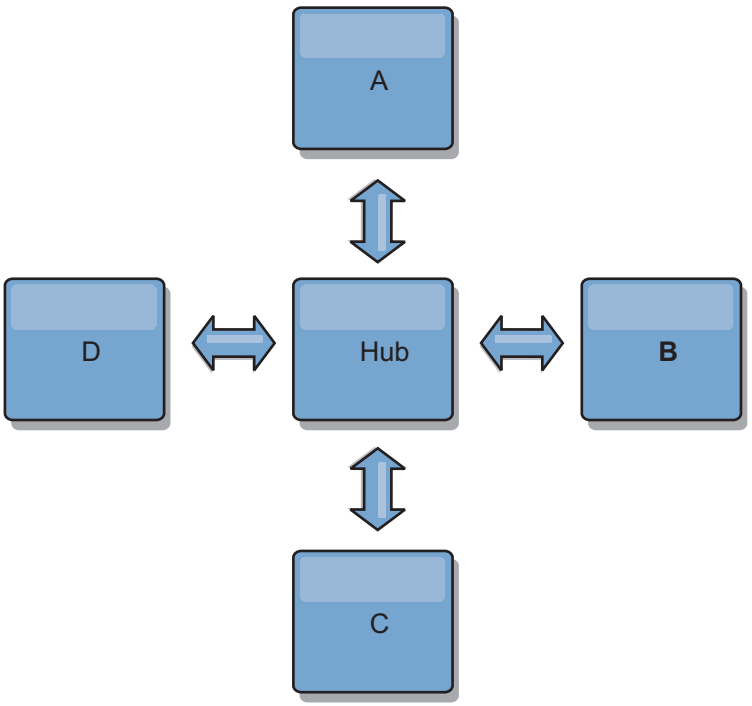
As topologias em anel são um exemplo de uma topologia mais resiliente. Quando um domínio do serviço de catálogo ou um único link falhar, os domínios do serviço de catálogo sobreviventes ainda podem obter as mudanças. Os domínios de serviço de catálogo se deslocam ao redor do anel, longe da falha. Cada domínio de serviço de catálogo tem no máximo dois links para outros domínios de serviço de catálogo, independente do tamanho da topologia em anel. A latência para propagar as mudanças pode ser grande. As mudanças a partir de um domínio de serviço de catálogo particular podem precisar percorrer vários links antes que todos os domínios de serviço de catálogo possuam as mudanças. Uma topologia em linha tem a mesma característica.



Também é possível implementar uma topologia em anel mais sofisticada, com um domínio de serviço de catálogo raiz no centro do anel. O domínio de serviço de catálogo raiz funciona como o ponto central da reconciliação. Os outros domínios de serviço de catálogo atuam como pontos remotos de reconciliação para que as mudanças ocorram no domínio de serviço de catálogo raiz. O domínio de serviço de catálogo raiz pode arbitrar mudanças entre os domínios de serviço de catálogo. Se uma topologia em anel contiver mais de um anel ao redor de um domínio de serviço de catálogo raiz, o domínio poderá arbitrar apenas as mudanças entre o anel mais interno. No entanto, os resultados da arbitragem se propagam por todos os domínios de serviço de catálogo nos outros anéis.

Topologias Hub-and-Spoke

Com uma topologia hub-and-spoke, as mudanças se deslocam por um domínio de serviço de catálogo hub. Como o hub é apenas o domínio de serviço de catálogo intermediário que é especificado, as topologias hub e spoke possuem baixa latência. O domínio hub é conectado a cada domínio de spoke por meio de um link. O hub distribui as mudanças entre os domínios de serviço de catálogo. O hub atua como um ponto de reconciliação de colisões. Em um ambiente com uma alta taxa de atualização, o hub pode precisar executar em mais hardware do que os spokes devem permanecer sincronizados. O WebSphere eXtreme Scale é projetado para escalar linearmente, o que significa que é possível aumentar o hub, conforme necessário, sem dificuldade. No entanto, se o hub falhar, as mudanças não são distribuídas até ele ser reiniciado. As mudanças nos domínios de serviço de catálogo spoke serão distribuídas depois que o hub for reconectado.



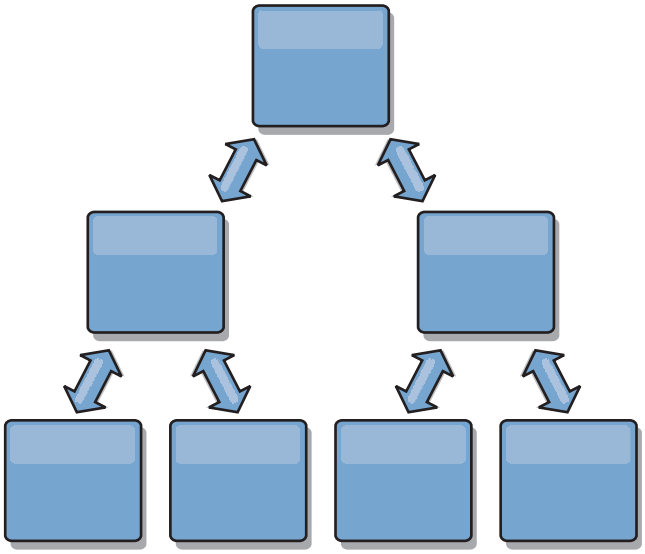
Também é possível usar uma estratégia com clientes totalmente replicados, uma variação de topologia que usa um par de servidores eXtreme Scale sendo executados como um hub. Cada cliente cria uma grade de dados autocontida de contêiner único com um catálogo na JVM do cliente. Um cliente usa a grade de dados para se conectar com o catálogo do hub. Esta conexão faz com que o cliente seja sincronizado com o hub assim que o cliente estabelecer uma conexão com o hub.

Todas as mudanças feitas pelo cliente são locais para o cliente e são replicadas de forma assíncrona para o hub. O hub age como um domínio de arbitragem, distribuindo mudanças para todos os clientes conectados. A topologia de clientes totalmente replicados fornece um cache L2 confiável para um mapeador relacional de objeto, como OpenJPA. As mudanças serão distribuídas rapidamente entre as JVMs de cliente por meio do hub. Se o tamanho do cache puder estar contido no espaço de heap disponível, a topologia será uma arquitetura confiável para este estilo de L2.

Use várias partições para escalar o domínio do hub em várias JVMs, se necessário. Como todos os dados ainda devem se ajustar em uma única JVM de cliente, diversas partições aumentam a capacidade do hub para distribuir e arbitrar as mudanças. No entanto, ter diversas partições não altera a capacidade de um único domínio.

Topologias em Árvore

Também é possível usar uma árvore direcionada acíclica. Uma árvore acíclica não tem ciclos ou loops e uma configuração direcionada limita a criação de links apenas entre pais e filhos. Esta configuração pode ser útil para topologias que possuem muitos domínios de serviço de catálogo e não é viável ter um hub central conectado a cada spoke possível. Este tipo de topologia também pode ser útil quando os domínios de serviço de catálogo filho tiverem que ser incluídos sem atualizar o domínio de serviço de catálogo raiz.



Uma topologia em árvore ainda pode ter um ponto central de reconciliação no domínio de serviço de catálogo raiz. O segundo nível pode ainda funcionar como um ponto de reconciliação remoto para que as mudanças ocorram no domínio de serviço de catálogo abaixo deles. O domínio do serviço de catálogo raiz pode arbitrar as mudanças entre os domínios de serviço de catálogo apenas no segundo nível. Também é possível usar N árvores, cada uma delas possuindo N filhos em cada nível. Cada domínio de serviço de catálogo se conecta com *n* links .

Clientes Totalmente Replicados

Essa variação de topologia envolve um par de servidores do eXtreme Scale sendo executados como um hub. Cada cliente cria uma grade de dados autocontida de contêiner único com um catálogo na JVM do cliente. Um cliente usa a grade de dados para se conectar com o catálogo do hub, fazendo com que o cliente seja sincronizado com o hub assim que estabelecer uma conexão com o hub.

Todas as mudanças feitas pelo cliente são locais para o cliente e são replicadas de forma assíncrona para o hub. O hub age como um domínio de arbitragem, distribuindo mudanças para todos os clientes conectados. A topologia de clientes totalmente replicada fornece um bom cache L2 para um mapeador relacional de objeto, como OpenJPA. As mudanças serão distribuídas rapidamente entre as JVMs de cliente por meio do hub. Contanto que o tamanho do cache possa estar contido no espaço de heap disponível dos clientes, esta topologia será uma boa arquitetura para este estilo de L2.

Use várias partições para escalar o domínio do hub em várias JVMs, se necessário. Como todos os dados ainda devem se ajustar em uma única JVM de cliente, usar várias partições aumenta a capacidade do hub para distribuir e arbitrar mudanças, mas não altera a capacidade de um único domínio.

Tarefas relacionadas

Configurando Diversas Topologias do Datacenter
Com a replicação assíncrona multimestre, um conjunto de domínios de serviço de catálogo é vinculado. Os domínios de serviço de catálogo conectados são então sincronizado usando a replicação sobre os links. É possível definir os links usando arquivos de propriedades, no tempo de execução com programas Java Management Extensions (JMX) ou com utilitários de linha de comandos. O conjunto de links atuais para um domínio é armazenado no serviço de catálogo. É possível incluir e remover links sem reiniciar o domínio de serviço de catálogo que hospeda a grade de dados.

"Desenvolvendo Árbitros Customizados para a Replicação Multimestre" na página 304
Poderão ocorrer colisões de mudanças se os mesmos registros puderem ser alterados simultaneamente em dois locais. Em uma topologia de replicação multimestre, os domínios do serviço de catálogo detectam colisões automaticamente. Quando um domínio de serviço de catálogo detecta uma colisão, ele chama um árbitro. Geralmente, as colisões são resolvidas com o árbitro de colisão padrão. No entanto, um aplicativo pode fornecer um árbitro de colisão customizado.

Considerações de Configuração para Topologias Multimestre

Considere os seguintes problemas ao decidir se e como usar as topologias de replicação multimestre.

- **Requisitos do conjunto de mapa**

  Os conjuntos de mapa devem ter as seguintes características para replicar as mudanças nos links de domínio do serviço de catálogo:

  – O nome do ObjectGrid e o nome do conjunto de mapas dentro de um domínio do serviço de catálogo devem corresponder ao nome do ObjectGrid e ao nome do conjunto de mapas de outros domínios de serviço de catálogo. Por exemplo, o ObjectGrid "og1" e o conjunto de mapas "ms1" devem ser configurados no domínio do serviço de catálogo A e no domínio do serviço de catálogo B para replicar os dados no conjunto de mapas entre os domínios do serviço de catálogo.

  – Ser uma grade de dados `FIXED_PARTITION`. As grades de dados `PER_CONTAINER` não podem ser replicadas.

  – Ter o mesmo número de partições em cada domínio do serviço de catálogo. O conjunto de mapa pode ou não ter o mesmo número e tipos de réplicas.

  – Ter os mesmos tipos de dados sendo replicados em cada domínio do serviço de catálogo.

  – Contém os mesmos mapas e modelos de mapas dinâmicos em cada domínio do serviço de catálogo.

  – Não usar o gerenciador de entidades. Um conjunto de mapas contendo um mapa de entidade não é replicado entre os domínios do serviço de catálogo.

  – Não usar o suporte de armazenamento em cache write-behind. Um conjunto de mapas contendo um mapa configurado com o suporte write-behind não é replicado entre os domínios de serviço de catálogo.

  Quaisquer conjuntos de mapas com as características anteriores começam a serem replicados depois que os domínios do serviço de catálogo na topologia forem iniciados.

- **Carregadores de classe com diversos domínios do serviço de catálogo**

  Os domínios do serviço de catálogo devem ter acesso a todas as classes que são usadas como chaves e valores. Todas as dependências devem ser refletidas em

todos os caminhos da classe para as Java virtual machine do contêiner da grade para todos os domínios. Se um plug-in CollisionArbiter recuperar o valor para uma entrada de cache, as classes para os valores deverão estar presentes para o domínio que está iniciando o mecanismo de resolução de conflitos.

Tarefas relacionadas

Configurando Diversas Topologias do Datacenter
Com a replicação assíncrona multimestre, um conjunto de domínios de serviço de catálogo é vinculado. Os domínios de serviço de catálogo conectados são então sincronizado usando a replicação sobre os links. É possível definir os links usando arquivos de propriedades, no tempo de execução com programas Java Management Extensions (JMX) ou com utilitários de linha de comandos. O conjunto de links atuais para um domínio é armazenado no serviço de catálogo. É possível incluir e remover links sem reiniciar o domínio de serviço de catálogo que hospeda a grade de dados.

"Desenvolvendo Árbitros Customizados para a Replicação Multimestre" na página 304
Poderão ocorrer colisões de mudanças se os mesmos registros puderem ser alterados simultaneamente em dois locais. Em uma topologia de replicação multimestre, os domínios do serviço de catálogo detectam colisões automaticamente. Quando um domínio de serviço de catálogo detecta uma colisão, ele chama um árbitro. Geralmente, as colisões são resolvidas com o árbitro de colisão padrão. No entanto, um aplicativo pode fornecer um árbitro de colisão customizado.

Considerações Sobre o Carregador em uma Topologia Multimestre

Quando estiver usando os carregadores em uma topologia multimestre, você deve considerar a possibilidade de colisão e desafios de manutenção das informações de revisão. A grade de dados mantém as informações de revisão sobre os itens nela para que colisões possam ser detectadas quando outros shards primários na configuração gravarem entradas na grade de dados. Quando as entradas são incluídas a partir de um carregador, essas informações de revisão não são incluídas e a entrada assume uma nova revisão. Como a revisão da entrada parece ser uma nova inserção, uma colisão false poderá ocorrer se outro shard primário também alterar esse estado ou obtiver as mesmas informações a partir de um carregador.

As mudanças na replicação chamam o método get no carregador com uma lista das chaves que ainda não estão na grade de dados, porém serão alteradas durante a transação da replicação. Quando a replicação ocorre, essas entradas são entradas de colisão. Quando as colisões são definidas e a revisão é aplicada, uma atualização em lote é chamada no carregador para aplicar as mudanças no banco de dados. Todos os mapas que foram alterados na janela de revisão são atualizados na mesma transação.

Desafio do Pré-Carregamento

Considere uma topologia de dois datacenters, com o datacenter A e o datacenter B. Ambos os datacenters possuem bancos de dados independente, mas apenas o datacenter A possui uma grade de dados em execução. Quando você estabelece um link entre os datacenters por uma configuração multimestre, as grades de dados no datacenter A começam a enviar dados para as novas grades de dados do datacenter B, causando uma colisão com cada entrada. Outro maior problema que ocorre é com os dados que estão no banco de dados do datacenter B, mas não no banco de dados no datacenter A. Essas linhas não são preenchidas e definidas, resultando em inconsistências que não são resolvidas.

Solução para o Desafio de Pré-Carregamento

Como os dados que residem apenas no banco de dados não podem ter revisões, a grade de dados sempre deve ser pré-carregada completamente a partir do banco de dados local antes de estabelecer o link multimestre. Em seguida, ambas as grades de dados podem revisar e definir os dados, eventualmente chegando a um estado consistente.

Desafio de Cache Disperso

Com um cache disperso, o primeiro aplicativo tenta localizar os dados na grade de dados. Se os dados não estiverem na grade de dados, eles serão procurados no banco de dados usando o carregador. As entradas são despejadas da grade de dados periodicamente para manter um tamanho de cache pequeno.

Este tipo de cache pode ser problemático em um cenário de configuração multimestre porque as entradas dentro da grade de dados possuem metadados de revisão que ajudam a detectar quando colisões ocorrem e qual lado fez as mudanças. Quando os links entre os datacenters não estiverem funcionando, um datacenter pode atualizar uma entrada e depois, eventualmente, atualizar o banco de dados e invalidar a entrada na grade de dados. Quando o link é recuperado, os datacenters tentam sincronizar as revisões entre si. No entanto, como o banco de dados foi atualizado e a entrada da grade de dados foi invalidada, a mudança é perdida a partir da perspectiva do datacenter que ficou inativo. Como resultado, os dois lados da grade de dados estão fora de sincronização e não estão consistentes.

Solução para o Desafio de Cache Disperso

Topologia Hub e Spoke:

É possível executar o carregador apenas no hub de uma topologia de hub e spoke, mantendo a consistência dos dados, enquanto a escala da grade de dados é ajustada. No entanto, se você estiver considerando essa implementação, observe que os carregadores podem permitir que a grade de dados seja parcialmente carregada, significando que um evictor foi configurado. Se o spokes de sua configuração forem caches dispersos, mas não possuírem nenhum carregador, quaisquer perdas de cache não terão como recuperar dados do banco de dados. Devido a esta restrição, você deve usar uma topologia de cache totalmente preenchida com uma configuração de hub e spoke.

Invalidações e Despejo

Uma invalidação cria inconsistência entre a grade de dados e o banco de dados. Os dados podem ser removidos da grade de dados, seja de modo programático ou com o despejo. Ao desenvolver seu aplicativo, você deve estar ciente de que a manipulação de revisão não replica mudanças que são invalidadas, resultando em inconsistências entre os shards primários.

Os eventos de invalidação não são mudanças de estado de cache e não resultam em replicação. Todos os evictors configurados são executados independentemente de outros evictors na configuração. Por exemplo, você pode ter um evictor configurado para um limite de memória em um domínio do serviço de catálogo, e um tipo diferente de evictor menos agressivo no outro domínio do serviço de catálogo vinculado. Quando as entradas da grade dedados são removidas devido à política de limite de memória, as entradas no outro domínio do serviço de catálogo não são afetadas.

Atualizações do banco de dados e invalidação da grade de dados

Problemas ocorrem quando o banco de dados é atualizado diretamente no plano de fundo ao chamar a invalidação na grade de dados para as entradas atualizadas em uma configuração multimestre. Esse problema ocorre porque a grade de dados não pode replicar a mudança para os outros shards primários até que algum tipo de acesso ao cache mova a entrada para a grade de dados.

Diversos Gravadores para um Único Banco de Dados Lógico

Quando um banco de dados único é usado com diversos shards primários conectados por meio de um carregador, isso pode resultar em conflitos transacionais. Sua implementação do carregador deve manipular especialmente esses tipos de cenários.

Espelhando Dados Usando Replicação Multimestre

É possível configurar bancos de dados independentes que estão conectados a domínios do serviço de catálogo independentes. Nessa configuração, o carregador pode enviar mudanças de um datacenter para o outro.

Tarefas relacionadas

Configurando Diversas Topologias do Datacenter
Com a replicação assíncrona multimestre, um conjunto de domínios de serviço de catálogo é vinculado. Os domínios de serviço de catálogo conectados são então sincronizado usando a replicação sobre os links. É possível definir os links usando arquivos de propriedades, no tempo de execução com programas Java Management Extensions (JMX) ou com utilitários de linha de comandos. O conjunto de links atuais para um domínio é armazenado no serviço de catálogo. É possível incluir e remover links sem reiniciar o domínio de serviço de catálogo que hospeda a grade de dados.

"Desenvolvendo Árbitros Customizados para a Replicação Multimestre" na página 304
Poderão ocorrer colisões de mudanças se os mesmos registros puderem ser alterados simultaneamente em dois locais. Em uma topologia de replicação multimestre, os domínios do serviço de catálogo detectam colisões automaticamente. Quando um domínio de serviço de catálogo detecta uma colisão, ele chama um árbitro. Geralmente, as colisões são resolvidas com o árbitro de colisão padrão. No entanto, um aplicativo pode fornecer um árbitro de colisão customizado.

Considerações de Design para Replicação Multimestre

Ao implementar da replicação multimestre, você deve considerar aspectos de design, como arbitragem, vinculação e desempenho.

Considerações sobre Arbitragem no Design de Topologia

Poderão ocorrer colisões de mudanças se os mesmos registros puderem ser alterados simultaneamente em dois locais. Configure cada domínio de serviço de catálogo para ter aproximadamente a mesma quantia de processador, memória e recursos de rede. Observe que os domínios de serviço de catálogo que executam a manipulação da colisão de mudanças (arbitragem) usam mais recursos que outros domínios de serviço de catálogo. As colisões são detectadas automaticamente. Elas são manipuladas com um desses dois mecanismo:

- **Árbitro de conflito padrão**: O protocolo padrão deve usar as mudanças a partir do domínio de serviço de catálogo mais baixo denominado de maneira lexical.

Por exemplo, se os domínios de serviço de catálogo A e B gerarem um conflito para um registro, a mudança no domínio de serviço de catálogo B será ignorada. O domínio de serviço de catálogo A mantém sua versão e o registro no domínio de serviço de catálogo B é alterado para corresponder ao registro do domínio de serviço de catálogo A. Esse comportamento também se aplica aos aplicativos nos quais os usuários ou as sessões são normalmente ligados ou têm afinidade com uma das grades de dados.

- **Árbitro de colisão customizado**: Os aplicativos podem fornecer um árbitro customizado. Quando um domínio do serviço de catálogo detecta uma colisão, ele inicia o árbitro. Para obter informações sobre como desenvolver um árbitro útil customizado, consulte "Desenvolvendo Árbitros Customizados para a Replicação Multimestre" na página 304.

Para topologias nas quais as colisões são possíveis, considere implementar uma topologia hub-and-spoke ou uma topologia em árvore. Essas duas topologias tendem a evitar colisões constantes, o que pode acontecer nos seguintes cenários:

1. Diversos domínios de serviço de catálogo experimentam uma colisão.
2. Cada domínio de serviço de catálogo manipula a colisão localmente, produzindo revisões.
3. As revisões colidem, resultando em revisões de revisões

Para evitar colisões, escolha um domínio de serviço de catálogo específico, chamado de *domínio de serviço de catálogo de arbitragem* como o árbitro de colisão para um subconjunto de domínios de serviço de catálogo. Por exemplo, uma topologia hub-and-spoke pode usar o hub como o manipulador de colisão. O manipulador de colisão spoke ignora quaisquer colisões que forem detectadas pelos domínios de serviço de catálogo spoke. O domínio de serviço de catálogo do hub cria revisões, evitando revisões de colisão inesperadas. O domínio de serviço de catálogo designado para manipular colisões deve ser vinculado a todos os domínios os quais ele é responsável por manipular as colisões. Em uma topologia em árvore, os domínios pais internos manipulam colisões para seus filhos imediatos. Em contraste, se usar uma topologia em anel, não será possível designar um domínio de serviço de catálogo no anel como o árbitro.

A tabela a seguir resume as abordagens de arbitragem que são mais compatíveis com várias topologias.

Tabela 1. Abordagens de Arbitragem. Esta tabela define se a arbitragem de aplicativo é compatível com várias tecnologias.

Topologia	Arbitragem do Aplicativo?	Notes
Uma linha de dois domínios de serviço de catálogo	Sim	Escolha um domínio do serviço de catálogo como o árbitro.
Uma linha de três domínios de serviço de catálogo	Sim	O domínio de serviço de catálogo médio deve ser o árbitro. Considere o domínio de serviço de catálogo médio como sendo o hub em uma topologia hub-and-spoke simples.
Uma linha de mais de três domínios de serviço de catálogo	Não	A arbitragem de aplicativo não é suportada.

Tabela 1. Abordagens de Arbitragem (continuação). Esta tabela define se a arbitragem de aplicativo é compatível com várias tecnologias.

Topologia	Arbitragem do Aplicativo?	Notes
Um hub com N spokes	Sim	O hub com links para todos os spokes deve ser o domínio de serviço de catálogo de arbitragem.
Um anel de N domínios de serviço de catálogo	Não	A arbitragem de aplicativo não é suportada.
Uma árvore acíclica, direcionada (árvore n-ary)	Sim	Todos os nós-raiz devem classificar apenas seus descendentes diretos.

Considerações sobre Links no Design de Topologia

De forma ideal, uma topologia inclui um número mínimo de links enquanto otimiza trade-offs entre latência de mudança, tolerância a falhas e características de desempenho.

- **Latência de mudança**

  A latência de mudança é determinada pelo número de domínios de serviço de catálogo intermediários onde uma mudança deve passar antes de chegar a um domínio do serviço de catálogo específico.

  Uma topologia tem a melhor latência de mudança quando ele elimina os domínios de serviço de catálogo intermediários ao vincular cada domínio de serviço de catálogo a outro domínio. No entanto, um domínio de serviço de catálogo deve executar o trabalho de replicação proporcionalmente ao seu número de links. Para topologias grandes, o número de links absoluto a ser definido pode causar uma carga administrativa.

  A velocidade com que uma mudança é copiada para outros domínios do serviço de catálogo depende de fatores adicionais, como:

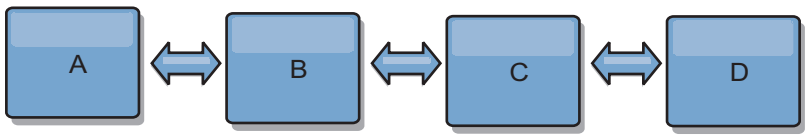
  – Processador e largura da banda da rede no domínio de serviço de catálogo de origem
  – O número de domínios de serviço de catálogo intermediários e de links entre o domínio de serviço de catálogo de origem e de destino
  – Os recursos de processador e de rede disponíveis para os domínios de serviço de catálogo de origem, de destino e intermediários

- **Tolerância a falhas**

  A tolerância a falhas é determinada pela quantia de caminhos que existem entre dois domínios de serviço de catálogo para a replicação de mudança.

  Se houver apenas um link entre um determinado par de domínios de serviço de catálogo, uma falha de link não permitirá a propagação das mudanças. Da mesma forma, as mudanças não serão propagadas entre os domínios de serviço de catálogo se ocorrer uma falha de link em qualquer um dos domínios intermediários. Sua topologia pode ter um único link a partir de um domínio de serviço de catálogo para outro, de modo que o link passe pelos domínios intermediários. Caso positivo, as mudanças não serão propagadas se qualquer um dos domínios de serviço de catálogo intermediários estiver inativo.

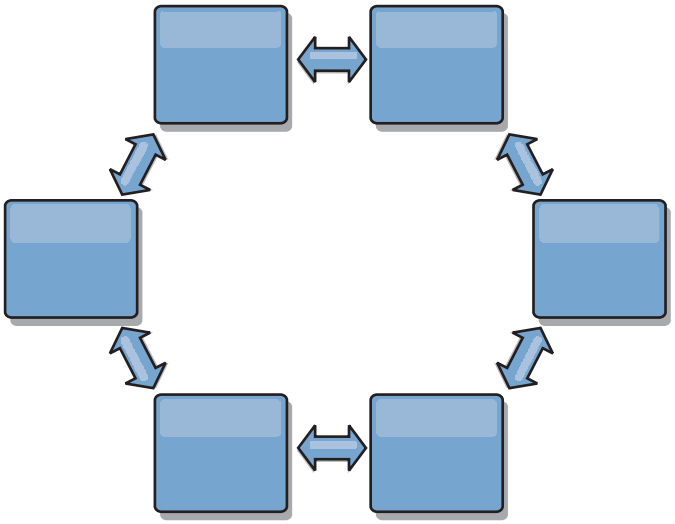
  Considere a topologia em linha com quatro domínios de serviço de catálogo, A, B, C e D:



Se qualquer uma dessas condições for mantida, o Domínio D não verá nenhuma mudança a partir do A:

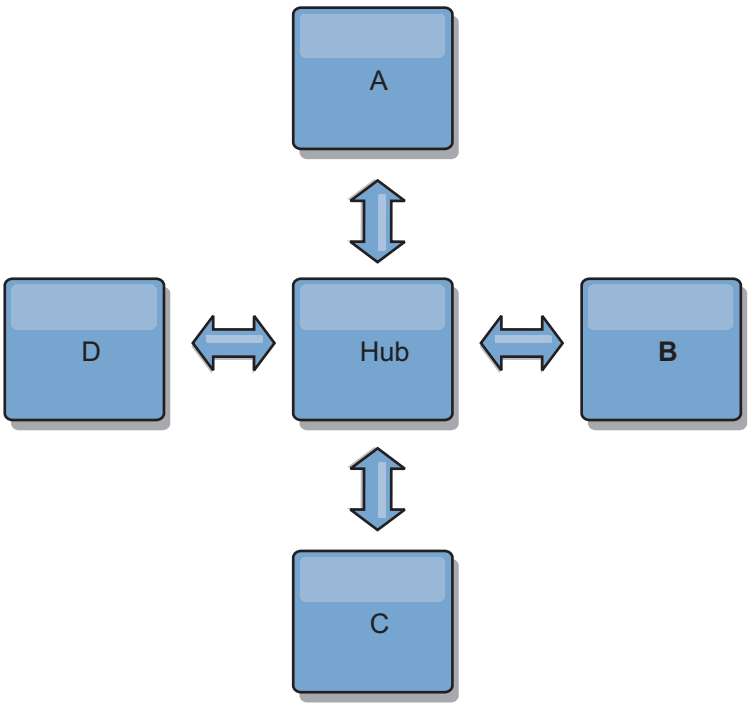
- O Domínio A está ativo e o B está inativo
- Os Domínios A e B estão ativos e C está inativo
- O link entre A e B está inativo
- O link entre B e C está inativo
- O link entre C e D está inativo

Em contraste, com uma topologia em anel, cada domínio de serviço de catálogo pode receber mudanças a partir de qualquer direção.



Por exemplo, se um determinado serviço de catálogo em sua topologia em anel estiver inativo, os dois domínios adjacentes ainda poderão obter as mudanças diretamente entre si.

Todas as mudanças são propagadas por meio do hub. Assim, em oposição às topologia de linha e em anel, o design hub-and-spoke será suscetível à interrupção se o hub falhar.



Um domínio de serviço de catálogo único é resiliente a uma determinada perda de quantidade de serviço. No entanto, falhas maiores, como interrupções ou perda de links em uma rede maior entre os datacenters físicos pode interromper qualquer um dos domínios de serviço de catálogo.

- **Vínculo e desempenho**

  O número de links definido em um domínio de serviço de catálogo afeta o desempenho. Mais links usam mais recursos e o desempenho de replicação pode diminuir como resultado. A capacidade de recuperar mudanças para um domínio A por meio de outros domínios isenta efetivamente o domínio A da replicação de suas transações em todo lugar. O carregamento de distribuição de mudança em um domínio é limitado pelo número de links que ele usa, e não pela quantia de domínios presentes na topologia. Esta propriedade carregamento fornece escalabilidade, de modo que os domínios na topologia possam compartilhar a carga de distribuição de mudança.

  Um domínio de serviço de catálogo pode recuperar as mudanças indiretamente por meio de outros domínios de serviço de catálogo. Considere uma topologia em linha com cinco domínios de serviço de catálogo.

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

  – A pega mudanças de B, C, D e E por meio de B
  – B pega mudanças de A e C diretamente e mudanças de D e E por meio de C
  – C pega mudanças de B e D diretamente e mudanças de A por meio de B e E por meio de D
  – D pega mudanças de C e E diretamente e mudanças de A e B por meio de C
  – E pega mudanças de D diretamente e mudanças de A, B e C por meio de D

  O carregamento de distribuição nos domínios de serviço de catálogo A e E é o mais baixo, porque cada um deles possui um link apenas para um domínio de serviço de catálogo único. Cada um dos domínios B, C e D possui um link para dois domínios. Assim, o carregamento de distribuição nos domínios B, C e D é o dobro do carregamento nos domínios A e E. A carga de trabalho depende do número de links em cada domínio, e não do número geral de domínios na

topologia. Assim, a distribuição dos carregamentos descrita permaneceria constante, mesmo se a linha contivesse 1.000 domínios.

Considerações de Desempenho de Replicação Multimestre

Leve em consideração as seguintes limitações quando usar topologias de replicação multimestre:

- **Alterar ajuste de distribuição**, conforme discutido na seção anterior.
- **Desempenho do link de replicação** O WebSphere eXtreme Scale cria um único soquete TCP/IP entre qualquer par de JVMs. Todo o tráfego entre as JVMs ocorre por meio deste soquete único, incluindo o tráfego de replicação multimestre. Os domínios do serviço de catálogo são hospedados em pelo menos *n* JVMs de contêiner, fornecendo pelo menos *n* links TCP para os domínios de serviço de catálogo equivalentes. Assim, os domínios de serviço de catálogo com números maiores de contêineres têm níveis maiores de desempenho de replicação. Mais contêineres requerem mais recursos de processador e de rede.
- **Ajuste da janela deslizante TCP e o RFC 1323** O suporte do RFC 1323 em ambas as extremidades de um link geram mais dados para um roundtrip. Este suporte resulta em maior rendimento, expandindo a capacidade da janela em um fator de aproximadamente 16.000.

  A rechamada desses soquetes TCP usa um mecanismo de janela deslizante para controlar o fluxo de dados em massa. Este mecanismo geralmente limita o soquete para 64 KB para um intervalo de roundtrip. Se o intervalo de roundtrip for de 100 ms, a largura de banda será limitada a 640 KB/segundo sem ajuste adicional. Usar totalmente a largura de banda disponível em um link pode exigir um ajuste específico para um sistema operacional. A maioria dos sistemas operacionais inclui parâmetros de ajuste, inclusive opções RFC 1323, para aprimorar o rendimento por meio dos links de alta latência.

  Vários fatores podem afetar o desempenho de replicação:

  – A velocidade com que o eXtreme Scale recupera as mudanças.
  – A velocidade com que o eXtreme Scale pode atender às solicitações de replicação de recuperação.
  – A capacidade da janela de deslizamento.
  – Com o ajuste do buffer de rede em ambos os lados de um link, o eXtreme Scale recupera as mudanças por meio do soquete de modo eficiente.
- **Serialização de Objeto** Todos os dados devem ser serializáveis. Se um domínio de serviço de catálogo não estiver usando `COPY_TO_BYTES`, o domínio de serviço de catálogo deverá usar a serialização Java ou ObjectTransformers para otimizar o desempenho da serialização.
- **Compactação** O WebSphere eXtreme Scale compacta todos os dados enviados entre os domínios de serviço de catálogo por padrão. Desativar a compactação não está disponível atualmente.
- **Ajuste de memória** O uso de memória para uma topologia de replicação multimestre é altamente independente do número de domínios de serviço de catálogo na topologia.

  A replicação multimaster inclui uma quantidade fixa de processamento por entrada de Mapa para manipular a versão. Cada contêiner também controla uma quantia fixa de dados em cada domínio de serviço de catálogo na topologia. Uma topologia com dois domínios de serviço de catálogo usa aproximadamente a mesma memória que uma topologia com 50 domínios de serviço de catálogo. O WebSphere eXtreme Scale não usa logs de reprodução ou filas semelhantes em

sua implementação. Assim, não há nenhuma estrutura de recuperação pronta caso um link de replicação esteja indisponível por um período prolongado e reinícios posteriores.

Tarefas relacionadas

Configurando Diversas Topologias do Datacenter
Com a replicação assíncrona multimestre, um conjunto de domínios de serviço de catálogo é vinculado. Os domínios de serviço de catálogo conectados são então sincronizado usando a replicação sobre os links. É possível definir os links usando arquivos de propriedades, no tempo de execução com programas Java Management Extensions (JMX) ou com utilitários de linha de comandos. O conjunto de links atuais para um domínio é armazenado no serviço de catálogo. É possível incluir e remover links sem reiniciar o domínio de serviço de catálogo que hospeda a grade de dados.

"Desenvolvendo Árbitros Customizados para a Replicação Multimestre" na página 304
Poderão ocorrer colisões de mudanças se os mesmos registros puderem ser alterados simultaneamente em dois locais. Em uma topologia de replicação multimestre, os domínios do serviço de catálogo detectam colisões automaticamente. Quando um domínio de serviço de catálogo detecta uma colisão, ele chama um árbitro. Geralmente, as colisões são resolvidas com o árbitro de colisão padrão. No entanto, um aplicativo pode fornecer um árbitro de colisão customizado.

Planejando para Desenvolver Aplicativos do WebSphere eXtreme Scale

Configure seu ambiente de desenvolvimento e aprenda onde localizar detalhes sobre as interfaces de programação disponíveis.

Visão Geral da API

O WebSphere eXtreme Scale fornece diversos recursos que são acessados programaticamente usando a linguagem de programação Java através de interfaces de programação de aplicativos (APIs) e interfaces de programação do sistema.

APIs do WebSphere eXtreme Scale

Quando estiver utilizando APIs do eXtreme Scale, você deve distinguir entre operações transacionais e não-transacionais. Uma operação transitória é uma operação executada dentro de uma transação. As APIs de ObjectMap, EntityManager, Query e DataGrid são APIs transacionais contidas no Session que é um contêiner transacional. As operações não-transitórias não estão relacionadas a uma transação, como por exemplo operações de configuração.

As APIs ObjectGrid, BackingMap e de plug-in não são transitórias. ObjectGrid, BackingMap e outras APIs de configuração são categorizadas como API Principal do ObjectGrid. Os plug-ins servem para customizar o cache para obter as funções desejadas e são categorizados como a API de Programação do Sistema. Um plug-in no eXtreme Scale é um componente que fornece um determinado tipo de função aos componentes conectáveis do eXtreme Scale que incluem ObjectGrid e BackingMap. Um recurso representa uma função ou característica específica de um componente do eXtreme Scale, incluindo ObjectGrid, Session, BackingMap, ObjectMap e assim por diante. Geralmente, os recursos são configuráveis com APIs de configuração. Os plug-ins podem ser internos, mas podem requerer o desenvolvimento de seus próprios plug-ins em algumas situações.

É possível configurar normalmente o ObjectGrid e o BackingMap para atender aos requisitos do seu aplicativo. Quando o aplicativo possui requisitos especiais, considere o uso de plug-ins especializados. O WebSphere eXtreme Scale pode ter plug-ins integrados que atendam aos seus requisitos. Por exemplo, se for necessário um modelo de replicação ponto a ponto entre duas instâncias do ObjectGrid ou duas grades distribuídas do eXtreme Scale, o JMSObjectGridEventListener integrado estará disponível. Se nenhum dos plug-ins internos puder resolver seus problemas de negócios, consulte a API de Programação do Sistema para fornecer seus próprios plug-ins.

ObjectMap é uma API baseada em mapa simples. Se os objetos armazenados em cache forem simples e nenhum relacionamento estiver envolvido, a API do ObjectMap será ideal para seu aplicativo. Se os relacionamentos de objetos estiverem envolvidos, use a API EntityManager, que suporta relacionamentos como gráfico.

Query é um mecanismo poderoso para localização de dados no ObjectGrid. Session e EntityManager fornecem o recurso de consulta tradicional.

A API do DataGrid é um recurso de computação poderoso em um ambiente distribuído do eXtreme Scale que envolve muitas máquinas, réplicas e partições. Os aplicativos podem executar lógica de negócios em paralelo a todos os nós do ambiente distribuído do eXtreme Scale. O aplicativo pode obter a API do DataGrid por meio da API do ObjectMap.

O serviço de dados REST do WebSphere eXtreme Scale é um serviço HTTP Java que é compatível com Microsoft WCF Data Services (formalmente, ADO.NET Data Services) e que implementa o Open Data Protocol (OData). O serviço de dados REST permite que qualquer cliente HTTP acesse uma grade do eXtreme Scale. Ele é compatível com o suporte do WCF Data Services fornecido com o Microsoft .NET Framework 3.5 SP1. Aplicativos RESTful podem ser desenvolvidos com um rico conjunto de ferramentas fornecido pelo Microsoft Visual Studio 2008 SP1. Para obter mais detalhes, consulte o Guia do Usuário do Serviço de Dados REST do eXtreme Scale.

Visão Geral de Plug-ins

Um plug-in do WebSphere eXtreme Scale é um componente que fornece um certo tipo de função para os componentes conectáveis que incluem ObjectGrid e BackingMap. O WebSphere eXtreme Scale fornece vários pontos de conexão para permitir que os aplicativos e provedores de cache se integrem com vários armazéns de dados, APIs de cliente alternativo e para melhorar o desempenho geral do cache. O produto é fornecido com vários plug-ins padrão pré-construídos, mas também é possível criar plug-ins customizados com o aplicativo.

Todos os plug-ins são classes concretas que implementam uma ou mais interfaces do plug-in eXtreme Scale. Tais classes são divididas em instâncias e chamadas pelo ObjectGrid nos momentos apropriados. O ObjectGrid e os BackingMaps permitem que plug-ins customizados sejam registrados.

Plug-ins do ObjectGrid

Os seguintes plug-ins estão disponíveis para uma instância do ObjectGrid. Se o plug-in for apenas do lado do servidor, os plug-ins serão removidas nas instâncias ObjectGrid e BackingMap do cliente. As instâncias do ObjectGrid e BackingMap são apenas do lado do servidor.

- **TransactionCallback**: Um plug-in TransactionCallback fornece eventos do ciclo de vida de transação. Se o plug-in TransactionCallback for a implementação de classe JPATxCallback integrada (com.ibm.websphere.objectgrid.jpa.JPATxCallback), o plug-in será apenas do lado do servidor. Entretanto, as subclasses da classe JPATxCallback não são apenas do lado do servidor.
- **ObjectGridEventListener**: Um plug-in ObjectGridEventListener fornece eventos de ciclo de vida ObjectGrid para o ObjectGrid, shards e transações.
- **ObjectGridLifecycleListener**: Um plug-in ObjectGridLifecycleListener fornece eventos de ciclo de vida ObjectGrid para a instância do ObjectGrid. O plug-in ObjectGridLifecycleListener pode ser usado como uma interface combinada opcional para todos os outros plug-ins do ObjectGrid.
- **ObjectGridPlugin**: Um ObjectGridPlugin é uma interface combinada opcional que fornece eventos de gerenciamento de ciclo de vida estendida para todos os outros plug-ins do ObjectGrid.
- **SubjectSource, ObjectGridAuthorization, SubjectValidation**: eXtreme Scale fornece vários terminais de segurança para permitir que mecanismos de autenticação customizados sejam integrados ao eXtreme Scale. (Apenas do lado do servidor)
- **MapAuthorization**: (Apenas do lado do servidor)

Requisitos Comuns do Plug-in do ObjectGrid

O ObjectGrid instancia e inicializa as instâncias de plug-in usando as convenções JavaBeans. Todas as implementações do plug-in apresentam os seguintes requisitos:

- A classe de plug-in deve ser uma classe pública de alto nível
- Ela deve apresentar um construtor público, sem argumentos.
- A classe de plug-in deve estar disponível no caminho de classe dos servidores e clientes (como apropriado).
- Os atributos devem ser definidos utilizando os métodos de propriedade de estilo do JavaBeans.
- Os plug-ins, exceto quando indicado de outra forma, são registrados antes da inicialização do ObjectGrid e não podem ser alterados depois de tal inicialização.

Plug-ins do BackingMap

Os seguintes plug-ins estão disponíveis para um BackingMap:

- **Evictor** - Um plug-in evictor é um mecanismo padrão fornecido para descartar entradas de cache e um plug-in para criar evictors customizados.
- **ObjectTransformer**: Um plug-in ObjectTransformer permite serializar, desserializar e copiar objetos no cache.
- **OptimisticCallback** - Utilize o plug-in OptimisticCallback para customizar as operações de versão e comparação dos objetos de cache ao utilizar a estratégia de bloqueio otimista.
- **MapEventListener** - Um plug-in MapEventListener fornece notificações de retorno de chamada e alterações de estado de cache significativas que ocorrem para um BackingMap.
- **BackingMapLifecycleListener**: Um plug-in BackingMapLifecycleListener fornece eventos de ciclo de vida BackingMap para a instância do BackingMap. O plug-in BackingMapLifecycleListener pode ser usado como uma interface combinada opcional para todos os outros plug-ins do BackingMap.

- **BackingMapPlugin**: Um BackingMapPlugin é uma interface combinada opcional que fornece eventos de gerenciamento de ciclo de vida estendida para todos os outros plug-ins do BackingMap.
- **Indexação** - Use o recurso de indexação, que é representado pelo plug-in MapIndexplug-in, para construir um ou mais índices em um mapa BackingMap para suportar acesso a dados sem chave.
- **Loader**: Um plug-in Loader em um mapa ObjectGrid atua como um cache de memória para os dados que são normalmente mantidos em um armazenamento persistente no mesmo sistema ou em outro sistema diferente. (Apenas do lado do servidor)

Visão Geral do Serviço de Dados REST

O serviço de dados REST WebSphere eXtreme Scale é um serviço HTTP Java compatível com Microsoft WCF Data Services (formalmente, ADO.NET Data Services) e implementa o Open Data Protocol (OData). O Microsoft WCF Data Services é compatível com essa especificação quando utiliza Visual Studio 2008 SP1 e .NET Framework 3.5 SP1.

Requisitos de Compatibilidade

O serviço de dados REST permite que qualquer cliente HTTP acesse uma grade de dados. O serviço de dados REST é compatível com o suporte do WCF Data Services fornecido com o Microsoft .NET Framework 3.5 SP1. Aplicativos RESTful podem ser desenvolvidos com um rico conjunto de ferramentas fornecido pelo Microsoft Visual Studio 2008 SP1. A figura fornece uma visão geral de como o WCF Data Services interage com clientes e bancos de dados.

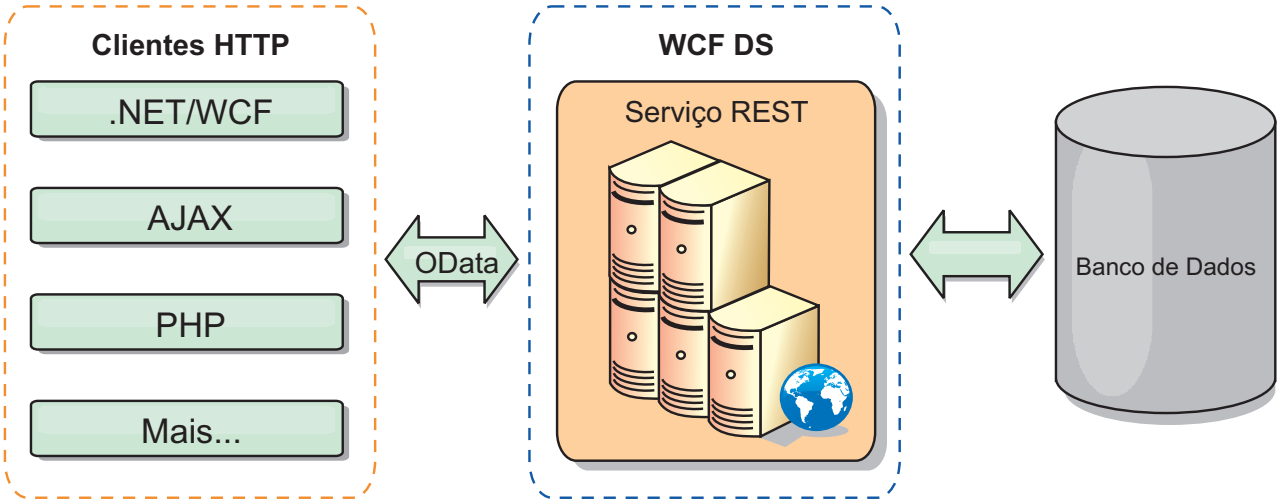


Figura 22. Microsoft WCF Data Services

O WebSphere eXtreme Scale inclui um conjunto de APIs com várias funções para clientes Java. Conforme mostrado na figura a seguir, o serviço de dados REST é um gateway entre clientes HTTP e a grade de dados do WebSphere eXtreme Scale, comunicando-se com a grade por meio de um cliente do WebSphere eXtreme Scale. O serviço de dados REST é um servlet Java, que permite implementações flexíveis para Plataforma Java comum, plataformas Enterprise Edition (JEE), como o WebSphere Application Server. O serviço de dados REST se comunica com a grade de dados WebSphere eXtreme Scale usando as APIs Java WebSphere eXtreme Scale. Ele permite clientes do WCF Data Services ou qualquer outro cliente que possa se

comunicar com HTTP e XML.

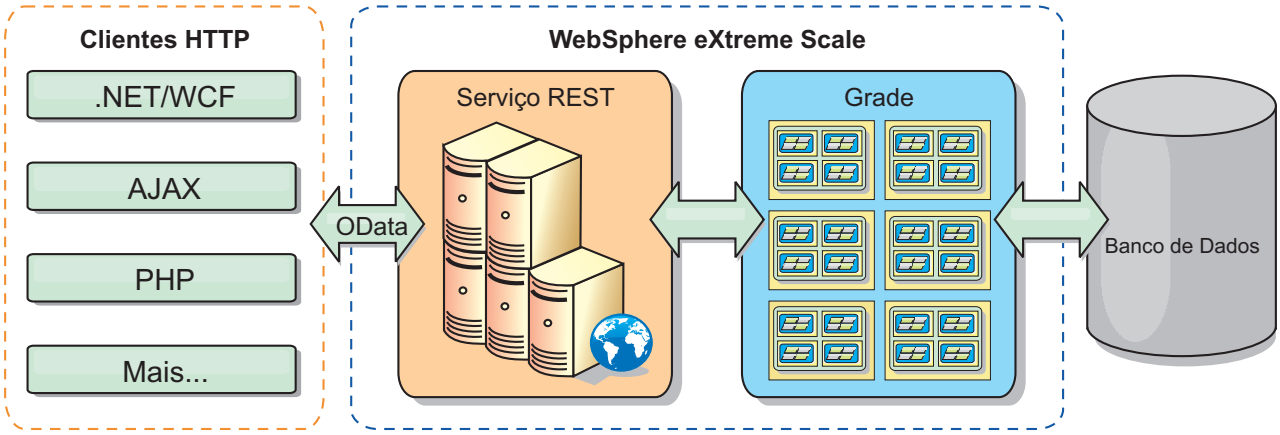


Figura 23. Serviço de Dados REST do WebSphere eXtreme Scale

Consulte o Configurando Serviços de Dados REST ou use os seguintes links para aprender mais sobre o WCF Data Services.

- Microsoft WCF Data Services Developer Center
- Visão geral do ADO.NET Data Services no MSDN
- White Paper: Utilizando ADO.NET Data Services
- Atom Publish Protocol: Data Services URI and Payload Extensions
- Conceptual Schema Definition File Format
- Entity Data Model for Data Services Packaging Format
- Open Data Protocol
- FAQ do Open Data Protocol

Características

Esta versão do serviço de dados REST do eXtreme Scale suporta os seguintes recursos:

- Modelagem automática de entidades de API EntityManager do eXtreme Scale como entidades do WCF Data Services, que inclui o seguinte suporte:
  - Tipo de dados Java para conversão de tipo do Entity Data Model
  - Suporte de associação de entidade
  - Suporte à raiz do esquema e à associação de chave, que é requerido para grades de dados particionadas

  Consulte o Modelo de Entidade para obter informações adicionais.
- Formato de carga útil de dados Atom Publish Protocol (AtomPub ou APP) XML e JavaScript Object Notation (JSON).
- Operações Create, Read, Update and Delete (CRUD) utilizando os respectivos métodos de pedido de HTTP: POST, GET, PUT e DELETE. Além disso, a extensão da Microsoft MERGE é suportada.
- Consultas simples, usando filtros
- Pedidos de recuperação de lote e do conjunto de mudanças
- Suporte de grade de dados particionada para alta disponibilidade
- Interoperabilidade com clientes de API EntityManager do eXtreme Scale
- Suporte para servidores da Web JEE padrão

- Simultaneidade otimista
- Autorização e autenticação de usuário entre o serviço de dados REST e a grade de dados do eXtreme Scale

Limitações e Problemas Conhecidos

- Pedidos em túnel não são suportados.

Tarefas relacionadas

Configurando Serviços de Dados REST
É possível usar o serviço de dados REST do WebSphere eXtreme Scale com o WebSphere Application Server versão 7.0, o WebSphere Application Server Community Edition e o Apache Tomcat.

"Acessando Dados com o Serviço de Dados REST" na página 268
Desenvolva aplicativos que executam operações usando protocolos do serviço de dados REST.

Referências relacionadas

"Simultaneidade Otimista no Serviço de Dados REST" na página 273
O serviço de dados REST do eXtreme Scale usa um modelo de bloqueio otimista ao usar cabeçalhos HTTP nativos: If-Match, If-None-Match e ETag. Esses cabeçalhos são enviados em mensagens de pedido e de resposta para retransmitir informações da versão da entidade do servidor para o cliente e do cliente para o servidor.

"Protocolos de Pedido para o Serviço de Dados REST" na página 274
No geral, os protocolos para interação com o serviço REST são os mesmos que os descritos no protocolo AtomPub de Serviços de Dados WCF. No entanto, o eXtreme Scale fornece detalhes adicionais, da perspectiva Modelo de Entidade do eXtreme Scale. Os usuários devem estar familiarizados com os protocolos WCF Data Services antes de lerem esta seção. Alternativamente, os usuários podem ler esta seção com a seção do protocolo WCF Data Services.

"Recuperar Pedidos com Serviço de Dados REST" na página 275
Um Pedido RetrieveEntity é usado por um cliente para recuperar uma entidade do eXtreme Scale. A carga útil da resposta contém os dados da entidade no formato AtomPub ou JSON. Além disso, o operador do sistema $expand pode ser utilizado para expandir as relações. As relações são representadas em sequência dentro da resposta do serviço de dados como um Atom Feed Document, que é uma relação para-muitos, ou um Atom Entry Document que é uma relação para-um.

"Recuperando Não Entidades com Serviços de Dados REST" na página 282
O serviço de dados REST permite recuperar mais que apenas entidades, como coletas e propriedades das entidades.

"Pedidos de Inserção com Serviço de Dados REST" na página 288
Um Pedido InsertEntity pode ser usado para inserir uma nova instância de entidade do eXtreme Scale, potencialmente com novas entidades relacionadas, no serviço de dados REST do eXtreme Scale.

"Pedidos de Atualização com Serviço de Dados REST" na página 292
O serviço de dados REST do WebSphere eXtreme Scale suporta pedidos de atualização para entidades, propriedades de primitivas de entidades e assim por diante.

"Pedidos de Exclusão com Serviços de Dados REST" na página 297
O serviço de dados REST do WebSphere eXtreme Scale pode excluir entidades, valores da propriedade e links.

Visão Geral da Estrutura Spring

O Spring é uma estrutura para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie as transações e configure clientes e servidores que compõem a grade de dados em memória implementada.

Transações Nativas Gerenciadas do Spring

O Spring fornece transações gerenciadas por contêiner que são similares a um servidor de aplicativos do Java Platform, Enterprise Edition. Porém, o mecanismo do Spring pode usar diferentes implementações. O WebSphere eXtreme Scale fornece a integração do gerenciador de transações que permite ao Spring para gerenciar os ciclos de vida da transação do ObjectGrid. Consulte o informações sobre transações nativas no *Guia de Programação* para obter detalhes.

Beans de Extensão Gerenciados do Spring e Suporte a Espaço de Nomes

Além disso, o eXtreme Scale se integra ao Spring para permitir que os beans de estilo do Spring definidos para pontos de extensão ou plug-ins. Este recurso fornece configurações mais sofisticadas e mais flexíveis para configuração dos pontos de extensão.

Além dos beans de extensão gerenciados do Spring, o eXtreme Scale fornece um espaço de nomes Spring chamado "objectgrid". Beans e implementações integradas são predefinidos neste espaço de nomes, o que facilita aos usuários configurar o eXtreme Scale.

Suporte ao Escopo Shard

Com a configuração do Spring estilo tradicional, um bean ObjectGrid pode se do tipo singleton ou prototype. O ObjectGrid também suporta um novo escopo chamado de escopo "shard". Se um bean for definido como escopo shard, então somente um bean será criado por shard. Todas as solicitações de beans com um ID ou IDs correspondentes a essa definição de bean no mesmo shard resulta nessa instância de bean específica sendo retornada pelo contêiner Spring.

O exemplo a seguir mostra que um bean com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl é definido com o escopo configurado para shard. Portanto, apenas uma instância da classe JPAPropFactoryImpl é criada por shard.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Fluxo da Web do Spring

O Fluxo da Web do Spring armazena seu estado de sessão em uma sessão HTTP por padrão. Se um aplicativo da web usar o eXtreme Scale para gerenciamento de sessões, o Spring armazenará automaticamente o estado com o eXtreme Scale. Além disso, a tolerância a falhas é ativada da mesma forma que a sessão.

Consulte as informações de gerenciamento de sessões HTTP no *Visão Geral do Produto* para obter detalhes adicionais.

compactando

As extensões Spring do eXtreme Scale estão no arquivo `ogspring.jar`. Este arquivo Java archive (JAR) deve estar no caminho de classe para o suporte ao Spring funcionar. Se um aplicativo Java EE que estiver em execução em um WebSphere Application Server Network Deployment aumentado pelo WebSphere Extended Deployment, coloque o arquivo `spring.jar` e seus arquivos associados nos módulos EAR (enterprise archive). Você também deve colocar o arquivo `ogspring.jar` no mesmo local.

Tarefas relacionadas

"Desenvolvendo Aplicativos com a Estrutura Spring" na página 414
Aprenda como integrar seus aplicativos eXtreme Scale com o Spring Framework popular.

"Iniciando um Servidor de Contêiner com o Spring" na página 424
É possível iniciar um servidor de contêiner usando beans de extensão gerenciados pelo Spring e o suporte ao namespace.

"Gerenciando Transações com o Spring" na página 417
O Spring é uma estrutura popular para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie transações do eXtreme Scale e configure clientes e servidores eXtreme Scale.

Referências relacionadas

"Beans de Extensão Gerenciados pelo Spring" na página 419
É possível declarar que os POJOs sejam usados como pontos de extensão no arquivo `objectgrid.xml`. Se você nomear os beans e, em seguida, especificar o nome de classe, o eXtreme Scale normalmente cria instâncias da classe especificada e usa essas instâncias como o plug-in. O WebSphere eXtreme Scale pode delegar para que o Spring aja como o bean factory para obter instâncias desses objetos de plug-in.

Arquivo XML descritor do Spring
Use um arquivo XML descritor do Spring para configurar e integrar o eXtreme Scale com o Spring.

Arquivo `objectgrid.xsd` Spring
Use o arquivo `objectgrid.xsd` Spring para integrar o eXtreme Scale ao Spring para gerenciar as transações do eXtreme Scale e configurar os clientes e servidores.

Considerações do Carregador de Classes e do Caminho de Classe

Como o eXtreme Scale armazena objetos Java no cache por padrão, você deve definir classes no caminho de classe sempre que os dados forem acessados.

Especificamente, os processos do cliente e de contêiner do eXtreme Scale devem incluir as classes ou arquivos JAR no caminho de classe ao iniciar o processo. Quando você projeta um aplicativo para uso com o eXtreme Scale, separe qualquer lógica de negócios dos objetos de dados persistentes.

Consulte Carregamento de Classe no centro de informações do WebSphere Application Server para obter mais informações.

Para obter as considerações dentro de uma configuração de Estrutura Spring, consulte a seção de pacotes no tópico sobre a integração com a estrutura Spring no *Guia de Programação*.

Gerenciamento de Relacionamentos

Linguagens orientadas a objetos como Java, e relacionamentos ou associações de suporte a bancos de dados relacionais. Os relacionamentos diminuem a quantidade de armazenamento através do uso de referências de objetos ou chaves estrangeiras.

Ao usar relacionamentos em uma grade de dados, os dados deverão ser organizados em uma árvore limitada. Um tipo de raiz deve existir na árvore e todos os filhos devem estar associados a apenas uma raiz. Por exemplo: Departamento pode ter muitos Funcionários e um Funcionário pode ter muitos Projetos. Porém um Projeto não pode ter muitos Funcionários pertencentes a

diferentes departamentos. Depois de uma raiz ser definida, todo o acesso a este objeto raiz e seus descendentes será gerenciado através da raiz. O WebSphere eXtreme Scale usa o código hash da chave do objeto raiz para escolher uma partição. Por exemplo:

```
partition = (hashCode MOD numPartitions).
```

Quando todos os dados para um relacionamento estiverem ligados a um única instância do objeto, toda a árvore pode ser co-localizada em uma única partição e pode ser acessada muito eficientemente usando uma transação. Se os dados englobarem múltiplos relacionamentos, então múltiplas partições devem estar envolvidas que envolvem chamadas remotas adicionais, o que pode levar a gargalos no desempenho.

Dados de Referência

Alguns relacionamentos incluem dados de consulta ou de referência como: `CountryName`. Para dados de consulta ou de referência, os dados devem existir em cada partição. Os dados podem ser acessados por qualquer chave raiz e o mesmo resultado é retornado. Os dados de referência como estes devem ser usados apenas nos casos em que os dados forem razoavelmente estáticas. Atualizar esses dados pode ser dispendioso porque eles precisam ser atualizados em cada partição. A API DataGrid é uma técnica comum para manter os dados de referência atualizados.

Custos e Benefícios de Normalização

A normalização dos dados usando os relacionamentos pode ajudar a reduzir a quantidade de memória usada pela grade de dados pois a duplicação dos dados é diminuída. Porém, em geral, quanto mais dados relacionais forem incluídos, menos eles irão expandir. Quando os dados são agrupados juntos, torna-se mais caro manter os relacionamentos e manter os tamanhos gerenciáveis. Como os dados das partições da grade baseiam-se na chave da raiz da árvore, o tamanho da árvore não é levado em consideração. Assim, se você tiver uma grande quantidade de relacionamentos para uma instância da árvore, a grade de dados poderá ficar desequilibrada, fazendo com que uma partição mantenha mais dados do que as outras.

Quando os dados forem não normalizados ou simplificados, os dados que normalmente seriam compartilhados entre os dois objetos são duplicados e cada tabela pode ser particionada de modo independente, oferecendo uma grade de dados muito mais equilibrada. Apesar disto aumentar a quantidade de memória usada, permite que o aplicativo escale pois uma única linha de dados pode ser acessada que pode ter todos os dados necessários. Isto é ideal para grades com maior quantidade de leituras pois a manutenção dos dados se torna mais cara.

Para obter informações adicionais, consulte Classificação de sistemas XTP e escalamento.

Gerenciamento de Relacionamentos Usando as APIs de Acesso a Dados

A API ObjectMap é a mais rápida, mais flexível e granular das APIs de acesso a dados, oferecendo uma abordagem transacional baseada em sessão no acesso aos dados na grade de mapas. A API ObjectMap permite que os clientes usem operações comuns, como create, read, update e delete (CRUD), para gerenciar pares de valores de chave de objetos na grade de dados distribuída.

Ao usar a API ObjectMap, os relacionamentos de objetos devem ser expressos pela incorporação da chave estrangeira para todos os relacionamentos no objeto-pai.

A seguir, está um exemplo.

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

A API EntityManager simplifica o gerenciamento de relacionamentos através da extração de dados persistentes a partir de objetos incluindo as chaves estrangeiras. Quando o objeto é posteriormente recuperado da grade de dados, o gráfico de relacionamentos é reconstruído, como no seguinte exemplo.

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

A API EntityManager é muito semelhante a outras tecnologias de persistência do objeto Java como JPA e Hibernate na qual ela sincroniza um gráfico de instâncias de objetos Java gerenciados com o armazenamento persistente. Nesse caso, o armazenamento persistente é uma grade de dados do eXtreme Scale, em que cada entidade é representada como um mapa e o mapa contém os dados da entidade em vez das instâncias do objeto.

Considerações-Chave sobre Cache

O WebSphere eXtreme Scale usa mapas hash para armazenar dados na grade, na qual um objeto Java é usado para a chave.

Diretrizes

Ao escolher uma chave, considere os seguintes requisitos.

- As chaves nunca podem ser alteradas. Se uma parte da chave precisar ser alterada, a entrada do cache deverá ser removida e reinserida.
- As chaves devem ser pequenas. Como as chaves são utilizadas em toda operação de acesso a dados, é recomendável manter a chave pequena para que ela possa ser serializada eficientemente e utilize menos memória.
- Implementa um bom hash e algoritmo de igualdade. Os métodos hashCode e equals(Object o) devem ser sempre substituídos para cada objeto-chave.
- Armazene o hashCode da chave. Se possível, armazene em cache o código hash na instância do objeto-chave para acelerar os cálculos de hashCode(). Como a chave é imutável, o hashCode deve ser armazenável em cache.
- Evite duplicar a chave no valor. Ao usar a API ObjectMap, é conveniente armazenar a chave dentro do objeto do valor. Quando isso é feito, os dados-chave são duplicados na memória.

Dados para Diferentes Fusos Horários

Ao inserir dados com os atributos calendar, java.util.Date e timestamp em um ObjectGrid, você deve garantir que esses atributos de data e hora sejam criados com base no mesmo fuso horário, principalmente quando implementados em vários servidores em vários fusos horários. Usar o mesmo objeto de data e hora baseado em fuso horário pode garantir que o aplicativo esteja protegido por fuso horário e que os dados possam ser consultados pelos predicados calendar, java.util.Date e timestamp.

Sem especificar explicitamente um fuso horário ao criar objetos de data e hora, o Java usa o fuso horário local e pode causar valores de data e hora inconsistentes nos clientes e servidores.

Considere um exemplo em uma implementação distribuída na qual o client1 está no fuso horário [GMT-0] e o client2 está no [GMT-6], e ambos querem criar um objeto java.util.Date com o valor '1999-12-31 06:00:00'. Então, o client1 criará o objeto java.util.Date com o valor '1999-12-31 06:00:00 [GMT-0]' e o client2 criará o objeto java.util.Date com o valor '1999-12-31 06:00:00 [GMT-6]'. Os objetos java.util.Date não são iguais porque o fuso horário é diferente. Um problema semelhante ocorre quando você pré-carrega os dados nas partições que residem em servidores em fusos horários diferentes se o fuso horário local for utilizado para criar objetos de data e hora.

Para evitar o problema descrito, o aplicativo pode escolher um fuso horário como [GMT-0] como fuso horário base para criar objetos calendar, java.util.Date e timestamp.

Configurando um Ambiente de Desenvolvimento Independente

Configure um ambiente de desenvolvimento integrado baseado no Eclipse para construir e executar um aplicativo Java SE com a versão independente do WebSphere eXtreme Scale.

Antes de Iniciar

Instale o produto WebSphere eXtreme Scale em um diretório novo ou vazio e aplique o fix pack acumulativo mais recente do WebSphere eXtreme Scale. Também é possível usar a versão de teste do WebSphere eXtreme Scale ao descompactar o arquivo zip. Para obter mais informações sobre a instalação, consulte o informações sobre a instalação do WebSphere eXtreme Scale ou do WebSphere eXtreme Scale Client independente no *Guia de Administração*.

Procedimento

- Configure o Eclipse para construir e executar um aplicativo SE Java com o WebSphere eXtreme Scale.
  1. Defina uma biblioteca de usuário para permitir que seu aplicativo faça referência a interfaces de programação de aplicativo do WebSphere eXtreme Scale.
     a. No Eclipse ou no ambiente do IBM® Rational Application Developer, clique em **Janela** > **Preferências**.
     b. Expanda a ramificação **Java** > **Caminho de Construção** e selecione **Bibliotecas de Usuário**. Clique em **Novo**.
     c. Selecione a biblioteca de usuário do eXtreme Scale. Clique em **Incluir JARs**.
        1) Procure por e selecione o arquivo `objectgrid.jar` ou `ogclient.jar` a partir do diretório *wxs_root*`/lib`. Clique em **OK**. Selecione o arquivo `ogclient.jar` se estiver desenvolvendo caches de aplicativos clientes ou locais em memória. Se estiver desenvolvendo e testando servidores eXtreme Scale, use o arquivo `objectgrid.jar`.
        2) Para incluir Javadoc nas APIs do ObjectGrid, selecione o local do Javadoc para o arquivo `objectgrid.jar` ou `ogclient.jar` incluído na etapa anterior. Clique em **Editar**. Na caixa do caminho do local do Javadoc, digite o seguinte endereço da Web:
           `http://www.ibm.com/developerworks/wikis/extremescale/docs/api/`

d. Clique em **OK** para aplicar as configurações e fechar a janela Preferências.

As bibliotecas do eXtreme Scale estão agora no caminho de construção para o projeto.

2. Inclua a biblioteca de usuário em seu projeto Java.
   a. No Package Explorer, clique com o botão direito do mouse no projeto e selecione **Propriedades**.
   b. Selecione a guia **Bibliotecas**.
   c. Clique em **Incluir Biblioteca**.
   d. Selecione **Biblioteca de Usuário**. Clique em **Next**.
   e. Selecione a biblioteca de usuário do eXtreme Scale configurada anteriormente.
   f. Clique em **OK** para aplicar as mudanças e fechar a janela Propriedades.

- Execute um aplicativo SE Java com o eXtreme Scale com Eclipse. Crie uma configuração de execução para executar seu aplicativo.
  1. Configure o Eclipse para construir e executar um aplicativo SE Java com o eXtreme Scale. No menu **Executar**, selecione **Executar Configurações**.
  2. Clique com o botão direito do mouse na categoria Aplicativo Java e selecione **Novo**.
  3. Selecione a nova configuração de execução, denominada *New_Configuration*.
  4. Configure o perfil.
     - **Projeto** (na página tabulada principal): *your_project_name*
     - **Classe Principal** (na página tabulada principal): *your_main_class*
     - **Argumentos VM** (na página tabulada de argumentos): -Djava.endorsed.dirs=`wxs_root/lib/endorsed`

     Problemas com os **Argumentos VM** ocorrem com frequência porque o caminho para `java.endorsed.dirs` deve ser um caminho absoluto sem variáveis ou atalhos.

     Outros problemas comuns de configuração envolvem o Object Request Broker (ORB). Você deve ver o seguinte erro. Consulte o Configurando um Object Request Broker Customizado para obter mais informações.

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

     Se você não tiver `objectGrid.xml` ou `deployment.xml` acessível ao aplicativo, poderá ver o seguinte erro:

     ```
     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. Clique em **Aplicar** e feche a janela ou clique em **Executar**.

Executando um Aplicativo Cliente ou do Servidor do WebSphere eXtreme Scale com o Apache Tomcat no Rational Application Developer

Independente se você tiver um aplicativo cliente ou do servidor, use as mesmas etapas básicas para executar o aplicativo no Apache Tomcat no Rational

Application Developer. Para um aplicativo cliente, você deseja configurar e executar um aplicativo da web para usar um cliente do WebSphere eXtreme Scale no Rational Application Developer. Siga estas instruções para criar um projeto da Web para executar um contêiner ou serviço de catálogo do WebSphere eXtreme Scale. Para um aplicativo do servidor, você deseja ativar um aplicativo Java EE na interface do Rational Application Developer com uma instalação independente do WebSphere eXtreme Scale. Siga estas instruções para configurar um projeto do aplicativo Java EE para usar a biblioteca do cliente do WebSphere eXtreme Scale.

Antes de Iniciar

Instale o Teste do WebSphere eXtreme Scale ou o produto integral.

- Instale a versão independente do produto WebSphere eXtreme Scale.
- Faça download e extraia a versão de teste do WebSphere eXtreme Scale.
- Instale o Apache Tomcat Versão 6.0 ou posterior.
- Instale o Rational Application Developer e crie um aplicativo da web Java EE.

Procedimento

1. Inclua a biblioteca de tempo de execução do WebSphere eXtreme Scale no caminho de construção do Java EE.

   Aplicativo cliente Neste cenário, você deseja configurar e executar um aplicativo da web para usar um cliente do WebSphere eXtreme Scale no Rational Application Developer.

   a. **Janela** > **Preferências** > **Java** > **Caminho de Construção** > **Bibliotecas de Usuário**. Clique em **Novo**.
   b. Digite um **Nome da biblioteca de usuário** de `eXtremeScaleClient` e clique em **OK**.
   c. Clique em **Incluir Jars...**, navegue para o arquivo `wxs_home/lib/ogclient.jar` e selecione-o. Clique em **Abrir**.
   d. Opcional: (Opcional) Para incluir Javadoc, selecione o local Javadoc e clique em **Editar....** No caminho do local Javadoc, é possível digitar a URL da documentação da API ou pode fazer download da documentação da API.
      - Para usar a documentação da API on-line, digite `http://www.ibm.com/developerworks/wikis/extremescale/docs/api/` no caminho do local Javadoc.
      - Para fazer download da documentação da API, vá para a WebSphere eXtreme Scale Página de download de documentação da API. Para o caminho do local Javadoc, digite seu local de download local.
   e. Clique em **OK**.
   f. Clique em **OK** para fechar o diálogo Bibliotecas de Usuário.
   g. Clique em **Projeto** > **Propriedades**.
   h. Clique em **Caminho de Construção Java**.
   i. Clique em **Incluir Biblioteca**.
   j. Selecione **Biblioteca de Usuário**. Clique em **Avançar**.
   k. Marque a biblioteca **eXtremeScaleClient** e clique em **Concluir**.
   l. Clique em **OK** para fechar o diálogo **Propriedades do Projeto**.

   Aplicativo do servidor Neste cenário, você deseja configurar e executar um aplicativo da web para executar um servidor WebSphere eXtreme Scale integrado no Rational Application Developer.

a. Clique em **Janela** > **Preferências** > **Java** > **Caminho de Construção** > **Bibliotecas de Usuário**. Clique em **Novo**.
b. Digite um **Nome de biblioteca de usuário** de `eXtremeScale` e clique em **OK**.
c. Clique em **Incluir Jars...** e selecione *wxs_home*`/lib/objectgrid.jar`. Clique em Abrir.
d. (Opcional) Para incluir Javadoc, selecione o local Javadoc e clique em **Editar....** No caminho do local Javadoc, digite `http://www.ibm.com/developerworks/wikis/extremescale/docs/api/`.
e. Clique em **OK**.
f. Clique em **OK** para fechar o diálogo Bibliotecas de Usuário.
g. Clique em **Projeto** > **Propriedades**.
h. Clique em **Caminho de Construção Java**.
i. Clique em **Incluir Biblioteca**.
j. Selecione **Biblioteca de Usuário**. Clique em **Avançar**.
k. Marque a biblioteca **eXtremeScaleClient** e clique em **Concluir**.
l. Clique em **OK** para fechar o diálogo **Propriedades do Projeto**.

2. Defina Tomcat Server para nosso projeto.
   a. Certifique-se de estar na perspectiva J2EE e clique na guia **Servidores** na área de janela inferior. Você também pode clicar em **Janela** > **Mostrar Visualização** > **Servidores**.
   b. Clique com o botão direito do mouse na área de janela Servidores e escolha **Novo** > **Servidor**.
   c. Escolha **Apache, Tomcat v6.0 Server**. Clique em **Avançar**.
   d. Clique em **Procurar** e selecione *tomcat_root*. Clique em **OK**.
   e. Clique em **Avançar**.
   f. Selecione seu aplicativo Java EE na área de janela Disponível à esquerda, clique em **Incluir >** para movê-lo para a área de janela Configurado à direita no servidor e clique em **Concluir**.
3. Resolva quaisquer erros restantes para o Projeto. Use as seguintes etapas para eliminar erros na área de janela Problemas:
   a. Clique em **Projeto** > **Limpar** > ***project_name***. Clique em **OK**. Construa o projeto.
   b. Clique com o botão direito no projeto Java EE e escolha **Caminho de Construção** > **Configurar Caminho de Construção**.
   c. Clique na guia **Bibliotecas**. Certifique-se de que o caminho seja configurado adequadamente:
      - **Para aplicativos cliente:** Certifique-se de que o Apache Tomcat, o eXtremeScaleClient e o Java 1.5 JRE estejam no caminho.
      - **Para aplicativos de servidor:** Certifique-se de que o Apache Tomcat, o eXtremeScale e o Java 1.5 JRE estejam no caminho.
4. Crie uma configuração de execução para executar seu aplicativo.
   a. No menu **Executar**, selecione **Executar Configurações**.
   b. Clique com o botão direito do mouse na categoria Aplicativo Java e selecione **Novo**.
   c. Selecione a nova configuração de execução, denominada *New_Configuration*.
   d. Configure o perfil.
      - **Projeto** (na página tabulada principal): *your_project_name*

- **Classe Principal** (na página tabulada principal): *your_main_class*
- **Argumentos VM** (na página tabulada de argumentos): `-Djava.endorsed.dirs=wxs_root/lib/endorsed`

Problemas com os **Argumentos VM** ocorrem com frequência porque o caminho para `java.endorsed.dirs` deve ser um caminho absoluto sem variáveis ou atalhos.

Outros problemas comuns de configuração envolvem o Object Request Broker (ORB). Você deve ver o seguinte erro. Consulte o Configurando um Object Request Broker Customizado para obter mais informações:

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

Se você não tiver os arquivos `objectGrid.xml` ou `deployment.xml` acessíveis ao aplicativo, poderá ver o seguinte erro:

```
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. Clique em **Aplicar** e feche a janela ou clique em **Executar**.

O que Fazer Depois

Depois de configurar e de executar um aplicativo da web com o cliente do WebSphere eXtreme Scale no Rational Application Developer, será possível desenvolver um servlet. Esse servlet usa as APIs do WebSphere eXtreme Scale para armazenar e recuperar dados de uma grade de dados remota.

Depois de ativar um aplicativo Java EE na interface do Rational Application Developer com uma instalação independente do WebSphere eXtreme Scale, será possível desenvolver um servlet que use as APIs de sistema do WebSphere eXtreme Scale para iniciar e parar serviços de catálogo.

Executando um Cliente ou um Servidor de Aplicativos Integrado com o WebSphere Application Server no Rational Application Developer

Configure e execute um aplicativo Java EE com um cliente ou servidor WebSphere eXtreme Scale com o tempo de execução integrado do WebSphere Application Server no Rational Application Developer. Se estiver configurando um servidor, iniciar o WebSphere Application Server inicia automaticamente o WebSphere eXtreme Scale.

Antes de Iniciar

As etapas a seguir são para o WebSphere Application Server Versão 7.0 com Rational Application Developer Versão 7.5. As etapas a seguir poderão variar se você estiver usando versões diferentes desses produtos.

Instale o Rational Application Developer com as extensões do Ambiente de Teste do WebSphere Application Server.

Instale o cliente ou servidor do WebSphere eXtreme Scale no Ambiente de Teste do WebSphere Application Server Versão 7.0 no diretório *rad_home*`\runtimes\base_v7`.

Consulte Instalando o WebSphere eXtreme Scale ou o WebSphere eXtreme Scale Client com WebSphere Application Server para obter mais informações.

Procedimento

1. Defina o servidor eXtreme Scale que está integrado ao WebSphere Application Server para seu projeto.
   a. Na perspectiva J2EE, clique em **Janela > Mostrar Visualização > Servidores.**
   b. Clique com o botão direito do mouse na área de janela **Servidores**. Escolha **Novo > Servidor**.
   c. Escolha **IBM WebSphere Application Server v7.0**. Clique em Avançar.
   d. Selecione um perfil para usar. O padrão é was70profile1.
   e. Digite o nome do servidor. O padrão é server1.
   f. Clique em **Next**.
   g. Selecione seu aplicativo Java EE na área de janela **Disponível**. Clique em **Incluir >** para movê-lo para a área de janela **Configurado** no servidor. Clique em **Finalizar**.
2. Para executar o aplicativo Java EE, inicie o servidor de aplicativos. Clique com o botão direito do mouse no **WebSphere Application Server v7.0** e selecione **Iniciar**.

Capítulo 5. Desenvolvendo Aplicativos

Desenvolva um aplicativo que use a grade de dados. As tarefas para desenvolver aplicativos incluem:

- Acessar Dados
- APIs e Plug-ins do Sistema
- Integração do JPA
- Integração do Spring

Acessando Dados com Aplicativos Cliente

Após configurar seu ambiente de desenvolvimento, é possível começar a desenvolver aplicativos que criam, acessam e gerenciam os dados em sua grade de dados.

Sobre Esta Tarefa

Da perspectiva de um aplicativo cliente, usar o WebSphere eXtreme Scale envolve as seguintes etapas principais:

- Conexão com o serviço de catálogo por meio da obtenção de uma instância de ClientClusterContext.
- Obtenção de uma instância do ObjectGrid do cliente.
- Obtenção de uma instância da Sessão.
- Obtenção de uma instância do ObjectMap.
- Uso de métodos ObjectMap.

Conectando-se às Instâncias do ObjectGrid Distribuído Programaticamente

É possível se conectar a um ObjectGrid distribuído com um terminal de conexão para o domínio de serviço de catálogo. É necessário ter o nome do host e a porta do listener de cada servidor de catálogos no domínio de serviço de catálogo com o qual deseja se conectar.

Antes de Iniciar

- Para se conectar com uma grade de dados distribuída, você deve configurar o ambiente do lado do servidor com um serviço de catálogo e com os servidores de contêiner.
- Você deve ter a porta de listener para cada serviço de catálogo. Consulte o Planejamento para Portas de Rede para obter informações adicionais.

Sobre Esta Tarefa

O método getObjectGrid(ClientClusterContext ccc, String objectGridName) conecta-se com o domínio de serviço de catálogo especificado e retorna uma instância ObjectGrid de cliente correspondente a uma instância ObjectGrid do lado do servidor. As etapas variam, dependendo se você estiver usando uma configuração independente ou o WebSphere Application Server.

Procedimento

- Conecte-se a uma grade de dados distribuída independente usando terminais de serviço de catálogo explícitos.

```
// Create an ObjectGridManager instance.

    ObjectGridManager ogm = ObjectGridManagerFactory.getObjectGridManager();

    // Obtain a ClientClusterContext by connecting to a catalog
    // server based distributed ObjectGrid.  You have to provide
    // a connection end point for your catalog server in the format
    // of hostName:endPointPort.  The hostName is the machine
    // where the catalog server resides, and the endPointPort is
    // the catalog server's listening port, whose default is 2809.
    // Catalog server end-points for a given domain must be in
    // the format of a comma-delimited list.

        String catalogServiceEndpoints = "host1:2809,host2:2809";
        ClientClusterContext ccc = ogm.connect
  (catalogServiceEndpoints, null, null);

    // Obtain a distributed ObjectGrid using ObjectGridManager and providing
    // the ClientClusterContext.

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

- Conecter-se a um domínio de serviço de catálogo a partir de um aplicativo cliente hospedado no WebSphere Application Server, onde o domínio de serviço de catálogo foi configurado usando o console administrativo ou a tarefa administrativa e onde os terminais de serviço de catálogo podem ser recuperados usando a API do servidor integrada:

```
...

// Retrieve the catalog service endpoints from the ServerPropeties
// singleton, which is configured in the WebSphere administration
// console or admin task.

String catalogServiceEndpoints = ServerFactory.getServerProperties()
  .getCatalogServiceBootstrap();
ClientClusterContext ccc = ogm.connect(catalogServiceEndpoints,
  null, null);

...
```

  Se o domínio do serviço de catálogo no WebSphere Application Server for hospedado pelo gerenciador de implementação, os clientes fora da célula, incluindo clientes do Java Platform, Standard Edition, deverão se conectar ao serviço de catálogo usando o nome do host do gerenciador de implementação e a porta de autoinicialização IIOP. Quando o serviço de catálogo for executado nas células do WebSphere Application Server enquanto os clientes forem executados fora das células, consulte as páginas de configuração de domínio do eXtreme Scale no console administrativo do WebSphere Application Server para obter as informações necessárias para apontar um cliente para o serviço de catálogo.

Rastreando Atualizações de Mapas por um Aplicativo

Quando um aplicativo está fazendo alterações em um Mapa durante uma transação, um objeto LogSequence rastreia estas alterações. Se o aplicativo alterar uma entrada no mapa, um objeto LogElement correspondente fornecerá os detalhes da mudança.

Os Loaders recebem um objeto LogSequence para um mapa específico sempre que um aplicativo solicita uma limpeza ou confirmação da transação. O Utilitário de Carga é repetido pelos objetos LogElement, no objeto LogSequence, e aplica cada objeto LogElement ao backend.

Os listeners ObjectGridEventListener registrados com um ObjectGrid também utilizam os objetos LogSequence. Estes listeners recebem um objeto LogSequence para cada mapa em uma transação confirmada. Os aplicativos podem utilizar estes listeners para esperar alterações de algumas entradas, como um acionador em um banco de dados convencional.

As seguintes interfaces ou classes relacionadas ao log são fornecidas pela estrutura do 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

Interface LogElement

Um LogElement representa uma operação em uma entrada durante uma transação. Um objeto LogElement possui vários métodos para obter os vários atributos. Os atributos mais usados são de tipo e os atributos de valor atual procurados por getType() e getCurrentValue().

O tipo é representado por uma das seguintes constantes definidas na interface LogElement: INSERT, UPDATE, DELETE, EVICT, FETCH ou TOUCH.

O valor atual representa o novo valor para a operação INSERT, UPDATE ou FETCH. Se a operação for TOUCH, DELETE ou EVICT, o valor atual será nulo. Este valor pode ser lançado no ValueProxyInfo quando um ValueInterface estiver sendo utilizado.

Consulte a documentação da API para obter mais detalhes sobre a interface LogElement.

Interface LogSequence

Na maioria das transações, ocorrem operações em mais de uma entrada em um mapa, portanto, são criados vários objetos LogElement. É necessário criar um objeto que atue como uma composição de vários objetos LogElement. A interface LogSequence atende esta finalidade contendo uma lista de objetos LogElement.

Consulte a documentação da API para obter mais detalhes sobre a interface LogSequence.

Usando o LogElement e o LogSequence

O LogElement e o LogSequence são amplamente utilizados no eXtreme Scale e por plug-ins do ObjectGrid que são gravados por usuários quando operações são propagadas de um componente ou servidor para outro componente ou servidor. Por exemplo, um objeto LogSequence pode ser utilizado pela função de propagação de transação do ObjectGrid distribuído para propagar as alterações para outros servidores ou pode ser aplicado ao armazenamento de persistência pelo loader. LogSequence é utilizado principalmente pelas seguintes interfaces.

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

Exemplo de Utilitário de Carga

Esta seção demonstra como os objetos LogSequence e LogElement são utilizados em um Utilitário de Carga. Um Utilitário de Carga é utilizado para fins de carregamento ou persistência de dados num armazenamento persistente. O método batchUpdate da interface do Utilitário de Carga utiliza o objeto LogSequence:

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

O método batchUpdate é chamado quando um ObjectGrid precisa aplicar todas as alterações atuais no Utilitário de Carga. O Utilitário de Carga recebe uma lista de objetos LogElement para o mapa, encapsulados em um objeto LogSequence. A implementação do método batchUpdate deve ser repetida nas alterações e aplicada ao backend. O trecho de código a seguir demonstra como um Utilitário de Carga utiliza um objeto LogSequence. O fragmento é repetido no conjunto de mudanças e constrói três instruções de Java Database Connectivity (JDBC) em lote: inserts, updates e deletes:

```
public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException {
    // Obter uma conexão SQL para utilizar.
    Connection conn = getConnection(tx);
    try
    {
    // Processar a lista de alterações e construir um conjunto de
 instruções preparadas
    // para executar uma operação SQL update, insert ou delete
    // . As instruções são armazenadas em cache em 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;
        }
    }
    // Executar as instruções de lote que foram construídas pelo loop acima.
    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;
}
}
```

O exemplo anterior mostra a lógica de alto nível de processamento do argumento LogSequence. Entretanto, o exemplo não mostra os detalhes de como construir uma instrução SQL insert, update ou delete. O método getPendingChanges é chamado no argumento LogSequence para obter um iterador dos objetos LogElement que o Utilitário de Carga precisa para processar e o método LogElement.getType().getCode() é usado para determinar se um LogElement destina-se para uma operação SQL insert, update ou delete.

Amostra de Evictor

Também é possível usar os objetos LogSequence e LogElement com um Evictor. Um Evictor é utilizado para liberar as entradas do mapa de suporte com base em alguns critérios. O método apply da interface do Evictor utiliza LogSequence.

```
/**
* É chamado durante a confirmação de cache para permitir que
o evictor rastreie o uso de objetos
* em um mapa de apoio. Também relatará as entradas que foram liberadas com
* êxito.
*
* Sequência @param LogSequence de alterações no mapa
*/
void apply(LogSequence sequence);
```

Interfaces LogSequenceFilter e LogSequenceTransformer

Às vezes, é necessário filtrar os objetos LogElement para que apenas os objetos LogElement com alguns critérios sejam aceitos e rejeitar outros objetos. Por exemplo, você pode serializar um determinado LogElement com base em algum critério.

LogSequenceFilter resolve este problema com o seguinte método:

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

Este método retorna true se o LogElement especificado tiver que ser utilizado na operação e retorna false se o LogElement especificado não tiver que ser utilizado.

LogSequenceTransformer é uma classe que utiliza a função LogSequenceFilter. Utiliza LogSequenceFilter para filtrar alguns objetos LogElement e, em seguida, serializar os objetos LogElement aceitos. Esta classe possui dois métodos. O primeiro método é o seguinte:

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

Este método permite que o responsável pela chamada forneça um filtro para determinar quais LogElements incluir no processo de serialização. O parâmetro DistributionMode permite que o responsável pela chamada controle o processo de serialização. Por exemplo, se o modo de distribuição for apenas de invalidação, não será necessário serializar o valor. O segundo método desta classe é o método inflate, da seguinte forma:

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

Esse método lê o formulário serializado de sequência de log, que foi criado pelo método serialize a partir do fluxo de entrada de objetos fornecido.

Interagindo com um ObjectGrid Usando a Interface ObjectGridManager

A classe ObjectGridManagerFactory e a interface ObjectGridManager fornecem um mecanismo para criar, acessar e incluir dados para instâncias do ObjectGrid. A classe ObjectGridManagerFactory é uma classe auxiliar estática para acessar a interface ObjectGridManager, que é um singleton. A interface ObjectGridManager inclui vários métodos de conveniência para criar instâncias de um objeto do ObjectGrid. A interface ObjectGridManager também facilita a criação e armazenamento em cache de instâncias do ObjectGrid que podem ser acessadas por vários usuários.

Criando Instâncias do ObjectGrid com a Interface ObjectGridManager

Cada um desses métodos cria uma instância local de um ObjectGrid.

Instância na Memória Local

O trecho de código a seguir ilustra como obter e configurar uma instância de ObjectGrid local com o eXtreme Scale.

```
// Obtain a local ObjectGrid reference
        // you can create a new ObjectGrid, or get configured ObjectGrid
        // defined in ObjectGrid xml file
        ObjectGridManager objectGridManager =
     ObjectGridManagerFactory.getObjectGridManager();
        ObjectGrid ivObjectGrid =
     objectGridManager.createObjectGrid("objectgridName");

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

        // Define a BackingMap
        // if the BackingMap is configured in ObjectGrid xml
        // file, you can just get it.
        BackingMap ivBackingMap = ivObjectGrid.defineMap("myMap");

        // Add a Loader into BackingMap
        Loader ivLoader = new HeapCacheLoader();
        ivBackingMap.setLoader(ivLoader);

        // initialize ObjectGrid
        ivObjectGrid.initialize();

        // Obtain a session to be used by the current thread.
        // Session can not be shared by multiple threads
        Session ivSession = ivObjectGrid.getSession();

        // Obtaining ObjectMap from ObjectGrid Session
        ObjectMap objectMap = ivSession.getMap("myMap");
```

Configuração Compartilhada Padrão

O código a seguir é um caso simples de como criar um ObjectGrid para compartilhar entre muitos usuários.

```
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.
/*sample continues..*/
```

O trecho de código Java precedente cria e armazena em cache o Employees ObjectGrid. O Employees ObjectGrid é inicializado com a configuração padrão e já está pronto para uso. O segundo parâmetro no método createObjectGrid é configurado como true, o que instrui o ObjectGridManager a armazenar em cache a instância do ObjectGrid que ele cria. Se este parâmetro for configurado como false, a instância não é armazenada em cache. Cada instância do ObjectGrid possui um nome e a instância pode ser compartilhada entre vários clientes ou usuários com base em tal nome.

Se a instância do objectGrid for utilizada no compartilhamento ponto a ponto, o armazenamento em cache deve ser configurado como true. Para obter informações adicionais sobre o compartilhamento ponto a ponto, consulte Distribuição de alterações entre as Java Virtual Machines de peer.

Configuração XML

O WebSphere eXtreme Scale é altamente configurável. O exemplo anterior demonstra como criar um ObjectGrid simples sem nenhuma configuração. Este exemplo mostra como criar uma instância do ObjectdGrid pré-configurada que é baseada em um arquivo de configuração XML. É possível configurar uma instância do ObjectGrid programaticamente ou utilizando um arquivo de configuração baseado em XML. Também é possível configurar o ObjectGrid utilizando uma combinação de duas abordagens. A interface ObjectGridManager permite a criação de uma instância do ObjectGrid baseada na configuração XML. A interface ObjectGridManager possui vários métodos que utilizam uma URL como argumento. Cada arquivo XML transmitido para o ObjectGridManager deve ser validado no esquema. A validação XML pode ser desativada apenas quando o arquivo está previamente validado e nenhuma alteração foi feita no arquivo desde sua última validação. A desativação da validação poupa uma pequena quantidade de sobrecarga, mas introduz a possibilidade de utilizar um arquivo XML inválido. O IBM Java Developer Kit (JDK) Versão 5 possui suporte para validação de XML. Ao utilizar um JDK que não tem este suporte, o Apache Xerces pode ser requerido para validar o XML.

O trecho de código Java a seguir demonstra como transmitir um arquivo de configuração XML para criar um 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; // ativar a validação XML
boolean cacheInstance = true; // Armazenar a instância em cache
String objectGridName="Employees"; // Nome da URL do Object Grid
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
```

```
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees =
oGridManager.createObjectGrid(objectGridName, allObjectGrids,
 bvalidateXML, cacheInstance);
```

O arquivo XML pode conter informações de configuração para vários ObjectGrids. O trecho de código anterior retorna especificamente o ObjectGrid Employees, supondo que a configuração de Employees esteja definida no arquivo.

Métodos createObjectGrid

```
.
/**
 * Um método de depósito de informações do provedor simples para retornar uma instância de um
 * Object Grid. É designado um nome exclusivo.
 * A instância do ObjectGrid não é armazenada em cache.
 * Os usuários podem então utilizar {@link ObjectGrid#setName(String)} para alterar o
 * nome do ObjectGrid.
 *
 * @return ObjectGrid uma instância do ObjectGrid com um nome exclusivo designado
 * @throws ObjectGridException qualquer erro encontrado durante a
 * criação do ObjectGrid
 */
public ObjectGrid createObjectGrid() throws ObjectGridException;

/**
 * Um método de depósito de informações do provedor simples para
 retornar uma instância de um ObjectGrid com
 * o nome especificado. As instâncias do ObjectGrid podem
 ser armazenadas em cache. Se um ObjectGrid
 * com este nome já tiver sido armazenado em cache, será emitida uma
 * ObjectGridException.
 *
 * @param objectGridName o nome do ObjectGrid a ser criado.
 * @param cacheInstance true, se a instância do ObjectGrid tiver
 que ser armazenada em cache
 * @return an ObjectGrid instance
 * @this o nome já foi armazenado em cache ou
 * qualquer erro durante a criação do ObjectGrid.
 */
public ObjectGrid createObjectGrid(String objectGridName, boolean cacheInstance)
  throws ObjectGridException;

/**
 * Crie uma instância do ObjectGrid com o nome do ObjectGrid especificado. The
 * instância do ObjectGrid criada será armazenada em cache.
 * @param objectGridName o Nome da instância do ObjectGrid a ser criada.
 * @return an ObjectGrid instance
 * @throws ObjectGridException se um ObjectGrid com este nome já tiver sido
 * armazenado em cache ou algum erro encontrado durante a criação do ObjectGrid
 */
public ObjectGrid createObjectGrid(String objectGridName)
  throws ObjectGridException;

/**
 * Crie uma instância do ObjectGrid com base no nome do ObjectGrid e no
 * XML do cluster. A instância do ObjectGrid definida no arquivo XML com o nome do
 * ObjectGrid especificado será criada e retornada. Se tal ObjectGrid
 * não puder ser localizado no arquivo xml, será emitida uma exceção.
 *
 * Esta instância do ObjecGrid não pode ser armazenada em cache.
 *
 * Se a URL for nula, ela simplesmente será ignorada.
 * Neste caso, este método se comportará
 * igual a {@link #createObjectGrid(String, boolean)}.
```

```
 *
 * @param objectGridName o Nome da instância do ObjectGrid a ser retornada. Ela
 * não deve ser nula.
 * @param xmlFile uma URL para um arquivo xml bem formado baseado
 no esquema do ObjectGrid.
 * @param enableXmlValidation se true o XML será validado
 * @param cacheInstance um valor booleano que indica se a(s) instância(s) do
 * ObjectGrid
 * definida(s) no arquivo XML será(ão) ou não armazenada(s) em cache.
 Se true, a(s) instância(s) será(ão)
 * armazenada(s) em cache.
 *
 * @throws ObjectGridException se um ObjectGrid com o mesmo nome
 * tiver sido armazenado em cache anteriormente, nenhum nome do
 ObjectGrid poderá ser localizado no arquivo xml
 * ou qualquer outro erro durante a criação do ObjectGrid.
 * @return an ObjectGrid instance
 * @see ObjectGrid
 */
public ObjectGrid createObjectGrid(String objectGridName, final URL xmlFile,
final boolean enableXmlValidation, boolean cacheInstance)
 throws ObjectGridException;

/**
 * Processar um arquivo XML e criar uma Lista de objetos do
 * ObjectGrid com base no arquivo.
 * Estas instâncias do ObjecGrid podem ser armazenadas em cache.
 * Será emitida uma ObjectGridException ao tentar armazenar em cache
 * um ObjectGrid recém-criado que
 * tenha o mesmo nome que um ObjectGrid já armazenado em cache.
 *
 * @param xmlFile o arquivo que define um ObjectGrid ou vários
 * ObjectGrids
 * @param enableXmlValidation a configuração como true validará o arquivo XML
 * no esquema
 * @param cacheInstances configurado como true para armazenar
 em cache todas as instâncias do ObjectGrid
 * criadas com base no arquivo
 * @return an ObjectGrid instance
 * @throws ObjectGridException ao tentar criar e armazenar em cache um
 * ObjectGrid com o mesmo nome que um
 * ObjectGrid já armazenado em cache ou qualquer outro erro
 * ocorrido durante a
 * criação do ObjectGrid
 */
public List createObjectGrids(final URL xmlFile, final boolean
enableXmlValidation, boolean cacheInstances) throws
ObjectGridException;

/** Criar todos os ObjectGrids localizados no arquivo XML.
 * O arquivo XML será validado no esquema. Cada instância do
 ObjectGrid criada será armazenada(s) em cache. Será emitida uma
 ObjectGridException ao tentar armazenar em cache um
 * ObjectGrid recém-criado com o mesmo nome que um ObjectGrid
 * já armazenado em cache.
 * @param xmlFile O arquivo XML a ser processado.
 Os ObjectGrids serão criados com base
 * no conteúdo do arquivo.
 * @return Uma Lista de instâncias do ObjectGrid que foram criadas.
 * @throws ObjectGridException se um ObjectGrid que tenha o
 mesmo nome que qualquer um dos
 * localizados no XML já tiver sido armazenado em cache ou
 * qualquer outro erro encontrado durante a criação do ObjectGrid.
 */
public List createObjectGrids(final URL xmlFile) throws ObjectGridException;

/**
```

```
 * Processar o arquivo XML e criar uma única instância do ObjectGrid com o
 * objectGridName especificado apenas se um ObjectGrid com esse nome for localizado
 * no arquivo. Se não houver nenhum ObjectGrid com este nome definido no arquivo XML,
 * será emitida uma
 * ObjectGridException. A instância do ObjectGrid criada será armazenada em cache.
 * @param objectGridName nome do ObjectGrid a ser criado. Este ObjectGrid
 * deve ser definido no arquivo XML.
 * @param xmlFile o arquivo XML a ser processado
 * @return Um ObjectGrid recém-criado
 * @throws ObjectGridException se um ObjectGrid com o mesmo nome tiver sido
 * armazenado em cache anteriormente, nenhum nome do ObjectGrid poderá
 ser localizado no arquivo xml
 * ou qualquer outro erro durante a criação do ObjectGrid.
 */
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
  throws ObjectGridException;
```

Tarefas relacionadas

"Resolução de Problemas de Conectividade do Cliente" na página 513
Há vários problemas comuns específicos para clientes e de conectividade do cliente que podem ser resolvidos conforme descrito nas seções a seguir.

Recuperando Dados em Cache com a Interface ObjectGridManager

Use os métodos ObjectGridManager.getObjectGrid para recuperar instâncias de cache.

Recuperando uma Instância de Cache

Como a instância Employees ObjectGrid foi armazenada em cache pela interface ObjectGridManager, outro usuário pode acessá-la com o seguinte trecho de código:

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

A seguir estão os dois métodos getObjectGrid que retornam instâncias de ObjectGrid em cache:

- **Recuperando Todas as Instâncias em Cache**

  Para obter todas as instâncias do ObjectGrid que foram previamente armazenadas em cache, use o método getObjectGrids, que retorna uma lista de cada instância. Se nenhuma instância em cache existir, o método retornará null.
- **Recuperando uma Instância em Cache por Nome**

  Para obter uma instância em cache única de um ObjectGrid, use getObjectGrid(StringobjectGridName), passando o nome da instância em cache para o método. O método retorna a instância de ObjectGrid com o nome especificado ou retorna nulo se não houver nenhuma instância de ObjectGrid com esse nome.

Nota: Também é possível usar o método getObjectGrid para se conectar a uma grade distribuída. Consulte o "Conectando-se às Instâncias do ObjectGrid Distribuído Programaticamente" na página 131 para obter informações adicionais.

Removendo Instâncias do ObjectGrid com a Interface ObjectGridManager

Você pode usar dois métodos removeObjectGrid diferentes para remover as instâncias do ObjectGrid do cache.

Remover uma Instância do ObjectGrid

Para remover instâncias do ObjectGrid do cache, utilize um dos métodos removeObjectGrid. A interface do ObjectGridManager não mantém uma referência das instâncias que são removidas. Existem dois métodos remove. Um método utiliza um parâmetro booleano. Se o parâmetro booleano for configurado como `true`, o método destroy é chamado no ObjectGrid. A chamada para o método destroy no ObjectGrid encerra o ObjectGrid e libera todos os recursos que o ObjectGrid está usando. Uma descrição de como usar os dois métodos removeObjectGrid a seguir:

```
/**
 * Remover um ObjectGrid do cache de instâncias do ObjectGrid
 *
 * @param objectGridName o nome da instância do ObjectGrid a ser removida
 * do cache
 *
 * @throws ObjectGridException se um ObjectGrid com o objectGridName
 * não tiver sido localizado no cache
 */
public void removeObjectGrid(String objectGridName) throws ObjectGridException;

/**
 * Remover um ObjectGrid do cache de instâncias do ObjectGrid e
 * destruir seus recursos associados
 *
 * @param objectGridName o nome da instância do ObjectGrid a ser removida
 * do cache
 *
 * @param destroy destruir a instância do objectgrid e seus
 * recursos associados
 *
 * @throws ObjectGridException se um ObjectGrid com o objectGridName
 * não tiver sido localizado no cache
 */
public void removeObjectGrid(String objectGridName, boolean destroy)
 throws ObjectGridException;
```

Controlando o Ciclo de Vida de um ObjectGrid com a Interface ObjectGridManager

É possível usar a interface ObjectGridManager para controlar o ciclo de vida de uma instância ObjectGrid usando um bean ou um servlet de inicialização.

Gerenciando o Ciclo de Vida com um Bean de Inicialização

Um bean de inicialização é usado para controlar o ciclo de vida de uma instância do ObjectGrid. Um bean de inicialização é carregado quando um aplicativo é iniciado. Com um bean de inicialização, o código pode ser executado sempre que um aplicativo for iniciado ou parado conforme o esperado. Para criar um bean de inicialização, utilize a interface home com.ibm.websphere.startupservice.AppStartUpHome e a interface remota com.ibm.websphere.startupservice.AppStartUp. Implemente os métodos start e stop no bean. O método start é chamado sempre que o aplicativo é inicializado. O método stop é chamado quando o aplicativo é encerrado. O método start é usado para criar instâncias do ObjectGrid. O método stop é usado para remover as instâncias do ObjectGrid. A seguir há um trecho de código que demonstra este gerenciamento de ciclo de vida do ObjectGrid em um bean de inicialização:

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

    /* The methods on the SessionBean interface have been
     * left out of this example for the sake of brevity */
```

```
    public boolean start(){
        // Iniciando o bean de inicialização
        // Este método é chamado quando o aplicativo é iniciado
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // criar 2 ObjectGrids e armazenar estas instâncias em cache
            ObjectGrid bookstoreGrid =
      objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
            ObjectGrid videostoreGrid =
      objectGridManager.createObjectGrid("videostore", true);
            // na JVM,
            // estes ObjectGrids agora podem ser recuperados do
            //ObjectGridManager utilizando o método getObjectGrid(String)
        } catch (ObjectGridException e) {
            e.printStackTrace();
            return false;
        }

        return true;
    }

    public void stop()  {
        // Parando o bean de inicialização
        // Este método é chamado quando o aplicativo é parado
        try {
            // remover os ObjectGrids armazenados em cache e destruí-los
            objectGridManager.removeObjectGrid("bookstore", true);
            objectGridManager.removeObjectGrid("videostore", true);
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }
}
```

Depois que o método start for chamado, as instâncias recém criadas do ObjectGrid serão recuperadas a partir da interface do ObjectGridManager. Por exemplo, se um servlet está incluído no aplicativo, o servlet acessa o eXtreme Scale utilizando o seguinte trecho de código:

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

Gerenciando o Ciclo de Vida com um Servlet

Para gerenciar o ciclo de vida de um ObjectGrid em um servlet, será possível usar o método init para criar uma instância do ObjectGrid e o método destroy para remover a instância do ObjectGrid. Se a instância do ObjectGrid estiver em cache, ela será recuperada e manipulada no código de servlet. A seguir há um código que mostra a criação, a manipulação e a destruição de um ObjectGrid em um servlet:

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

    public MyObjectGridServlet() {
        super();
    }

    public void init(ServletConfig arg0) throws ServletException {
        super.init();
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // criar e armazenar em cache um ObjectGrid denominado 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");
        // desempenhar operações no ObjectGrid armazenado em cache
        // ...
    }

    public void destroy() {
        super.destroy();
        try {
            // remover e destruir o ObjectGrid bookstore armazenado em cache
            objectGridManager.removeObjectGrid("bookstore", true);
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }
}
```

Acesso ao Shard ObjectGrid

O WebSphere eXtreme Scale atinge altas taxas de processamento movendo a lógica para onde os dados estão e retornando apenas resultados ao cliente.

A lógica de aplicativo em uma Java Virtual Machine (JVM) cliente precisa executar o pull de dados do servidor JVM que contém os dados e executar o push back quando ocorre o commit da transação. Esse processo reduz a taxa em que os dados podem ser processados. Se a lógica de aplicativo estivesse no mesmo JVM que o shard que está hospedando os dados, então a latência de rede e o custo de delegação seriam eliminados e poderia fornecer um significativo impulso no desempenho.

Referência Local para Dados do Shard

As APIs do ObjectGrid fornecem uma Session para o método do lado do servidor. Tal objeto Session é uma referência direta aos dados do shard. Não há nenhuma lógica de roteamento nesse caminho. A lógica de aplicativo pode trabalhar com os dados desse shard diretamente. O Session não pode ser utilizado para acessar os dados em outra partição porque não há nenhuma lógica de roteamento.

Um plug-in do Utilitário de Carga também fornece um meio para receber um evento quando um shard assume a função de partição primária. Um aplicativo pode implementar um Utilitário de Carga e implementar a interface do ReplicaPreloadController. O método check preload status é chamado apenas quando o shard assume a função de primário. O objeto Session fornecido a esse método é uma referência local aos dados dos shards. Essa abordagem é utilizada normalmente se o shard primário de uma partição precisar iniciar alguns encadeamentos ou assinar uma malha de mensagens para o tráfego relacionado à partição. Ele pode iniciar um encadeamento para escutar mensagens num Mapa local, por meio da API getNextKey.

Otimização Colocada do Cliente-Servidor

Se um aplicativo utiliza as APIs do cliente para acessar uma partição que pode ter sido colocada com a JVM que contém o cliente, a rede será evitada mas ainda ocorrerá alguma delegação devido a problemas de implementação atuais. Se uma grade particionada é utilizada, então não há impacto no desempenho do aplicativo porque (N-1)/número N de chamadas são roteadas para uma JVM diferente. Se você sempre precisar de acesso local com um shard, então, utilize o Utilitário de Cargo ou as APIs do ObjectGrid para chamar est lógica.

Acessando Dados com Índices (API de Índice)

Use indexação para acesso a dados mais eficiente.

Sobre Esta Tarefa

A classe HashIndex é a implementação de plug-in de índice integrada que pode suportar ambas as interfaces integradas de índice a seguir: MapIndex e MapRangeIndex. Também é possível criar seus próprios índices. É possível incluir HashIndex como um índice estático ou dinâmico no mapa de apoio, obter um objeto proxy do índice MapIndex ou MapRangeIndex e usar o objeto proxy do índice para localizar objetos em cache.

Se desejar iterar por meio das chaves em um mapa local, o índice padrão poderá ser usado. Este índice não requer nenhuma configuração, porém ele deve ser usado com relação ao shard, usando um agente ou uma instância do ObjectGrid recuperados a partir do método ShardEvents.shardActivated(shard do ObjectGrid).

Nota: Em um ambiente distribuído, se o objeto do índice for obtido de um cliente do ObjectGrid, o índice possuirá um objeto de índice do tipo cliente e todas as operações de índice executadas em um ObjectGrid do servidor remoto. Se o mapa for particionado, as operações de índice executarão em cada partição remotamente. Os resultados de cada partição são mesclados antes de retornar os resultados para o aplicativo. O desempenho é determinado pelo número de partições e pelo tamanho do resultado retornado por cada partição. Um desempenho insuficiente poderá ocorrer se ambos os fatores forem altos.

Procedimento

1. Se desejar usar índices diferente do índice local padrão, inclua plug-ins de índice no mapa de apoio.
   - **Configuração 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>
```

     Neste exemplo de configuração XML, a classe HashIndex integrada é usada como o plug-in de índice. A classe HashIndex suporta propriedades que os usuários podem configurar, como Name, RangeIndex e AttributeName no exemplo anterior.
     - A propriedade **Name** é configurada como CODE, uma sequência que identifica este plug-in de índice. O valor da propriedade Name deve ser

único dentro do escopo do BackingMap, e pode ser usado para recuperar o objeto do índice pelo nome a partir da instância de ObjectMap para o BackingMap.

– A propriedade **RangeIndex** é configurada como `true`, o que significa que o aplicativo pode efetuar cast do objeto do índice recuperado para a interface MapRangeIndex. Se a propriedade RangeIndex for configurada como `false`, o aplicativo poderá apenas efetuar cast do objeto do índice recuperado para a interface MapIndex. Um MapRangeIndex suporta funções para localizar dados usando funções de intervalo, como maior que, menor que, ou ambas, enquanto um MapIndex suporta apenas funções iguais. Se o índice for usado por consulta, a propriedade **RangeIndex** deverá ser configurada para `true` em índices de atributo único. Para um índice de relacionamento e um índice composto, a propriedade RangeIndex deve ser configurada para `false`.

– A propriedade **AttributeName** é configurada como `employeeCode`, o que significa que o atributo **employeeCode** do objeto em cache é usado para construir um índice de atributo único. Se um aplicativo precisar procurar por objetos em cache
com diversos atributos, a propriedade **AttributeName** poderá ser configurada para uma lista de atributos delimitados por vírgula, rendendo um índice composto.

- **Configuração programática:**

  A interface BackingMap tem dois métodos que podem ser usados para incluir plug-ins de índice estático: addMapIndexplugin e setMapIndexplugins. Para obter mais informações, consulte a documentação da API. O exemplo a seguir cria a mesma configuração que o exemplo de configuração 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");

    //utilize a classe HashIndex interna como a classe de plugin do índice.
    HashIndex mapIndexplugin = new HashIndex();
    mapIndexplugin.setName("CODE");
    mapIndexplugin.setAttributeName("EmployeeCode");
    mapIndexplugin.setRangeIndex(true);
    personBackingMap.addMapIndexplugin(mapIndexplugin);
```

2. Acesse as chaves e valores de mapa com índices.

   - **Índice de local:**

     Para iterar por meio das chaves e valores em um mapa local, o índice padrão poderá ser usado. O índice padrão funciona apenas com relação ao shard, usando um agente ou usando a instância do ObjectGrid recuperados com o método ShardEvents.shardActivated(shard do ObjectGrid). Consulte o seguinte exemplo:

```
MapIndex keyIndex = (MapIndex)
objMap.getIndex(MapIndexPlugin.SYSTEM_KEY_INDEX_NAME);
Iterator keyIterator = keyIndex.findAll();
```

   - **Índices estáticos:**

     Após um plug-in de índice estático ser incluído em uma configuração de BackingMap e a instância de ObjectGrid de abrangência ser inicializada, os aplicativos podem recuperar o objeto do índice por nome a partir da instância do ObjectMap para o BackingMap. Lance o objeto de índice para a interface de índice do aplicativo. As operações que a interface do índice do aplicativo suporta podem executar agora.

```
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 dinâmicos:**

  É possível criar e remover índices dinâmicos a partir de uma instância do BackingMap programaticamente a qualquer momento. Um índice dinâmico se difere de um índice estático porque o índice dinâmico pode ser criado mesmo depois que a instância do ObjectGrid de abrangência tiver sido inicializada. Diferentemente da indexação estática, a indexação dinâmica é um processo assíncrono e precisa estar em estado de pronto antes de ser usada. Este método utiliza a mesma abordagem para recuperação e utilização de índices dinâmicos como índices estáticos. É possível remover um índice dinâmico se ele não for mais necessário. A interface BackingMap possui métodos para criar e remover índices dinâmicos.

  Consulte a Documentação da API BackingMap para obter mais informações sobre os métodos createDynamicIndex e 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();

        // create index after ObjectGrid initialization without DynamicIndexCallback.
        bm.createDynamicIndex("CODE", true, "employeeCode", null);

        try {
            //Se não estiver utilizando DynamicIndexCallback, será necessário aguardar que o Índice esteja pronto.
            //O tempo de espera depende do tamanho atual do mapa
            Thread.sleep(3000);
        } catch (Throwable t) {
            // ...
        }

        // When the index is ready, applications can try to get application index
        // interface instance.
        // Applications have to find a way to ensure that the index is ready to use,
        // if not using DynamicIndexCallback interface.
        // The following example demonstrates the way to wait for the index to be ready
        // Consider the size of the map in the total waiting time.

        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) {
                // implies index is not ready, ...
                System.out.println("Index is not ready. continue to wait.");
                try {
                    Thread.sleep(3000);
                } catch (Throwable tt) {
                    // ...
                }
            } catch (Throwable t) {
                // unexpected exception
                t.printStackTrace();
            }
        }

        if(!ready){
            System.out.println("Index is not ready.  Need to handle this situation.");
        }

        // Use the index to peform queries
        // Refer to the MapIndex or MapRangeIndex interface for supported operations.
        // The object attribute on which the index is created is the EmployeeCode.
        // Assume that the EmployeeCode attribute is Integer type: the
        // parameter that is passed into index operations has this data type.

        Iterator iter = codeIndex.findLessEqual(new Integer(15));

        // remove the dynamic index when no longer needed

        bm.removeDynamicIndex("CODE");
```

O que Fazer Depois

É possível usar a interface DynamicIndexCallback para obter notificações nos eventos de indexação. Consulte o "Interface DynamicIndexCallback" para obter informações adicionais.

Conceitos relacionados

"Plug-ins para Indexar Dados" na página 327
O HashIndex integrado, a classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, é um plug-in MapIndexPlugin que pode ser incluído no BackingMap para construir índices estáticos ou dinâmicos. Essa classe suporta ambas as interfaces MapIndex e MapRangeIndex. A definição e a implementação de índices podem aprimorar significativamente o desempenho da consulta.

"Plug-ins para Indexação Customizada de Objetos de Cache" na página 333
Com um plug-in MapIndexPlugin, ou índice, é possível gravar estratégias de indexação customizadas que vão além de índices integrados fornecidos pelo eXtreme Scale.

"Usando um Índice Composto" na página 336
O HashIndex composto aprimora o desempenho da consulta e evita a custosa varredura de mapa. O recurso também fornece uma maneira conveniente para a API HashIndex localizar objetos em cache quando os critérios de busca envolvem muitos atributos.

"Indexação" na página 97
Use o plug-in MapIndexPlugin para construir um índice ou vários índices em um BackingMap para suportar acesso a dados sem chave.

Referências relacionadas

"Atributos do Plug-in HashIndex" na página 330
É possível usar os seguintes atributos para configurar o plug-in HashIndex. Esses atributos definem propriedades, como se você estiver usando um atributo ou HashIndex composto ou se a indexação do intervalo estiver ativada.

Interface DynamicIndexCallback

A interface DynamicIndexCallback foi projetada para aplicativos que desejam obter notificações nos eventos de indexação de ready, error ou destroy. O DynamicIndexCallback é um parâmetro opcional para o método createDynamicIndex do BackingMap. Com uma instância DynamicIndexCallback registrada, os aplicativos podem executar a lógica de negócios ao receber notificação de um evento de indexação.

Eventos de Indexação

Por exemplo, o evento ready significa que o índice está pronto para utilização. Quando uma notificação para este evento for recebida, um aplicativo poderá tentar recuperar a instância da interface do índice do aplicativo e utilizá-la. Consulte a API DynamicIndexCallback na Documentação da API para obter mais informações.

Exemplo: Usando a Interface EntityIndexCallback

```
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);

            // Simulate what an application would do when notified that the index is ready.
```

```
            // Normally, the application would wait until the ready state is reached and then proceed
            // with any index usage logic.
            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);
        }
    }
```

Uso de Sessões para Acessar Dados na Grade

Os aplicativos podem iniciar e terminar transações através da interface Session. A interface Session também fornece acesso ao aplicativo com base nas interfaces ObjectMap e JavaMap.

Cada instância de ObjectMap ou de JavaMap está diretamente ligada a um objeto de Sessão específico. Cada encadeamento que desejar acesso a um eXtreme Scale deve primeiro obter uma Sessão do objeto ObjectGrid. Uma instância de Sessão não pode ser compartilhada simultaneamente entre encadeamentos. O WebSphere eXtreme Scale não usa qualquer armazenamento local do encadeamento, mas restrições de plataforma podem limitar a oportunidade de passar uma Sessão de um encadeamento para outro.

Métodos

Método Get

Um aplicativo obtém uma instância da Sessão a partir de um objeto ObjectGrid usando o método ObjectGrid.getSession. O exemplo a seguir demonstra como obter uma instância de Session:

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

Após uma Sessão ter sido obtida, o encadeamento mantém uma referência à sessão para sua própria utilização. Chamar o método getSession várias vezes sempre retorna, cada vez, um novo objeto de Sessão.

Métodos de Transações e Sessão

Uma Sessão pode ser utilizada para iniciar, confirmar ou efetuar rollback de transações. As operações em BackingMaps que utilizam ObjectMaps e JavaMaps são desempenhadas de maneira mais eficiente em uma transação de Sessão. Quando uma transação é iniciada, as alterações em um ou mais BackingMaps nesse escopo de transação são armazenadas em um cache de transações especiais até que a transação seja confirmada. Quando uma transação é confirmada, as alterações pendentes são aplicadas aos BackingMaps e Loaders e se tornam visíveis a outros clientes desse ObjectGrid.

O WebSphere eXtreme Scale também suporta a habilidade de automaticamente consolidar transações, também conhecida como auto-consolidação. Se quaisquer operações do ObjectMap forem desempenhadas fora do contexto de uma transação ativa, uma transação implícita será iniciada antes da operação e a transação será automaticamente confirmada antes de retornar o controle ao aplicativo.

```
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

O método Session.flush faz sentido apenas quando um Loader está associado a um BackingMap. O método flush chama o Loader com o conjunto atual de alterações no cache de transações. O Loader aplica as alterações ao backend. Estas alterações não são confirmadas quando o flush é chamado. Se uma transação de Sessão for confirmada após uma chamada de flush, apenas as atualizações que ocorrem após a chamada do flush serão aplicadas ao Loader. Se uma transação de Sessão receber rollback após uma chamada de flush, as alterações limpas serão descartadas com todas as demais alterações pendentes na transação. Utilize o método Flush com moderação, pois ele limita a oportunidade de operações de batch em um Loader. A seguir está um exemplo do uso do método Session.flush:

```
Session session = objectGrid.getSession();
session.begin();
// fazer algumas alterações
...
session.flush(); // enviar estas alterações para o Loader, mas ainda não confirmar
// fazer mais algumas alterações
...
session.commit();
```

Método NoWriteThrough

Alguns mapas são suportados por um Loader, que fornece armazenamento persistente aos dados no mapa. Algumas vezes, é útil consolidar dados apenas no mapa do eXtreme Scale e não executar o push de dados para fora do Utilitário de Carga. A interface Session fornece o método beginNoWriteThough para esta finalidade. O método beginNoWriteThrough inicia uma transação como o método begin. Com o método beginNoWriteThrough, quando a transação é confirmada, os dados são confirmados apenas para o mapa na memória e não são confirmados para o armazenamento persistente que é fornecido pelo Loader. Este método é muito útil ao desempenhar o pré-carregamento de dados no mapa.

Ao utilizar uma instância do ObjectGrid distribuído, o método beginNoWriteThrough é útil para fazer alterações apenas no near cache, sem modificar o far cache no servidor. Se os dados forem considerados stale no near cache, a utilização do método beginNoWriteThrough pode permitir que entradas sejam invalidadas no near cache sem invalidá-las também no servidor.

A interface Session também fornece o método isWriteThroughEnabled para determinar qual o tipo de transação está ativo no momento.

```
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
// fazer algumas alterações...
session.commit(); // estas alterações não serão enviadas ao Loader
```

Obter o Método de Objeto TxID

O objeto TxID é um objeto opaco que identifica a transação ativa. Utilize o objeto TxID para as seguintes finalidades:

- Para comparação quando estiver procurando uma transação específica.
- Para armazenar dados compartilhados entre os objetos TransactionCallback e Loader.

Consulte o plug-in TransactionCallback e Loaders para obter informações adicionais sobre o recurso de slot do Objeto.

Método de monitoramento de desempenho

Se estiver usando o eXtreme Scale dentro do WebSphere Application Server, pode ser necessário reiniciar o tipo de transação para monitoramento de desempenho. É possível configurar o tipo de transação com o método setTransactionType. Consulte Monitoramento do Desempenho do ObjectGrid com a PMI (Performance Monitoring Infrastructure) do WebSphere Application Server para obter informações adicionais sobre o método setTransactionType.

Processar um Método LogSequence Completo

O WebSphere eXtreme Scale pode propagar conjuntos de alterações de mapas para listeners de ObjectGrid como um meio de distribuição de mapas de um Java Virtual Machine para outro. Para facilitar o processamento pelo listener de LogSequences recebidos, a interface Session fornece o método processLogSequence. Este método examina cada LogElement no LogSequence e desempenha uma operação apropriada, por exemplo, inserção, atualização, invalidação e outros, no BackingMap identificado pelo MapName LogSequence. Uma Sessão do ObjectGrid deve estar disponível antes de o método processLogSequence ser chamado. O aplicativo também é responsável por emitir as chamadas de confirmação ou de rollback apropriadas para concluir a Sessão. O processamento da confirmação automática não está disponível para esta chamada de método. O processamento normal pelo ObjectGridEventListener receptor na JVM remota seria iniciar uma Sessão utilizando o método beginNoWriteThrough, que evita a propagação sem fim de alterações, seguido de uma chamada para este método processLogSequence e, então, consolidando ou retrocedendo a transação.

```
// Utilizar o objeto de Sessão transmitido durante
//ObjectGridEventListener.initialization...
session.beginNoWriteThrough();
// processar o LogSequence recebido
try {
 session.processLogSequence(receivedLogSequence);
} catch(Exception e) {
 session.rollback(); throw e;
}
// confirmar as alterações
session.commit();
```

Método markRollbackOnly

Este método é utilizado para marcar a transação atual como "apenas rollback". Marcar uma transação como "apenas rollback" assegura que, mesmo que o método commit seja chamado pelo aplicativo, a transação receba rollback. Este método geralmente é utilizado pelo próprio ObjectGrid ou pelo aplicativo quando ele sabe que pode ocorrer danos nos dados se a transação tiver permissão para ser confirmada. Após este método ser chamado, o objeto Throwable que é passado a este método é encadeado na exceção com.ibm.websphere.objectgrid.TransactionException que resulta do método commit se for chamado em uma Sessão que foi anteriormente marcada como "apenas retrocesso". As chamadas subseqüentes para este método para uma transação que

já está marcada como "apenas rollback" serão ignoradas. Ou seja, apenas a primeira chamada que transmite uma referência Throwable não nula é utilizada. Quando a transação marcada estiver concluída, a marca "apenas rollback" será removida para que a próxima transação iniciada pela Sessão possa ser confirmada.

Método isMarkedRollbackOnly

Retorna se a Sessão está marcada como "apenas rollback". O true booleano será retornado por este método apenas se um método markRollbackOnly tiver sido chamado anteriormente nesta Sessão e a transação iniciada pela Sessão ainda estiver ativa.

Método setTransactionTimeout

Configure o tempo limite de transação para a próxima transação iniciada por esta Sessão como um número de segundos especificado. Este método não afeta o tempo limite da transação de nenhuma transação iniciada anteriormente por esta Sessão. Afeta apenas as transações iniciadas após este método ter sido chamado. Se este método nunca for chamado, o valor de tempo limite transmitido para o método setTxTimeout do método com.ibm.websphere.objectgrid.ObjectGrid será utilizado.

Método getTransactionTimeout

Este método retorna o valor de tempo limite da transação em segundos. O último valor que foi transmitido como o valor de tempo limite para o método setTransactionTimeout é retornado por este método. Se o método setTransactionTimeout nunca for chamado, o valor de tempo limite transmitido para o método setTxTimeout do método com.ibm.websphere.objectgrid.ObjectGrid será utilizado.

transactionTimedOut

Este método retorna true booleano se a transação atual iniciada por esta Sessão tiver seu tempo limite excedido.

Método isFlushing

Este método retorna true booleano apenas se todas as alterações de transação forem esvaziadas do plug-in do Utilitário de Carga como resultado do método flush da interface Session que está sendo chamada. Um plug-in do Utilitário de Carga pode achar este método útil quando ele precisar saber porquê seu método batchUpdate foi chamado.

Método isCommitting

Este método retorna true booleano apenas se todas as alterações de transação forem confirmadas como resultado do método commit da interface Session que está sendo chamada. Um plug-in do Loader pode achar este método útil quando precisar saber por que seu método batchUpdate foi chamado.

Método setRequestRetryTimeout

Este método define o valor de tempo limite de nova tentativa para a sessão em milissegundos. Se o cliente definir um tempo limite de nova tentativa de solicitação, a configuração da sessão prevalece em relação ao valor do cliente.

Método getRequestRetryTimeout

Este método obtém a configuração atual de tempo limite de nova tentativa na sessão. Uma valor de `-1` indica que o tempo limite não está configurado. Um valor de `0` indica que ele está no modo fail-fast. Um valor maior que `0` indica a configuração de tempo limite em milissegundos.

SessionHandle para Roteamento

Quando estiver usando uma política de posicionamento de partição por contêiner, um objeto SessionHandle pode ser usado. Um objeto SessionHandle contém informações de partição para a Sessão atual e pode ser reusado para uma nova Sessão.

Um objeto SessionHandle inclui informações para a partição à qual a Sessão atual está limitada. O SessionHandle é extremamente útil para a política de posicionamento de partição por contêiner e pode ser serializado com a serialização Java padrão.

Se você tiver um objeto SessionHandle, esse identificador poderá ser aplicado em uma Sessão com o método setSessionHandle(SessionHandle target), passando o identificador como o destino. O objeto SessionHandle pode ser recuperado com o método Session.getSessionHandle.

Como ele se aplica apenas em um cenário de posicionamento por contêiner, obter o objeto SessionHandle emitirá uma IllegalStateException se uma determinada grade de dados tiver diversos conjuntos de mapas por contêiner ou não tiver nenhum conjunto de mapas por contêiner. Se o método setSessionHandle não for chamado antes de chamar o método getSessionHandle, o objeto SessionHandle apropriado será selecionado com base na configuração das propriedades do cliente.

Também é possível usar a classe auxiliar SessionHandleTransformer para converter o identificador em formatos diferentes. Os métodos desta classe podem alterar uma representação do identificador da matriz de bytes para a instância, da sequência para instância, e vice-versa, para ambos os casos e também podem gravar o conteúdo do identificador no fluxo de saída.

Para obter um exemplo de como é possível usar um objeto SessionHandle, consulte o tópico de roteamento preferencial por zona na *Visão Geral do Produto*.

Integração do SessionHandle

Um objeto SessionHandle inclui informações de partição para o Session ao qual ele está ligado e facilita o roteamento da solicitação. Os objetos SessionHandle se aplicam apenas no cenário de posicionamento de partição por contêiner.

Objeto SessionHandle para Roteamento de Solicitação

É possível ligar um objeto SessionHandle a um Session das seguintes formas:

Dica: Em cada uma das chamadas de método a seguir, depois de um objeto SessionHandle ser ligado a um Session, o objeto SessionHandle poderá ser obtido a partir do método Session.getSessionHandle.

- **Chame o método Session.getSessionHandle:** Quando este método é chamado, se nenhum objeto SessionHandle estiver ligado ao Session, um objeto SessionHandle será selecionado aleatoriamente e ligado ao Session.
- **Chame as operações transacionais de criação, leitura, atualização e de exclusão:** Quando esses métodos são chamados no tempo de confirmação, se nenhum

objeto SessionHandle estiver ligado ao Session, um objeto SessionHandle será selecionado aleatoriamente e ligado ao Session.

- **Chame o método ObjectMap.getNextKey:** Quando esse método é chamado, se nenhum objeto SessionHandle estiver ligado ao Session, a solicitação da operação será roteada aleatoriamente para partições individuais até a chave ser obtida. Se uma chave for retornada de uma partição, um objeto SessionHandle correspondente a essa partição será ligado ao Session. Se nenhuma chave for localizada, nenhum SessionHandle será ligado ao Session.
- **Chame os métodos QueryQueue.getNextEntity ou QueryQueue.getNextEntities:** Quando esse método é chamado, se nenhum objeto SessionHandle estiver ligado ao Session, a solicitação da operação será roteada aleatoriamente para partições individuais até um objeto ser obtido. Se um objeto for retornado de uma partição, um objeto SessionHandle correspondente a essa partição será ligado ao Session. Se nenhum objeto for localizado, nenhum SessionHandle será ligado ao Session.
- **Configure um SessionHandle com o método Session.setSessionHandle(SessionHandle sh):** Se SessionHandle for obtido a partir do método Session.getSessionHandle, o SessionHandle poderá ser ligado a um Session. A configuração de um SessionHandle influencia o roteamento de solicitações dentro do escopo do Session ao qual ele está ligado.

O método Session.getSessionHandle sempre retorna um objeto SessionHandle. O método também ligará automaticamente um SessionHandle no Session se nenhum objeto SessionHandle estiver ligado ao Session. Se desejar verificar se um Session tem apenas um objeto SessionHandle, chame o método Session.isSessionHandleSet. Se esse método retornar um valor `false`, nenhum SessionHandle estará ligado atualmente ao Session.

Principais Tipos de Operações no Cenário de Posicionamento por Contêiner

A seguir há um resumo do comportamento de roteamento dos principais tipos de operações no cenário de posicionamento de partição por contêiner em relação aos objetos SessionHandle.

- **Objeto Session com o objeto SessionHandle ligado**
  - Índice - API de MapIndex e MapRangeIndex: SessionHandle
  - Query e ObjectQuery: SessionHandle
  - Agente - API de MapGridAgent e ReduceGridAgent API: SessionHandle
  - ObjectMap.Clear: SessionHandle
  - ObjectMap.getNextKey: SessionHandle
  - QueryQueue.getNextEntity, QueryQueue.getNextEntities: SessionHandle
  - Operações transacionais de criação, recuperação, atualização e exclusão (API de ObjectMap e API de EntityManager): SessionHandle
- **Objeto Session sem o objeto SessionHandle ligado**
  - Índice - API de MapIndex e MapRangeIndex: Todas as partições ativas atuais
  - Query e ObjectQuery: Partição especificada com o método setPartition de Query e ObjectQuery
  - Agente - MapGridAgent e ReduceGridAgent
    - Não suportado: método ReduceGridAgent.reduce(Session s, ObjectMap map, Collection keys) e MapGridAgent.process(Session s, ObjectMap map, Object key).

- Todas as partições ativas atuais: método ReduceGridAgent.reduce(Session s, ObjectMap map) e MapGridAgent.processAllEntries(Session s, ObjectMap map).

– ObjectMap.clear: Todas as partições ativas atuais.
– ObjectMap.getNextKey: Liga um SessionHandle ao Session se uma chave for retornada de uma das partições selecionadas aleatoriamente. Se nenhuma chave for retornada, o Session não será ligado a nenhum SessionHandle.
– QueryQueue: Especifica uma partição com o método QueryQueue.setPartition. Se nenhuma partição for configurada, o método seleciona aleatoriamente uma partição para retornar. Se um objeto for retornado, o Session atual será ligado com o SessionHandle que é ligado à partição que retorna o objeto. Se nenhum objeto for retornado, o Session não será ligado a nenhum SessionHandle.
– Operações transacionais de criação, recuperação, atualização e exclusão (API de ObjectMap e API de EntityManager): Selecione uma partição aleatoriamente.

Na maioria dos casos, use SessionHandle para controlar o roteamento para uma determinada partição. É possível recuperar e armazenar em cache o SessionHandle a partir do Session que insere dados. Depois de armazenar em cache o SessionHandle, é possível configurá-lo em outro Session para que você possa rotear solicitações para a partição especificada pelo SessionHandle em cache. Para executar operações como ObjectMap.clear sem o SessionHandle, é possível configurar temporariamente o SessionHandle como nulo ao chamar Session.setSessionHandle(null). Sem um SessionHandle especificado, as operações são executadas em todas as partições ativas atuais.

- **Comportamento de roteamento de QueryQueue**

  No cenário de posicionamento de partição por contêiner, o SessionHandle pode ser usado para controlar o roteamento dos métodos getNextEntity e getNextEntities da API de QueryQueue. Se o Session estiver ligado a um SessionHandle, as solicitações são roteadas para a partição à qual o SessionHandle está ligado. Se o Session não estiver ligado a um SessionHandle, as solicitações serão roteadas para a partição configurada com o método QueryQueue.setPartition se uma partição tiver sido configurada dessa forma. Se o Session não tiver ligado SessionHandle ou à partição, uma partição selecionada aleatoriamente será retornada. Se nenhuma partição for localizada, o processo será interrompido e nenhum SessionHandle será ligado ao Session atual.

O fragmento de código a seguir mostra como usar os objetos SessionHandle.

```
Session ogSession = objectGrid.getSession();

// binding the SessionHandle
SessionHandle sessionHandle = ogSession.getSessionHandle();

ogSession.begin();
ObjectMap map = ogSession.getMap("planet");
map.insert("planet1", "mercury");

// transaction is routed to partition specified by SessionHandle
ogSession.commit();

// cache the SessionHandle that inserts data
SessionHandle cachedSessionHandle = ogSession.getSessionHandle();

// verify if SessionHandle is set on the Session
boolean isSessionHandleSet = ogSession.isSessionHandleSet();
```

```
// temporarily unbind the SessionHandle from the Session
if(isSessionHandleSet) {
    ogSession.setSessionHandle(null);
}

// if the Session has no SessionHandle bound,
// the clear operation will run on all current active partitions
// and thus remove all data from the map in the entire grid
map.clear();

// after clear is done, reset the SessionHandle back,
// if the Session needs to use previous SessionHandle.
// Optionally, calling getSessionHandle can get a new SessionHandle
ogSession.setSessionHandle(cachedSessionHandle);
```

Considerações de Design do Aplicativo

No cenário de estratégia de posicionamento por contêiner, use o objeto SessionHandle para a maioria das operações. O objeto SessionHandle controla o roteamento para as partições. Para recuperar dados, o objeto SessionHandle que é ligado ao Session deverá ser o mesmo objeto SessionHandle de qualquer transação de inserção de dados.

Quando desejar executar uma operação sem um SessionHandle configurado no Session, é possível desvincular um SessionHandle de um Session ao chamar o método Session.setSessionHandle(null).

Quando um Session for ligado a um SessionHandle, todas as solicitações de operação serão roteadas para a partição especificada pelo objeto SessionHandle. Sem o objeto SessionHandle configurado, as operações serão roteadas para todas as partições ou para uma partição selecionada aleatoriamente.

Objetos de Armazenamento em Cache sem Relacionamentos Envolvidos (API ObjectMap)

Os ObjectMaps são como Mapas Java que permitem que os dados sejam armazenados como pares chave-valor. Os ObjectMaps apresentam uma abordagem simples e intuitiva para o aplicativo que armazenará os dados. Um ObjectMap é ideal para o armazenamento em cache de objetos que não tenham nenhum relacionamento envolvido. Se os relacionamentos de objetos estiverem envolvidos, então você deve usar a API EntityManager.

Para obter informações adicionais sobre a API EntityManager, consulte “Objetos de Armazenamento em Cache e seus Relacionamentos (API EntityManager)” na página 166.

Os aplicativos normalmente obtêm uma referência do WebSphere eXtreme Scale e, então, obtêm um objeto Session da referência para cada encadeamento. As sessões não podem ser compartilhadas entre encadeamentos. O método getMap do Session retorna uma referência a um ObjectMap a ser utilizado para esse encadeamento.

Tarefas relacionadas

"Introdução ao Desenvolvimento de Aplicativos" na página 70
Para começar a desenvolver aplicativos WebSphere eXtreme Scale, configure um ambiente de desenvolvimento no Eclipse.

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Informações relacionadas

"Lição 2 do Tutorial de Introdução: Criando um Aplicativo Cliente" na página 63
Para inserir, excluir, atualizar e recuperar dados de sua grade de dados, você deverá gravar um aplicativo cliente. A introdução de amostra inclui um aplicativo cliente que pode ser usado para saber mais sobre a criação de seu próprio aplicativo cliente.

Introdução ao ObjectMap

A interface ObjectMap é utilizada para interação transacional entre aplicativos e BackingMaps.

Propósito

Uma instância do ObjectMap é obtida do objeto de Sessão que corresponde ao encadeamento atual. A interface ObjectMap é o principal veículo utilizado pelos aplicativos para fazer alterações em entradas em um BackingMap.

Obtenha uma Instância do ObjectMap

Um aplicativo obtém uma instância do ObjectMap a partir de um objeto Session utilizando método Session.getMap(String). O trecho de código a seguir demonstra como obter uma instância de ObjectMap:

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

Cada instância de ObjectMap corresponde a um objeto de Sessão específico. Chamar o método getMap várias vezes em um objeto Session particular com o mesmo nome de BackingMap sempre retorna a mesma instância do ObjectMap.

Executar Commit de Transações Automaticamente

As operações junto a BackingMaps que utilizam ObjectMaps e JavaMaps são executadas de maneira mais eficiente em uma transação Session. O WebSphere eXtreme Scale fornece suporte a autocommit quando métodos nas interfaces ObjectMap e JavaMap são chamados fora de uma transação Session. Os métodos iniciam uma transação implícita, desempenham a operação solicitada e confirmam a transação implícita.

Semântica do Método

A seguir, está uma explicação da semântica por trás de cada método nas interfaces ObjectMap e JavaMap. O método setDefaultKeyword, o método invalidateUsingKeyword e os métodos que possuem um argumento Serializable

são discutidos no tópico Palavras-chave. O método setTimeToLive é discutido no tópico Evictors. Consulte a documentação da API para obter informações adicionais sobre estes métodos.

Método containsKey

O método containsKey determina se uma chave possui um valor no BackingMap ou Utilitário de Carga. Se valores nulos forem suportados por um aplicativo, este método poderá ser utilizado para determinar se uma referência nula retornada de uma operação get refere-se a um valor nulo ou indica que o BackingMap e o Utilitário de Carga não contêm a chave.

Método flush

A semântica do método flush é semelhante ao método flush na interface Session. A diferença notável é que a limpeza de Sessão aplica as alterações pendentes atuais para todos os mapas que foram modificados na sessão atual. Com este método, o flush ocorre apenas nas alterações nesta instância do ObjectMap para o utilitário de carga.

Método get

O método get busca a entrada a partir da instância do BackingMap. Se a entrada não for localizada na instância do BackingMap mas um Utilitário de Carga estiver associado com a instância do BackingMap, a instância do BackingMap tenta buscar a entrada a partir do Utilitário de Carga. O método getAll é fornecido para permitir o processamento de busca em lote.

Método getForUpdate

O método getForUpdate é o mesmo que o método get, mas utilizar o método getForUpdate informa ao BackingMap e ao Utilitário de Carga que a intenção é atualizar a entrada. Um Utilitário de Carga pode utilizar esta sugestão para emitir uma consulta SELECT for UPDATE para um backend de banco de dados. Se uma estratégia de bloqueio pessimistic for definida para o BackingMap, o gerenciador de bloqueios bloqueia a entrada. O método getAllForUpdate é fornecido para permitir o processamento de busca em lote.

Método insert

O método insert insere uma entrada no BackingMap e no Utilitário de Carga. Utilize este método informa ao BackingMap e ao Utilitário de Carga que você deseja inserir uma entrada que não existia anteriormente. Quando você chama este método em uma entrada existente, ocorre uma exceção quando o método é chamado ou quando a transação atual é confirmada.

Método invalidate

A semântica do método invalidate depende do valor do parâmetro **isGlobal** que é passado para o método. O método invalidateAll é fornecido para permitir o processamento de invalidação em lote.

A invalidação local é especificada quando o valor false é passado como o parâmetro **isGlobal** do método invalidate. A invalidação local descarta as alterações na entrada no cache de transação. Se o aplicativo emitir um método get, a entrada será buscada no último valor confirmado no BackingMap. Se nenhuma entrada estiver presente no BackingMap, a entrada será buscada no último valor limpo ou confirmado no Utilitário de Carga. Quando uma transação é confirmada, todas as entradas que estão marcadas como estando invalidadas localmente não têm nenhum impacto no BackingMap. As alterações que foram limpas no Utilitário de Carga ainda serão confirmadas, mesmo que a entrada tenha sido invalidada.

A invalidação global é especificada quando true é passado como o parâmetro **isGlobal** do método invalidate. A invalidação global descarta as alterações pendentes na entrada no cache de transação e ignora o valor de BackingMap nas operações seguintes que são executadas na entrada. Quando ocorre um commit de uma transação, as entradas que estão marcadas como invalidadas globalmente são despejadas do BackingMap. Considere o seguinte caso de uso para invalidação como um exemplo: O BackingMap é suportado por uma tabela de banco de dados que tem uma coluna de auto-incremento. As colunas de incremento são úteis para designar números exclusivos a registros. O aplicativo insere uma entrada. Após a inserção, o aplicativo precisa saber o número de seqüência para a linha inserida. Ele sabe que sua cópia do objeto é antiga, por isso, utiliza a invalidação global para obter o valor do Utilitário de Carga. O código a seguir demonstra 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();
```

Esta amostra de código inclui uma entrada para `Billy`. O atributo version de Person é configurado utilizando uma coluna de auto-incremento no banco de dados. Primeiro, o aplicativo executa um comando insert. Em seguida, ele emite um flush, que faz a inserção ser enviada para o Utilitário de Carga e o banco de dados. O banco de dados configura a coluna de versão para o próximo número na sequência, que desatualiza o objeto Person na transação. Para atualizar o objeto, o aplicativo é globalmente invalidado. O próximo método get que é emitido obtém a entrada do Utilitário de Carga, ignorando o valor da transação. A entrada é buscada no banco de dados com o valor de versão atualizado.

Método put

A semântica do método put é independente de um método get anterior ter sido chamado na transação para a chave. Se o aplicativo emite uma operação get que retorna uma entrada que existe no BackingMap ou Utilitário de Carga, a chamada do método put é interpretada como uma atualização e retorna o valor anterior na transação. Se uma chamada do método put foi executada sem uma chama do método get anterior ou uma chamada do método get anterior não localizou uma entrada, a operação é interpretada como uma inserção. A semântica dos métodos insert e update é aplicável quando ocorre o commit da operação put. O método putAll é fornecido para ativar inserção em lote e processamento de atualizações.

Método remove

O método remove remove a entrada do BackingMap e do Utilitário de Carga, se um Utilitário de Carga estiver conectado. O valor do objeto que foi removido é retornado por este método. Se o objeto não existir, este método retornará um valor nulo. O método removeAll é fornecido para ativar o processamento de exclusão em lote sem os valores de retorno.

Método setCopyMode

O método setCopyMode especifica um valor CopyMode para este ObjectMap. Com este método, um aplicativo pode substituir o valor CopyMode que é especificado no BackingMap. O valor CopyMode especificado fica em efeito até que o método clearCopyMode seja chamado.

Os dois métodos são chamados fora dos limites transacionais. Um valor CopyMode não pode ser alterado no meio de uma transação.

Método touch
O método touch atualiza o horário do último acesso para uma entrada. Este método não recupera o valor do BackingMap. Utilize este método em sua própria transação. Se a chave fornecida não existir no BackingMap devido a uma invalidação ou remoção, ocorrerá uma exceção durante o processamento de confirmação.

Método update
O método update atualiza explicitamente uma entrada no BackingMap e no Utilitário de Carga. A utilização deste método indica ao BackingMap e ao Utilitário de Carga que você deseja atualizar uma entrada existente. Ocorrerá uma exceção se você chamar este método em uma entrada que não existe quando o método for chamado ou durante o processamento de confirmação.

Método getIndex
O método getIndex tenta obter um índice denominado que é baseado no BackingMap. O índice não pode ser compartilhado entre encadeamentos e funciona de acordo com as mesmas regras que um objeto Session. O objeto index retornado deve ser convertido para a interface de índice do aplicativo tal como a interface MapIndex, a interface MapRangeIndex ou uma interface de índice customizada.

Método clear
O método clear remove todas as entradas de cache de um mapa de todas as partições. Esta operação é uma função auto-commit, assim, nenhuma transação ativa deverá estar presente ao chamar o método clear.

Nota: O método clear limpa apenas o mapa no qual é chamado, deixando quaisquer mapas de entidade relacionados não-afetados. Este método não chama o plug-in do Utilitário de Carga.

Mapas Dinâmicos

Com mapas dinâmicos, é possível criar mapas depois que grade de dados já tiver sido inicializada.

Nas versões anteriores, o WebSphere eXtreme Scale precisava que você definisse mapas antes de inicializar o ObjectGrid. Como resultado, você precisava criar todos os mapas a serem usados antes de executar transações em relação a qualquer um dos mapas.

Vantagens dos mapas dinâmicos

A introdução de mapas dinâmicos reduz a necessidade de definir todos os mapas antes da inicialização. Por meio do uso de mapas modelos, os mapas podem agora ser criados após o ObjectGrid ter sido inicializado.

Mapas modelos são definidos no arquivo XML do ObjectGrid. As comparações de modelo são executadas quando uma Sessão requer um mapa que não foi previamente definido. Se o nome do novo mapa corresponder à expressão regular de um mapa modelo, o mapa é criado dinamicamente e recebe o nome do mapa solicitado. Este mapa recentemente criado herda todas as configurações do mapa modelo como definido pelo arquivo XML ObjectGrid.

Criando Mapas Dinâmicos

A criação de mapa dinâmico está ligada ao método Session.getMap(String). Chamadas para este mapa retornam um ObjectMap baseado no BackingMap que foi configurado pelo arquivo XML ObjectGrid.

Passar uma Sequência que corresponde à expressão regular de um mapa modelo resulta na criação de um ObjectMap e em um BackingMap associado.

Consulte a documentação da API para obter mais informações sobre o método Session.getMap(String cacheName).

Definir uma mapa modelo em XML é tão simples quanto configurar um atributo booleano modelo no elemento backingMap. Quando o modelo é configurado para true, o nome do backingMap é interpretado como uma expressão regular.

O WebSphere eXtreme Scale usa a correspondência padrão de expressão regular Java. Para obter mais informações sobre o mecanismo de expressão regular em Java, consulte a documentação da API para o pacote e as classes java.util.regex.

Nota: Quando estiver definindo mapas de modelo, verifique se os nomes dos mapas são exclusivos o suficiente para que o aplicativo possa corresponder a apenas um dos mapas de modelo com o método Session.getMap(String mapName). Se o método getMap() corresponder a mais de um padrão de mapa de modelo, uma exceção `IllegalArgumentException` resultará.

Um arquivo XML do ObjectGrid XML de amostra com um mapa modelo definido está a seguir.

```
<?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>
```

O arquivo XML anterior define um mapa modelo em uma mapa não-modelo. O nome do mapa modelo é uma expressão regular: `templateMap.*`. Quando o método Session.getMap(String) é chamado com um nome de mapa correspondente a esta expressão regular, o aplicativo cria um mapa.

Exemplo

A configuração de um mapa modelo é necessária para criar um mapa dinâmico. Inclua um booleano modelo em um backingMap no arquivo XML do ObjectGrid.

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

O nome do mapa modelo é tratado como uma expressão regular.

Chamar o método Session.getMap(String cacheName) com um cacheName que seja uma correspondência para a expressão regular resulta na criação do mapa dinâmico. Um objeto ObjectMap é retornado a partir desta chamada de método e um objeto BackingMap associado é criado.

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

O mapa criado mais recentemente é configurado com todos os atributos e plug-ins que foram definidos na definição do mapa modelo. Considere novamente o arquivo XML do ObjectGrid anterior.

Um mapa dinâmico criado com base no mapa modelo neste arquivo XML teria um evictor configurado e sua estratégia de bloqueio seria pessimista.

Nota: Um modelo não é um BackingMap real. Isto é, o ObjectGrid "contabilidade" não contém um mapa "templateMap.*" real. O modelo é usado apenas como uma base para a criação de mapa dinâmico. Entretanto, você deve incluir o mapa dinâmico no elemento mapRef no arquivo XML da política de implementação nomeado exatamente como no XML do ObjectGrid. Este elemento identifica quais mapas dinâmicos são definidos e em quais mapSet.

Considere a mudança no comportamento do método Session.getMap(String cacheName) ao usar mapas modelos. Antes do WebSphere eXtreme Scale Versão 7.0, todas as chamadas ao método Session.getMap(String cacheName) resultavam em uma exceção UndefinedMapException se o mapa solicitado não existisse. Com mapas dinâmicos, cada nome que corresponde à expressão regular para um mapa modelo resulta na criação do mapa. Certifique-se de anotar o número de mapas que o seu aplicativo cria, particularmente se a sua expressão regular for genérica.

Também, ObjectGridPermission.DYNAMIC_MAP é necessário para a criação de mapa dinâmico quando a segurança doeXtreme Scale está ativada. Esta permissão é verificada quando o método Session.getMap(String) é chamado. Para obter mais informações, consulte o informações sobre a autorização do aplicativo cliente na *Visão Geral do Produto*.

Exemplos Adicionais

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>
```

Limitações e Considerações

Limitações:

- O elemento QuerySchema não suporta o modelo para mapName.
- Não é possível usar entidades com mapas dinâmicos.
- Um BackingMap de entidade é implicitamente definido, mapeado para a entidade através do nome da classe.

Considerações:

- Muitos plug-ins não têm como determinar o mapa com o qual cada plug-in está associado.
- Outros plug-ins se diferenciam usando um nome de mapa ou nome de BackingMap como um argumento.
- Quando estiver definindo mapas de modelo, verifique se os nomes dos mapas são exclusivos o suficiente para que o aplicativo possa corresponder a apenas um dos mapas de modelo usando o método Session.getMap(String mapName). Se o método getMap() corresponder a mais de um padrão de mapa de modelo, uma exceção `IllegalArgumentException` resultará.

ObjectMap e JavaMap

Uma instância do JavaMap é obtida de um objeto ObjectMap. A interface JavaMap possui as mesmas assinaturas de método que ObjectMap, mas com manipulação de exceção diferente. O JavaMap estende a interface java.util.Map, portanto, todas as exceções são instâncias da classe java.lang.RuntimeException. Como o JavaMap estende a interface java.util.Map, é fácil utilizar o WebSphere eXtreme Scale rapidamente com um aplicativo existente que utiliza uma interface java.util.Map para armazenamento e cache do objeto.

Obter uma Instância do JavaMap

Um aplicativo obtém uma instância do JavaMap a partir de um objeto ObjectMap utilizando o método ObjectMap.getJavaMap. O trecho de código a seguir demonstra como obter uma instância 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;
```

Um JavaMap é suportado pelo ObjectMap a partir do qual ele foi obtido. Chamar o método getJavaMap várias vezes utilizando um ObjectMap particular sempre retorna a mesma instância do JavaMap.

Métodos

A interface JavaMap suporta apenas um subconjunto dos métodos na interface java.util.Map. A interface java.util.Map suporta os seguintes 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()

Todos os métodos herdados da interface java.util.Map resultam em uma exceção java.lang.UnsupportedOperationException.

Mapas como Filas FIFO

Com o WebSphere eXtreme Scale, é possível fornecer um recurso semelhante à fila first-in first-out (FIFO) para todos os mapas. O WebSphere eXtreme Scale controla a ordem de inserção para todos os mapas. Um cliente pode solicitar um mapa para a próxima entrada não-bloqueada em um mapa na ordem de inserção e bloqueia a entrada. Este processo permite que vários clientes consumam entradas do mapa de maneira eficiente.

Exemplo de FIFO

O trecho de código a seguir mostra um cliente entrando em um loop para processar entradas do mapa até que o mapa seja esvaziado. O loop inicia uma transação e, então, chama o método ObjectMap.getNextKey(5000). Este método retorna a chave da próxima entrada desbloqueada disponível e a bloqueia. Se a transação é bloqueada por mais de 5000 milissegundos, então, o método retorna null.

```
Session session = ...;
ObjectMap map = session.getMap("xxx");
// this needs to be set somewhere to stop this loop
boolean timeToStop = false;

while(!timeToStop)
{
  session.begin();
  Object msgKey = map.getNextKey(5000);
  if(msgKey == null)
  {
    // current partition is exhausted, call it again in
    // a new transaction to move to next partition
    session.rollback();
    continue;
  }
  Message m = (Message)map.get(msgKey);
  // now consume the message
  ...
  // need to remove it
  map.remove(msgKey);
  session.commit();
}
```

Modo Local versus Modo do Cliente

Se o aplicativo estiver utilizando um núcleo local, ou seja, se ele não for um cliente, então o mecanismo funcionará conforme descrito anteriormente.

Para o modo cliente, se a Java virtual machine (JVM) for um cliente, então, o cliente inicialmente se conecta ao primário de partição aleatório. Se não existir nenhum trabalho em tal partição, então, o cliente move para a próxima partição para procurar trabalho. O cliente localiza uma partição com entrada ou executa um loop pela partição aleatória inicial. Se o cliente executa um loop pela partição inicial, então, ele retorna um valor nulo para o aplicativo. Se o cliente localiza uma partição com um mapa que possui entradas, então ele consome entradas até que nenhuma esteja disponível para o período de tempo limite. Após o tempo limite passar, então, um nulo é retornado. Esta ação significa que quando um nulo é retornado e um mapa particionado é utilizado, então, ele deve iniciar uma nova transação e retomar o atendimento. O fragmento de amostra do código anterior possui este comportamento.

Exemplo

Quando você está executando com um cliente é uma chave é retornada, tal transação é vinculada à partição com a entrada para tal chave. Se não desejar atualizar nenhum outro mapa durante tal transação, então, não há um problema. Se você não desejar atualizar, então, será possívelapenas atualizar os mapas a partir da mesma partição que o mapa, a partir da qual obteve a chave. A entrada que é retornada do método getNextKey precisa fornecer ao aplicativo uma maneira de descobrir dados relevantes nesta partição. Como exemplo, se você possui dois mapas; um para eventos e um outro para tarefas que os eventos impactam. Você define os dois mapas com as seguintes 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;
}
```

A tarefa possui um ID, que é uma cadeia, e um estado, que é um número inteiro. Suponha que você deseja incrementar o estado sempre que chega um evento. Os

eventos são armazenados no Mapa de JobEvent. Cada entrada possui uma referência para a tarefa que o evento tem interesse. O código para o listener fazer isto é semelhante ao exemplo a seguir:

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);
   // process the event, here we just increment the
   // job state
   event.job.jobState++;
   em.getTransaction().commit();
  }
 }
}
```

Um listener é iniciado em um encadeamento pelo aplicativo. O listener é executado até que o método stopListening seja chamado. O método processJobEvents é executado no encadeamento até que o método stopListening seja chamado. O loop bloqueia a espera por um eventKey a partir do Mapa JobEvent e, em seguida, utiliza o EntityManager para acessar o objeto de evento, aponta para a tarefa e incrementa o estado.

A API do EntityManager não possui um método getNextKey, mas o ObjectMap possui. Portanto, o código utiliza o ObjectMap para JobEvent para obter a chave. Se um mapa é utilizado com entidades, então ele não mais armazena objetos. Em vez disso, ele armazena Tuplas; um objeto Tupla para a chave e um objeto Tupla para o valor. O método EntityManager.find aceita uma Tupla para a chave.

O código para criar um evento é semelhante ao exemplo a seguir:

```
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();
```

Você localiza a tarefa para o evento, constrói um evento, aponta o evento para a tarefa, insere o evento no Mapa de JobEvent e consolida a transação.

Utilitários de Carga e Mapas FIFO

Se você desejar apoiar um mapa que é utilizado como uma fila FIFO com um Utilitário de Carga, então, pode ser necessário executar algum trabalho adicional. Se a ordem das entradas no mapa não for uma preocupação, não há trabalho extra. Se a ordem for importante, então, é necessário incluir um número de sequência em todos os registros inseridos quando você está persistindo os registros para o backend. O mecanismo de pré-carregamento deve ser escrito para inserir os registros na inicialização utilizando esta ordem.

Objetos de Armazenamento em Cache e seus Relacionamentos (API EntityManager)

A maioria dos produtos de cache utiliza APIs baseadas em mapa para armazenar dados como pares de chave-valor. A API ObjectMap e o cache dinâmico no WebSphere Application Server, entre outros, usam essa abordagem. Entretanto, APIs baseadas em mapas têm limitações. A API EntityManager simplifica a interação com a grade de dados ao fornecer uma maneira fácil de declarar e interagir com um gráfico complexo de objetos relacionados.

Limitações de API Baseada em Mapa

Se estiver usando uma API baseada em mapa, como o cache dinâmico no WebSphere Application Server ou a API ObjectMap, leve em consideração as seguintes limitações:

- Os índices e consultas devem usar a reflexão para consultar campos e propriedades nos objetos em cache.
- A serialização de dados customizados é necessária para atingir o desempenho para objetos complexos.
- É difícil trabalhar com gráficos de objetos. Você deve desenvolver o aplicativo para armazenar referências artificiais entre objetos e uni-los manualmente.

Benefícios da API EntityManager

A API EntityManager usa a infraestrutura baseada em mapa existente, porém converte os objetos de entidade em, e a partir de, tuplas antes de serem armazenados ou lidos a partir do mapa. Um objeto de entidade é transformado em uma tupla de chave e uma tupla de valor, que são então armazenadas como pares chave-valor. Uma tupla é uma matriz de atributos primitivos.

Este conjunto de APIs segue o estilo de programação Plain Old Java Object (POJO) que é adotado pela maioria das estruturas.

Tarefas relacionadas

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Referências relacionadas

"Agente de Instrumentação de Desempenho da Entidade" na página 465
É possível melhorar o desempenho de entidades de acesso a campo ativando o agente de instrumentação do WebSphere eXtreme Scale ao utilizar o Java Development Kit (JDK) Versão 1.5 ou posterior.

"Definindo um Esquema de Entidade" na página 169
Um ObjectGrid pode ter inúmeros esquemas de entidade lógicos. As entidades são definidas usando as classes Java anotadas, o XML ou uma combinação de classes XML e Java. Entidades definidas são registradas com um servidor eXtreme Scale e ligadas a BackingMaps, índices e outros plug-ins.

"Listeners de Entidade e Métodos de Retorno de Chamada" na página 185
Os aplicativos podem ser notificados quando o estado de uma entidade é alterado de estado para estado. Dois mecanismos de retorno de chamada existem para os eventos de alteração de estado: os métodos de retorno de chamada do ciclo de vida, que são definidos em uma classe de entidade e são chamados sempre que o estado da entidade for alterado, e os listeners de entidade que são mais genéricos porque o listener da entidade pode ser registrado em várias entidades.

"Exemplos do Listener de Entidade" na página 189
É possível gravar EntityListeners com base em seus requisitos. Veja a seguir vários scripts de exemplo.

"Interface EntityTransaction" na página 201
É possível utilizar a interface EntityTransaction para demarcar transações.

Gerenciamento de Relacionamentos

Linguagens orientadas a objetos como Java, e relacionamentos ou associações de suporte a bancos de dados relacionais. Os relacionamentos diminuem a quantidade de armazenamento através do uso de referências de objetos ou chaves estrangeiras.

Ao usar relacionamentos em uma grade de dados, os dados deverão ser organizados em uma árvore limitada. Um tipo de raiz deve existir na árvore e todos os filhos devem estar associados a apenas uma raiz. Por exemplo: Departamento pode ter muitos Funcionários e um Funcionário pode ter muitos Projetos. Porém um Projeto não pode ter muitos Funcionários pertencentes a diferentes departamentos. Depois de uma raiz ser definida, todo o acesso a este objeto raiz e seus descendentes será gerenciado através da raiz. O WebSphere eXtreme Scale usa o código hash da chave do objeto raiz para escolher uma partição. Por exemplo:

```
partition = (hashCode MOD numPartitions).
```

Quando todos os dados para um relacionamento estiverem ligados a um única instância do objeto, toda a árvore pode ser co-localizada em uma única partição e pode ser acessada muito eficientemente usando uma transação. Se os dados englobarem múltiplos relacionamentos, então múltiplas partições devem estar envolvidas que envolvem chamadas remotas adicionais, o que pode levar a gargalos no desempenho.

Dados de Referência

Alguns relacionamentos incluem dados de consulta ou de referência como: `CountryName`. Para dados de consulta ou de referência, os dados devem existir em cada partição. Os dados podem ser acessados por qualquer chave raiz e o mesmo resultado é retornado. Os dados de referência como estes devem ser usados apenas nos casos em que os dados forem razoavelmente estáticas. Atualizar esses dados pode ser dispendioso porque eles precisam ser atualizados em cada partição. A API DataGrid é uma técnica comum para manter os dados de referência atualizados.

Custos e Benefícios de Normalização

A normalização dos dados usando os relacionamentos pode ajudar a reduzir a quantidade de memória usada pela grade de dados pois a duplicação dos dados é diminuída. Porém, em geral, quanto mais dados relacionais forem incluídos, menos eles irão expandir. Quando os dados são agrupados juntos, torna-se mais caro manter os relacionamentos e manter os tamanhos gerenciáveis. Como os dados das partições da grade baseiam-se na chave da raiz da árvore, o tamanho da árvore não é levado em consideração. Assim, se você tiver uma grande quantidade de relacionamentos para uma instância da árvore, a grade de dados poderá ficar desequilibrada, fazendo com que uma partição mantenha mais dados do que as outras.

Quando os dados forem não normalizados ou simplificados, os dados que normalmente seriam compartilhados entre os dois objetos são duplicados e cada tabela pode ser particionada de modo independente, oferecendo uma grade de dados muito mais equilibrada. Apesar disto aumentar a quantidade de memória usada, permite que o aplicativo escale pois uma única linha de dados pode ser acessada que pode ter todos os dados necessários. Isto é ideal para grades com maior quantidade de leituras pois a manutenção dos dados se torna mais cara.

Para obter informações adicionais, consulte Classificação de sistemas XTP e escalamento.

Gerenciamento de Relacionamentos Usando as APIs de Acesso a Dados

A API ObjectMap é a mais rápida, mais flexível e granular das APIs de acesso a dados, oferecendo uma abordagem transacional baseada em sessão no acesso aos dados na grade de mapas. A API ObjectMap permite que os clientes usem operações comuns, como create, read, update e delete (CRUD), para gerenciar pares de valores de chave de objetos na grade de dados distribuída.

Ao usar a API ObjectMap, os relacionamentos de objetos devem ser expressos pela incorporação da chave estrangeira para todos os relacionamentos no objeto-pai.

A seguir, está um exemplo.

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

A API EntityManager simplifica o gerenciamento de relacionamentos através da extração de dados persistentes a partir de objetos incluindo as chaves estrangeiras. Quando o objeto é posteriormente recuperado da grade de dados, o gráfico de relacionamentos é reconstruído, como no seguinte exemplo.

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

A API EntityManager é muito semelhante a outras tecnologias de persistência do objeto Java como JPA e Hibernate na qual ela sincroniza um gráfico de instâncias de objetos Java gerenciados com o armazenamento persistente. Nesse caso, o armazenamento persistente é uma grade de dados do eXtreme Scale, em que cada entidade é representada como um mapa e o mapa contém os dados da entidade em vez das instâncias do objeto.

Definindo um Esquema de Entidade

Um ObjectGrid pode ter inúmeros esquemas de entidade lógicos. As entidades são definidas usando as classes Java anotadas, o XML ou uma combinação de classes XML e Java. Entidades definidas são registradas com um servidor eXtreme Scale e ligadas a BackingMaps, índices e outros plug-ins.

Ao projetar uma esquema de entidade, é necessário concluir as seguintes tarefas:

1. Definir as entidades e seus relacionamentos.
2. Configurar o eXtreme Scale.
3. Registrar as entidades.
4. Criar aplicativos baseados em entidade que interajam com as APIs EntityManager do eXtreme Scale.

Configuração do Esquema de Entidade

Um esquema de entidade é um conjunto de entidades e relacionamentos entre as entidades. Em um aplicativo eXtreme Scale com várias partições, as opções e restrições a seguir aplicam-se aos esquemas de entidade:

- Cada esquema de entidade deve ter uma raiz única definida. Esta é conhecida como a raiz do esquema.
- Todas as entidades para um determinado esquema devem estar no mesmo conjunto de mapas, o que significa que todas as entidades que podem ser alcançadas a partir de uma raiz de esquema com relacionamentos de chave e não-chave devem ser definidas no mesmo conjunto de mapas como a raiz do esquema.
- Cada entidade pode pertencer a apenas um esquema de entidade.
- Cada aplicativo eXtreme Scale pode ter vários esquemas.

Entidades são registradas com uma instância de ObjectGrid antes de serem inicializadas. Cada entidade definida deve ser nomeada exclusivamente e ligada automaticamente a um BackingMap de ObjectGrid de mesmo nome. O método de inicialização varia dependendo da configuração que você utilizando:

Configuração do eXtreme Scale local

Se estiver utilizando um ObjectGrid local, será possível configurar programaticamente o esquema de entidade. Neste modo, é possível utilizar os métodos ObjectGrid.registerEntities para registrar classes de entidade anotadas ou um arquivo descritor de metadados.

Configuração do eXtreme Scale distribuída

Se você estiver utilizando uma configuração do eXtreme Scale distribuída, é necessário fornecer um arquivo descritor de metadados da entidade com o esquema de entidade.

Para obter detalhes adicionais, consulte “Entity Manager em um Ambiente Distribuído” na página 177.

Requisitos de Entidade

Os metadados de entidade são configurados com o uso de arquivos de classe Java, um XML do descritor de entidade ou ambos. No mínimo, o XML do descritor de entidade é requerido para identificar quais BackingMaps do eXtreme Scale devem ser associados às entidades. Os atributos persistentes da entidade e seus relacionamentos com outras entidades são descritos em uma classe Java anotada (classe de metadados da entidade) ou arquivo XML do descritor de entidade. A classe de metadados da entidade, quando especificada, também é utilizada pela API EntityManager para interagir com dados na grade.

Uma grade do eXtreme Scale pode ser definida sem fornecer quaisquer classes de entidade. Isso pode ser benéfico quando o servidor e o cliente estiverem interagindo diretamente com os dados da tupla armazenados nos mapas subjacentes. Tais entidades são definidas completamente no arquivo XML do descritor de entidade e são referidas como entidades sem classe.

Entidades sem Classe

As entidades sem classe são úteis quando não é possível incluir classes de aplicativo no caminho de classe do servidor ou do cliente. Tais entidades são definidas no arquivo XML do descritor de entidade, onde o nome da classe da entidade é especificado utilizando um identificador de entidade sem classe no formato: @<identificador de entidade>. O símbolo @ identifica a entidade como sem classe e é utilizado para mapear associações entre entidades. Consulte a figura "Metadados da Entidade sem Classe" para ver um exemplo de um arquivo XML do descritor de metadados da entidade com duas entidades sem classe definidas.

Se um cliente ou servidor eXtreme Scale não tiver acesso às classes, eles poderão utilizar a API EntityManager utilizando as entidades sem classes. Casos de uso comuns incluem o seguinte:

- O contêiner do eXtreme Scale é hospedado em um servidor que não permite classes de aplicativo no caminho de classe. Nesse caso, os clientes ainda podem acessar a grade utilizando a API EntityManager de um cliente em que as classes sejam permitidas.
- O cliente eXtreme Scale não requer acesso às classes de entidade porque o cliente está utilizando um cliente não Java, como o serviço de dados REST do eXtreme Scale, ou o cliente está acessando os dados da tupla na grade utilizando a API de ObjectMap.

Se os metadados da entidade forem compatíveis entre cliente e servidor, eles poderão ser criados utilizando classes de metadados da entidade, um arquivo XML ou ambos.

Por exemplo, a "Classe de Entidade Programática" na figura a seguir é compatível com o código de metadados sem classe na próxima seção.

Classe de entidade programática

```
@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;
}
```

Versões, Chaves e Campos sem Classe

Conforme mencionado anteriormente, as entidades sem classe são configuradas completamente no arquivo descritor XML da entidade. As entidades baseadas em classe definem seus atributos utilizando campos, propriedades e anotações Java. Portanto, as entidades sem classe precisam definir a estrutura de chave e atributo no descritor XML da entidade com as tags <basic> e <id>.

Metadados da entidade sem classe

```
<?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 entidade acima tem um elemento <id>. Uma entidade sem classe deve ter um ou mais de um elemento <id> definido, ou uma associação com valor único que represente a chave para a entidade. Os campos da entidade são representados pelos elementos <basic>. Os elementos <id>, <version> e <basic> requerem um nome e um tipo nas entidades sem classe. Consulte a seguinte seção de tipos de atributo suportados para obter detalhes sobre os tipos suportados.

Requisitos de Classe da Entidade

As entidades baseadas em classe são identificadas por meio da associação de vários metadados com uma classe Java. Os metadados podem ser especificados utilizando anotações do Java Platform, Standard Edition 5, um arquivo descritor de metadados de entidade ou uma combinação de anotações e o arquivo descritor. As classes de entidade precisam atender os seguintes critérios:

- A anotação `@Entity` é especificada no arquivo descritor XML da entidade.
- A classe tem um construtor no-argument público ou protegido.
- Ela deve ser uma classe de nível superior. Interfaces e tipos enumerados não são classes de entidade válidas.
- Não é possível utilizar a palavra-chave final.
- Não é possível utilizar herança.
- É necessário ter nome e tipo exclusivos para cada instância de ObjectGrid.

As entidades todas possuem um nome e tipo exclusivos. O nome, se utilizando anotações, é o nome simples (curto) da classe por padrão, mas pode ser substituído utilizando o atributo name da anotação `@Entity`.

Atributos Persistentes

O estado persistente de uma entidade é acessado por cliente e o entity manager utilizando qualquer um dos campos (variáveis de instância) ou acessores de propriedade do estilo Enterprise JavaBeans. Cada entidade deve definir um acesso baseado em campo ou em propriedade. Entidades anotadas são field-access se os campos de classe são anotados e property-access se o método getter da propriedade é anotada. Uma mistura de acesso por campo e acesso por propriedade não é permitida. Se o tipo não puder ser automaticamente determinado, o atributo **`accessType`** na anotação `@Entity` ou XML equivalente pode ser utilizado para identificar o tipo de acesso.

Campos persistentes

As variáveis de instância da entidade field-access são acessadas diretamente a partir do entity manager e dos clientes. Os campos que são marcados com o modificador transiente ou a anotação transiente são ignorados. Campos persistentes devem ter modificadores final ou static.

Propriedades persistentes

As entidades de acesso de propriedade devem seguir as convenções de assinatura do JavaBeans para as propriedades de leitura e gravação. Os métodos que não seguirem as convenções do JavaBeans ou possuírem a anotação Transient no método getter serão ignorados. Para uma propriedade do tipo `T`, deve haver um método getter getProperty que retorne um valor do tipo T e um método setter void setProperty(T). Para tipos booleanos, o método getter pode ser expresso como isProperty, retornando true ou false. As propriedades persistentes não podem ter um modificador estático.

Tipos de atributos suportados

O seguinte campo persistente e tipos de propriedades são suportados:

- Os tipos primitivos Java incluindo wrappers
- 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

Tipos de atributos serializáveis são suportados mas possuem limitações de desempenho, consulta e detecção de alterações. Dados persistentes que não podem ser colocados em proxy, como matrizes e objetos serializáveis de usuário, devem ser reatribuídos para a entidade se alterados.

Os atributos serializáveis são representados no arquivo XML do descritor de entidade utilizando o nome de classe do objeto. Se o objeto for uma matriz, o tipo de dado será representado utilizando a forma interna Java. Por exemplo, se um tipo de dado de atributo for java.lang.Byte[][], a representação de cadeia será [[Ljava.lang.Byte;

Os tipos serializáveis de usuários devem obedecer às seguintes boas práticas:

- Implemente métodos de serialização de alto desempenho. Implemente a interface java.lang.Cloneable e um método clone público.
- Implemente a interface java.io.Externalizable.
- Implemente iguais e hashCode

Associações de Entidade

Associações de entidades bidirecionais e unidirecionais, ou relacionamentos entre entidades podem ser definidos como um-para-um, muitos-para-um, um-para-muitos e muitos-para-muitos. O gerenciador de entidade resolve automaticamente os relacionamentos da entidade com as referências de chave apropriadas ao armazenar entidades.

A grade do eXtreme Scale é um cache de dados e não força integridade referencial como um banco de dados. Embora os relacionamentos permitam persistência em cascata e removam operações para entidades-filhas, eles não detectam ou impões links quebrados em objetos. Quando remover um objeto-filho, a referência a esse objeto deve ser removida do pai.

Se você definir uma associação bidirecional entre duas entidades, será preciso definir o proprietário do relacionamento. Em uma associação para-muitos, o lado muitos do relacionamento é sempre o lado do proprietário. Se a propriedade não puder ser determinada automaticamente, então, o atributo **mappedBy** da anotação ou equivalente XML, deve ser especificado. O atributo **mappedBy** identifica o campo na entidade de destino que é a proprietária do relacionamento. Este atributo também ajuda a identificar os campos relacionados quando há vários atributos do mesmo tipo e cardinalidade.

Associações com valor único

Associações um-para-um e muitos-para-um são indicadas utilizando anotações @OneToOne e @ManyToOne ou atributos XML equivalentes. O tipo de entidade de

destino é determinado pelo tipo de atributo. O exemplo a seguir define uma associação unidirecional entre Person e Address.s A entidade Cliente tem uma referência a uma entidade Endereço. Nesse caso, a associação poderia também ser de muitos-para-um, já que não há relacionamento inverso.

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

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

Para especificar um relacionamento bidirecional entre as classes Customer e Address, inclua uma referência à classe Customer a partir da classe Address e inclua a anotação apropriada para marcar o lado inverso do relacionamento. Como essa associação é um-para-um, você precisa especificar um proprietário do relacionamento utilizando o atributo mappedBy na anotação @OneToOne.

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

Associações com valor de coleta

Associações um-para-muitos e muitos-para-muitos são denotadas utilizando as anotações `@OneToMany` e `@ManyToMany` ou atributos XML XML equivalente. Todos os relacionamentos muito são representados utilizando os tipos: java.util.Collection, java.util.List ou java.util.Set. O tipo de entidade de destino é determinado pelo tipo genérico de Collection, List ou Set ou explicitamente utilizando atributo **`targetEntity`** na anotação `@OneToMany` ou `@ManyToMany` (ou XML equivalente).

No exemplo anterior, não é prático ter um objeto address por cliente porque muitos clientes podem compartilhar um endereço ou podem ter vários endereços. Esta situação é melhor resolvida utilizando uma associação many:

```
@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;
}
```

Neste exemplo, existem dois diferentes relacionamentos entre as mesmas entidades: um relacionamento de endereços Home e Work. Uma Collection não genérica é

utilizada para o atributo **workCustomers** para demonstrar como utilizar o atributo **targetEntity** quando genéricos não estiverem disponíveis.

Associações sem classe

As associações de entidade sem classe são definidas no arquivo XML do descritor de metadados da entidade de forma semelhante ao modo como as associações baseadas em classe são definidas. A única diferença é que em vez de a entidade de destino apontar para uma classe real, ela aponta para o identificador da entidade sem classe utilizado para o nome de classe da entidade.

Este é um exemplo:

```
<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>
```

Chaves Primárias

Todas as entidades devem ter uma chave primária, que pode ser uma chave simples (atributo único) ou composta (atributos múltiplos). Os atributos-chave são denotados utilizando a anotação Id ou definidos no arquivo descritor XML da entidade. Atributos-chave possuem os seguintes requisitos:

- O valor de uma chave primária não pode mudar.
- Um atributo de chave principal deve ser um dos seguintes tipos: tipo primitivo Java e wrappers, java.lang.String, java.util.Date ou java.sql.Date.
- Uma chave primária pode conter qualquer número de associações com valor único. A entidade de destino da associação de chave primária não pode ter uma associação inversa direta ou indiretamente à entidade de origem.

As chaves primárias compostas podem definir opcionalmente uma classe de chave primária. Uma entidade é associada a uma classe de chave primária com o uso da anotação `@IdClass` ou arquivo descritor XML da entidade. Uma anotação `@IdClass` é útil junto com o método EntityManager.find.

As classes de chave primária possuem os seguintes requisitos:

- Ela deve ser pública com um construtor no-argument.
- O tipo de acesso da classe de chave primária é determinado pela entidade que declara a classe da chave primária.
- Se acesso por propriedade, as propriedades da classe de chave primária precisam ser públicas ou protegidas.
- Os campos-chave primários ou propriedades devem corresponder aos nomes do atributo-chave e tipos definidos na entidade de referência.
- As classes de chave primária devem implementar os métodos equals e hashCode.

Este é um exemplo:

```
@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) {...}
}
```

Chaves primárias sem classe

Entidades sem classe devem ter pelo menos um elemento <id> ou uma associação no arquivo XML com o atributo `id=true`. Um exemplo de ambos seria semelhante ao seguinte:

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

Lembre-se:
A tag XML <id-class> não é suportada para entidades sem classe.

Proxies de Entidade e Intercepção de Campo

As classes de entidade e os tipos de atributo suportados mutáveis são estendidos pelas classes de proxy para entidades property-access e bytecode-enhanced para entidades field-access Java Development Kit (JDK) 5. Todos os acessos à entidade, mesmo por meio de métodos de negócios internos e dos métodos equals, devem utilizar o campo apropriado ou os métodos de acesso da propriedade.

Os proxies e interceptores de campo são utilizados para permitir que o entity manager controle o estado da entidade, determine se a entidade foi alterada e aprimore o desempenho. Os interceptores de campo estão disponíveis apenas em plataformas Java SE 5 quando o agente de instrumentação da entidade é configurado.

Atenção: Ao utilizar entidades property-access, o método equals deve utilizar o operador instanceof para comparar a instância atual com o objeto input. Toda a introspecção do objeto de destino deve ser por meio das propriedades do objeto, e não dos campos em si, pois a instância do objeto será o proxy.

Conceitos relacionados

"Ajustando o Desempenho da Interface EntityManager" na página 463
A interface EntityManager separa aplicativos do estado de suspensão no armazenamento de dados da grade do servidor.

"Objetos de Armazenamento em Cache e seus Relacionamentos (API EntityManager)" na página 166
A maioria dos produtos de cache utiliza APIs baseadas em mapa para armazenar dados como pares de chave-valor. A API ObjectMap e o cache dinâmico no WebSphere Application Server, entre outros, usam essa abordagem. Entretanto, APIs baseadas em mapas têm limitações. A API EntityManager simplifica a interação com a grade de dados ao fornecer uma maneira fácil de declarar e interagir com um gráfico complexo de objetos relacionados.

"Entity Manager em um Ambiente Distribuído"
É possível usar a API EntityManager com um ObjectGrid local ou em um ambiente distribuído do eXtreme Scale . A principal diferença é como você se conecta a esse ambiente remoto. Após você estabelecer uma conexão, não existe diferença entre o uso de um objeto Session ou uma API do EntityManager.

"Interagindo com EntityManager" na página 182
Geralmente os aplicativos primeiro obtêm uma referência do ObjectGrid e, depois, uma Sessão dessa referência para cada encadeamento. As sessões não podem ser compartilhadas entre encadeamentos. Um método extra em Session, o método getEntityManager, está disponível. Este método retorna uma referência para um gerenciador de entidades para uso para este encadeamento. A interface de EntityManager pode substituir as interfaces de Session e ObjectMap para todos os aplicativos. É possível utilizar essas APIs de EntityManager se o cliente tiver acesso às classes de entidade definidas.

"Suporte ao Plano de Carregamento do EntityManager" na página 191
Um FetchPlan é a estratégia que o gerenciador de entidade usa para recuperar objetos associados se o aplicativo precisar acessar relacionamentos.

"Filas de Consulta da Entidade" na página 196
Filas de consulte permitem que aplicativos criem uma fila qualificada por uma consulta no lado do servidor ou eXtreme Scale local sobre uma entidade. As entidades do resultado da consulta são armazenadas nesta fila. Atualmente, a fila de consulta é suportada apenas em um mapa que está utilizando a estratégia de bloqueio pessimista.

Tarefas relacionadas

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Entity Manager em um Ambiente Distribuído

É possível usar a API EntityManager com um ObjectGrid local ou em um ambiente distribuído do eXtreme Scale . A principal diferença é como você se conecta a esse ambiente remoto. Após você estabelecer uma conexão, não existe diferença entre o uso de um objeto Session ou uma API do EntityManager.

Arquivos de Configuração Necessários

Os seguintes arquivos de configuração XML são necessários:

- Arquivo XML descritor do ObjectGrid
- Arquivo XML descritor da entidade

- Arquivo XML descritor de implementação ou de grade de dados

Esses arquivos especificam as entidades e os BackingMaps que um servidor hospeda.

O arquivo descritor de metadados da entidade contém uma descrição das entidades que são utilizadas. No mínimo, você deve especificar o nome e a classe da entidade. Se você estiver executando em um ambiente Java Platform, Standard Edition 5, oeXtreme Scale automaticamente lê a classe da entidade e suas anotações. É possível definir atributos XML adicionais se a classe de entidade não tiver anotações ou se você precisar substituir os atributos de classe. Se estiver registrando as entidades sem classe, forneça todas as informações da entidade apenas no arquivo XML.

É possível usar o seguinte fragmento de configuração XML para definir uma grade de dados com entidades. Nesse fragmento, o servidor cria um ObjectGrid com o nome `bookstore` e um mapa de apoio associado com o nome `order`. O fragmento no arquivo `objectgrid.xml` refere-se ao arquivo `entity.xml`. Nesse caso, o arquivo `entity.xml` contém uma entidade, a 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 arquivo `objectgrid.xml` especifica o arquivo `entity.xml` com o atributo **`entityMetadataXMLFile`**. O valor pode ser um diretório relativo ou um caminho absoluto.

- **Para um relatório relativo:** Especifique o local relativo com o local do arquivo `objectgrid.xml`.
- **Para um caminho absoluto:** Especifique o local com um formato de URL, como `file://` ou `http://`.

Este é um exemplo do arquivo `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 exemplo assume que a classe Order teria os campos **`orderNumber`** e **`desc`** anotados de modo semelhante.

A seguir há um exemplo do arquivo `server.properties` sem classe equivalente:

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 obter informações sobre como iniciar os servidores, consulte o *Guia de Administração*. É possível usar ambos os arquivos `deployment.xml` e `objectgrid.xml` para iniciar o servidor de catálogos.

Conectando-se a um Servidor eXtreme Scale Distribuído

O seguinte código ativa o mecanismo de conexão para um cliente e servidor no mesmo computador:

```
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");
```

No fragmento de código anterior, observe a referência ao servidor eXtreme Scale remoto. Após estabelecer uma conexão, é possível invocar os métodos de API do EntityManager como persist, update, remove e find.

Atenção: Quando estiver usando entidades, passe o arquivo XML do descritor do ObjectGrid de substituição para o método connect. Se um valor nulo é passado para a propriedade clientOverrideURL e o cliente tem uma estrutura de diretório doferente da estrutura de diretório do servidor, então o cliente pode falhar em localizar os arquivos XML do descritor de entidade ou ObjectGrid. No mínimo, os arquivos XML de entidade e ObjectGrid para o servidor podem ser copiados para o cliente.

Anteriormente, o uso de entidades no cliente ObjectGrid exigia que você deixasse o XML do ObjectGrid e o XML da entidade disponíveis para o cliente de uma das duas maneiras a seguir:

1. Transmita um XML do ObjectGrid de substituição para o 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. Transmita nulo para o arquivo de substituição e certifique-se de que o XML do ObjectGrid e o XML da entidade referida estejam disponíveis para o cliente no mesmo caminho que no servidor.

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

Os arquivos XML eram necessários, independentemente de você querer ou não utilizar um subconjunto de entidades no lado do cliente. Esses arquivos não são mais necessários para o uso de entidades, conforme definido pelo servidor. Em vez disso, transmita nulo como o parâmetro overRideObjectGridXml como na opção 2 da seção anterior. Se o arquivo XML não estiver localizado no mesmo caminho configurado no servidor, o cliente usa a configuração da entidade no servidor.

Entretanto, se você utilizar entidades do subconjunto no cliente, será necessário fornecer um XML do ObjectGrid de substituição na opção 1.

Esquema do Lado do Cliente e do Servidor

O esquema do lado do servidor define o tipo de dado armazenado nos mapas em um servidor. O esquema do lado do cliente é um mapeamento para objetos de aplicativo do esquema no servidor. Por exemplo, é possível ter o seguinte esquema do lado do servidor:

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

Um cliente pode ter um objeto anotado como no exemplo a seguir:

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

Este cliente, então, toma uma entidade do lado do servidor e projeta o subconjunto da entidade no objeto do cliente. Esta projeção leva à economia de largura de banda e de memória em um cliente porque o cliente tem somente as informações que ele precisa em vez de todas as informações que estiverem na entidade do lado do servidor. Aplicativos diferentes podem utilizar seus próprios objetos em vez de forçarem todos os aplicativos a compartilharem um conjunto de classes para o acesso a dados.

O arquivo XML descritor da entidade do lado do cliente é necessário nos seguintes casos: se o servidor estiver em execução com entidades baseadas em classe enquanto o lado do cliente está em execução sem classes; ou se o servidor estiver sem classes e o cliente utilizar entidades baseadas em classe. Um modo de cliente sem classe permite que o cliente ainda execute consultas de entidade sem precisar acessar classes físicas. Supondo que o servidor tenha registrado a entidade ServerPerson acima, o cliente poderia substituir a grade de dados por um arquivo `entity.xml` como a seguir:

```
<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 arquivo atinge uma entidade de subconjunto equivalente no cliente, sem exigir que o cliente forneça a classe anotada real. Se o servidor não tiver classe e o cliente tiver, o cliente fornecerá um arquivo XML do descritor de entidade de substituição. Este arquivo XML do descritor de entidade contém uma substituição para a referência do arquivo de classe.

Fazendo Referência ao Esquema

Se seu aplicativo estiver em execução no Java SE 5, o aplicativo poderá ser incluído nos objetos usando anotações. O EntityManager pode ler o esquema nas anotações de tais objetos. O aplicativo fornece ao tempo de execução do eXtreme Scale referências a esses objetos utilizando o arquivo `entity.xml`, que é referido no arquivo `objectgrid.xml`. O arquivo `entity.xml` lista todas as entidades, cada uma

associada a uma classe ou a um esquema. Se um nome de classe apropriado for especificado, o aplicativo tentará ler as anotações do Java SE 5 a partir dessas classes para determinar o esquema. Se você não anotar o arquivo de classe ou especificar um identificador sem classe como o nome de classe, o esquema será tirado do arquivo XML. O arquivo XML é utilizado para especificar todos os atributos, chaves e relacionamentos para cada entidade.

Uma grade de dados local não precisa de arquivos XML. O programa pode obter uma referência do ObjectGrid e invocar o método ObjectGrid.registerEntities para especificar uma lista de classes anotadas do Java SE 5 ou um arquivo XML.

O tempo de execução utiliza o arquivo XML ou uma lista de classes anotadas para localizar nomes de entidades, nomes e tipos de atributos, tipos e campos chave e relacionamentos entre entidades. Se o eXtreme Scale estiver executando em um servidor ou em modo independente, então ele cria automaticamente um mapa nomeado após cada entidade. Estes mapas podem ser customizados adicionalmente utilizando o arquivo `objectgrid.xml` ou APIs configuradas pelo aplicativo ou por estruturas de injeção tais como Spring.

Arquivo Descritor de Metadados da Entidade

Consulte Arquivo `emd.xsd` para obter informações adicionais sobre o arquivo descritor de metadados.

Tarefas relacionadas

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Referências relacionadas

"Agente de Instrumentação de Desempenho da Entidade" na página 465
É possível melhorar o desempenho de entidades de acesso a campo ativando o agente de instrumentação do WebSphere eXtreme Scale ao utilizar o Java Development Kit (JDK) Versão 1.5 ou posterior.

"Definindo um Esquema de Entidade" na página 169
Um ObjectGrid pode ter inúmeros esquemas de entidade lógicos. As entidades são definidas usando as classes Java anotadas, o XML ou uma combinação de classes XML e Java. Entidades definidas são registradas com um servidor eXtreme Scale e ligadas a BackingMaps, índices e outros plug-ins.

"Listeners de Entidade e Métodos de Retorno de Chamada" na página 185
Os aplicativos podem ser notificados quando o estado de uma entidade é alterado de estado para estado. Dois mecanismos de retorno de chamada existem para os eventos de alteração de estado: os métodos de retorno de chamada do ciclo de vida, que são definidos em uma classe de entidade e são chamados sempre que o estado da entidade for alterado, e os listeners de entidade que são mais genéricos porque o listener da entidade pode ser registrado em várias entidades.

"Exemplos do Listener de Entidade" na página 189
É possível gravar EntityListeners com base em seus requisitos. Veja a seguir vários scripts de exemplo.

"Interface EntityTransaction" na página 201
É possível utilizar a interface EntityTransaction para demarcar transações.

Interagindo com EntityManager

Geralmente os aplicativos primeiro obtêm uma referência do ObjectGrid e, depois, uma Sessão dessa referência para cada encadeamento. As sessões não podem ser compartilhadas entre encadeamentos. Um método extra em Session, o método getEntityManager, está disponível. Este método retorna uma referência para um gerenciador de entidades para uso para este encadeamento. A interface de EntityManager pode substituir as interfaces de Session e ObjectMap para todos os aplicativos. É possível utilizar essas APIs de EntityManager se o cliente tiver acesso às classes de entidade definidas.

Obtendo uma Instância de EntityManager de uma Sessão

O método getEntityManager está disponível em um objeto Session. O exemplo de código a seguir ilustra como criar uma instância de ObjectGrid local e acessar EntityManager. Consulte a interface EntityManager na documentação da API para obter detalhes sobre todos os métodos suportados.

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

Existe um relacionamento um para um entre o objeto Session e o objeto EntityManager. É possível utilizar o objeto EntityManager mais de uma vez.

Persistindo uma Entidade

A persistência de uma entidade significa salvar o estado de uma nova entidade em um cache ObjectGrid. Depois de chamar o método "persist", a entidade fica no estado "managed". A operação persist é transacional e a nova entidade fica armazenada no cache do ObjectGrid depois da confirmação do cache.

Cada entidade tem um BackingMap correspondente no qual as tuplas são armazenadas. O BackingMap tem o mesmo nome que a entidade e será criado quando a classe for registrada. O seguinte exemplo de código demonstra como criar um objeto Order usando a operação persist.

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

O objeto Order é criado com a chave 123 e o objeto é passado para método persist. É possível continuar a modificar o estado do objeto antes de consolidar a transação.

Importante: O exemplo anterior não inclui quaisquer limites transacionais necessários, como begin e commit. Consulte o “Tutorial: Armazenando Informações de Pedido nas Entidades” na página 7 tutorial do gerenciador de entidade no *Visão Geral do Produto* para obter mais informações.

Localizando uma Entidade

É possível localizar a entidade no cache do ObjectGrid com o método find fornecendo uma chave após o armazenamento da entidade no cache. Esse método não requer nenhum limite transacional, o que será útil em caso de semântica de somente leitura. O exemplo a seguir ilustra que apenas uma linha de código é suficiente para localizar a entidade.

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

Removendo uma Entidade

O método remove, a exemplo do método persist, é uma operação transacional. O exemplo a seguir mostra o limite transacional ao chamar os métodos begin e commit.

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

Primeiro, a entidade deve estar no estado managed antes que a remoção seja possível; o que pode ser conseguido chamando o método find dentro do limite transacional. Depois, chame o método remove na interface de EntityManager.

Invalidando uma Entidade

O método invalidate se comporta de maneira muito semelhante ao método remove, mas não chama nenhum plug-in do Utilitário de Carga. Use este método para remover entidades do ObjectGrid, mas para preservá-las no armazém de dados de backend.

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

Primeiro, a entidade deve ser gerenciada antes que possa ser invalidada, o que pode ser conseguido chamando o método find dentro do limite transacional. Após chamar o método find, será possível chamar o método invalidate na interface de EntityManager.

Atualizando uma Entidade

O método update também é uma operação transacional. A entidade deve ser gerenciada antes que quaisquer atualizações possam ser aplicadas.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
foundOrder.date = new Date(); // update the date of the order
em.getTransaction().commit();
```

No exemplo precedente, o método persist não é chamado depois da atualização da entidade. A entidade é atualizada no cache do ObjectGrid quando a transação é confirmada.

Consultas e Filas de Consultas

Com o mecanismo de consulta flexível, é possível recuperar entidades utilizando a API do EntityManager. Crie consultas do tipo SELECT para uma entidade ou esquema baseado em Objeto, utilizando a linguagem de consulta do ObjectGrid. A interface de consulta explica em detalhes como é possível executar as consultas utilizando a API do EntityManager. Consulte a API de Query para obter mais informações sobre o uso de consultas.

Uma QueryQueue de entidade é uma estrutura de dados semelhante a uma fila associada com uma consulta de entidade. Ela seleciona todas as entidades que correspondem à condição WHERE no filtro de consulta e coloca as entidades do resultado em uma fila. Os clientes podem, então, recuperar iterativamente as entidades dessa fila. Consulte “Filas de Consulta da Entidade” na página 196 para obter mais informações.

Tarefas relacionadas

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Referências relacionadas

"Agente de Instrumentação de Desempenho da Entidade" na página 465
É possível melhorar o desempenho de entidades de acesso a campo ativando o agente de instrumentação do WebSphere eXtreme Scale ao utilizar o Java Development Kit (JDK) Versão 1.5 ou posterior.

"Definindo um Esquema de Entidade" na página 169
Um ObjectGrid pode ter inúmeros esquemas de entidade lógicos. As entidades são definidas usando as classes Java anotadas, o XML ou uma combinação de classes XML e Java. Entidades definidas são registradas com um servidor eXtreme Scale e ligadas a BackingMaps, índices e outros plug-ins.

"Listeners de Entidade e Métodos de Retorno de Chamada"
Os aplicativos podem ser notificados quando o estado de uma entidade é alterado de estado para estado. Dois mecanismos de retorno de chamada existem para os eventos de alteração de estado: os métodos de retorno de chamada do ciclo de vida, que são definidos em uma classe de entidade e são chamados sempre que o estado da entidade for alterado, e os listeners de entidade que são mais genéricos porque o listener da entidade pode ser registrado em várias entidades.

"Exemplos do Listener de Entidade" na página 189
É possível gravar EntityListeners com base em seus requisitos. Veja a seguir vários scripts de exemplo.

"Interface EntityTransaction" na página 201
É possível utilizar a interface EntityTransaction para demarcar transações.

Listeners de Entidade e Métodos de Retorno de Chamada:

Os aplicativos podem ser notificados quando o estado de uma entidade é alterado de estado para estado. Dois mecanismos de retorno de chamada existem para os eventos de alteração de estado: os métodos de retorno de chamada do ciclo de vida, que são definidos em uma classe de entidade e são chamados sempre que o estado da entidade for alterado, e os listeners de entidade que são mais genéricos porque o listener da entidade pode ser registrado em várias entidades.

Ciclo de Vida de uma Instância de Entidade

Uma instância de entidade apresenta os seguintes estados:

- **Novo**: Uma instância da entidade recentemente criada que não existe no cache do eXtreme Scale.
- **Gerenciado**: A instância da entidade existe no cacho do eXtreme Scale e é recuperada ou persistida utilizando o entity manager. É preciso que uma entidade esteja associada a uma transação ativa para ficar no estado gerenciado.
- **Separado**: A instância da entidade existe no cache do eXtreme Scale, mas não é mais associada com uma transação ativa.
- **Removido**: A instância da entidade é removida ou está planejada para ser removida do cache do eXtreme Scale quando ocorre o flush ou o commit da transação.

- **Invalidado**: A instância da entidade está invalidada ou está planejada para ser invalidada no cache do eXtreme Scale quando a ocorre o flush ou o commit da transação.

Quando as entidades são alteradas de estado para estado, é possível chamar métodos life cycle e callback.

As seções a seguir descrevem com mais detalhes os significados dos estados Novo, Gerenciado, Separado, Removido e Invalidado à medida que os estados se aplicam a uma entidade.

Métodos do Retorno de Chamada do Ciclo de Vida da Entidade

Os métodos do retorno de chamada do ciclo de vida da entidade podem ser definidos na classe de entidade e são chamados quando o estado da entidade for alterado. Estes métodos são úteis para validar campos de entidade e atualizar o estado temporário que normalmente não é persistido com a entidade. Os métodos do retorno de chamada do ciclo de vida da entidade também podem ser definidos nas classes que não estão usando as entidades. Tais classes são classes de listener da entidade, que podem ser associadas a vários tipos de entidades. Os métodos do retorno de chamada do ciclo de vida podem ser definidos usando as anotações de metadados e um arquivo descritor XML de metadados da entidade.

- **Anotações**: Os métodos do retorno de chamada do ciclo de vida podem ser denotados usando as anotações PrePersist, PostPersist, PreRemove, PostRemove, PreUpdate, PostUpdate e PostLoad em uma classe de entidade.
- **Descritor XML de Entidade**: Os métodos do retorno de chamada do ciclo de vida podem ser descritos usando o XML quando as anotações não estiverem disponíveis.

Listeners de Entidade

Uma classe de listener de entidade é uma classe que não usa as entidades que definem um ou mais métodos do retorno de chamada do ciclo de vida da entidade. Os listeners de entidade são úteis para aplicativos de auditoria e criação de log com propósito geral. Listeners de entidade podem ser definidos utilizando anotações de metadados e um arquivo descritor XML de metadados da entidade:

- **Anotação**: A anotação EntityListeners pode ser utilizada para denotar uma ou mais classes do listener de entidade em uma classe de entidade. Se vários listeners de entidade estiverem definidos, a ordem na qual eles são chamados é determinada pela ordem na qual estão especificados na anotação EntityListeners.
- **Descritor XML da entidade**: O descritor XML pode ser utilizado como uma alternativa para especificar a ordem de chamada dos listeners de entidade ou para substituir a ordem que é especificada nas anotações de metadados.

Requisitos do Método de Retorno de Chamada

Qualquer subconjunto ou combinação de anotações pode ser especificado em uma classe de entidade ou uma classe de listener. Uma única classe não pode ter mais que um método de retorno de chamada do ciclo de vida para o mesmo evento do ciclo de vida. Entretanto, o mesmo método pode ser utilizado para vários eventos de retorno de chamada. A classe de listener de entidade deve ter um construtor no-arg público. Listeners de entidade não têm definição de estado. O ciclo de vida de um listener de entidade não é especificado. O eXtreme Scale não suporta herança de entidades, portanto, os métodos callback podem ser definidos apenas na classe de entidade, mas não na superclasse.

Assinatura de Método callback

Os métodos do retorno de chamada do ciclo de vida da entidade podem ser definidos em uma classe de listener de entidade, diretamente em uma classe de entidade, ou em ambos. Os métodos do retorno de chamada do ciclo de vida da entidade podem ser definidos usando as anotações de metadados e o arquivo descritor XML da entidade. As anotações utilizadas para métodos callback de entidade e na classe de listener da entidade são as mesmas. As assinaturas dos métodos callback são diferentes quando definidas em uma classe de entidade versus uma classe de listener de entidade. Os métodos callback definidos em uma classe de entidade ou superclasse mapeada possuem a seguinte assinatura:

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

Os métodos callback que são definidos em uma classe de listener de entidade possuem a seguinte assinatura:

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

O argumento Object é a instância da entidade para a qual o método de retorno de chamada é chamado. O argumento Object pode ser declarado como um objeto java.lang.Object ou o tipo de entidade real.

Os métodos callback podem ter nível de acesso público, privado, protegido ou de pacote, mas não devem ser estáticos ou finais.

As seguintes anotações são definidas para designar os métodos de retorno de chamada de evento do ciclo de vida dos tipos correspondentes:

- 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 a Documentação da API para obter mais detalhes. Cada anotação possui um atributo XML equivalente definido no arquivo descritor XML de metadados de entidade.

Semântica do Método do Retorno de Chamada do Ciclo de Vida

Cada método do retorno de chamada do ciclo de vida diferente possui uma finalidade diferente e é chamado nas fases diferentes do ciclo de vida da entidade:

PrePersist
: Chamado para uma entidade antes da entidade ter sido persistida para o armazém, o que inclui entidades que foram persistidas devido a uma operação em cascata. Este método é chamado no encadeamento da operação EntityManager.persist.

PostPersist
: Chamado para uma entidade após a entidade ter sido persistida para o armazém, o que inclui entidades que foram persistidas devido a uma operação em cascata. Este método é chamado no encadeamento da operação EntityManager.persist. Ele é chamado após o EntityManager.flush ou EntityManager.commit ser chamado.

PreRemove
Chamado para um entidade antes da entidade ter sido removida, o que inclui entidades que foram removidas devido a uma operação em cascata. Este método é chamado no encadeamento da operação EntityManager.remove.

PostRemove
Chamado para uma entidade após a entidade ter sido removida, o que inclui entidades que foram removidas devido a uma operação em cascata. Este método é chamado no encadeamento da operação EntityManager.remove. Ele é chamado após o EntityManager.flush ou EntityManager.commit ser chamado.

PreUpdate
Chamado para uma entidade antes da entidade ter sido atualizada no armazém. Este método é chamado no encadeamento da operação flush ou commit da transação.

PostUpdate
Chamado para uma entidade após a entidade ter sido atualizada no armazém. Este método é chamado no encadeamento da operação flush ou commit da transação.

PostLoad
Chamado para uma entidade após a entidade ter sido carregada do armazém, o que inclui quaisquer entidades que são carregadas através de uma associação. Este método é chamado no encadeamento da operação de carregamento, tal como EntityManager.find ou uma consulta.

Duplicata dos Métodos do Retorno de Chamada do Ciclo de Vida

Se vários métodos do retorno de chamada forem definidos para um evento de ciclo de vida da entidade, a ordem de chamada desses métodos será a seguinte:

1. **Métodos de retorno de chamada do ciclo de vida definidos nos listeners de entidade**: Os métodos do retorno de chamada do ciclo de vida, que são definidos nas classes de listener de entidade para uma classe de entidade, são chamados na mesma ordem que a especificação das classes de listener de entidade na anotação EntityListeners ou no descritor XML.
2. **Superclasse do listener**: Os métodos callback definidos na superclasse do listener de entidade são chamados antes dos filhos.
3. **Métodos do ciclo de vida da entidade**: O WebSphere eXtreme Scale não suporta herança de entidade, portanto, os métodos do ciclo de vida da entidade podem ser definidos apenas na classe de entidade.

Exceções

Os métodos de retorno de chamada do ciclo de vida podem resultar em exceções de tempo de execução. Se um método de retorno de chamada do ciclo de vida causar uma exceção de tempo de execução em uma transação, a transação será recuperada. Nenhum outro método de retorno de chamada do ciclo de vida será chamado depois que ocorrer uma exceção no tempo de execução.

Conceitos relacionados

"Ajustando o Desempenho da Interface EntityManager" na página 463
A interface EntityManager separa aplicativos do estado de suspensão no armazenamento de dados da grade do servidor.

"Objetos de Armazenamento em Cache e seus Relacionamentos (API EntityManager)" na página 166
A maioria dos produtos de cache utiliza APIs baseadas em mapa para armazenar dados como pares de chave-valor. A API ObjectMap e o cache dinâmico no WebSphere Application Server, entre outros, usam essa abordagem. Entretanto, APIs baseadas em mapas têm limitações. A API EntityManager simplifica a interação com a grade de dados ao fornecer uma maneira fácil de declarar e interagir com um gráfico complexo de objetos relacionados.

"Entity Manager em um Ambiente Distribuído" na página 177
É possível usar a API EntityManager com um ObjectGrid local ou em um ambiente distribuído do eXtreme Scale . A principal diferença é como você se conecta a esse ambiente remoto. Após você estabelecer uma conexão, não existe diferença entre o uso de um objeto Session ou uma API do EntityManager.

"Interagindo com EntityManager" na página 182
Geralmente os aplicativos primeiro obtêm uma referência do ObjectGrid e, depois, uma Sessão dessa referência para cada encadeamento. As sessões não podem ser compartilhadas entre encadeamentos. Um método extra em Session, o método getEntityManager, está disponível. Este método retorna uma referência para um gerenciador de entidades para uso para este encadeamento. A interface de EntityManager pode substituir as interfaces de Session e ObjectMap para todos os aplicativos. É possível utilizar essas APIs de EntityManager se o cliente tiver acesso às classes de entidade definidas.

"Suporte ao Plano de Carregamento do EntityManager" na página 191
Um FetchPlan é a estratégia que o gerenciador de entidade usa para recuperar objetos associados se o aplicativo precisar acessar relacionamentos.

"Filas de Consulta da Entidade" na página 196
Filas de consulte permitem que aplicativos criem uma fila qualificada por uma consulta no lado do servidor ou eXtreme Scale local sobre uma entidade. As entidades do resultado da consulta são armazenadas nesta fila. Atualmente, a fila de consulta é suportada apenas em um mapa que está utilizando a estratégia de bloqueio pessimista.

Tarefas relacionadas

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Exemplos do Listener de Entidade:

É possível gravar EntityListeners com base em seus requisitos. Veja a seguir vários scripts de exemplo.

Exemplo de EntityListeners Utilizando Anotações

O exemplo a seguir mostra as chamadas de método de retorno de chamada do ciclo de vida e a ordem das chamadas. Assuma que existe uma classe de entidade Employee e dois listeners de entidade: EmployeeListener e 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) {
        ....
    }
}
```

Se um evento PrePersist ocorre em uma instância Employee, os seguintes métodos são chamados em ordem:

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

Exemplo de Listeners de Entidade Utilizando XML

O exemplo a seguir mostra como configurar um listener de entidade em uma entidade utilizando o arquivo XML descritor de entidade:

```
<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>
```

A entidade Employee é configurada com uma classe de listener de entidade com.ibm.websphere.objectgrid.sample.EmployeeListener, que possui dois métodos de retorno de chamada de ciclo de vida definidos. O método onListenerPrePersist é para o evento PrePersist e o método onListenerPostPersist é para o evento

PostPersist. Além disso, o método checkEmployeeID na classe Employee é configurado para atender o evento PrePersist.

Conceitos relacionados

"Ajustando o Desempenho da Interface EntityManager" na página 463
A interface EntityManager separa aplicativos do estado de suspensão no armazenamento de dados da grade do servidor.

"Objetos de Armazenamento em Cache e seus Relacionamentos (API EntityManager)" na página 166
A maioria dos produtos de cache utiliza APIs baseadas em mapa para armazenar dados como pares de chave-valor. A API ObjectMap e o cache dinâmico no WebSphere Application Server, entre outros, usam essa abordagem. Entretanto, APIs baseadas em mapas têm limitações. A API EntityManager simplifica a interação com a grade de dados ao fornecer uma maneira fácil de declarar e interagir com um gráfico complexo de objetos relacionados.

"Entity Manager em um Ambiente Distribuído" na página 177
É possível usar a API EntityManager com um ObjectGrid local ou em um ambiente distribuído do eXtreme Scale . A principal diferença é como você se conecta a esse ambiente remoto. Após você estabelecer uma conexão, não existe diferença entre o uso de um objeto Session ou uma API do EntityManager.

"Interagindo com EntityManager" na página 182
Geralmente os aplicativos primeiro obtêm uma referência do ObjectGrid e, depois, uma Sessão dessa referência para cada encadeamento. As sessões não podem ser compartilhadas entre encadeamentos. Um método extra em Session, o método getEntityManager, está disponível. Este método retorna uma referência para um gerenciador de entidades para uso para este encadeamento. A interface de EntityManager pode substituir as interfaces de Session e ObjectMap para todos os aplicativos. É possível utilizar essas APIs de EntityManager se o cliente tiver acesso às classes de entidade definidas.

"Suporte ao Plano de Carregamento do EntityManager"
Um FetchPlan é a estratégia que o gerenciador de entidade usa para recuperar objetos associados se o aplicativo precisar acessar relacionamentos.

"Filas de Consulta da Entidade" na página 196
Filas de consulte permitem que aplicativos criem uma fila qualificada por uma consulta no lado do servidor ou eXtreme Scale local sobre uma entidade. As entidades do resultado da consulta são armazenadas nesta fila. Atualmente, a fila de consulta é suportada apenas em um mapa que está utilizando a estratégia de bloqueio pessimista.

Tarefas relacionadas

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Suporte ao Plano de Carregamento do EntityManager

Um FetchPlan é a estratégia que o gerenciador de entidade usa para recuperar objetos associados se o aplicativo precisar acessar relacionamentos.

Exemplo

Suponha, por exemplo, que seu aplicativo tenha duas entidades: Department e Employee. O relacionamento entre a entidade Department e a entidade Employee é um relacionamento bidirecional um-para-muitos: Um departamento tem vários

funcionários, e um funcionário pertence a um único departamento. Como na maioria das vezes, quando é feita uma busca pela entidade Department, seus Employees provavelmente são buscados, e o tipo de busca desse relacionamento um-para-muitos é configurado como EAGER.

Aqui está um fragmento da classe 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;

}
```

Em um ambiente distribuído, quando um aplicativo chama `em.find(Department.class, "dept1")` para localizar uma entidade Department com a chave "dept1", essa operação find obtém a entidade Department e todas as suas relações eager-fetched. No caso do fragmento anterior, elas são todos os funcionários do departamento "dept1".

Antes do WebSphere eXtreme Scale 6.1.0.5, a recuperação de uma entidade Department e N entidades Employee incorria em N+1 trips de cliente-servidor porque o cliente recuperava uma entidade para um trip de cliente-servidor. É possível melhorar o desempenho se você recuperar essas N+1 entidades em um trip.

Plano de Carregamento

Um plano de carregamento pode ser utilizado para customizar a forma como você executa um carregamento ansioso de relacionamentos customizando a profundidade máxima dos relacionamentos. A profundidade da busca substitui relações eager maiores do que a profundidade especificada para relações lazy. Por padrão, a profundidade da busca é a profundidade máxima da busca. Isso significa que relacionamentos eager de todos os níveis navegáveis como eager a partir da entidade raiz serão buscados. Um relacionamento EAGER é navegável como eager a partir de uma entidade raiz se, e apenas se, todas as relações começando da entidade raiz para ela forem configuradas como eager-fetched.

No exemplo anterior, a entidade Employee é navegável como eager a partir da entidade Department porque o relacionamento Department-Employee é configurado como eager-fetched.

Se a entidade Employee tiver outro relacionamento eager com uma entidade Address, por exemplo, a entidade Address também será navegável como eager a partir da entidade Department. Entretanto, se os relacionamentos Department-Employee foram configurados como lazy-fetch, a entidade Address não será navegável como eager a partir da entidade Department, pois o relacionamento Department-Employee quebra a cadeia de eager fetch.

Um objeto Plano de Carregamento pode ser recuperado da instância EntityManager. O aplicativo pode utilizar o método setMaxFetchDepth para alterar a profundidade máxima da busca.

Um plano de carregamento é associado a uma instância EntityManager. O plano de carregamento aplica-se a qualquer operação de busca, mais especificamente da seguinte forma.

- Operações EntityManager `find(Class class, Object key)` e `findForUpdate(Class class, Object key)`
- Operações Query
- Operações QueryQueue

O objeto Plano de Carregamento é mutável. Após ser alterado, o valor será aplicado às operações de busca executadas posteriormente.

Um plano de carregamento é importante para uma implementação distribuída porque decide se as entidades de relacionamento eager-fetched são recuperadas com a entidade raiz em um trip de cliente/servidor ou em mais de uma.

Continuando com o exemplo anterior, considere que o plano de carregamento tenha profundidade máxima configurada como infinito. Nesse caso, quando o aplicativo chama `em.find(Department.class, "dept1")` para localizar Department, essa operação find obtém uma entidade Department e N entidades Employee em um trip de cliente/servidor. Entretanto, para um plano de carregamento com profundidade de busca máxima configurada como zero, apenas o objeto Department será recuperado do servidor, enquanto as entidades Employee são recuperadas do servidor apenas quando a coleta de Employees do objeto Department é acessada.

Planos de Carregamento Diferentes

Você tem diferentes planos de carregamentos baseados em seus requisitos, explicados nas seguintes seções.

Impacto em uma grade distribuída

- *Plano de carregamento de profundidade infinita:* Um Plano de Carregamento de profundidade infinita tem sua profundidade de busca máxima configurada como `FetchPlan.DEPTH_INFINITE`.

  Em um ambiente de cliente/servidor, se um plano de carregamento de profundidade infinita for utilizado, todas as relações navegáveis como eager a partir da entidade raiz serão recuperadas em um trip de cliente/servidor.

  Exemplo: Se o aplicativo estiver interessado em todas as entidades Address de todos os employees de um determinado Department, ele utilizará um plano de carregamento de profundidade infinita para recuperar todas as entidades Address associadas. O código a seguir incorre apenas em um trip de cliente/servidor.

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

  tran.begin();
  Department dept = (Department) em.find(Department.class, "dept1");
  // do something with Address object.
  for (Employee e: dept.employees) {
      for (Address addr: e.addresses) {
      // do something with addresses.
      }
  }
  tran.commit();
  ```

- *Plano de carregamento com profundidade zero:* Um plano de carregamento com profundidade zero tem sua profundidade máxima de busca configurada como 0.

Em um ambiente de cliente/servidor, se um plano de carregamento com profundidade zero for utilizado, apenas a entidade raiz será recuperada no primeiro trip de cliente/servidor. Todos os relacionamentos eager são tratados como se fossem lazy.

Exemplo: Neste exemplo, o aplicativo só está interessado no atributo da entidade Department. Ele não precisa acessar seus employees, portanto, o aplicativo configura a profundidade do plano de carregamento como 0.

```
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");
// do something with dept object.
tran.commit();
```

- *Plano de carregamento com espessura k* :

  Um plano de carregamento de espessura *k*- tem sua espessura máxima de carregamento configurada para *k*.

  Em um ambiente eXtreme Scale de cliente/servidor, se um plano de carregamento de profundidade *k*- for utilizado, todos os relacionamentos navegáveis como eager a partir da entidade raiz dentro de *k* etapas serão recuperados no primeiro trip de cliente/servidor.

  O plano de carregamento de profundidade infinita (*k* = infinito) e o plano de carregamento de profundidade zero (*k* = 0) são apenas dois exemplos do plano de carregamento de profundidade *k*-.

  Para continuar expandindo o exemplo anterior, suponha que exista outro relacionamento eager da entidade Employee com a entidade Address. Se o plano de carregamento tiver profundidade de busca máxima configurada como 1, a operação `em.find(Department.class, "dept1")` irá recuperar a entidade Department e todas as entidades Employee em um trip de cliente/servidor. Entretanto, as entidades Address não serão recuperadas porque não são navegáveis como eager para a entidade Department dentro de 1 etapa, mas sim de 2 etapas.

  Se você utilizar um plano de carregamento com profundidade configurada como 2, a operação `em.find(Department.class, "dept1")` irá recuperar a entidade Department, todas as suas entidades Employee e todas as entidades Address associadas às entidades Employee em um trip de cliente/servidor.

  Dica: O plano de carregamento padrão tem profundidade de busca máxima configurada como infinito, portanto, o comportamento padrão de uma operação fetch pode mudar. Todos os relacionamentos navegáveis como eager a partir da entidade raiz são recuperados. Em vez de várias viagens, agora a operação fetch incorre apenas um trip de cliente/servidor com o plano de carregamento padrão. Para manter as configurações para o produto da versão anterior, configure a profundidade da busca como 0.

- *Plano de carregamento utilizado na consulta:*

  Se você executar uma consulta de entidade, também será possível utilizar um plano de carregamento para customizar a recuperação de relacionamento.

  Por exemplo, o resultado da consulta `SELECT d FROM Department d WHERE "d.deptName='Department'"` tem um relacionamento com a entidade Department. Observe que a profundidade do plano de carregamento começa com a associação do resultado da consulta: nesse caso, a entidade Department, não o resultado da consulta em si. Ou seja, a entidade Department está no nível de profundidade de busca 0. Entretanto, um plano de carregamento com

profundidade de busca máxima de 1 irá recuperar a entidade Department e suas entidades Employee em um trip de cliente/servidor.

Exemplo: Neste exemplo, a profundidade do plano de carregamento está configurada como 1, portanto, a entidade Department e suas entidades Employee são recuperadas em um trip de cliente/servidor, mas as entidades Address não serão recuperadas no mesmo trip.

Importante: Se um relacionamento for solicitado, utilizando configuração ou anotação OrderBy, ele será considerado um relacionamento eager, mesmo que esteja configurado como lazy-fetch.

Considerações de Desempenho em um Ambiente Distribuído

Por padrão, todos os relacionamentos navegáveis como eager a partir da entidade raiz serão recuperados em um trip de cliente/servidor. Isso pode melhorar o desempenho se todos os relacionamentos forem utilizados. No entanto, em certos cenários de uso, nem todos os relacionamentos navegáveis como eager a partir da entidade raiz são utilizados, portanto eles incorrem em gasto adicional de tempo de execução e em gasto adicional de largura de banda ao recuperar essas entidades não utilizadas.

Para esses casos, o aplicativo pode configurar a profundidade de busca máxima para um número pequeno para diminuir a profundidade de entidades a serem recuperadas, tornando lazy todas as relações eager após essa profundidade específica. Essa configuração pode melhorar o desempenho.

Continuando com o exemplo Department-Employee-Address anterior, por padrão, todas as entidades Address associadas a employees de Department "dept1" serão recuperadas quando `em.find(Department.class, "dept1")` for chamado. Se o aplicativo não utilizar entidades Address, ele poderá configurar a profundidade máxima da busca como 1, portanto as entidades Address não serão recuperadas com a entidade Department.

Tarefas relacionadas

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Referências relacionadas

"Agente de Instrumentação de Desempenho da Entidade" na página 465
É possível melhorar o desempenho de entidades de acesso a campo ativando o agente de instrumentação do WebSphere eXtreme Scale ao utilizar o Java Development Kit (JDK) Versão 1.5 ou posterior.

"Definindo um Esquema de Entidade" na página 169
Um ObjectGrid pode ter inúmeros esquemas de entidade lógicos. As entidades são definidas usando as classes Java anotadas, o XML ou uma combinação de classes XML e Java. Entidades definidas são registradas com um servidor eXtreme Scale e ligadas a BackingMaps, índices e outros plug-ins.

"Listeners de Entidade e Métodos de Retorno de Chamada" na página 185
Os aplicativos podem ser notificados quando o estado de uma entidade é alterado de estado para estado. Dois mecanismos de retorno de chamada existem para os eventos de alteração de estado: os métodos de retorno de chamada do ciclo de vida, que são definidos em uma classe de entidade e são chamados sempre que o estado da entidade for alterado, e os listeners de entidade que são mais genéricos porque o listener da entidade pode ser registrado em várias entidades.

"Exemplos do Listener de Entidade" na página 189
É possível gravar EntityListeners com base em seus requisitos. Veja a seguir vários scripts de exemplo.

"Interface EntityTransaction" na página 201
É possível utilizar a interface EntityTransaction para demarcar transações.

Filas de Consulta da Entidade

Filas de consulte permitem que aplicativos criem uma fila qualificada por uma consulta no lado do servidor ou eXtreme Scale local sobre uma entidade. As entidades do resultado da consulta são armazenadas nesta fila. Atualmente, a fila de consulta é suportada apenas em um mapa que está utilizando a estratégia de bloqueio pessimista.

Uma fila de consulta é compartilhada por várias transações e clientes. Após a fila de consulta ficar vazia, a consulta da entidade associada a esta fila é executada novamente e novos resultados são incluídos na fila. Uma fila de consulta é identificada exclusivamente pela cadeia e os parâmetro de consulta da entidade. Há apenas uma instância para cada fila de consulta exclusiva em uma instância do ObjectGrid. Consulte a documentação da API do EntityManager para obter mais informações.

Exemplo de Fila de Consulta

O exemplo a seguir mostra como a fila de consulta pode ser utilizada.

```
/**
 * Get a unassigned question type task
 */
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();
}
```

O exemplo anterior primeiro cria um QueryQueue com uma cadeia de consulta de entidade, "SELECT t FROM Task t WHERE t.type=?1 AND t.status=?2". Depois ele configura os parâmetros para o objeto QueryQueue. Esta fila de consulta representa todas as tarefas "não-designadas" do tipo "questão". O objeto QueryQueue é muito semelhante a um objeto Query da entidade.

Após o QueryQueue ser criado, uma transação de entidade é iniciada e o método getNextEntity é chamado, que recupera a próxima entidade disponível sem um valor de tempo limite de 10 segundos. Após a entidade ser recuperada, ela é processada no método assignTask. O assignTask modifica a instância da entidade Task e altera o status para "designado" que efetivamente a remove da fila já que não corresponde mais ao filtro do QueryQueue. Uma vez designada, ocorre o commit da transação.

A partir deste exemplo, é possível visualizar uma fila de consulta que é semelhante a uma consulta de entidade. Entretanto, elas diferem nas seguintes maneiras:

1. As entidades na fila de consulta podem ser recuperadas de uma maneira iterativa. O aplicativo de usuário decide o número de entidades a ser recuperado. Por exemplo, se QueryQueue.getNextEntity(timeout) é utilizado, apenas uma entidade é recuperada e se QueryQueue.getNextEntities(5, timeout) é utilizado, 5 entidades são recuperadas. Em um ambiente distribuído, o número de entidades decide diretamente o número de bytes a ser transferido do servidor para o cliente.
2. Quando uma entidade é recuperada da fila de consulta, um bloqueio U é colocado na entidade, assim nenhuma outra transação pode acessá-la.

Recuperar Entidades em um Loop

É possível recuperar entidades em um loop. A seguir está um exemplo que ilustra como obter todas as tarefas não-designadas do tipo question concluídas.

```
/**
 * Get all unassigned question type tasks
 */
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);
}
```

Se houver 10 tarefas não-designadas do tipo question no mapa de entidade, é possível esperar que você terá 10 entidades impressas no console. Entretanto, se este exemplo for executado, visualizará que o programa nunca sai, o que pode ser contrário ao que você assumiu.

Quando uma fila de consulta é criada e o getNextEntity é chamado, a consulta de entidade associada com a fila é executada e os 10 resultados são colocadas na fila. Quando o getNextEntity é chamado, uma entidade é retirada da fila. Após 10 chamadas do getNextEntity serem executadas, a fila fica vazia. A consulta da entidade será novamente executada automaticamente. Como estas 10 entidades ainda existem e correspondem aos critérios de filtro da fila de consulta, elas são colocadas na fila novamente.

Se a linha a seguir for incluída após a instrução println(), você verá apenas 10 entidades impressas.

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

Para obter informações sobre o uso de SessionHandle com QueryQueue em uma implementação de posicionamento por contêiner, leia sobre Integração de SessionHandle.

Filas de Consulta Implementadas em todas as Partições

Em um eXtreme Scale distribuído, uma fila de consulta pode ser criada para uma partição ou todas as partições. Se uma fila de consulta é criada para todas as partições, haverá uma instância de fila de consulta em cada partição.

Quando um cliente tenta obter a próxima entidade utilizando o método QueryQueue.getNextEntity ou QueryQueue.getNextEntities, o cliente envia um pedido para uma das partições. Um cliente envia pedidos de peek e pin para o servidor:

- Com um pedido de peek, o cliente envia um pedido para uma partição e o servidor retorna imediatamente. Se houver uma entidade na fila, o servidor envia uma resposta com a entidade; se não houver, o servidor envia uma resposta sem nenhuma entidade. Em qualquer um dos casos, o servidor retornará imediatamente.
- Com um pedido de pin, o cliente envia um pedido para uma partição e o servidor aguarda até que uma entidade esteja disponível. Se houver uma entidade na fila, o servidor envia uma resposta com a entidade imediatamente; se não houver, o servidor aguarda na fila até que uma entidade esteja disponível ou o pedido atinja o tempo limite.

A seguir, está um exemplo de como uma entidade é recuperada para uma fila de consulta que é implementada em todas as partições (n):

1. Quando um método QueryQueue.getNextEntity ou QueryQueue.getNextEntities é chamado, o cliente seleciona um número de partição aleatório de 0 a n-1.

2. O cliente envia o pedido de peek para a partição aleatória.
   - Se uma entidade estiver disponível, o método QueryQueue.getNextEntity ou QueryQueue.getNextEntities sai retornando a entidade.
   - Se uma entidade não estiver disponível e não for a última partição não-visitada, o cliente envia um pedido de peek para a próxima partição.
   - Se uma entidade não estiver disponível e for a última partição não-visitada, o cliente envia um pedido de pin.
   - Se o pedido de pin para a última partição atingir o tempo limite e ainda não houver dados disponíveis, o cliente fará um último esforço enviado o pedido de peek para todas as partições serialmente um ciclo a mais. Portanto, se alguma entidade estiver disponível nas partições anteriores, o cliente estará a obtê-la.

Suporte a Entidade e Não-entidade de Subconjunto

A seguir está o método para criar um objeto QueryQueue no entity manager:

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

O resultado na fila de consulta deve ser projetado para o objeto definido pelo segundo parâmetro para o método, Class entityClass.

Se este parâmetro for especificado, a classe deve ter o mesmo nome da entidade conforme especificado na cadeia de consultas. Isto é útil se você desejar projetar uma entidade em uma entidade de subconjunto. Se um valor nulo é utilizado com a classe de entidade, então, o resultado não será projetado. O valor armazenado no mapa estará em um formato de tupla da entidade.

Colisão de Chaves do Lado do Cliente

No ambiente eXtreme Scale distribuído, a fila de consulta é suportada apenas para mapas do eXtreme Scale com o modo de bloqueio pessimista. Portanto, não há um cache local no lado do cliente. Entretanto, um cliente poderia ter dados (chave e valor) no mapa transacional. Isto potencialmente poderia levar a uma colisão de chaves quando uma entidade recuperada do servidor compartilha a mesma chave que uma entrada já no mapa da transação.

Quando ocorre uma colisão de chaves, o tempo de execução do cliente do eXtreme Scale utiliza a seguinte regra para emitir uma exceção ou substituir os dados silenciosamente.

1. Se a chave colidida for a chave da entidade especificada na consulta de entidade associada com a fila de consulta, então, uma exceção é lançada. Neste caso, ocorre o rollback da transação e o bloqueio U nesta entidade será liberado no lado do servidor.
2. Caso contrário, se a chave colidida for a chave de uma associação de entidade, os dados no mapa transacional serão substituídos sem aviso.

A colisão de chaves acontece apenas quando há dados no mapa transacional. Em outras palavras, isto ocorre apenas quando uma chamada getNextEntity ou getNextEntities é chamada em uma transação que já foi suja (um novo dado foi inserido ou um dado foi atualizado). Se um aplicativo não deseja que aconteça uma colisão de chaves, ele deve sempre chamar getNextEntity ou getNextEntities em uma transação que não foi suja.

Falhas do Cliente

Após um cliente enviar um pedido getNextEntity ou getNextEntities para o servidor, o cliente poderia falhar da seguinte maneira:

1. O cliente envia um pedido para o servidor e, então, fica inativo.
2. O cliente obtém uma ou mais entidades do servidor e, então, fica inativo.

No primeiro caso, o servidor descobre que o cliente está ficando inativo quando ele tenta enviar de volta a resposta para o cliente. No segundo caso, quando o cliente obtém uma ou mais entidades do servidor, um bloqueio X é colocado nestas entidades. Se o cliente fica inativo, a transação eventualmente atingirá o tempo limite e o bloqueio X será liberado.

Consulta com a cláusula ORDER BY

Geralmente, as filas de consulta não honram a cláusula ORDER BY. Se você chamar getNextEntity ou getNextEntities a partir da fila de consulta, não há garantia de que as entidades serão retornadas de acordo com a ordem. O motivo é que as entidades não podem ser ordenadas entre partições. No caso em que a fila de consulta é implementada em todas as partições, quando uma chamada getNextEntity ou getNextEntities é executada, uma partição aleatória é selecionada para processar o pedido. Portanto, a ordem não é garantida.

ORDER BY será honrado se uma fila de consulta for implementada em uma partição única.

Para obter mais informações, consulte “API de Consulta EntityManager” na página 213.

Uma Chamada Por Transação

Cada chamada QueryQueue.getNextEntity ou QueryQueue.getNextEntities recupera entidades correspondentes de uma partição aleatória. Os aplicativos devem chamar exatamente uma QueryQueue.getNextEntity ou QueryQueue.getNextEntities em uma transação. Caso contrário, o eXtreme Scale pode encerrar o toque a entidades de várias partições, fazendo com que uma exceção seja lançada na hora da confirmação.

Tarefas relacionadas

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Referências relacionadas

"Agente de Instrumentação de Desempenho da Entidade" na página 465
É possível melhorar o desempenho de entidades de acesso a campo ativando o agente de instrumentação do WebSphere eXtreme Scale ao utilizar o Java Development Kit (JDK) Versão 1.5 ou posterior.

"Definindo um Esquema de Entidade" na página 169
Um ObjectGrid pode ter inúmeros esquemas de entidade lógicos. As entidades são definidas usando as classes Java anotadas, o XML ou uma combinação de classes XML e Java. Entidades definidas são registradas com um servidor eXtreme Scale e ligadas a BackingMaps, índices e outros plug-ins.

"Listeners de Entidade e Métodos de Retorno de Chamada" na página 185
Os aplicativos podem ser notificados quando o estado de uma entidade é alterado de estado para estado. Dois mecanismos de retorno de chamada existem para os eventos de alteração de estado: os métodos de retorno de chamada do ciclo de vida, que são definidos em uma classe de entidade e são chamados sempre que o estado da entidade for alterado, e os listeners de entidade que são mais genéricos porque o listener da entidade pode ser registrado em várias entidades.

"Exemplos do Listener de Entidade" na página 189
É possível gravar EntityListeners com base em seus requisitos. Veja a seguir vários scripts de exemplo.

"Interface EntityTransaction"
É possível utilizar a interface EntityTransaction para demarcar transações.

Interface EntityTransaction

É possível utilizar a interface EntityTransaction para demarcar transações.

Propósito

Para demarcar uma transação, é possível utilizar a interface EntityTransaction, que é associada com uma instância do gerenciador de entidades. Utilize o método EntityManager.getTransaction para recuperar a instância do EntityTransaction para o gerenciador de entidades. Cada instância do EntityManager e do EntityTransaction é associada com o objeto Session. É possível demarcar transações com EntityTransaction ou Session. Os métodos na interface EntityTransaction não possuem nenhuma exceção verificada. Apenas as exceções de tempo de execução do tipo PersistenceException ou seu resultado de subclasses.

Para obter mais informações sobre a interface EntityTransaction, consulte Documentação da APIa interface EntityTransaction na Documentação de API .

Conceitos relacionados

"Ajustando o Desempenho da Interface EntityManager" na página 463
A interface EntityManager separa aplicativos do estado de suspensão no armazenamento de dados da grade do servidor.

"Objetos de Armazenamento em Cache e seus Relacionamentos (API EntityManager)" na página 166
A maioria dos produtos de cache utiliza APIs baseadas em mapa para armazenar dados como pares de chave-valor. A API ObjectMap e o cache dinâmico no WebSphere Application Server, entre outros, usam essa abordagem. Entretanto, APIs baseadas em mapas têm limitações. A API EntityManager simplifica a interação com a grade de dados ao fornecer uma maneira fácil de declarar e interagir com um gráfico complexo de objetos relacionados.

"Entity Manager em um Ambiente Distribuído" na página 177
É possível usar a API EntityManager com um ObjectGrid local ou em um ambiente distribuído do eXtreme Scale . A principal diferença é como você se conecta a esse ambiente remoto. Após você estabelecer uma conexão, não existe diferença entre o uso de um objeto Session ou uma API do EntityManager.

"Interagindo com EntityManager" na página 182
Geralmente os aplicativos primeiro obtêm uma referência do ObjectGrid e, depois, uma Sessão dessa referência para cada encadeamento. As sessões não podem ser compartilhadas entre encadeamentos. Um método extra em Session, o método getEntityManager, está disponível. Este método retorna uma referência para um gerenciador de entidades para uso para este encadeamento. A interface de EntityManager pode substituir as interfaces de Session e ObjectMap para todos os aplicativos. É possível utilizar essas APIs de EntityManager se o cliente tiver acesso às classes de entidade definidas.

"Suporte ao Plano de Carregamento do EntityManager" na página 191
Um FetchPlan é a estratégia que o gerenciador de entidade usa para recuperar objetos associados se o aplicativo precisar acessar relacionamentos.

"Filas de Consulta da Entidade" na página 196
Filas de consulte permitem que aplicativos criem uma fila qualificada por uma consulta no lado do servidor ou eXtreme Scale local sobre uma entidade. As entidades do resultado da consulta são armazenadas nesta fila. Atualmente, a fila de consulta é suportada apenas em um mapa que está utilizando a estratégia de bloqueio pessimista.

Tarefas relacionadas

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Recuperando Entidades e Objetos (API de Consulta)

O WebSphere eXtreme Scale fornece um mecanismo de consulta flexível para recuperar entidades usando a API do EntityManager e objetos Java usando a API do ObjectQuery.

Recursos de Consulta do WebSphere eXtreme Scale

Com o mecanismo de consulta do eXtreme Scale, é possível executar consultas do tipo SELECT sobre uma entidade ou esquema baseado em objeto usando a linguagem de consulta do eXtreme Scale.

Esta linguagem de consulta fornece os seguintes recursos:

- Resultados únicos e com diversos valores
- Funções agregadas
- Classificação e agrupamento
- Junções
- Expressões condicionais com subconsultas
- Parâmetros nomeados e posicionais
- Uso de índice do eXtreme Scale
- Sintaxe da expressão de caminho para navegação de objetos
- Paginação

Interface de Consulta

Utilize a interface de consulta para controlar a execução da consulta da entidade de controle.

Utilize o método EntityManager.createQuery(String) para criar uma Query. É possível usar cada instância de consulta múltiplas vezes com a instância de EntityManager na qual ela foi recuperada.

Cada resultado da consulta produz uma entidade na qual a chave da entidade é o ID da linha (do tipo longo) e o valor da entidade contém os resultados do campo da cláusula SELECT. É possível usar cada resultado da consulta em consultas subsequentes.

Os seguintes métodos estão disponíveis na interface com.ibm.websphere.objectgrid.em.Query.

public ObjectMap getResultMap()

O método getResultMap executa uma consulta SELECT e retorna os resultados em um objeto ObjectMap com os resultados na ordem especificada pela consulta. O ObjectMap resultante é válido apenas para a transação atual.

A chave do mapa é o número de resultado, do tipo long, iniciando em 1. O valor do mapa é do tipo com.ibm.websphere.projector.Tuple, em que cada atributo e associação é nomeado com base em sua posição ordinal dentro da cláusula select da consulta. Utilize o método para recuperar o EntityMetadata para o objeto Tupla armazenado dentro do mapa.

O método getResultMap é o mais rápido para recuperar dados do resultado da consulta nas qual múltiplos resultados podem existir. É possível recuperar o nome da entidade resultante usando os métodos ObjectMap.getEntityMetadata() e EntityMetadata.getName().

Exemplo: A seguinte consulta retorna duas linhas.

```
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; // starts with index 1
 Tuple tResult = (Tuple) resultMap.get(new Long(rowID));
 while(tResult != null) {
     // The first attribute is name and has an attribute name of 1
     // But has an ordinal position of 0.
     String name = (String)tResult.getAttribute(0);
     Integer id = (String)tResult.getAttribute(1);

     // Dept is an association with a name of 3, but
```

```
    // an ordinal position of 0 since it's the first association.
    // The association is always a OneToOne relationship,
    // so there is only one key.
    Tuple deptKey = tResult.getAssociation(0,0);
    ...
    ++rowID;
    tResult = (Tuple) resultMap.get(new Long(rowID));

}
```

public Iterator getResultIterator

O método getResultIterator executa uma consulta SELECT e retorna os resultados da consulta usando um Agente iterativo no qual cada resultado é um Objeto para uma consulta com valor único, ou uma matriz de Objetos para uma consulta com múltiplos valores. Os valores no resultado são armazenados na ordem da consulta. O Iterator de resultado é válido apenas para a transação atual.

Este método é preferencial para recuperar resultados da consulta dentro do contexto de EntityManager. É possível utilizar o método setResultEntityName(String) opcional para nomear a entidade resultante, para que ela possa ser utilizada em consultas adicionais.

Exemplo: A seguinte consulta retorna duas linhas.

```
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)

O método getResultIterator(Class resultType) executa uma consulta SELECT e retorna os resultados da consulta usando um Agente Iterativo da entidade. O tipo de entidade é determinado pelo parâmetro resultType. O Iterator de resultado é válido apenas para a transação atual.

Utilize esse método quando você quiser utilizar as APIs EntityManager para acessar as entidades resultantes.

Exemplo: A consulta a seguir retorna todos os funcionários e o departamento ao qual pertencem para uma divisão, com classificação por salário. Para imprimir os cinco funcionários com os salários mais altos e, então, selecionar o trabalho com funcionários de apenas um departamento no mesmo conjunto de trabalhos, utilize o seguinte 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);
 // Remove the employee from the resultset.
 em.remove(curEmp);
}

// Flush the changes to the result map.
```

```
em.flush();

// Run a query against the local working set without the employees we
// removed
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

O método getSingleResult executa uma consulta SELECT que retorna um único resultado.

Se a cláusula SELECT tiver mais de um campo definido, então o resultado é uma matriz de objetos, na qual cada elemento na matriz se baseia em sua posição original dentro da cláusula SELECT da 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)

O método setResultEntityName(String entityName) especifica o nome da entidade do resultado da consulta.

Toda vez que os métodos getResultIterator ou getResultMap são chamados, uma entidade com um ObjectMap é dinamicamente criada para manter os resultados da consulta. Se a entidade não for especificada, ou estiver nula, a entidade e o nome do ObjectMap serão automaticamente gerados.

Como todos os resultados da consulta estão disponíveis para a duração de uma transação, um nome de consulta não pode ser reutilizado em uma única transação.

public Query setPartition(int partitionId)

Configure a partição para onde é roteada a consulta.

Este método é necessário se os mapas na consulta estão particionados e se o gerenciador de entidades não tem afinidade com uma partição da entidade-raiz do esquema único.

Use a Interface PartitionManager para determinar o número de partições para o mapa de apoio de uma determinada entidade.

A tabela a seguir fornece descrições dos outros métodos que estão disponíveis por meio da interface da consulta.

Tabela 2. Outros Métodos

Método	Resultado
public Query setMaxResults(int maxResult)	Configure o número máximo de resultados para recuperar.
public Query setFirstResult(int startPosition)	Configure a posição do primeiro resultado para recuperar.
public Query setParameter(String name, Object value)	Ligue um argumento a um parâmetro nomeado.
public Query setParameter(int position, Object value)	Ligue um argumento a um parâmetro posicional.
public Query setFlushMode(FlushModeType flushMode)	Configure o tipo de modo de limpeza a ser utilizado quando a consulta for executada, substituindo esse tipo de modo configurado no EntityManager.

Elementos de Consulta do eXtreme Scale

Com o mecanismo de consulta do eXtreme Scale, é possível utilizar uma linguagem de consulta única para procurar o cache do eXtreme Scale. Esta linguagem de consulta pode consultar objetos Java que são armazenados em objetos ObjectMap e objetos Entity. Utilize a sintaxe a seguir para criar uma cadeia de consultas.

Uma consulta do eXtreme Scale é uma cadeia que contém os seguintes elementos:

- Uma cláusula SELECT que especifica os objetos ou valores a retornar.
- Uma cláusula FROM que nomeia as coletas de objeto.
- Uma cláusula WHERE opcional que contém predicados de procura sobre as coletas.
- Uma cláusula GROUP BY e HAVING (consulte as funções de agregação de consulta do eXtreme Scale).
- Uma cláusula ORDER BY opcional que especifica a classificação da coleta de resultados.

Conjuntos de objetos Java são identificados em consultas por meio do uso de seu nome na cláusula FROM da consulta.

Os elementos da linguagem de consulta são discutidos em mais detalhes nos tópicos relacionados a seguir:

- Sintaxe do "Backus-Naur Form de Consulta do ObjectGrid" na página 225
- "Referência para Consultas do eXtreme Scale" na página 217

Os tópicos a seguir descrevem os meios para usar a API de Consulta:

- "API de Consulta EntityManager" na página 213
- "Uso da API ObjectQuery" na página 208

Consultando Dados em vários Fusos Horários

Em um cenário distribuído, consultas são realmente executadas em servidores. Ao consultar dados com predicados do tipo calendar, java.util.Date e timestamp, o valor de data / hora especificado em uma consulta é baseado no fuso horário local do servidor.

Em um sistema de fuso horário único no qual todos os clientes e servidores são executados no mesmo fuso horário, você não precisa considerar os problemas relacionados aos tipos de predicado com calendar, java.util.Date e timestamp. Entretanto, quando clientes e servidores estão em fusos horários diferentes, o valor de data / hora especificado nas consultas é baseado no fuso horário do servidor e pode retornar dados indesejados para o cliente. Sem saber o fuso horário do servidor, o valor de data / hora especificado não tem sentido. Portanto, o valor de data / hora especificado deve considerar a diferença do deslocamento de fuso horário entre o fuso horário de destino e o fuso horário do servidor.

Deslocamento de Fuso Horário

Por exemplo, suponha que um cliente esteja em um fuso horário [GMT-0] e que o servidor esteja em um fuso horário [GMT-6]. O fuso horário do servidor está 6 horas atrás do cliente. O cliente gostaria de executar a seguinte consulta:

```
SELECT e FROM Employee
e WHERE e.birthDate='1999-12-31 06:00:00'
```

Supondo que a entidade Employee tenha um atributo birthDate do tipo java.util.Date, o cliente está no fuso horário [GMT-0] e quer recuperar Employees com o valor de birthDate de '1999-12-31 06:00:00 [GMT-0]' com base em seu fuso horário.

A consulta será executada no servidor e o valor de birthDate utilizado pelo mecanismo de consulta será '1999-12-31 06:00:00 [GMT-6]', que é igual a '1999-12-31 12:00:00 [GMT-0]'. Employees com valor de birthDate igual a '1999-12-31 12:00:00 [GMT-0]' serão retornados ao cliente. Além disso, o cliente não obterá Employees desejados com valor de birthDate de '1999-12-31 06:00:00 [GMT-0]'.

O problema descrito ocorre devido à diferença de fuso horário entre cliente e servidor. Para resolver esse problema, uma abordagem é calcular o deslocamento de fuso horário entre cliente e servidor e aplicar esse deslocamento no valor de data / hora de destino na consulta. No exemplo de consulta anterior, o deslocamento de fuso horário é de -6 horas, e o predicado birthDate ajustado deve ser “birthDate='1999-12-31 00:00:00'” se o cliente pretende recuperar Employees com valor de birthDate de '12-31 06:00:00 [GMT-0]'. Com o valor de birthDate ajustado, o servidor utilizará '1999-12-31 00:00:00 [GMT-6]' que é igual ao valor de destino '12-31 06:00:00 [GMT-0]', e Employees necessários serão retornados ao cliente.

Implementação Distribuída em vários Fusos Horários

Se a grade do eXtreme Scale distribuída for implementada em vários servidores ObjectGrid em vários fusos horários, a abordagem de ajuste do deslocamento de fuso horário não funcionará porque o cliente não saberá qual servidor executará a consulta e, assim, não poderá determinar o deslocamento de fuso horário a ser utilizado. A única solução é utilizar o sufixo ‘Z' (sem distinção de maiúsculas e minúsculas) no formato de escape de data e hora do JDBC para indicar o uso do fuso horário GMT com base no valor de data e hora. O sufixo ‘Z' (sem distinção de maiúsculas e minúsculas) indica o uso do fuso horário GMT com base no valor de data e hora. Sem o sufixo ‘Z', o valor de data e hora baseado no fuso horário local será utilizado no processo que executa a consulta.

A consulta a seguir é equivalente ao exemplo anterior, mas utiliza o sufixo ‘Z':

```
SELECT e FROM Employee e WHERE e.birthDate='1999-12-31 06:00:00Z'
```

A consulta deve localizar Employees com valor de birthDate de ‘1999-12-31 06:00:00'. O sufixo ‘Z' indica que o valor de birthDate especificado é baseado no fuso horário GMT, portanto, o valor de birthDate ‘1999-12-31 06:00:00 [GMT-0]' baseado no fuso horário GMT será utilizado pelo mecanismo de consulta para o valor do critério de correspondência. Employees com valor de atributo birthDate igual a esse valor de birthDate baseado em GMT ‘1999-12-31 06:00:00 [GMT-0]' serão incluídos no resultado da consulta. O uso do sufixo ‘Z' no formato de escape de data e hora do JDBC em qualquer consulta é crucial para tornar o fuso horário dos aplicativos seguro. Sem essa abordagem, o valor de data e hora é baseado no fuso horário do servidor e não tem sentido a partir da perspectiva do cliente quando clientes e servidores estão em fusos horários diferentes.

Para obter mais informações, consulte o tópico sobre como inserir dados para fusos horários diferentes em *Visão Geral do Produto*

Dados para Diferentes Fusos Horários

Ao inserir dados com os atributos calendar, java.util.Date e timestamp em um ObjectGrid, você deve garantir que esses atributos de data e hora sejam criados com base no mesmo fuso horário, principalmente quando implementados em vários servidores em vários fusos horários. Usar o mesmo objeto de data e hora baseado em fuso horário pode garantir que o aplicativo esteja protegido por fuso horário e que os dados possam ser consultados pelos predicados calendar, java.util.Date e timestamp.

Sem especificar explicitamente um fuso horário ao criar objetos de data e hora, o Java usa o fuso horário local e pode causar valores de data e hora inconsistentes nos clientes e servidores.

Considere um exemplo em uma implementação distribuída na qual o client1 está no fuso horário [GMT-0] e o client2 está no [GMT-6], e ambos querem criar um objeto java.util.Date com o valor '1999-12-31 06:00:00'. Então, o client1 criará o objeto java.util.Date com o valor '1999-12-31 06:00:00 [GMT-0]' e o client2 criará o objeto java.util.Date com o valor '1999-12-31 06:00:00 [GMT-6]'. Os objetos java.util.Date não são iguais porque o fuso horário é diferente. Um problema semelhante ocorre quando você pré-carrega os dados nas partições que residem em servidores em fusos horários diferentes se o fuso horário local for utilizado para criar objetos de data e hora.

Para evitar o problema descrito, o aplicativo pode escolher um fuso horário como [GMT-0] como fuso horário base para criar objetos calendar, java.util.Date e timestamp.

Uso da API ObjectQuery

A API ObjectQuery fornece métodos para consulta de dados no ObjectGrid que são armazenados usando a API ObjectMap. Quando um esquema é definido na instância do ObjectGrid, a API do ObjectQuery pode ser usada para criar e executar consultas sobre os objetos heterogêneos armazenados nos mapas de objetos.

Consulta e Mapas de Objetos

Você pode usar uma capacidade de consulta avançada para objetos que são armazenados usando a API do ObjectMap. Essas consultas permitem que os objetos sejam recuperados utilizando os atributos não-chave e executem agregações simples, como sum, avg, min e max junto a todos os dados que correspondem a uma consulta. Os aplicativos podem construir uma consulta usando o método

Session.createObjectQuery. Ese método retorna um objeto ObjectQuery que pode ser então interrogado para se obter os resultados da consulta. O objeto de consulta também permite que a consulta seja customizada antes de executá-la. A consulta é executada automaticamente quando qualquer método que retorna o resultado é chamado.

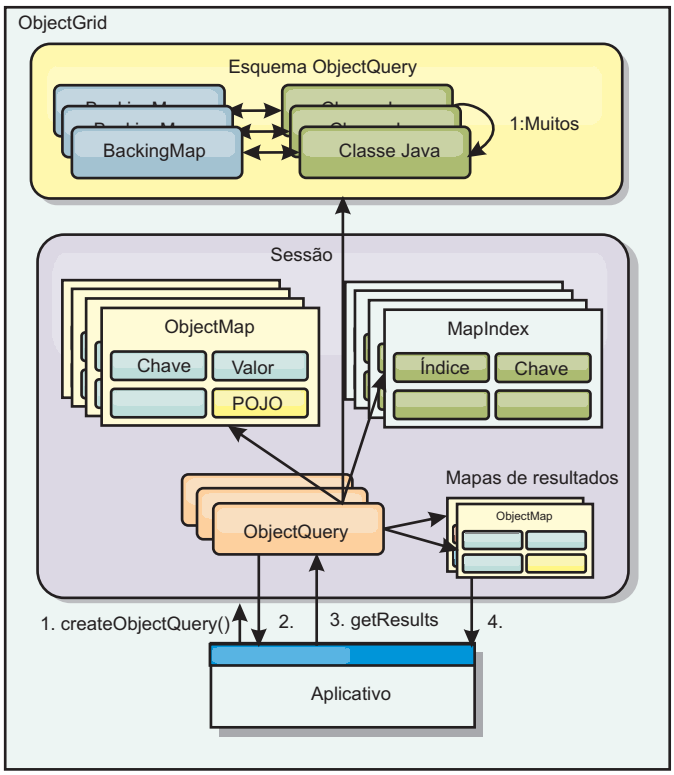


Figura 24. A interação da consulta com os mapas de objetos e como um esquema é definido para classes e associado a um mapa ObjectGrid

Definindo um Esquema do ObjectMap

Os mapas de objetos são usados para armazenar objetos em várias formas e são largamente livres de formatos. Um esquema deve ser definido no ObjectGrid que define o formato dos dados. Um esquema é composto das seguintes partes:

- O tipo de objeto armazenado no ObjectMap
- Relacionamentos entre ObjectMaps
- O método ao qual cada consulta acessa os atributos de dados nos objetos (campos e métodos de propriedade)
- O nome do atributo da chave primária no objeto.

Consulte o esquema Configurando um ObjectQuery para obter detalhes.

Para obter um exemplo sobre a criação de um esquema programaticamente ou usando o arquivo XML descritor do ObjectGrid, consulte "Tutorial do ObjectQuery - Etapa 3" na página 3o tutorial no ObjectQuery no *Visão Geral do Produto*.

Consultando Objetos com a API do ObjectQuery

A interface ObjectQuery permite a consulta de objetos de não-entidade, que sejam objetos heterogêneos e estejam armazenados nos ObjectMaps do ObjectGrid. A API

do ObjectQuery fornece uma forma fácil de localizar objetos ObjectMap sem usar o teclado e os mecanismos de índice diretamente.

Há dois métodos para recuperação de resultados de um ObjectQuery: getResultIterator e getResultMap.

Recuperando resultados da consulta utilizando getResultIterator

Basicamente, os resultados de consultas são uma lista de atributos. Suponha que a consulta for selecionar a,b,c de X, em que y=z. Esta consulta retorna uma lista de linhas contendo a, b e c. Esta lista é armazenada em um Mapa com escopo em transação, o que significa que você deve associar uma chave artificial a cada linha e utilizar um número inteiro que aumenta a cada linha. Este mapa é obtido utilizando o método ObjectQuery.getResultMap(). É possível acessar os elementos de cada linha utilizando o código semelhante ao seguinte:

```
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 ]);
  }
```

Recuperando resultados da consulta utilizando getResultMap

Os resultados da consulta também podem ser recuperados utilizando diretamente o mapa de resultados. O exemplo a seguir mostra uma consulta recuperando partes específicas dos Clientes correspondentes e demonstra como acessar as linhas resultantes. Observe que, se você utilizar o objeto ObjectQuery para acessar os dados, então o identificador de linha longa gerado será ocultado. A linha longa é visível somente ao utilizar o ObjectMap para acessar o resultado.

Quando a transação é concluída, este mapa desaparece. O mapa também está visível apenas na sessão utilizada, ou seja, geralmente apenas no encadeamento que o criou. O mapa utiliza uma chave do tipo Long que representa o ID da linha. Os valores armazenados no mapa são do tipo Object ou Object[], em que cada elemento corresponde ao tipo do elemento na cláusula select da 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 obter exemplos sobre o uso do ObjectQuery, consulte “Tutorial: Consultando uma Grade de Dados na Memória Local” na página 1o tutorial na API do ObjectQuery no *Visão Geral do Produto*.

Configurando um Esquema ObjectQuery:

O ObjectQuery conta com o esquema ou informações de formato para executar a verificação semântica e avaliar expressões de caminho. Esta seção descreve como definir o esquema no XML ou programaticamente.

Definindo o Esquema

O esquema do ObjectMap é definido no XML do descritor de implementação do ObjectGrid ou programaticamente utilizando as técnicas de configuração normais do eXtreme Scale. Para obter um exemplo de como criar um esquema, consulte "Configurando um Esquema ObjectQuery"

As informações do esquema descrevem os POJOs (Plain Old Java Objects): cujos atributos que os compõem e quais os tipos de atributos que podem ter, sejam os atributos de campos de chave primária, relacionamentos de um único valor ou de múltiplos valores, ou relacionamentos bidirecionais. As informações do esquema conduzem o ObjectQuery a utilizar acesso de campo ou acesso de propriedade.

Atributos que Podem Ser Consultados

Quando o esquema é definido no ObjectGrid, os objetos no esquema são examinados automaticamente utilizando reflexão para determinar quais atributos estão disponíveis para consulta. Você pode consultar os tipos de atributos a seguir:

- Os tipos primitivos Java incluindo wrappers
- 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[]
- Enum J2SE

Tipos serializáveis integrados que não aqueles indicados anteriormente também podem ser incluídos em um resultado da consulta, mas não podem ser incluídos na cláusula WHERE ou FROM da consulta. Atributos serializáveis não são navegáveis.

Os tipos de atributo podem ser excluídos do esquema se o tipo não for serializável, o campo ou propriedade é estática, ou o campo é temporário. Como todos os objetos de mapa devem ser serializáveis, o ObjectGrid inclui somente atributos que podem ser persistidos a partir do objeto. Outros objetos são ignorados.

Atributos de campo

Quando um esquema é configurado para acessar o objeto utilizando campos, todos serializáveis, os campos não-temporários são incorporados automaticamente no esquema. Para selecionar um atributo de campo em uma consulta, utilize o nome identificador de campo como ele existe na definição de classe.

Todos os campos público, privado, protegido e de pacote protegido são incluídos no esquema.

Atributos de propriedade

Quando o esquema está configurado para acessar o objeto usando propriedades, todos os métodos serializáveis que seguem as convenções de nomenclatura da propriedade JavaBeans serão automaticamente incorporadas no esquema. Para selecionar um atributo de propriedade para a consulta, use as convenções de nome de propriedade de estilo JavaBeans.

Todas as propriedades pública, privada, protegida e de pacote protegido são incluídas no esquema.

Na classe a seguir, os seguintes atributos são incluídos no esquema: name, birthday, valid.

```
public class Person {
  public String getName(){}
  private java.util.Date getBirthday(){}
  boolean isValid(){}
  public NonSerializableObject getData(){}
}
```

Ao utilizar um CopyMode de COPY_ON_WRITE, o esquema da consulta deve sempre utilizar o acesso baseado em propriedade. COPY_ON_WRITE cria objetos proxy sempre que os objetos são recuperados do mapa e podem acessar apenas aqueles objetos utilizando métodos de propriedade. A falha ao fazer isso resultará em cada resultado da consulta sendo configurado como nulo.

Relacionamentos

Cada relacionamento deve ser definido explicitamente na configuração do esquema. A cardinalidade do relacionamento é determinada automaticamente pelo tipo do atributo. Se o atributo implementa a interface java.util.Collection, então o relacionamento é um relacionamento de um-para-muitos ou de muitos-para-muitos.

Diferente das consultas de entidade, os atributos que se referem a outros objetos de cache não podem armazenar diretamente referências no objeto. As referências a outros objetos são serializadas como parte dos dados do objeto que as contém. Em vez disso, armazene a chave no objeto relacionado.

Por exemplo, se houver relacionamento de muitos-para-um entre um Cliente e o Pedido:

Incorrect. Storing an object reference.

```
public class Customer {
  String customerId;
  Collection<Order> orders;
}
```

```
public class Order {
  String orderId;
  Customer customer;
}
```

Correto. A chave para o objeto relacionado.

```
public class Customer {
  String customerId;
  Collection<String> orders;
}

public class Order {
  String orderId;
  String customer;
}
```

Quando uma consulta é executada de modo a unir os dois objetos de mapa, a chave será automaticamente aumentada. Por exemplo, a seguinte consulta retorna objetos de Cliente:

```
SELECT c FROM Order o JOIN Customer c WHERE orderId=5
```

Utilizando Índices

O ObjectGrid utiliza plugins de índice para incluir índices nos mapas. O mecanismo de consulta incorpora automaticamente todos os índices definidos em um elemento de mapa do esquema do tipo: com.ibm.websphere.objectgrid.plugins.index.HashIndex e a propriedade rangeIndex é configurada para true. Se o tipo do índice não for HashIndex e a propriedade rangeIndex não estiver configurada para true, então o índice é ignorado pela consulta. Consulte o "Tutorial do ObjectQuery - Etapa 2" na página 2 tutorial do ObjectQuery no *Visão Geral do Produto* para obter um exemplo de como incluir um índice no esquema.

API de Consulta EntityManager

A API do EntityManager fornece métodos para consultar dados no ObjectGrid armazenado utilizando a API do EntityManager. A API de Consulta EntityManager é usada para criar e executar consultas sobre uma ou mais entidades definidas no eXtreme Scale.

Consulta e ObjectMaps para Entidades

O WebSphere Extended Deployment v6.1 introduziu um recurso de consulta avançado para entidades armazenadas no eXtreme Scale. Essas consultas permitem que os objetos sejam recuperados utilizando os atributos não-chave e executem agregações simples, como sum, average, minimum e maximum junto a todos os dados que correspondem a uma consulta. Os aplicativos constroem uma consulta utilizando a API EntityManager.createQuery. Isso retorna um objeto de consulta e pode, então, ser interrogado para obter os resultados da consulta. O objeto de consulta também permite que a consulta seja customizada antes de executá-la. A consulta é executada automaticamente quando qualquer método que retorna o resultado é chamado.

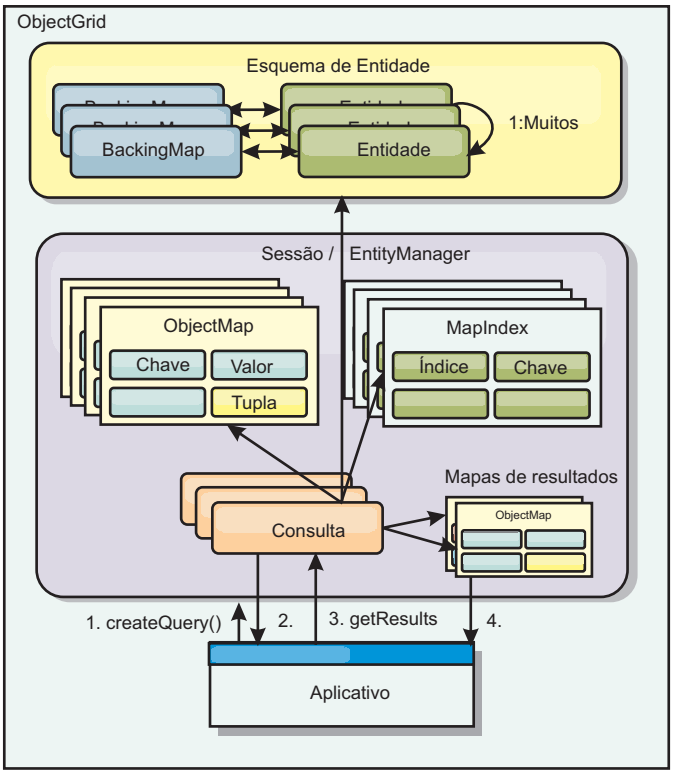


Figura 25. A interação da consulta com os mapas de objetos ObjectGrid e como o esquema da entidade é definido e associado com um mapa ObjectGrid.

Recuperando resultados da consulta usando o método getResultIterator

Os resultados da consulta são uma lista de atributos. Se uma consulta foi selecionar a,b,c de X, em que y=z, então, uma lista de linhas contendo a, b e c é retornada. Essa lista é armazenada em um Mapa de transações com escopo definido, o que significa que é necessário associar uma chave artificial com cada linha e utilizar um inteiro que aumenta com cada linha. Este mapa é obtido usando o método Query.getResultMap. O mapa possui EntityMetaData, que descreve cada linha no Mapa associado a ele. É possível acessar os elementos de cada linha utilizando o código semelhante ao seguinte:

```
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 ]);
  }
```

Recuperando Resultados da Consulta Utilizando getResultMap

O código a seguir mostra a recuperação de partes específicas dos Clientes correspondentes e mostra como acessar as linhas resultantes. Se você utilizar o objeto Query para acessar os dados, então o identificador de linha longa gerado será ocultado. O Long é visível somente ao utilizar o ObjectMap para acessar o resultado. Quando a transação é concluída, este mapa desaparece. O Mapa está visível apenas na Session utilizada, ou seja, geralmente apenas no encadeamento

que o criou. O Mapa utiliza uma Tupla para a chave com um único atributo, um Long com o ID da linha. O valor é uma outra tupla com um atributo para cada coluna no conjunto de resultados.

O código de amostra a seguir demonstra isto:

```
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));
}
```

Recuperando Resultados da Consulta Utilizando um Agente Iterativo de Resultado da Entidade

O código a seguir mostra a consulta e o loop que recupera cada linha resultante utilizando as APIs de mapas normais. A chave para o Mapa é uma Tupla. Portanto, construa uma dos tipos corretos utilizando o resultado do método createTuple em keyTuple. Tente recuperar todas as linhas com rowIds de 0 em diante. Quando você obtém retornos de nulo (indicando que a chave não foi localizada), então o loop é finalizado. Configure o primeiro atributo de keyTuple como o long que deseja localizar. O valor retornado por get também é uma Tupla com um atributo para cada coluna no resultado da consulta. Em seguida, empurre cada atributo da Tupla de valor utilizando getAttribute.

A seguir está o próximo trecho de código:

```
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 is the id not the firstName.
  System.out.println("Found a Claus with id " + row.id
    + ", firstName: " + row.firstName
    + ", surname: " + row.surname);
}

em.getTransaction().commit();
```

Um valor de ResultEntityName está especificado na consulta. Este valor informa ao mecanismo de consulta que você deseja projetar cada linha para um objeto específico, CustomerQueryResult, neste caso. A classe é a seguinte:

```
@Entity
public class CustomerQueryResult {
 @Id long rowId;
 String id;
 String firstName;
 String surname;
};
```

No primeiro snippet, observe que cada linha da consulta é retornada como um objeto CustomerQueryResult em vez de um Object[]. As colunas resultantes da consulta são projetadas para o objeto CustomerQueryResult. Projetar o resultado é

ligeiramente mais lento no tempo de execução, porém mais legível. O resultado da consulta Entities não deve ser registrado com o eXtreme Scale na inicialização. Se as entidades são registradas, então um Mapa global com o mesmo nome é criado, e a consulta falha com um erro indicando nome de Mapa duplicado.

Consultas Simples com EntityManager:

WebSphere eXtreme Scale é fornecido com a API de consulta de EntityManager.

A API de consulta EntityManager é muito semelhante a outros mecanismos de consulta SQL que pesquisam sobre objetos. Uma consulta é definida, então o resultado é recuperado da consulta utilizando vários métodos getResult.

Os exemplos a seguir referem-se às entidades usadas no tutorial do EntityManager na Visão Geral do Produto.

Executando uma Consulta Simples

Neste exemplo, os clientes com o sobrenome Claus são consultados:

```
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();
```

Utilizando Parâmetros

Como você quer localizar todos os clientes com um sobrenome Claus, um parâmetro para especificar o sobrenome é utilizado, visto que você pode utilizar essa consulta mais de uma vez.

Exemplo de Parâmetro Posicional

```
Query q = em.createQuery("select c from Customer c where c.surname=?1");
  q.setParameter(1, "Claus");
```

O uso de parâmetros é muito importante quando a consulta é utilizada mais de uma vez. O EntityManager precisa analisar a cadeia de consultas e construir um plano para a consulta, o qual é caro. Utilizando um parâmetro, o EntityManager armazena em cache o plano para a consulta, reduzindo, assim, o tempo que leva para executar uma consulta.

Os parâmetros posicionais e nomeados são utilizados:

Exemplo de Parâmetro Nomeado

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
  q.setParameter("name", "Claus");
```

Utilizando um Índice para Melhorar o Desempenho

Se houver milhões de clientes, a consulta anterior precisará varrer sobre todas as linhas no Mapa do Cliente. Isso não é muito eficiente. Mas o eXtreme Scale fornece

um mecanismo para definição de índices sobre atributos individuais em uma entidade. A consulta automaticamente utiliza este índice quando apropriado, o que pode acelerar as consultas dramaticamente.

Você pode especificar quais atributos relacionar muito simples, utilizando a anotação @Index no atributo entity:

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

O EntityManager cria um índice ObjectGrid apropriado para o atributo surname na entidade Customer e o mecanismo de consulta utiliza automaticamente o índice, o qual diminui bastante o tempo da consulta.

Utilizando a Paginação para Melhorar o Desempenho

Se houver um milhão de clientes chamados Claus, provavelmente você não vai querer exibir uma página que mostra um milhão de clientes. É mais provável que você queira exibir 10 ou 25 clientes de uma vez.

Os métodos Query setFirstResult e setMaxResults ajudam apenas a retornar um subconjunto dos resultados.

Exemplo de Paginação

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
  q.setParameter("name", "Claus");
  // Display the first page
  q.setFirstResult=1;
  q.setMaxResults=25;
  displayPage(q.getResultIterator());

  // Display the second page
  q.setFirstResult=26;
  displayPage(q.getResultIterator());
```

Referência para Consultas do eXtreme Scale

WebSphere eXtreme Scale tem sua própria linguagem por meio da qual o usuário pode consultar dados.

Cláusula FROM de Consulta do ObjectGrid

A cláusula FROM especifica as coleções de objetos aos quais a consulta deve ser aplicada. Cada coleta é identificada ou por um nome de esquema abstrato e uma variável de identificação, chamada uma variável de intervalo, ou por uma declaração de membro de coleta que identifica um relacionamento com valor único e com vários valores e uma variável de identificação.

Conceitualmente, a semântica da consulta serve para primeiro formar uma coleta temporária de tuplas, referidas como R. As tuplas são compostas de elementos das coletas que são identificadas na cláusula FROM. Cada tupla contém um elemento de cada uma das coleções na cláusula FROM. Todas as combinações possíveis são formadas sujeitas às restrições impostas pelas declarações de membros da coleta. Se algum nome de esquema identificar uma coleta para a qual não existem

registros no armazenamento persistente, a coleta temporária R será vazia.

Exemplos utilizando FROM

O objeto DeptBean contém os registros 10, 20 e 30. O objeto EmpBean contém os registros 1, 2 e 3 que são relacionados ao departamento 10 e os registros 4 e 5 que são relacionados ao departamento 20. O departamento 30 não possui funcionários.

```
FROM DeptBean d, EmpBean e
```

Essa cláusula forma uma coleta temporária R que contém 15 tuplas.

```
FROM  DeptBean d,  DeptBean d1
```

Essa cláusula forma uma coleta temporária R que contém 9 tuplas.

```
FROM  DeptBean d,  IN (d.emps) AS e
```

Essa cláusula forma uma coleta temporária R que contém 5 tuplas. O departamento 30 não está na coleta temporária R porque não contém funcionários. O departamento 10 estará contido na coleta temporária R três vezes e o departamento 20 estará contido em R duas vezes.

Em vez de utilizar IN(d.emps) como e, será possível utilizar um predicado JOIN:

```
FROM DeptBean d JOIN d.emps as e
```

Depois de formar a coleta temporária, as condições de procura da cláusula WHERE serão aplicadas à coleta temporária R e isso produzirá uma nova coleta temporária R1. As cláusulas ORDER BY e SELECT são aplicadas a R1 para resultar no conjunto de resultados final.

Uma variável de identificação é uma variável declarada na cláusula FROM utilizando o operador IN ou o operador AS opcional.

```
FROM DeptBean AS d,  IN (d.emps) AS e
```

é equivalente a:

```
FROM DeptBean d, IN (d.emps) e
```

Uma variável de identificação que é declarada para ser um nome de esquema abstrato é chamada uma variável de faixa. Na consulta anterior, "d" é uma variável de intervalo. Uma variável de identificação que é declarada para ser uma expressão de caminho com diversos valores é chamada uma declaração de membro de coleta. Os valores "d" e "e" no exemplo anterior são declarações do membro de coleta.

A seguir, está um exemplo de utilização de uma expressão de caminho com valor único na cláusula FROM:

```
FROM EmpBean e, IN(e.dept.mgr) as m
```

Cláusula SELECT de Consulta do ObjectGrid

A sintaxe da cláusula SELECT é ilustrada no seguinte exemplo:

```
SELECT { ALL | DISTINCT }  [ seleção , ]*  seleção
selection  ::= {single_valued_path_expression |
                identification_variable |
                OBJECT ( identification_variable) |
        aggregate_functions } [[ AS ] id ]
```

A cláusula SELECT consiste em um ou mais dos seguintes elementos: uma única variável de identificação, definida na cláusula FROM, ou uma expressão de caminho com valor único que é avaliada para referências ou valores do objeto e uma função agregada. É possível utilizar a palavra-chave DISTINCT para eliminar as referências duplicadas.

Uma subseleção-escalar é uma subseleção que retorna um único valor.

Exemplos utilizando SELECT

Localizar todos os funcionários que ganham mais que João:

```
SELECT OBJECT(e) FROM EmpBean ej, EmpBean eWHERE  ej.name = 'John'  and
e.salary > ej.salary
```

Localizar todos os departamentos que têm um ou mais funcionários que ganham menos de 20000:

```
SELECT DISTINCT e.dept  FROM EmpBean e where e.salary < 20000
```

Uma consulta que pode ter uma expressão de caminho que seja avaliada para um valor arbitrário:

```
SELECT e.dept.name  FROM EmpBean e where e.salary < 20000
```

A consulta anterior retorna uma coleta de valores de nomes para os departamentos que possuem funcionários que ganham menos de 20000.

Uma consulta pode retornar um valor agregado:

```
SELECT  avg(e.salary)  FROM EmpBean e
```

Uma consulta que recupera os nomes e as referências do objeto para funcionário não remunerados a seguir:

```
SELECT e.name as name, object(e) as emp from EmpBean e where e.salary <
50000
```

Cláusula WHERE de Consulta do ObjectGrid

A cláusula WHERE contém condições de procura que são compostos dos elementos apresentados abaixo. Quando uma condição de procura é avaliada como TRUE, a tupla é incluída no conjunto de resultados.

Literais de Consulta do ObjectGrid

Um literal de cadeia é delimitado por aspas simples. Uma aspa simples que ocorre dentro de uma cadeia literal é representada por duas aspas simples, por exemplo: 'Tom''s'.

Um literal numérico pode ser qualquer um dos seguintes valores:
- Um valor exato como 57, -957 ou +66
- Qualquer valor suportado pelo tipo long Java
- Um literal decimal como 57,5 ou -47,02
- Um valor numérico aproximado como 7E3 ou -57.4E-2
- Tipos float devem incluir o qualificador "F", por exemplo, 1.0F
- Tipos long devem incluir o qualificador "L", por exemplo, 123L

Os literais booleanos são TRUE e FALSE.

Os literais temporais seguem a sintaxe de escape JDBC baseada no 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...

Os literais de enumeração são expressos usando a sintaxe de literal de enumeração Java com um nome de classe de enumeração completo.

Parâmetros de Entrada de Consulta do ObjectGrid

É possível especificar parâmetros de entrada utilizando uma posição ordinal ou um nome de variável. A gravação de consultas que utilizam parâmetros de entrada é bastante incentivada, porque o uso de parâmetros de entrada aumenta o desempenho, permitindo que o ObjectGrid capture o plano da consulta entre ações em execução.

Um parâmetro de entrada pode ser um dos seguintes tipos: 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, uma enumeração Java SE 5, uma Entity ou POJO Object ou uma cadeia de dados binários no formato Java byte[].

Um parâmetro de entrada não pode ter um valor NULL. Para procurar pela ocorrência de um valor NULL, utilize o predicado NULL.

Parâmetros Posicionais

Os parâmetros de entrada posicionais são definidos utilizando um ponto de interrogação seguido por um número positivo:

```
?[inteiro positivo].
```

Parâmetros de entrada posicionais são numerados iniciando em 1 e correspondem aos argumentos da consulta; portanto, uma consulta não deve conter um parâmetro de entrada que exceda o número de argumentos de entrada.

Exemplo: `SELECT e FROM Employee e WHERE e.city = ?1 and e.salary >= ?2`

Parâmetros Denominados

Os parâmetros de entrada denominados são definidos utilizando um nome de variável no formato: :[nome do parâmetro].

Exemplo: `SELECT e FROM Employee e WHERE e.city = :city and e.salary >= :salary`

Predicado BETWEEN de Consulta do ObjectGrid

O predicado BETWEEN determina se um valor dado está entre dois outros valores dados.

```
expressão  [NOT]  BETWEEN expressão-2  AND expressão-3
```

Exemplo 1

```
e.salary BETWEEN 50000 AND 60000
```

é equivalente a:

```
e.salary >= 50000  AND e.salary <= 60000
```

Exemplo 2

```
e.name NOT BETWEEN 'A' AND 'B'
```

é equivalente a:

```
e.name < 'A'  OR e.name > 'B'
```

Predicado IN de Consulta do ObjectGrid

O predicado IN compara um valor a um conjunto de valores. É possível utilizar o predicado IN em um dos dois formatos:

```
expression [NOT] IN ( subselect )expression [NOT] IN  ( value1, value2, .... )
```

O valor ValorN pode ser um valor literal ou um parâmetro de entrada. A expressão não pode ser avaliada para um tipo de referência.

Exemplo 1

```
e.salary IN ( 10000, 15000 )
```

é equivalente a

```
( e.salary = 10000  OR  e.salary = 15000 )
```

Exemplo 2

```
e.salary IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

é equivalente a

```
e.salary = ANY   ( select  e1.salary from EmpBean e1 where e1.dept.deptno =
10)
```

Exemplo 3

```
e.salary NOT IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno =
10)
```

é equivalente a

```
e.salary <> ALL    ( select  e1.salary from EmpBean e1 where e1.dept.deptno
= 10)
```

Predicado LIKE de Consulta do ObjectGrid

O predicado LIKE pesquisa um valor de cadeia para um certo padrão.

```
expressão-de-cadeia   [NOT]  LIKE  padrão    [ ESCAPE caractere-de-escape ]
```

O valor padrão é uma cadeia literal ou marcador de parâmetro do tipo string no qual o sublinhado (_) representa qualquer caractere único e o símbolo de percentual (%) representa qualquer sequência de caracteres, incluindo uma sequência vazia. Qualquer outro caractere significa ele próprio. O caractere de escape pode ser utilizado para pesquisar os caracteres _ e %. O caractere de escape pode ser especificado como um literal de cadeia ou um parâmetro de entrada.

Se a expressão-de-cadeia for nula, o resultado será desconhecido.

Se a expressão-de-cadeia e o padrão forem ambos vazios, o resultado será true.

Exemplo

```
'' 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 do ObjectGrid

O predicado NULL testa a ocorrência de valores nulos.

```
{expressão-de-caminho-com-valor-único | parâmetro_de_entrada} IS [NOT] NULL
```

Exemplo

```
e.name IS NULL
e.dept.name IS NOT NULL
e.dept IS NOT NULL
```

Predicado de Coleta EMPTY de Consulta do ObjectGrid

Utilize o predicado de coleta EMPTY para testar uma coleta vazia.

Para testar se um relacionamento com vários valores é vazio, utilize a seguinte sintaxe:

```
expressão-de-caminho-com-valor-de-coleção  IS [NOT]  EMPTY
```

Exemplo

Predicado de coleta vazio para localizar todos os departamentos que não possuem funcionários:

```
SELECT OBJECT(d) FROM DeptBean d  WHERE  d.emps IS EMPTY
```

Predicado MEMBER OF de Consulta do ObjectGrid

A expressão a seguir testa se a referência de objeto especificada pela expressão de caminho com valor único ou parâmetro de entrada é um membro da coleta designada. Se a expressão de caminho com valor da coleta designar uma coleta vazia o valor da expressão MEMBER OF será FALSE.

```
{ expressão-de-caminho-com-valor-único | parâmetro_de_entrada } [ NOT ]
MEMBER [ OF ] expressão-de-caminho-com-valor-de-coleção
```

Exemplo

Localizar funcionários que não são membros de um número de departamento dado:

```
SELECT OBJECT(e) FROM EmpBean e , DeptBean d
WHERE e NOT MEMBER OF d.emps AND d.deptno = ?1
```

Localizar funcionários cujo gerente é um membro de um 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 do ObjectGrid

O predicado EXISTS testa a presença ou ausência de uma condição especificada por uma subseleção.

```
EXISTS ( subseleção )
```

O resultado de EXISTS será true se a subseleção retornar pelo menos um valor; caso contrário, o resultado será false.

Para negar um predicado EXISTS, preceda-o com o operador lógico NOT.

Exemplo

Retornar departamentos que têm pelo menos um funcionário ganhando mais que 1000000:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE EXISTS ( SELECT  e  FROM IN (d.emps) e WHERE  e.salary > 1000000 )
```

Retornar departamentos que não têm funcionários

```
SELECT OBJECT(d) FROM DeptBean d
WHERE NOT EXISTS  ( SELECT e FROM IN (d.emps) e)
```

É possível reescrever a consulta anterior como no exemplo a seguir:

```
SELECT OBJECT(d) FROM DeptBean d WHERE SIZE(d.emps)=0
```

Cláusula ORDER BY de Consulta do ObjectGrid

A cláusula ORDER BY especifica uma ordenação dos objetos na coleta de resultados. Este é um exemplo:

ORDER BY [order_element ,]* order_element order_element ::={ path-expression }[ASC | DESC]

A expressão de caminho deve especificar um campo de valor único que é um tipo primitivo de byte, short, int, long, float, double, char, ou um tipo de wrapper de Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp e java.util.Calendar. O elemento de ordem ASC especifica que os resultados são exibidos em ordem crescente, que é o padrão. Um elemento de ordem DESC especifica que os resultados são exibidos em ordem decrescente.

Exemplo

Retornar objetos do departamento. Exibir os números de departamento em ordem decrescente:

```
SELECT  OBJECT(d) FROM  DeptBean d ORDER BY  d.deptno DESC
```

Retornar objetos de funcionário, classificados por número de departamento e nome:

```
SELECT  OBJECT(e) FROM EmpBean e ORDER BY e.dept.deptno ASC,  e.name DESC
```

Funções de Agregação de Consulta do ObjectGrid

As funções de agregação operam em um conjunto de valores para retornar um valor escalar único. É possível utilizar essas funções nos métodos select e subselect. O exemplo a seguir ilustra uma agregação:

```
SELECT  SUM (e.salary) FROM EmpBean e WHERE e.dept.deptno =20
```

Essa agregação calcula o salário total para o departamento 20.

As funções de agregação são: AVG, COUNT, MAX, MIN e SUM. A sintaxe de uma função de agregação é ilustrada no exemplo a seguir:

```
função-de-agregação  (    [ ALL |  DISTINCT ]  expressão )
```

ou:

```
COUNT( [ ALL |  DISTINCT ] variável de identificação )
```

A opção DISTINCT elimina valores duplicados antes de aplicar a função. A opção ALL é a opção padrão e não elimina valores duplicados. Os valores nulos são ignorados no cálculo da função agregada, exceto quando você utiliza a função COUNT(identification-variable), que retorna uma contagem de todos os elementos no conjunto.

Definindo o Tipo de Retorno

As funções MAX e MIN podem se aplicar a qualquer tipo de dados numéricos, de cadeia ou de data-hora e retornam o tipo de dados correspondente. As funções SUM e AVG pegam um tipo numérico como entrada. A função AVG retorna um tipo double. A função SUM retorna um tipo long se o tipo de entrada for um tipo integer, exceto quando a entrada for um tipo Java BigInteger, e, em seguida, a função retornar um tipo Java BigInteger. A função SUM retorna um tipo double se o tipo de entrada não for um tipo integer, exceto quando a entrada for um tipo Java BigDecimal, e, em seguida, a função retornar um tipo Java BigDecimal. A função COUNT pode tomar qualquer tipo de dados, exceto coletas, e retorna um tipo long.

Quando aplicadas a um conjunto vazio, as funções SUM, AVG, MAX e MIN podem retornar um valor nulo. A função COUNT retorna zero (0) quando aplicada a um conjunto vazio.

Utilizando cláusulas GROUP BY e HAVING

O conjunto de valores utilizado para a função agregada é determinado pela coleta que resulta da cláusula FROM e WHERE da consulta. É possível dividir o conjunto em grupos e aplicar a função de agregação a cada grupo. Para executar essa ação, utilize uma cláusula GROUP BY na consulta. A cláusula GROUP BY define os membros do agrupamento que compreendem uma lista de expressões de caminho. Cada expressão de caminho especifica um campo que é um tipo primitivo de byte, short, int, long, float, double, boolean, char ou um tipo de wrapper de Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp e java.util.Calendar ou um Java SE 5 enum.

O exemplo a seguir ilustra a utilização da cláusula GROUP BY em uma consulta que calcula o salário médio para cada departamento:

```
SELECT e.dept.deptno,  AVG ( e.salary) FROM EmpBean e GROUP BY
e.dept.deptno
```

Na divisão de um conjunto em grupos, um valor NULL é considerado igual a outro valor NULL.

Os grupos podem ser filtrados utilizando uma cláusula HAVING, que testa as propriedades de grupo antes de envolver funções agregadas ou membros do agrupamento. Essa filtragem é similar a como a cláusula WHERE filtra tuplas (isto é, registros dos valores de coleta de retorno) da cláusula FROM. Um exemplo da cláusula HAVING é o seguinte:

```
SELECT e.dept.deptno,  AVG ( e.salary) FROM EmpBean e
GROUP BY e.dept.deptno
HAVING COUNT(e) > 3 AND e.dept.deptno > 5
```

Essa consulta retorna o salário médio para departamentos que possuem mais de três funcionários e o número de departamentos é maior que cinco.

É possível utilizar a cláusula HAVING sem uma cláusula GROUP BY. Nesse caso, todo o conjunto é tratado como um único grupo, ao qual a cláusula HAVING é aplicada.

Backus-Naur Form de Consulta do ObjectGrid:

A seguir, está um resumo da Notação Backus-Naur Form (BNF) de Consulta do ObjectGrid.

Tabela 3. Chave para o Resumo BNF

Representação	Descrição
{...}	Agrupamento
[...]	Construções opcionais
negrito	Palavras-Chave
*	Zero ou mais
|	Alternativas

```
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)
```

Usando Objetos Diferentes de Chaves para Localizar Partições (Interface PartitionableKey)

Quando a configuração do eXtreme Scale usa a estratégia de colocação de partição fixa, ela dependerá do hash da chave para uma partição inserir, obter, atualizar ou remover o valor. O método hashCode é chamado na chave e ele deverá ser bem definido se uma chave customizada for criada. Porém, outra opção é usar a interface PartitionableKey. Com a interface PartitionableKey, será possível usar um objeto diferente da chave para efetuar hash de uma partição.

É possível usar a interface PartitionableKey em situações em que houver vários mapas e os dados que você consolidar serão relatados e, assim, deverão ser colocados na mesma partição. WebSphere eXtreme Scale não suporta two-phase commit, portanto, várias transações de mapa não deverão ser confirmadas se incluírem várias partições. Se o PartitionableKey efetuar hash para a mesma partição para as chaves em mapas diferentes no mesmo conjunto de mapas, eles poderão ser consolidados em conjunto.

Também é possível usar a interface PartitionableKey quando grupos de chaves tiverem que ser colocados na mesma partição, mas não necessariamente durante uma única transação. Se as chaves tiverem que ser submetidas a hash com base no local, departamento, tipo de domínio ou algum outro tipo de identificador, as chaves filha poderão receber um PartitionableKey pai.

Por exemplo, os funcionários devem efetuar hash para a mesma partição do seu departamento. Cada chave de funcionário terá um objeto PartitionableKey que pertence ao mapa do departamento. Então, tanto o funcionário quanto o departamento efetuarão hash para a mesma partição.

A interface PartitionableKey fornece um método, chamado ibmGetPartition. O objeto retornado desse método deve implementar o método hashCode. O resultado retornado do uso de um hashCode alternativo será usado para rotear as chaves para uma partição.

Programação para Transações

Os aplicativos que requerem transações introduzem considerações como bloqueios de manipulação, colisões de manipulação e isolamento de transação.

Visão Geral do Processamento de Transações

O WebSphere eXtreme Scale usa transações de acordo com seu mecanismo de interação com os dados.

Para interagir com os dados, o encadeamento em seu aplicativo precisa de sua própria sessão. Quando o aplicativo desejar usar o ObjectGrid em um encadeamento, chame um dos métodos ObjectGrid.getSession para obter um encadeamento. Com a sessão, o aplicativo pode trabalhar com dados que são armazenados nos mapas ObjectGrid.

Quando um aplicativo usa um objeto de Sessão, a sessão deve estar no contexto de uma transação. Uma transação inicia e é consolidada ou inicia e é recuperada usando os métodos begin, commit e rollback no objeto de Sessão. Os aplicativos também podem trabalhar em modo de auto-consolidação, no qual a Sessão inicia e consolida automaticamente uma transação sempre que uma operação é executada no mapa. O modo de auto-confirmação não pode agrupar várias operações em uma única transação, assim, ele é a opção mais lenta se você estiver criando um lote de várias operações em uma única transação. Porém, para transações que contêm uma operação, a auto-consolidação é a opção mais rápida.

Acesso a Dados e Transações:

Após um aplicativo fazer uma referência a uma instância de ObjectGrid ou uma conexão do cliente com uma grade remota, é possível acessar e interagir com dados em sua configuração do WebSphere eXtreme Scale. Com a API do ObjectGridManager, utilize um dos métodos createObjectGrid para criar uma instância local ou o método getObjectGrid para uma instância de cliente com uma grade distribuída.

Um encadeamento em um aplicativo precisa de sua própria Sessão. Quando um aplicativo deseja usar o ObjectGrid em um encadeamento, ele deve simplesmente chamar um dos métodos getSession para obter um encadeamento. Esta operação é barata--não há necessidade de agrupar essas operações na maioria dos casos. Se o aplicativo estiver usando uma estrutura de injeção independente como Spring, é possível injetar uma Sessão em um bean de aplicativo quando necessário.

Após obter uma Sessão, o aplicativo pode acessar dados armazenados em mapas no ObjectGrid. Se o ObjectGrid usar entidades, é possível usar a API de EntityManager, que pode ser obtida com o método Session.getEntityManager. Porque ele está mais próxima das especificações Java, a interface EntityManager é mais simples que a API baseada em mapa. Porém, a API de EntityManager transporta um gasto adicional de desempenho porque controla as alterações em objetos. A API baseada em mapa é obtida usando o método Session.getMap.

O WebSphere eXtreme Scale usa transações. Quando um aplicativo interage com Session, ele deve estar no contexto de uma transação. Uma transação é iniciada e consolidade ou retrocedida usando os métodos Session.begin, Session.commit e

Session.rollback no objeto Sessão. Os aplicativos também podem funcionar no modo auto-commit, no qual Session inicia automaticamente e executa o commit de uma transação sempre que o aplicativo interagem com Mapas. Entretanto, o modo de auto-consolidação é mais lento.

A Lógica de Utilização de Transações

As transações podem parecer lentas, mas o eXtreme Scale usa transações por três motivos:

1. Para permitir o retrocesso das alterações se uma exceção ocorrer ou se a lógica de negócios precisar desfazer alterações de estado.
2. Para manter bloqueios nos dados e liberar bloqueios dentro do ciclo de vida de uma transação, permitindo que um conjunto de alterações seja feito atomicamente, ou seja, todas as alterações ou nenhuma alteração nos dados.
3. Para produzir uma unidade atômica de replicação.

O WebSphere eXtreme Scale permite que uma Sessão customize quanta transação realmente é necessária. Um aplicativo pode desligar o suporte à recuperação e bloquear, mas com um custo para o aplicativo. O aplicativo deve manipular a falta desses recursos.

Por exemplo, um aplicativo pode desligar o bloqueio configurando a estratégia de bloqueio de BackingMap para que seja NONE. Esta estratégia é rápida, mas transações simultâneas agora podem modificar os mesmos dados sem nenhuma proteção uma da outra. O aplicativo é responsável por todos os bloqueios e consistências de dados quando NONE é utilizado.

Um aplicativo também pode alterar a maneira como os objetos são copiados quando acessados pela transação. O aplicativo pode especificar como os objetos são copiados com o método ObjectMap.setCopyMode. Com este método, é possível desligar CopyMode. Desligar CopyMode normalmente é usado para transações somente de leitura se diferentes valores podem ser retornados para o mesmo objeto dentro de uma transação. Valores diferentes podem ser retornados para o mesmo objeto dentro de uma transação.

Por exemplo, se a transação chamou o método ObjectMap.get para o objeto em T1, ela obtém o valor naquele ponto no tempo. Se ela chamar o método get novamente dentro dessa transação em um tempo posterior T2, outro encadeamento pode ter alterado o valor. Porque o valor foi alterado por outro encadeamento, o aplicativo vê um valor diferente. Se o aplicativo modifica um objeto recuperado usando um valor de CopyMode NONE, ele está alterando a cópia consolidada desse objeto diretamente. A recuperação da transação não faz sentido neste modo. Você está alterando a única cópia no ObjectGrid. Apesar do uso de CopyMode NONE ser rápido, esteja ciente de suas consequências. Um aplicativo que usa o CopyMode NONE nunca deve retroceder a transação. Se o aplicativo retroceder a transação, os índices não são atualizados com as alterações *e* as alterações não são replicadas se a replicação estiver desativada. Os valores padrão são fáceis de usar e menos propensos a erros. Se você começar a trocar desempenho por dados menos confiáveis, o aplicativo precisará estar ciente do que está fazendo para evitar problemas indesejados.

CUIDADO:
Tome cuidado ao alterar os valores de bloqueio ou de CopyMode. Se você alterar os valores, um comportamento imprevisível do aplicativo pode ocorrer.

Interagindo com Dados Armazenados

Após a obtenção de uma sessão, é possível usar o seguinte fragmento de código para usar a API do Mapa para inserir dados.

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

O mesmo exemplo usando a API do EntityManager está a seguir. Esta amostra de código supõe que o objeto Pessoal é mapeado para uma Entidade.

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

O padrão é projetado para obter referências aos ObjectMaps para os Mapas com os quais o encadeamento trabalhará, iniciará uma transação, trabalhará com os dados, e depois consolidará a transação.

A interface ObjectMap tem as operações de Mapa comuns, como put, get e remove. Porém, use os nomes de operação mais específicos como: get, getForUpdate, insert, update e remove. Esses nomes de métodos expressam o intento mais precisamente que as APIs de Mapa tradicionais.

Também é possível usar o suporte à indexação, que é flexível.

A seguir está um exemplo para atualização de um 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 o aplicativo usa o método getForUpdate em vez de um get simples para bloquear o registro. O método update deve ser chamado para fornecer o valor atualizado para o mapa. Se o método update não for chamado, então o mapa não é alterado. A seguir está o mesmo fragmento usando a API de EntityManager:

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

A API de EntityManager é mais simples que a abordagem de Mapa. Neste caso, o eXtreme Scale localiza a Entidade e retorna um objeto gerenciado para o aplicativo. O aplicativo modifica o objeto e consolida a transação, e o eXtreme Scale controla as alterações para objetos gerenciados automaticamente no tempo de consolidação e executa as atualizações necessárias.

Transações e Partições

As transações do WebSphere eXtreme Scale somente podem ser atualizadas em uma partição única. As transações de um cliente podem ler de múltiplas partições, mas somente podem atualizar uma partição. Se um aplicativo tentar atualizar duas partições, então a transação falha e é retrocedida. Uma transação que está usando um ObjectGrid (lógica da grade) integrado não tem capacidade de roteamento e somente pode ver dados na partição local. Esta lógica de negócios sempre pode obter uma segunda sessão que é uma sessão do cliente true para acessar outras partições. Porém, esta transação seria uma transação independente.

Consultas e Partições

Se uma transação já buscou por uma Entidade, a transação é associada com a partição para essa Entidade. Quaisquer consultas que executam em uma transação que está associada com uma Entidade são roteadas para a partição associada.

Se uma consulta é executada em uma transação antes de ser associada com uma partição, você deve configurar o ID da partição a ser usado para a consulta. O ID da partição é um valor de número inteiro. A consulta é, então, roteada para essa partição.

As consultas pesquisam somente dentro de uma única partição. Porém, é possível usar as PIs do DataGrid para executar a mesma consulta em paralelo em todas as partições ou em um subconjunto de partições. Use as APIs do DataGrid para localizar uma entrada que podem estar em qualquer partição.

O serviço de dados REST permite que qualquer cliente HTTP acesse uma grade do WebSphere eXtreme Scale, além de ser compatível com WCF Data Services no Microsoft .NET Framework 3.5 SP1. Para obter mais informações, consulte o guia do usuário para o Serviço de Dados REST do eXtreme Scale

.

Transações:

As transações possuem muitas vantagens para o armazenamento e a manipulação de dados. É possível usar as transações para proteger a grade de dados contra mudanças simultâneas, aplicar várias mudanças como uma unidade simultânea, replicar dados e implementar um ciclo de vida para bloqueios nas mudanças.

Quando uma transação inicia, o WebSphere eXtreme Scale aloca um mapa de diferença especial para conter as alterações atuais ou cópias dos pares chave e valor que a transação utiliza. Normalmente, quando um par de chave e valor é acessado, o valor é copiado antes de o aplicativo receber o valor. O mapa de diferenças controla todas as alterações de operações, como inserir, atualizar, obter, remover e assim por diante. As chaves não são copiadas porque elas são assumidas como imutáveis. Se um objeto ObjectTransformer for especificado, então, ele será utilizado para copiar o valor. Se a transação estiver utilizando o bloqueio optimistic, as imagens anteriores dos valores também serão rastreadas para comparação quando a transação for confirmada.

Se uma transação for recuperada, as informações do mapa de diferenças serão descartadas e os bloqueios nas entradas serão liberados. Quando uma transação é consolidada, as alterações são aplicadas nos mapas e os bloqueios são liberados. Se o bloqueio otimista estiver sendo utilizado, o eXtreme Scale compara as versões de

imagens anteriores dos valores com os valores que estão no mapa. Esses valores devem corresponder para que a transação seja confirmada. Essa comparação permite um esquema de bloqueio de várias versões, mas a um custo de duas cópias sendo feitas quando a transação acessa a entrada. Todos os valores são copiados novamente e a nova cópia é armazenada no mapa. O WebSphere eXtreme Scale executa esta cópia para se proteger do aplicativo alterando a referência do aplicativo para o valor após um commit.

É possível evitar o uso de diversas cópias das informações. O aplicativo pode salvar uma cópia, utilizando o bloqueio pessimistic em vez do bloqueio optimistic como o custo da limitação da simultaneidade. A cópia do valor no momento da confirmação também pode ser evitada se o aplicativo concordar em não alterar um valor após uma confirmação.

Vantagens das Transações

Utilize as transações pelos seguintes motivos:

Usando as transações, você pode:

- Recuperar alterações se ocorrer uma exceção ou a lógica de negócios precisar desfazer mudanças de estado.
- Para aplicar várias alterações como uma unidade atômica no momento commit.
- Mantém e libera bloqueios em dados para aplicar múltiplas alterações como uma unidade atômica no momento da consolidação.
- Protege um encadeamento de alterações concorrentes.
- Implementa um ciclo de vida para bloqueios nas alterações.
- Produz uma unidade atômica de replicação.

Tamanho da Transação

Transações maiores são mais eficientes, especificamente para replicação. No entanto, as transações maiores podem causar impacto adverso na simultaneidade porque os bloqueios nas entradas são retidos por um período maior de tempo. Se você usar transações maiores, é possível aumentar o desempenho de replicação. Este aumento de desempenho é importante quando você estiver pré-carregando um Mapa. Experimente diferentes tipos de batch para determinar qual funciona melhor para o seu cenário.

Transações maiores também ajudam com os utilitários de carga. Se estiver sendo usado um utilitário de carga que possa executar SQL em lote, então ganhos consideráveis no desempenho são possíveis dependendo da transação e de reduções significativas de carga no lado do banco de dados. Esse ganho no desempenho depende da implementação do Carregador.

Modo de Commit Automático

Se nenhuma transação for ativamente iniciada, então quando um aplicativo interage com um objeto ObjectMap, uma operação automática é iniciada e uma consolidação é executada em nome do aplicativo. Esta operação automática de início e consolidação funciona, mas evita que a recuperação e o bloqueio funcionem efetivamente. A velocidade de replicação síncrona sofre um impacto devido ao tamanho de transação muito reduzido. Se estiver usando um aplicativo gerenciador de entidades, então não use o modo de consolidação automática pois os objetos que estiverem bloqueados com o método EntityManager.find se tornarão

imediatamente não gerenciados no retorno do método e inutilizáveis.

Coordenadores de Transação Externos

Normalmente, as transações iniciam com o método session.begin e terminam com o método session.commit. Porém, quando o eXtreme Scale está incorporado, as transações podem ser iniciadas e encerradas por um coordenador externo de transações. Se você estiver usando um coordenador de transação externo, não é necessário chamar o método session.begin e terminar com o método session.commit. Se você estiver usando o WebSphere Application Server, é possível usar o plug-in WebSphereTranscationCallback.

Atributo CopyMode:

É possível ajustar o número de cópias ao definir o atributo CopyMode do BackingMap ou objetos ObjectMap no arquivo XML do descritor do ObjectGrid.

É possível ajustar o número de cópias definindo o atributo CopyMode do BackingMap ou objetos ObjectMap. O modo de cópia possui os seguintes valores:

- COPY_ON_READ_AND_COMMIT
- COPY_ON_READ
- NO_COPY
- COPY_ON_WRITE
- COPY_TO_BYTES
- COPY_TO_BYTES_RAW

O valor COPY_ON_READ_AND_COMMIT é o padrão. O valor COPY_ON_READ copia os dados iniciais recuperados, mas não copia no momento da consolidação. Este modo é seguro se o aplicativo não modificar um valor depois de consolidar uma transação. O valor NO_COPY não copia os dados, que são seguros apenas para dados de leitura. Se ele nunca for alterado, não será necessário copiá-lo por motivos de isolamento.

Seja cauteloso ao usar o valor do atributo NO_COPY com mapas que possam ser atualizados. O WebSphere eXtreme Scale utiliza a cópia no primeiro acesso para permitir o retrocesso da transação. O aplicativo alterou apenas a cópia e, como resultado, o eXtreme Scale descarta a cópia. Se o valor de atributo NO_COPY for utilizado e o aplicativo modificar o valor confirmado, não será possível concluir a recuperação. Modificar o valor confirmado conduz a problemas nos índices, replicação e assim por diante porque os índices e as réplicas são atualizadas quando a transação é confirmada. Se você modificar os dados confirmados e, em seguida, recuperar a transação, o que na realidade não é recuperada, os índices não serão atualizados e a replicação não ocorrerá. Os outros encadeamentos podem ver as alterações não confirmadas imediatamente, mesmo se tiverem bloqueios. Utilize o valor de atributo NO_COPY apenas para mapas somente leitura ou para aplicativos que concluem a cópia apropriada antes de modificar o valor. Se você utilizar o valor de atributo NO_COPY e chamar o suporte IBM com um problema de integridade de dados, será necessário reproduzir o problema com o modo de cópia definido como COPY_ON_READ_AND_COMMIT.

O valor COPY_TO_BYTES armazena os valores no mapa de maneira serializada. No tempo de leitura, o eXtreme Scale aumenta o valor de um formato serializado e, no tempo de consolidação, ele armazena o valor em um formato serializado. Com esse método, uma cópia é feita no tempo de leitura e de consolidação.

O modo de cópia padrão para um mapa pode ser configurado no objeto BackingMap. Também é possível alterar o modo de cópia antes de iniciar uma transação usando o método ObjectMap.setCopyMode.

A seguir há um exemplo de um fragmento de mapa de apoio de um arquivo `objectgrid.xml` que mostra como configurar o modo de cópia para um determinado mapa de apoio. Este exemplo assume que você esteja utilizando `cc` como o espaço de nomes `objectgrid/config`.

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

Referências relacionadas

Arquivo XML descritor do ObjectGrid
Para configurar o WebSphere eXtreme Scale, utilize um arquivo XML descritor do ObjectGrid e a API do ObjectGrid.

Gerenciador de Bloqueio:

Quando configurar uma estratégia de bloqueio, um gerenciador de bloqueio é criado para o mapa de apoio para manter a consistência da entrada de cache.

Configuração do Gerenciador de Bloqueios

Quando a estratégia de bloqueio pessimista ou otimista for utilizada, será criado um gerenciador de bloqueios para o BackingMap. O gerenciador de bloqueios utiliza um mapa hash para controlar entradas bloqueadas por uma ou mais transações. Se existirem muitas entradas de mapa no mapa hash, mais depósitos de bloqueio podem resultar em melhor desempenho. O risco de colisões de sincronização Java é inferior conforme a quantidade de depósitos aumenta. Mais depósitos de bloqueios também resultam em maior simultaneidade. Os exemplos anteriores mostram como um aplicativo pode configurar o número de depósitos de bloqueio para utilizar para uma determinada instância de BackingMap.

Para evitar uma exceção java.lang.IllegalStateException, o método setNumberOfLockBuckets deve ser chamado antes de chamar os métodos initialize ou getSession na instância do ObjectGrid. O parâmetro do método setNumberOfLockBuckets é um inteiro primitivo Java que especifica a quantidade de depósitos de bloqueio para uso. Utilizar um número primo permite uma distribuição uniforme de entradas do mapa sobre os depósitos de bloqueios. Um bom ponto de partida para obter melhor desempenho é configurar o número de depósitos de bloqueios para aproximadamente dez por cento do número esperado de entradas do BackingMap.

Estratégias de Bloqueio:

As estratégias de bloqueio incluem pessimista, otimista e nenhum. Para escolher uma estratégia de bloqueio, é necessário considerar questões como a porcentagem de cada tipo de operações que você tem, se você utiliza um utilitário de carga, entre outras.

Os bloqueios são limitados pelas transações. É possível especificar as seguintes configurações de bloqueio:

- **Ausência de bloqueio**: Executar sem a configuração de bloqueio é a opção mais rápida. Se estiver utilizando dados de leitura, você talvez não precise do bloqueio.

- **Bloqueio pessimistic**: Adquire bloqueios em entrada e, em seguida, contém o bloqueio até o momento do commit. Essa estratégia de bloqueio fornece boa consistência à custa do rendimento.
- **Bloqueio optimistic**: Obtém uma imagem anterior de cada registro que a transação acessa e compara a imagem com os valores de entrada atuais quando ocorre o commit da transação. Se os valores de entrada forem alterados, a transação será recuperada. Nenhum bloqueio será retido até o momento da confirmação. Esta estratégia de bloqueio fornece melhor simultaneidade do que a estratégia pessimista, no risco da transação sendo recuperada e no custo da memória de criar a cópia extra da entrada.

Configure a estratégia de bloqueio no BackingMap. Não é possível alterar a estratégia de bloqueio para cada transação. A seguir há um exemplo de fragmento XML que mostra como configurar o modo de bloqueio em um mapa utilizando o arquivo XML, assumindo que `cc` é o espaço de nomes para o espaço de nomes `objectgrid/config`.

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Bloqueio Pessimista

Use a estratégia de bloqueio pessimista para ler e gravar mapas quando outras estratégias de bloqueio não foram possíveis. Quando um mapa do ObjectGrid map é configurado para utilizar a estratégia de bloqueio pessimista, um bloqueio de transação pessimista para uma entrada de mapa é obtido quando uma transação primeiro obtém a entrada do BackingMap. O bloqueio pessimistic fica retido até que o aplicativo conclua a transação. Geralmente, a estratégia de bloqueio pessimistic é utilizada nas seguintes situações:

- Quando o BackingMap é configurado com ou sem um utilitário de carga e as informações de controle de versões não estão disponíveis.
- Quando o BackingMap é utilizado diretamente por um aplicativo que precisa de ajuda do eXtreme Scale para controle de simultaneidade.
- Quando as informações de controle de versões estão disponíveis, mas as transações de atualização colidem freqüentemente nas entradas de suporte, resultando em falhas de atualização otimistas.

Como a estratégia de bloqueio pessimista tem o maior impacto no desempenho e escalabilidade, esta estratégia deve ser utilizada apenas para ler e gravar mapas quando outras estratégias de bloqueio não são viáveis. Por exemplo, essas situações incluem quando ocorrem falhas de atualização otimista com frequência ou quando a recuperação da falha otimista é difícil para um aplicativo manipular.

Bloqueio Otimista

A estratégia de bloqueio otimista assume que nenhuma transação em execução simultânea pode tentar atualizar a mesma entrada de mapa. Devido a esta convicção, o modo de bloqueio não precisa ser retido pelo ciclo de vida da transação porque é improvável que mais de uma transação possa atualizar a entrada de mapa simultaneamente. A estratégia de bloqueio optimistic geralmente é utilizada nas seguintes situações:

- Quando um BackingMap é configurado com ou sem um utilitário de carga e as informações de controle de versões estão disponíveis.
- Quando um BackingMap possui em sua maior parte, transações que executam operações de leitura. As operações insert, update ou remove nas entradas de mapa não ocorrem com frequência no BackingMap.

- Quando um BackingMap é inserido, atualizado ou removido mais freqüentemente do que é lido, mas as transações raramente colidem na mesma entrada do mapa.

Como a estratégia de bloqueio pessimistic, o métodos na interface ObjectMap determinam como o eXtreme Scale automaticamente tenta adquirir um modo de bloqueio para a entrada de mapa que está sendo acessada. Entretanto, existem as seguintes diferenças entre as estratégias pessimistic e optimistic:

- Como a estratégia de bloqueio pessimistic, um modo de bloqueio S é adquirido pelos métodos get e getAll quando o método é chamado. No entanto, com o bloqueio optimistic, o modo de bloqueio S não fica retido até que a transação seja concluída. Em vez disso, o modo de bloqueio S é liberado antes de o método retornar ao aplicativo. O propósito de adquirir o modo de bloqueio é para que o eXtreme Scale possa garantir que apenas dados com commit de outras transações fiquem visíveis para a transação atual. Após o eXtreme Scale ter verificado que ocorreu commit nos dados, o modo de bloqueio S é liberado. No momento do commit, uma verificação de versão optimistic é executada para garantir que nenhuma outra transação tenha alterado a entrada do mapa após a transação atual ter liberado seu modo de bloqueio S. Se uma entrada não for procurada a partir do mapa antes de ser atualizada, invalidada ou excluída, o tempo de execução do eXtreme Scale implicitamente procura a entrada a partir do mapa. Esta operação get implícita é desempenhada para obter o valor atual no momento em que foi solicitada a modificação da entrada.
- Diferente da estratégia de bloqueio pessimista, os métodos getForUpdate e getAllForUpdate são tratados exatamente como os métodos get e getAll quando a estratégia de bloqueio otimista é utilizada. Ou seja, um modo de bloqueio S é adquirido no início do método e o modo de bloqueio S é liberado antes de retornar para o aplicativo.

Todos os outros métodos ObjectMap são tratados exatamente como são tratados para a estratégia de bloqueio pessimistic. Ou seja, quando o método commit é chamado, um modo de bloqueio X é obtido para qualquer entrada do mapa que tenha sido inserida, atualizada, removida, tocada ou invalidada e o modo de bloqueio X é retido até que a transação tenha concluído o processamento de consolidação.

A estratégia de bloqueio optimistic assume que nenhuma transação em execução simultânea tenta atualizar a mesma entrada de mapa. Devido a esta suposição, o modo de bloqueio não precisa ser mantido pela duração da transação porque é improvável que mais de uma transação possa atualizar a entrada de mapa simultaneamente. Entretanto, como um modo de bloqueio não foi mantido, outra transação simultânea poderia potencialmente atualizar a entrada do mapa após a transação atual ter liberado seu modo de bloqueio S.

Para tratar esta possibilidade, o eXtreme Scale obtém um bloqueio X no momento do commit e executa uma verificação de versão optimistic para verificar se nenhuma outra transação alterou a entrada do mapa após a transação atual ter lido a entrada do mapa a partir do BackingMap. Se outra transação alterar a entrada do mapa, a verificação de versão falhará e ocorrerá uma exceção OptimisticCollisionException. Esta exceção força a transação atual a ser retrocedida e o aplicativo deve tentar novamente a transação inteira. A estratégia de bloqueio optimistic é muito útil quando um mapa é lido em sua maior parte e é improvável que ocorram atualizações na mesma entrada do mapa.

Ausência de Bloqueio

Quando um BackingMap é configurado para usar nenhuma estratégia de bloqueio, nenhum bloqueio de transação para uma entrada de mapa é obtido.

Não utilizar uma estratégia de bloqueio é útil quando um aplicativo é um gerenciador de persistência, como um contêiner Enterprise JavaBeans (EJB) ou quando um aplicativo utiliza Hibernate para obter dados persistentes. Neste cenário, o BackingMap é configurado sem um utilitário de carga e o gerenciador de persistência utiliza o BackingMap como um cache de dados. Neste cenário, o gerenciador de persistência fornece controle de simultaneidade entre transações que estão acessando as mesmas entradas de Mapa.

O WebSphere eXtreme Scale não precisa obter nenhum bloqueio de transação para o propósito de controle de simultaneidade. Essa situação presume que o gerenciador de persistência não libera os bloqueios da transação antes de atualizar o mapa ObjectGrid com as alterações confirmadas. Se o gerenciador de persistência libera seus bloqueios, então uma estratégia de bloqueio pessimistic ou optimistic deve ser utilizada. Por exemplo, suponha que o gerenciador de persistência de um contêiner EJB esteja atualizando o mapa do ObjectGrid com dados que foram confirmados na transação gerenciada por contêiner de EJB. Se a atualização do mapa do ObjectGrid ocorrer antes dos bloqueios de transação do gerenciador de persistência serem liberados, então é possível não utilizar nenhuma estratégia de bloqueio. Se o mapa do ObjectGrid ocorrer após os bloqueios de transação do gerenciador de persistência serem liberados, será necessário utilizar a estratégia de bloqueio otimista ou pessimista.

Outro cenário onde a ausência de estratégia de bloqueio pode ser utilizada é quando o aplicativo utiliza um BackingMap diretamente e um Utilitário de Carga é configurado para o mapa. Neste cenário, o utilitário de carga utiliza o suporte de controle de simultaneidade que é fornecido por um Relational Database Management System (RDBMS) utilizando Java Database Connectivity (JDBC) ou Hibernate para acessar dados em um banco de dados relacional. A implementação do utilitário de carga pode utilizar uma abordagem optimistic ou pessimistic. Um utilitário de carga que utiliza um bloqueio optimistic ou uma abordagem de controle de versões ajuda a obter a maior quantidade de simultaneidade e desempenho. Para obter mais informações sobre como implementar uma abordagem de bloqueio otimista, consulte a seção OptimisticCallback em informações sobre as considerações do carregador no *Guia de Administração*. Se estiver usando um utilitário de carga que usa suporte de bloqueio pessimistic de um backend subjacente, é possível querer usar o parâmetro forUpdate que é transmitido no método get da interface do utilitário de carga. Configure este parâmetro como true se o método getForUpdate da interface ObjectMap foi utilizado pelo aplicativo para obter os dados. O utilitário de carga pode utilizar esse parâmetro para determinar se solicitará um bloqueio atualizável na linha que está sendo lida. Por exemplo, o DB2 obtém um bloqueio atualizável quando uma instrução select SQL contém uma cláusula `FOR UPDATE`. Esta abordagem oferece a mesma prevenção de conflito que está descrita em “Bloqueio Pessimista” na página 236.

Distribuindo Transações:

Use Java Message Service (JMS) para mudanças de transação distribuída entre diferentes camadas ou em ambientes em plataformas mistas.

O JMS é um protocolo ideal para alterações distribuídas entre diferentes camadas ou em ambientes em plataformas mistas. Por exemplo, alguns aplicativos que usam o eXtreme Scale podem ser implementados noIBM WebSphere Application Server Community Edition, Apache Geronimo ou Apache Tomcat, considerando que outros aplicativos podem executar no WebSphere Application Server Versão 6.x. O JMS é ideal para alterações distribuídas entre peers do eXtreme Scale nesses diferentes ambientes. O transporte de mensagens do gerenciador de alta disponibilidade é muito rápido, mas pode apenas distribuir alterações para as Java Virtual Machines que estão em um grupo principal único. O JMS é mais lento, mas permite que conjuntos maiores e mais diversos de aplicativos clientes compartilhem um ObjectGrid. O JMS é ideal no compartilhamento de dados em um ObjectGrid entre um cliente Swing rápido e um aplicativo implementado no WebSphere Extended Deployment.

O Mecanismo de Invalidação do Cliente e a Replicação Ponto a Ponto incorporados são exemplos da distribuição de alterações transacionais com base no JMS. Consulte o informações sobre como configurar a replicação ponto a ponto com o JMS no *Guia de Administração* para obter mais informações.

Implementando o JMS

O JMS é implementado para distribuir alterações de transação usando um objeto Java que se comporta como um ObjectGridEventListener. Este objeto pode propagar o estado nas quatro maneiras a seguir:

1. Invalidar: Qualquer entrada que é despejada, atualizada ou excluída é removida em todas as Java Virtual Machines peer quando elas recebem a mensagem.
2. Invalidar condicional: A entrada é despejada somente se a versão local for a mesma ou mais antiga que a versão no publicador.
3. Push: Qualquer entrada que foi despejada, atualizada, excluída ou inserida é incluída ou sobrescrita em todas as Java Virtual Machines peer quando elas recebem a mensagem JMS.
4. Push condicional: A entrada é atualizada ou incluída no lado de recebimento apenas se a entrada local for menos recente que a versão que está sendo publicada.

Atender Alterações de Publicação

O plug-in implementa a interface ObjectGridEventListener para interceptar o evento transactionEnd. Quando o eXtreme Scale chama este método, o plug-in tenta converter a lista LogSequence list para cada mapa que é acessado pela transação em uma mensagem JMS e, então, a publica. O plug-in pode ser configurado para publicar alterações para todos os mapas ou um subconjunto de mapas. Os objetos LogSequence são processados para os mapas com a publicação ativada. A classe LogSequenceTransformer do ObjectGrid serializa um LogSequence filtrado para cada mapa em um fluxo. Após todas as LogSequences serem serializadas para o fluxo, então, um ObjectMessage JMS é criado e publicado em um tópico bem conhecido.

Atender Mensagens JMS e Aplicá-las ao ObjectGrid Local

O mesmo plug-in também inicia um encadeamento que gira em um loop, recebendo todas as mensagens publicadas no tópico bem conhecido. Quando chega uma mensagem, ele transmite o conteúdo da mensagem para a classe LogSequenceTransformer para convertê-lo em um conjunto de objetos

LogSequence. Em seguida, uma transação não-write-through é iniciada. Cada objeto LogSequence é fornecido ao método Session.processLogSequence, que atualiza os Mapas locais com as alterações. O método processLogSequence entende o modo de distribuição. A transação é confirmada e o cache local agora reflete as alterações. Para obter mais informações sobre como usar o JMS para distribuir as mudanças transacionais, consulte o informações sobre como distribuir as mudanças entre as Java Virtual Machines equivalentes no *Guia de Administração*.

Transações de Partição Única e entre Grade de Dados:

A maior diferença entre as soluções do WebSphere eXtreme Scale e de armazenamento de dados tradicional, como bancos de dados relacionais ou bancos de dados em memória, é o uso do particionamento, que permite que o cache seja escalado de maneira linear. Os tipos de transações importantes a serem considerados são transações de partição única e de cada partição (grade de dados cruzada).

Em geral, as interações com o cache podem ser categorizadas como transações de partição única ou transações de grade de dados cruzada, conforme abordado na seguinte seção.

Transações de Partição Única

As transações de partição única são o método preferido para interagir com os caches que são hospedados pelo WebSphere eXtreme Scale. Quando uma transação é limitada a uma única partição, por padrão, ela é limitada a uma única Java Virtual Machine e, portanto, a um único computador de servidor. Um servidor pode executar *M* número dessas transações por segundo e, se você tiver *N* computadores, poderá executar `M*N` transações por segundo. Se os negócios aumentarem e você precisar executar o dobro dessas transações por segundo, poderá dobrar *N* ao adquirir mais computadores. Em seguida, é possível atender as demandas de capacidade sem alterar o aplicativo, fazer upgrade de hardware ou até mesmo usar o aplicativo off-line.

Além de permitir que o cache seja escalado de maneira significativa, as transações de partição única também maximizam a disponibilidade do cache. Cada transação depende apenas de um computador. Qualquer um dos outros `(N-1)` computadores podem falhar sem afetar o sucesso ou o tempo de resposta da transação. Assim, se você estiver executando 100 computadores e um deles falhar, apenas 1% das transações em andamento no momento em que esse servidor falhou é recuperado. Depois que o servidor falhar, o WebSphere eXtreme Scale relocará as partições que são hospedadas pelo servidor com falha nos outros 99 computadores. Durante esse breve período, antes de concluir a operação, os outros 99 computadores ainda poderão concluir as transações. Apenas as transações que envolveriam as partições que estão sendo relocadas são bloqueadas. Depois que o processo de failover ser concluído, o cache poderá continuar executando, totalmente operacional, a 99% de sua capacidade de rendimento original. Depois que o servidor com falha for substituído e retornado para a grade de dados, o cache voltará para 100% da capacidade de rendimento.

Transações da Grade de Dados Cruzada

Em termos de desempenho, disponibilidade e escalabilidade, as transações de grade de dados cruzada são o oposto de transações de partição única. As transações de grade de dados cruzada acessam cada partição e, portanto, cada computador na configuração. Cada computador na grade de dados é instruído a

procurar alguns dados e retornar o resultado. A transação não pode ser concluída até que cada computador tenha respondido e, dessa forma, o rendimento da grade de dados inteira ficará limitado em função do computador mais lento. Incluir computadores não agiliza o computador mais lento e, assim, não melhora o rendimento do cache.

As transações de grade de dados cruzada possuem um efeito semelhante em termos de disponibilidade. Estendendo o exemplo anterior, se você estiver executando 100 servidores e um deles falhar, então, 100% das transações que estão em andamento no momento em que esse servidor falhou será recuperado. Depois que o servidor falhar, o WebSphere eXtreme Scale relocará as partições que são hospedadas por esse servidor nos outros 99 computadores. Durante esse tempo, antes de o processo de failover ser concluído, a grade de dados não pode processar nenhuma dessas transações. Depois que o processo de failover ser concluído, o cache poderá continuar executando, porém com capacidade reduzida. Se cada computador na grade de dados atender 10 partições, então 10 dos 99 computadores restantes receberão, pelo menos, uma partição extra como parte do processo de failover. Incluir uma partição extra aumenta a carga de trabalho desse computador em pelo menos 10%. Como o rendimento da grade de dados é limitado ao rendimento do computador mais lento em uma transação de grade de dados cruzada, em média, o rendimento é reduzido em 10%.

As transações de partição única são preferidas para as transações de grade de dados cruzada para efetuar scale out com um cache de objeto distribuído e altamente disponível, como o WebSphere eXtreme Scale. Aumentar o desempenho desses tipos de sistemas requer o uso de técnicas que são diferentes das metodologias relacionais tradicionais, porém é possível transformar as transações de grade de dados cruzada em transações de partição única escalável.

Boas práticas para criar modelos de dados escaláveis

As boas práticas para construir aplicativos escaláveis com produtos como o WebSphere eXtreme Scale incluem duas categorias: princípios básicos e dicas de implementação. Os princípios básicos são ideias principais que precisam ser capturadas no projeto dos próprios dados. Um aplicativo que não observa esses princípios podem não ser escalados tão bem, mesmo para as transações de linha principal. Por outro lado, as dicas de implementação são aplicadas em transações problemáticas em um aplicativo bem projetado que observa os princípios gerais para modelos de dados escaláveis.

Princípios Básicos

Algumas das maneiras importantes de otimizar a escalabilidade são conceitos ou princípios básicos que devem ser mantidos em mente.

Duplicar em vez de normalizar

O que mais deve-se ter em mente sobre os produtos como o WebSphere eXtreme Scale é que eles são designados para propagar dados entre um grande número de computadores. Se o objetivo é concluir a maioria ou todas as transações em uma única partição, o design do modelo de dados precisa garantir que todos os dados que a transação possa precisar estejam localizados na partição. Na maioria das vezes, a única maneira de fazer isso é duplicar os dados.

Por exemplo, considere um aplicativo, como um quadro de avisos. Duas transações muito importantes para um quadro de mensagens mostram

todas as postagens de um determinado usuário e todas as postagens de um determinado tópico. Primeiro considere como essas transações trabalhariam com um modelo de dados normalizado que contenha um registro de usuário, um registro de tópico e um registro de postagem que contenha o texto real. Se as postagens forem particionadas com os registros do usuário, a exibição do tópico torna-se uma transação de grade cruzada e vice-versa. Os tópicos e os usuários não podem ser particionados juntos porque eles possuem um relacionamento muitos-para-muitos.

A melhor maneira de fazer com que esse quadro de avisos seja escalável é duplicar as postagens, armazenar uma cópia com o registro de tópico e uma cópia com o registro do usuário. Em seguida, a exibição das postagens de um usuário é uma transação de partição única, exibir as postagens em um tópico é uma transação de partição única e atualizar ou excluir uma postagem é uma transação de duas partições. Todas essas três transações serão escaladas de maneira linear já que o número de computadores na grade de dados aumenta.

Escalabilidade Em Vez de Recursos

O maior obstáculo a ser superado ao considerar os modelos de dados não-normalizados é o impacto que esse modelos causam nos recursos. Manter duas, três ou mais cópias de alguns dados pode parecer que muitos recursos usados são práticos. Ao se deparar com esse cenário, lembre-se dos seguintes fatos: os recursos de hardware se tornam mais baratos a cada dia. Segundo e o mais importante, o WebSphere eXtreme Scale elimina a maioria dos custos implícitos associados à implementação de mais recursos.

Os recursos devem ser medidos em termos de custo em vez de computador, como megabytes e processadores. Os armazenamentos de dados que trabalham com dados relacionais normalizados geralmente precisam estar localizados no mesmo computador. Essa colocação necessária significa que um único computador corporativo maior precisa ser adquirido em vez de vários computadores menores. Com o hardware corporativo, um computador que executa um milhão de transações por segundo normalmente é bem mais barato que 10 computadores capazes de executar 100 mil transações por segundo cada um.

Incluir recursos também gera custos de negócios. Um negócio em crescimento normalmente pode ficar sem capacidade. Quando não houver capacidade, é necessário encerrar para mudar para um computador maior e mais rápido ou é necessário criar um segundo ambiente de produção para o qual você possa mudar. De uma das formas, custos adicionais serão acarretados na forma de negócios perdidos ou ao manter quase o dobro da capacidade necessária durante o período de transação.

Com o WebSphere eXtreme Scale, o aplicativo não precisa ser encerrado para incluir capacidade. Se seus projetos de negócios requererem 10% de capacidade a mais para o próximo ano, aumente 10% o número de computadores na grade de dados. É possível aumentar essa porcentagem sem ocorrer tempo de inatividade do aplicativo e sem adquirir capacidade em excesso.

Evitar transformações de dados

Quando estiver usando o WebSphere eXtreme Scale, os dados deverão ser armazenados em um formato que possa ser consumado diretamente pela lógica de negócios. Dividir os dados em um formato mais primitivo gera custos. A transformação precisa ser feita quando os dados forem gravados

e lidos. Com os bancos de dados relacionais, essa transformação é feita sem necessidade porque os dados são definitivamente persistidos no disco muito frequentemente, mas com o WebSphere eXtreme Scale, essas transformações não precisam ser executadas. Na maioria das vezes os dados são armazenados na memória e podem, portanto, serem armazenados no formato exato em que o aplicativo precisa.

Observar essa regra de amostra ajuda a desnormalizar os dados de acordo com o primeiro princípio. O tipo mais comum de transformação dos dados de negócios é as operações JOIN que são necessárias para tornar os dados normalizados em um conjunto de resultados que se ajusta às necessidades do aplicativo. Armazenar os dados no formato correto implicitamente evita a execução dessas operações JOIN e produz um modelo de dados não-normalizado.

Eliminar consultas ilimitadas

Independente de como os seus dados são estruturados, as consultas ilimitadas não são bem escaladas. Por exemplo, não tenha uma transação que solicite uma lista de todos os itens classificados por valor. Essa transação pode funcionar inicialmente quando o número total de itens for 1.000, mas quando o número total de itens chegar a 10 milhões, a transação retornará todos os 10 milhões de itens. Se você executar essa transação, os dois resultados mais prováveis são o tempo limite da transação se esgotar ou o cliente receber um erro de falta de memória.

A melhor opção é alterar a lógica de negócios para que apenas os 10 ou 20 itens principais possam ser retornados. Essa mudança de lógica mantém o tamanho da transação gerenciável, independente de quantos itens estão no cache.

Definir esquema

A principal vantagem de normalizar os dados é que o sistema de banco de dados controla a consistência de dados em segundo plano. Quando os dados são desnormalizados para escalabilidade, esse gerenciamento de consistência de dados automático já não existe mais. É necessário implementar um modelo de dados que funcione na camada de aplicativos ou como um plug-in para a grade de dados distribuída para garantir a consistência dos dados.

Considere o exemplo do quadro de mensagens. Se uma transação remover uma postagem de um tópico, a postagem duplicada no registro do usuário precisará ser removida. Sem um modelo de dados, um desenvolvedor pode gravar o código do aplicativo para remover a postagem do tópico e se esquecer de remover a postagem do registro do usuário. Entretanto, se o desenvolvedor estiver usando um modelo de dados em vez de interagir diretamente com o cache, o método removePost no modelo de dados poderá extrair o ID do usuário da postagem, procurar pelo registro do usuário e remover a postagem duplicada em segundo plano.

Como alternativa, é possível implementar um listener que é executado na partição real que detecta a alteração no tópico e ajusta automaticamente o registro do usuário. Um listener pode ser benéfico porque o ajuste do registro do usuário pode ocorrer localmente caso a partição possua o registro do usuário ou, mesmo se o registro do usuário estiver em uma partição diferente, a transação ocorrerá entre os servidores em vez de ocorrer entre o cliente e o servidor. A conexão de rede entre os servidores provavelmente é mais rápida do que a conexão de rede entre o cliente e o servidor.

Evitar contenção

Evite cenários, como ter um contador global. A grade de dados não será escalável se um registro único estiver sendo usado por um número de vezes desproporcional, comparado com o restante dos registros. O desempenho da grade de dados será limitado pelo desempenho do computador que mantém o registro fornecido.

Nessas situações, tente dividir o registro para que ele seja gerenciado por partição. Por exemplo, considere uma transação que retorna o número total de entradas no cache distribuído. Em vez de fazer com que cada operação de inserção e remoção acesse um único registro que incrementa, faça com que o listener em cada partição controle as operações de inserção e remoção. Com o controle desse listener, a inserção e a remoção poderá se transformar nas operações de partição única.

A leitura do contador se transformará em uma operação de grade de dados cruzada, mas para a maior parte, isso já era ineficiente como uma operação de grade de dados cruzada porque o desempenho dependia do desempenho do computador que hospeda o registro.

Dicas de Implementação

Também é possível considerar as seguintes dicas para obter a melhor escalabilidade.

Índices de Procura Reversa

Considere um modelo de dados não-normalizado adequadamente em que os registros são particionados com base no número do ID do cliente. Esse método de particionamento é a opção lógica porque quase cada operação de negócios executada com o registro do cliente usa o número de ID do cliente. Entretanto, uma transação importante que não usa o número do ID do cliente é a transação de login. É mais comum ter nomes de usuário ou endereços de e-mail para login em vez de números do ID de cliente.

A abordagem simples com o cenário de login é usar uma transação de grade de dados cruzada para localizar o registro do cliente. Conforme explicado anteriormente, essa abordagem não é escalada.

A próxima opção pode ser uma partição no nome do usuário ou e-mail. Essa opção não é viável porque todas as operações baseadas no ID do cliente se transformam em transações de grade de dados cruzada. Além disso, os clientes do seu site podem alterar o nome do usuário ou o endereço de e-mail. Produtos como o WebSphere eXtreme Scale precisam do valor usado para particionar os dados que permanecerem constantes.

A solução correta é usar um índice de consulta reversa. Com o WebSphere eXtreme Scale, um cache pode ser criado na mesma grade distribuída que o cache que mantém todos os registros do usuário. Esse cache é altamente escalável, particionado e escalável. Esse cache pode ser usado para mapear um nome de usuário ou endereço de e-mail para um ID de cliente. Esse cache transforma o login em uma operação de duas partições em vez de uma operação de grade cruzada. Esse cenário não é tão bom quanto uma transação de partição única, mas o rendimento ainda pode ser escalado linearmente conforme o número de computadores aumenta.

Computar no Momento da Gravação

Valores normalmente calculados, como médias ou totais, podem ser despendiosos para serem produzidos porque essas operações normalmente

requerem leitura de um grande número de entradas. Como as leituras são mais comuns do que as gravações na maioria dos aplicativos, é eficiente calcular esses valores no momento da gravação e, em seguida, armazenar o resultado no cache. Essa prática torna as operações mais rápidas e mais escaláveis.

Campos Opcionais

Considere um registro de usuário que mantém um número de negócios, doméstico e de telefone. Um usuário pode ter todos, nenhum ou qualquer combinação desses números definida. Se os dados forem normalizados, uma tabela do usuário e um número de telefone existirão. Os números de telefone para um determinado usuário pode ser localizado usando uma operação JOIN entre as duas tabelas.

Desnormalizar esse registro não requer duplicação de dados, porque a maioria dos usuários não compartilha números de telefone. Em vez disso, slots vazios no registro do usuário devem ser permitidos. Em vez de ter uma tabela de número de telefone, inclua três atributos em cada registro do usuário, um para cada tipo de número de telefone. Essa inclusão de atributos elimina a operação JOIN e torna uma procura por número de telefone de um usuário uma operação de partição única.

Colocação de relacionamentos muitos-para-muitos

Considere um aplicativo que controla os produtos e as lojas nas quais os produtos são vendidos. Um único produto é vendido em muitas lojas e uma única loja vende muitos produtos. Suponha que esse aplicativo controla 50 grandes varejistas. Cada produto é vendido em um máximo de 50 lojas, com cada uma vendendo milhares de produtos.

Mantenha uma lista de lojas dentro da entidade do produto (organização A), em vez de manter uma lista de produtos dentro de cada entidade de loja (organização B). Observar algumas das transações que esse aplicativo teria que executar mostra o porquê a organização A é mais escalável.

Primeiro observe as atualizações. Com a organização A, remover um produto do inventário de uma loja bloqueia a entidade do produto. Se a grade de dados mantiver 10.000 produtos, apenas 1/10.000 da grade de dados precisará ser bloqueada para executar a atualização. Com a organização B, a grade de dados contém apenas 50 lojas, de modo que 1/50 da grade deverá ser bloqueada para concluir a atualização. Portanto, embora os dois possam ser considerados operações de partição única, a organização A é escalada mais eficientemente.

Agora, considerando as leituras na organização A, observar as lojas nas quais um produto é vendido é uma transação de partição única que é escalada e é rápido porque a transação transmite apenas uma pequena quantia de dados. Com a organização B, essa transação se torna uma transação de grade de dados cruzada porque cada entidade de loja deve ser acessada para ver se o produto é vendido nessa loja, que revela uma enorme vantagem de desempenho para a organização A.

Escalando com Dados Normalizados

Um uso legítimo de transações de grade de dados cruzada é escalar o processamento de dados. Se uma grade de dados tiver 5 computadores e uma transação de grade de dados cruzada for despachada, que classifica cerca de 100.000 registros em cada computador, essa transação classificará entre 500.000 registros. Se o computador mais lento na grade de dados puder executar 10 dessas transações por segundo, a grade de dados poderá

classificar cerca de 5.000.000 de registros por segundo. Se os dados da grade dobrarem, cada computador deverá classificar cerca de 200.000 registros e cada transação classificará cerca de 1.000.000 de registros. Esse aumento de dados diminui o rendimento do computador mais lento para 5 transações por segundo, reduzindo, assim, o rendimento da grade de dados para 5 transações por segundo. Além disso, a grade de dados classifica entre 5.000.000 de registros por segundo.

Neste cenário, dobrar o número de computadores permite que cada computador volte para o carregamento anterior de classificação de 100.000 registros, permitindo que o computador mais lento processe 10 dessas transações por segundo. O rendimento da grade de dados permanece o mesmo nas 10 solicitações por segundo, mas agora cada transação processa 1.000.000 de registros dobrando, assim, a capacidade da grade em processar 10.000.000 de registros por segundo.

Aplicativos, como um mecanismo de procura, que precisam ser escalados tanto em termos de processamento de dados para acomodar o tamanho crescente da Internet e de rendimento para acomodar o crescimento de usuários, requer a criação de diversas grades de dados, com um round robin das solicitações entre as grades. Se você precisar efetuar scale up do rendimento, inclua computadores e outra grade de dados para solicitações de serviço. Se for necessário efetuar scale up do processamento de dados, inclua mais computadores e mantenha o número de grades de dados constante.

Usando Bloqueio

Bloqueios têm ciclos de vida e os tipos diferentes de bloqueios são compatíveis com outros de várias maneiras. Os bloqueios devem ser manipulados na ordem correta para evitar cenários de conflito.

Bloqueios:

Bloqueios têm ciclos de vida e os tipos diferentes de bloqueios são compatíveis com outros de várias maneiras. Os bloqueios devem ser manipulados na ordem correta para evitar cenários de conflito.

Bloqueios Compartilhados, Passíveis de Upgrade e Exclusivos

Quando um aplicativo chama qualquer método da interface ObjectMap, usa os métodos de localização em um índice ou faz uma consulta, o eXtreme Scale tenta automaticamente adquirir um bloqueio para a entrada de mapa que está sendo acessada. O WebSphere eXtreme Scale utiliza os seguintes modos de bloqueio com base no método em que o aplicativo chama na interface ObjectMap.

- Os métodos get e getAll na interface ObjectMap, os métodos de índice e as consultas adquirem um *bloqueio S* ou um modo de bloqueio compartilhado para a chave em uma entrada de mapa. A duração em que o bloqueio S é mantido depende do nível de isolamento da transação usado. Um modo de bloqueio S permite a simultaneidade entre transações que tentam adquirir um modo de bloqueio S ou de bloqueio para upgrade (bloqueio U) para a mesma chave, mas bloqueia outras transações que tentam obter um modo de bloqueio exclusivo (bloqueio X) para a mesma chave.
- Os métodos getForUpdate e getAllForUpdate adquirem um *bloqueio U* ou um modo de bloqueio para upgrade para a chave de uma entrada do mapa. O bloqueio U fica retido até que a transação seja concluída. Um modo de bloqueio U permite a simultaneidade entre transações que adquirem um modo de

bloqueio S para a mesma chave, mas bloqueia outras transações que tentam adquirir um modo de bloqueio U ou de bloqueio X para a mesma chave.

- Os métodos put, putAll, remove, removeAll, insert, update e touch adquirem um *bloqueio X* ou um modo de bloqueio exclusivo para a chave de uma entrada de mapa. O bloqueio X fica retido até que a transação seja concluída. Um modo de bloqueio X assegura que apenas uma transação esteja inserindo, atualizando ou removendo uma entrada do mapa de um valor de chave especificado. Um bloqueio X bloqueia todas as demais transações que tentam adquirir um modo de bloqueio S, U ou X para a mesma chave.
- Os métodos global invalidate e global invalidateAll adquirem um bloqueio X para cada entrada do mapa que é invalidada. O bloqueio X fica retido até que a transação seja concluída. Não são adquiridos bloqueios para os métodos local invalidate e local invalidateAll, porque nenhuma das entradas de BackingMap é invalidada por chamadas de métodos locais invalidate.

Das definições anteriores, é óbvio que um modo de bloqueio S é mais fraco que um modo de bloqueio U, porque permite que mais transações sejam executadas simultaneamente durante o acesso à mesma entrada do mapa. O modo de bloqueio U é um pouco mais forte do que o modo de bloqueio S, porque bloqueia outras transações que estão solicitando um modo de bloqueio U ou X. O modo de bloqueio S bloqueia apenas outras transações que estão solicitando um modo de bloqueio X. Esta pequena diferença é importante na prevenção de alguns conflitos. O modo de bloqueio X é o modo de bloqueio mais forte, porque bloqueia todas as demais transações que estão tentando obter um modo de bloqueio S, U ou X para a mesma entrada do mapa. O efeito de rede de um modo de bloqueio X é para garantir que apenas uma transação possa inserir, atualizar ou remover uma entrada de mapa e para evitar que atualizações sejam perdidas quando mais de uma transação esteja tentando atualizar a mesma entrada de mapa.

A tabela a seguir é uma matriz de compatibilidade de modo de bloqueio que resume os modos de bloqueio descritos, que você pode utilizar para determinar quais modos de bloqueio são compatíveis com outros. Para ler esta matriz, a linha na matriz indica um modo de bloqueio já concedido. A coluna indica o modo de bloqueio solicitado por outra transação. Se Yes for exibido na coluna, então, o modo de bloqueio solicitado por outra transação é concedido porque é compatível com o modo de bloqueio que já foi concedido. No, indica que o modo de bloqueio não é compatível e a outra transação deve esperar a primeira transação liberar o bloqueio que ela possui.

Tabela 4. Matriz de Compatibilidade do Modo de Bloqueio

Bloqueio	Tipo de bloqueio S (compartilhado)	Tipo de bloqueio U (para upgrade)	Tipo de bloqueio X (exclusivo)	Força
S (compartilhado)	Sim	Sim	Não	mais fraco
U (para upgrade)	Sim	Não	Não	normal
X (exclusivo)	Não	Não	Não	mais forte

Conflitos de Bloqueio

Considere a seguinte seqüência de pedidos de modo de bloqueio:

1. O bloqueio X é concedido à transação 1 para key1.
2. O bloqueio X é concedido à transação 2 para key2.
3. O bloqueio X é solicitado pela transação 1 para key2. (Transaction 1 blocks waiting for lock owned by transaction 2.)

4. O bloqueio X é solicitado pela transação 2 para key1. (Transaction 2 blocks waiting for lock owned by transaction 1.)

A sequência anterior é o exemplo clássico de conflito de duas transações que tentam adquirir mais de um bloqueio único e cada transação adquire os bloqueios em uma ordem diferente. Para evitar este conflito, cada transação deve obter vários bloqueios na mesma ordem. Se a estratégia de bloqueio OPTIMISTIC for utilizada e o método flush na interface ObjectMap nunca for utilizado pelo aplicativo, os modos de bloqueio serão solicitados pela transação apenas durante o ciclo de confirmação. Durante o ciclo de commit, o eXtreme Scale determina as chaves para as entradas de mapa que precisam ser bloqueadas e solicita os modos de bloqueio na sequência de chaves (comportamento determinístico). Com este método, o eXtreme Scale evita a grande maioria dos conflitos clássicos. Entretanto, o eXtreme Scale não evita e não pode evitar todos os cenários de conflito possíveis. Existem poucos cenários que o aplicativo precisa considerar. A seguir estão os cenários que o aplicativo deve considerar e executar uma ação preventiva contra.

Existe um cenário em que eXtreme Scale está apto a detectar um conflito sem precisar aguardar que um tempo limite de espera de bloqueio ocorra. Se este cenário ocorrer, isto resultará em uma exceção com.ibm.websphere.objectgrid.LockDeadlockException. Considere o seguinte trecho de código:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Lynn");
// Lynn had a birthday, so we make her 1 year older.
p.setAge( p.getAge() + 1 );
person.put( "Lynn", p );
sess.commit();
```

Nessa situação, o namorado de Lynn quer que ela seja mais velha do que sua idade atual e tanto Lynn quanto seu namorado executam essa transação simultaneamente. Nesta situação, as duas transações possuem um modo de bloqueio S na entrada Lynn do mapa PERSON como resultado da chamada de método person.get("Lynn"). Como resultado da chamada de método person.put ("Lynn", p), as duas transações tentam fazer upgrade do modo de bloqueio S para um modo de bloqueio X. As duas transações bloqueiam a espera para que a outra transação libere o modo de bloqueio S que ela possui. Como resultado, ocorre um conflito porque existe uma condição de espera circular entre as duas transações. É gerada uma condição de espera circular quando mais de uma transação tenta promover um bloqueio de um modo mais fraco para um mais forte para a mesma entrada do mapa. Neste cenário, o resultado é uma exceção LockDeadlockException ao invés de uma exceção LockTimeoutException.

O aplicativo pode evitar a exceção LockDeadlockException para o exemplo anterior utilizando a estratégia de bloqueio optimistic ao invés da estratégia de bloqueio pessimistic. Utilizar a estratégia de bloqueio optimistic é a solução preferida quando o mapa é em maior parte lido e as atualizações no mapa são infrequentes. Se a estratégia de bloqueio pessimistic deve ser utilizada, o método getForUpdate pode ser utilizado ao invés do método get no exemplo acima ou pode ser utilizado um nível de isolamento de transação de TRANSACTION_READ_COMMITTED.

Para obter mais informações, consulte o tópico sobre as estratégias de bloqueio na *Visão Geral do Produto.*

Utilizar o nível de isolamento da transação TRANSACTION_READ_COMMITTED evita o bloqueio S adquirido pelo método get seja mantido até a transação ser concluída. Como a chave nunca é invalidada no cache transacional, as leituras repetitíveis ainda são garantidas.

Consulte o tópico sobre bloqueio de entrada de mapa no *Guia de Administração* para obter mais informações.

Uma alternativa para alterar o nível de isolamento de transação é utilizar o método getForUpdate. A primeira transação para chamar o método getForUpdate adquire um modo de bloqueio U ao invés de um bloqueio S. Este modo de bloqueio faz com que a segunda transação seja bloqueada quando ela chama o método getForUpdate, porque apenas uma transação recebe um modo de bloqueio U. Como a segunda transação é bloqueada, ela não possui nenhum modo de bloqueio na entrada do mapa de Lynn. A primeira transação não é bloqueada quando tenta atualizar o modo de bloqueio U para um modo de bloqueio X como um resultado da chamada de método put a partir da primeira transação. Este recurso demonstra porque o modo de bloqueio U é chamado no modo de bloqueio *atualizável*. Quando a primeira transação é concluída, a segunda transação é desbloqueada e recebe o modo de bloqueio U. Um aplicativo pode evitar o cenário de conflito de promoção de bloqueio utilizando o método getForUpdate ao invés do método get quando a estratégia de bloqueio pessimistic está sendo utilizada.

Importante: Esta solução não evita que transações somente leitura sejam capazes de ler uma entrada de mapa. As transações de leitura chamam o método get, mas nunca chamam os métodos put, insert, update ou remove. A simultaneidade é alta apenas quando o método get regular é utilizado. A única redução na simultaneidade ocorre quando o método getForUpdate é chamado por mais de uma transação para a mesma entrada do mapa.

Você deve estar ciente de que uma transação chama o método getForUpdate em mais de uma entrada de mapa para garantir que os bloqueios U sejam adquiridos na mesma ordem para cada transação. Por exemplo, suponha que a primeira transação chame o método getForUpdate para a chave 1 e o método getForUpdate para a chave 2. Outra transação simultânea chama o método getForUpdate para as mesmas chaves, mas em ordem inversa. Esta sequência causa o conflito clássico, porque vários bloqueios são obtidos em diferentes ordens por diferentes transações. O aplicativo ainda precisará assegurar que cada transação acesse várias entradas do mapa na seqüência de chaves para assegurar que não ocorrerá o conflito. Como o bloqueio U é obtido no momento em que o método getForUpdate é chamado ao invés de no momento do commit, o eXtreme Scale não pode ordenar os pedidos de bloqueio como ele faz durante o ciclo do commit. O aplicativo deve controlar a ordem de bloqueios neste caso.

A utilização do método flush na interface ObjectMap antes de uma confirmação pode introduzir considerações adicionais sobre ordem de bloqueios. O método flush geralmente é utilizado para forçar alterações no mapa fora do backend por meio do plug-in do Loader. Nesta situação, o backend utiliza seu próprio gerenciador de bloqueios para controlar a simultaneidade, assim, a condição de espera do bloqueio e o conflito podem ocorrer no backend ao invés de no gerenciador de bloqueios do eXtreme Scale. Considere a seguinte transação:

```
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();
}
```

Suponha que outra transação também tenha atualizado a pessoa Tom person, chamado o método flush e, em seguida, atualizado a pessoa Lynn. Se esta situação tiver ocorrido, a seguinte intercalação das duas transações resultará em uma condição de conflito do banco de dados:

```
X lock is granted to transaction 1 for "Lynn" when flush is executed.
X lock is granted to transaction 2 for "Tom" when flush is executed..
X lock requested by transaction 1 for "Tom" during commit processing.
(Transaction 1 blocks waiting for lock owned by transaction 2.)
X lock requested by transaction 2 for "Lynn" during commit processing.
(Transaction 2 blocks waiting for lock owned by transaction 1.)
```

Este exemplo demonstra que o uso do método flush pode causar um conflito no banco de dados ao invés de no eXtreme Scale. Este exemplo de conflito pode ocorrer, independentemente da estratégia de bloqueio utilizada. O aplicativo deve ter atenção para evitar que ocorra este tipo de conflito ao utilizar o método flush e quando um Utilitário de Carga for conectado ao BackingMap. O exemplo anterior também ilustra outro motivo pelo qual o eXtreme Scale tem um mecanismo de tempo limite de espera de bloqueio. Uma transação que está aguardando um bloqueio de banco de dados poderia estar aguardando enquanto possuía um bloqueio de entrada de mapa do eXtreme Scale. Consequentemente, problemas no nível do banco de dados podem causar tempos de espera excessivos para um modo de bloqueio do eXtreme Scale e resultam em uma exceção LockTimeoutException.

Tarefas relacionadas

"Resolvendo Problemas de Conflitos" na página 519
As seções a seguir descrevem alguns dos cenários de conflitos mais comuns e sugestões sobre como evitá-los.

Implementando Manipulação de Exceção em Cenários de Bloqueio:

Para evitar que bloqueios sejam mantidos por quantidades excessivas de tempo quando ocorre uma exceção LockTimeoutException ou LockDeadlockException, um aplicativo deve garantir que obtenha exceções inesperadas e chama o método rollback quando ocorre algo inesperado.

Procedimento

1. Capture a exceção e exiba a mensagem resultante.

   ```
   try {
   ...
   } catch (ObjectGridException oe) {
   System.out.println(oe);
   }
   ```

   A seguinte exceção é exibida como resultado:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: Message
```

Essa mensagem representa a cadeia que é transmitida como um parâmetro quando a exceção é criada e emitida.

2. Retroceda a transação após uma exceção:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
    sess.begin();
    activeTran = true;
    Person p = (IPerson)person.get("Lynn");
    // Lynn had a birthday, so we make her 1 year older.
    p.setAge( p.getAge() + 1 );
    person.put( "Lynn", p );
    sess.commit();
    activeTran = false;
}
finally
{
    if ( activeTran ) sess.rollback();
}
```

O bloco de `finally` no fragmento de código assegura que uma transação é retrocedida quando ocorrer uma exceção inesperada. Ele não apenas manipula uma exceção LockDeadlockException, mas qualquer outra exceção inesperada que possa ocorrer. O bloco finally manipula o caso em que ocorre uma exceção durante uma chamada de método commit. Este exemplo não é a única maneira de lidar com exceções inesperadas e pode haver casos em que um aplicativo deseja capturar algumas das exceções inesperadas que podem ocorrer e exibir uma de suas exceções do aplicativo. É possível incluir blocos de captura conforme apropriado, mas o aplicativo deve assegurar que o trecho de código não seja encerrado sem concluir a transação.

Configurando uma Estratégia de Bloqueio:

É possível definir uma estratégia otimista, pessimista ou de nenhum bloqueio em cada BackingMap na configuração do WebSphere eXtreme Scale.

Sobre Esta Tarefa

Cada instância do BackingMap pode ser configurada para usar uma das seguintes estratégias de bloqueio:

1. Modo de Bloqueio Otimista
2. Modo de Bloqueio Pessimista
3. Nenhum

A estratégia de bloqueio padrão é OPTIMISTIC. Utilize o bloqueio optimistic quando os dados são alterados de maneira infrequente. Os bloqueios são mantidos apenas por uma curta duração enquanto os dados estão sendo lidos do cache e copiados para a transação. Quando o cache da transação é sincronizado com o cache principal, quaisquer objetos de cache que foram atualizados são verificados junto à versão original. Se a verificação falhar, então, ocorre o rollback da transação e o resultado é uma exceção OptimisticCollisionException.

A estratégia de bloqueio PESSIMISTIC adquire bloqueios para entradas de cache e deve ser utilizada quando os dados são alterados frequentemente. Sempre que uma

entrada de cache é lida, um bloqueio é adquirido e mantido condicionalmente até que a transação seja concluída. A duração de alguns bloqueios pode ser ajustada utilizando níveis de isolamento de transação para a sessão.

Se o bloqueio não for necessário porque os dados nunca são atualizados ou são atualizados apenas durante períodos tranquilos, você pode desativar o bloqueio utilizando a estratégia de bloqueio NONE. Esta estratégia é muito rápida porque um gerenciador de bloqueio não é necessário. A estratégia de bloqueio NONE é ideal para tabelas de consulta ou mapas somente leitura.

Para obter mais informações sobre as estratégicas de bloqueio, consulte "Estratégias de Bloqueio" na página 235 informações sobre as estratégias de bloqueio *Visão Geral do Produto*.

Procedimento

- **Configure uma estratégia de bloqueio otimista**
  - Usando o método setLockStrategy programaticamente:

```
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 );
```

  - Usando o atributo lockStrategy no :

```
<?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 uma estratégia de bloqueio pessimista**
  - Usando o método setLockStrategy programaticamente:

```
especificar programaticamente
estratégia pessimista
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 );
```

  - Usando o atributo lockStrategy no :

```
Especificar estratégia pessimista usando 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 uma estratégia de ausência de bloqueio**
  - Usando o método setLockStrategy programaticamente:

```
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 );
```

  - Usando o atributo lockStrategy no :

```
<?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>
```

O que Fazer Depois

Para evitar uma exceção java.lang.IllegalStateException, é necessário chamar o método setLockStrategy antes de chamar os métodos initialize ou getSession na instância ObjectGrid.

Configurando o Valor de Tempo Limite de Bloqueio:

O valor de tempo limite de bloqueio é usado em uma instância BackingMap para garantir que um aplicativo não aguarde infinitamente por um modo de bloqueio a ser concedido devido a uma condição de conflito que ocorre por causa de um erro de aplicativo.

Antes de Iniciar

Para configurar o valor de tempo limite de bloqueio, a estratégia de bloqueio deve ser configurada para um OPTIMISTIC ou PESSIMISTIC. Consulte o “Configurando uma Estratégia de Bloqueio” na página 251 para obter informações adicionais.

Sobre Esta Tarefa

Quando ocorrer uma exceção LockTimeoutException, o aplicativo deve determinar se o tempo limite está ocorrendo porque o aplicativo está em execução mais lento do que o esperado ou se o tempo limite ocorreu devido a uma condição de conflito. Se tiver ocorrido uma condição de conflito real, aumentar o valor de tempo limite de espera de bloqueio não elimina a exceção. Aumentar o tempo limite implica na exceção demorar mais tempo para ocorrer. No entanto, se o aumento do valor de tempo limite de espera de bloqueio eliminar a exceção, isto

indica que o problema ocorreu porque o aplicativo estava em execução de forma mais lenta que o esperado. Neste caso, o aplicativo deve determinar por que o desempenho é lento.

Para evitar a ocorrência de conflitos, o gerenciador de bloqueios possui um valor de tempo limite de 15 segundos. Se o tempo limite for excedido, ocorre uma exceção LockTimeoutException. Se o sistema estiver muito carregado, o valor de tempo limite padrão poderá fazer com que ocorram exceções LockTimeoutException quando não existir nenhum conflito. Nesta situação, é possível aumentar o valor de tempo limite de bloqueio programaticamente ou no arquivo descritor XML do ObjectGrid.

Procedimento

- Configure um valor de tempo limite de bloqueio programaticamente em uma instância do BackingMap com o método setLockTimeout.

  O exemplo a seguir ilustra como configurar o valor de tempo limite de espera de bloqueio para o mapa de apoio map1 em 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 uma exceção java.lang.IllegalStateException, chame tanto o método setLockStrategy quanto o método setLockTimeout antes da chamada dos métodos initialize ou getSession na instância ObjectGrid. O parâmetro do método setLockTimeout é um número inteiro de primitiva Java que especifica o número de segundos que o eXtreme Scale aguarda para que um modo de bloqueio seja concedido. Se uma transação esperar mais do que o valor de tempo limite de espera de bloqueio configurado para o BackingMap, isto resultará em uma exceção com.ibm.websphere.objectgrid.LockTimeoutException.

- Configure o valor de tempo limite de bloqueio usando o atributo lockTimeout no Arquivo XML descritor do ObjectGridArquivo Descritor XML do 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>
  ```

- Substitua o tempo limite de espera de bloqueio para uma única instância do ObjectMap. Use o método ObjectMap.setLockTimeout para substituir o valor de tempo limite de bloqueio por uma instância do ObjectMap específica. O valor de tempo limite do bloqueio afeta todas as transações iniciadas após o novo valor de tempo limite ser configurado. Este método pode ser útil quando colisões de bloqueio são possível ou esperadas nas transações select.

Bloqueios de Entrada de Mapa com Consultas e Índices:

Este tópico descreve como as APIs de consulta do eXtreme Scale e o plug-in de indexação MapRangeIndex interagem com bloqueios e algumas boas práticas para aumentar a concorrência e diminuir os conflitos ao usar a estratégia de bloqueio pessimista para mapas.

Visão Geral

A API de Consulta do ObjectGrid permite consultas SELECT sobre objetos e entidades de cache do ObjectMap. Quando uma consulta é executada, o mecanismo de consulta utiliza um MapRangeIndex quando possível para localizar chaves correspondentes na cláusula WHERE da consulta ou para vincular relacionamentos. Quando um índice não está disponível, o mecanismo de consulta varrerá cada entrada em um ou mais mapas para localizar as entradas apropriadas. O mecanismo de consulta e os plug-ins de índice adquirirão bloqueios para verificar dados consistentes, dependendo da estratégia de bloqueio, nível de isolamento de transação e estado da transação.

Bloqueio com o Plug-in HashIndex

O plug-in HashIndex do eXtreme Scale permite localizar chaves baseadas em um único atributo armazenado no valor de entrada do cache. O índice armazena o valor indexado em uma estrutura de dados separada do mapa de cache. O índice valida as chaves em relação a entradas de mapa antes de retornar para o usuário para tentar alcançar um conjunto de resultados exato. Quando a estratégia de bloqueio pessimista é utilizada e o índice é utilizado junto a uma instância local do ObjectMap (versus um ObjectMap do cliente/servidor), o índice adquirirá bloqueios para cada entrada correspondente. Ao utilizar o bloqueio optimistic ou um ObjectMap remoto, os bloqueios são sempre liberados imediatamente.

O tipo de bloqueio que é adquirido depende do argumento forUpdate passado para o método ObjectMap.getIndex. O argumento forUpdate especifica o tipo de bloqueio que o índice deve adquirir. Se false, um bloqueio compartilhável (S) é adquirido e se true, um bloqueio atualizável (U) é adquirido.

Se o tipo de bloqueio for compartilhável, a configuração de isolamento de transação para a sessão é aplicada e afeta a duração do bloqueio. Consulte o tópico Isolamento de Transação para obter detalhes sobre como o nível de isolamento é utilizado para incluir simultaneidade nos aplicativos.

Bloqueios Compartilhados com Consultas

O mecanismo de consulta do eXtreme Scale adquire bloqueios S quando necessário para introspectar as entrada do cache para descobrir se elas satisfazem os critérios de filtro da consulta. Ao utilizar o isolamento de transação de leitura repetível com bloqueio pessimistic, os bloqueios S são retidos apenas para os elementos que estão incluídos no resultado da consulta e não são incluídos no resultado. Se utilizando um nível de isolamento de transação inferior ou o bloqueio optimistic, os bloqueios S não são retidos.

Bloqueios Compartilhados com o Cliente para Consulta do Servidor

Ao usar a consulta do eXtreme Scale a partir de um cliente, a consulta tipicamente é executada no servidor, a menos que todos os mapas ou entidades referenciadas na consulta sejam locais para o cliente (por exemplo: um mapa replicado pelo cliente ou uma entidade de resultado da consulta). Todas as consultas que são executadas em uma transação de leitura/gravação reterão bloqueios S, conforme

descrito na seção anterior. Se a transação não for uma transação de leitura/gravação, então, uma sessão não será retida no servidor e os bloqueios S serão liberados.

Uma transação de leitura/gravação é roteada apenas para uma partição primária e uma sessão é mantida no servidor para a sessão do cliente. Uma transação pode ser promovida para leitura/gravação sob as seguintes condições:

1. Qualquer mapa configurado para utilizar o bloqueio pessimistic é acessado utilizando os métodos de API get e getAll do ObjectMap ou os métodos EntityManager.find.
2. Ocorreu o flush da transação, fazendo com que as atualizações sejam enviadas para o servidor.
3. Qualquer mapa configurado para utilizar o bloqueio optimistic é acessado utilizando o método ObjectMap.getForUpdate ou EntityManager.findForUpdate.

Bloqueios Atualizáveis com Consultas

Bloqueios compartilháveis são úteis quando a simultaneidade e a consistência são importantes. Isto garante que um valor de entrada não seja alterado durante a duração da transação. Nenhuma outra transação pode alterar o valor enquanto qualquer outro bloqueio S é mantido, e apenas uma outra transação possa estabelecer uma tentativa de atualizar a entrada. Consulte o tópico Modo de Bloqueio Pessimistic para obter mais detalhes sobre os modos S, U e X.

Bloqueios atualizáveis são utilizados para identificar a tentativa de atualizar uma entrada de cache ao utilizar a estratégia de bloqueio pessimistic. Isto permite a sincronização entre transações que desejam modificar uma entrada de cache. As transações ainda podem visualizar a entrada utilizando um bloqueio S, mas outras transações são impedidas de adquirir um bloqueio U ou um bloqueio X. Em muitos cenários, adquirir um bloqueio U sem primeiro adquirir um bloqueio S é necessário para evitar conflitos. Consulte o tópico Modo de Bloqueio Pessimistic para obter exemplos de conflitos comuns.

As interfaces de Consulta ObjectQuery e EntityManager fornecem o método setForUpdate para identificar o uso destinado para o resultado da consulta. Especificamente, o mecanismo de consulta adquire bloqueios U ao invés de bloqueios S para cada entrada de mapa envolvida no resultado da 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();
// Run the query.  Each order has  U lock
Iterator result = q.getResultIterator();
// For each order, update the status.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
    orderMap.update(o.getId(), o);
}
// When committed, the
session.commit();
```

```
Query q = em.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
emTran.begin();
// Run the query.  Each order has  U lock
```

```
Iterator result = q.getResultIterator();
// For each order, update the status.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
}
tmTran.commit();
```

Quando o atributo **setForUpdate** está ativado, a transação é automaticamente convertida em uma transação de leitura/gravação e os bloqueios são mantidos no servidor, conforme esperado. Se a consulta não puder utilizar nenhum índice, então, o mapa deve ser varrido, o que resultará em bloqueios U temporários para entradas de mapas que não satisfazem o resultado da consulta, e bloqueios U para as entradas que estão incluídas no resultado.

Isolamento de transação

Para transações, é possível configurar cada configuração de mapa de apoio com uma das três estratégias de bloqueio: pessimistic, optimistic ou none. Quando você usa os bloqueios pessimistic e optimistic, o eXtreme Scale usa bloqueios compartilhado(S), atualizável (U) e exclusivo (X) para manter a consistência. Este comportamento de bloqueio é mais perceptível quando o bloqueio pessimistic é usado, pois os bloqueios optimistic não são mantidos. É possível usar um dos três níveis de isolamento de transação para ajustar as semânticas de bloqueio que o eXtreme Scale usa para manter a consistência em cada mapa de cache: repeatable read, read committed e read uncommitted.

Visão Geral de Isolamento da Transação

O isolamento de transação define como as alterações que são feitas por uma operação se tornam visíveis a outras operações concorrentes.

O WebSphere eXtreme Scale suporta três níveis de isolamento de transação com os quais é possível ajustar adicionalmente as semânticas de bloqueio que o eXtreme Scale usa para manter a consistência em cada mapa do cache: repeatable read, read committed e read uncommitted. O nível de isolamento de transação é configurado na interface Sessão usando o método setTransactionIsolation. O isolamento de transação pode ser alterado a qualquer momento durante a duração da sessão, se uma transação não estiver atualmente em andamento.

O produto força várias semânticas de isolamento de transação ajustando a forma na qual os bloqueios compartilhados (S) são solicitados e mantidos. O isolamento de transação não tem efeito nos mapas configurado para utilizar as estratégias de bloqueio optimistic ou none ou quando bloqueios atualizáveis (U) são necessários.

Leitura Repetível com Bloqueio Pessimistic

O nível de isolamento de transação repeatable read é o padrão. Este nível de isolamento evita leituras sujas e leituras não repetitivas, mas não evita leituras fantasmas. Uma leitura suja é uma operação de leitura que ocorre nos dados que foram modificados por uma transação mas não foi consolidada. Uma leitura não repetitiva pode ocorrer quando os bloqueios de leitura não são adquiridos na execução de uma operação de leitura. Uma leitura fantasma pode ocorrer quando duas operações de leitura idênticas são executadas, mas dois conjuntos diferentes de resultados são retornados porque uma atualização ocorreu nos dados entre as operações de leitura. O produto atinge uma leitura repetitiva se mantendo em qualquer bloqueio S até que a transação que possui o bloqueio seja concluída.

Como um bloqueio X não é concedido até que todos os bloqueios S sejam liberados, todas as transações contendo o bloqueio S são garantidas para visualizar o mesmo valor quando lidas novamente.

```
map = session.getMap("Order");
session.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session.begin();

// An S lock is requested and held and the value is copied into
// the transactional cache.
Order order = (Order) map.get("100");
// The entry is evicted from the transactional cache.
map.invalidate("100", false);

// The same value is requested again.  It already holds the
// lock, so the same value is retrieved and copied into the
// transactional cache.
Order order2 (Order) = map.get("100");

// All locks are released after the transaction is synchronized
// with cache map.
session.commit();
```

As leituras fantasmas são possíveis quando você usa consultas ou índices pois os bloqueios não são adquiridos para intervalos de dados, somente para as entradas de cache que correspondem ao índice ou critérios de busca. Por exemplo:

```
session1.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session1.begin();

// A query is run which selects a range of values.
ObjectQuery query = session1.createObjectQuery
    ("SELECT o FROM Order o WHERE o.itemName='Widget'");

// In this case, only one order matches the query filter.
// The order has a key of "100".
// The query engine automatically acquires an S lock for Order "100".
Iterator result = query.getResultIterator();

// A second transaction inserts an order that also matches the query.
Map orderMap = session2.getMap("Order");
orderMap.insert("101", new Order("101", "Widget"));

// When the query runs again in the current transaction, the
// new order is visible and will return both Orders "100" and "101".
result = query.getResultIterator();

// All locks are released after the transaction is synchronized
// with cache map.
session.commit();
```

Leitura com Commit com Bloqueio Pessimistic

O nível de isolamento de transação consolidado da leitura pode ser usado com o eXtreme Scale, o que evita leituras sujas, mas não evita leituras não repetitivas ou leituras fantasmas, assim o eXtreme Scale continua a usar bloqueios S para ler dados a partir do mapa de cache, mas libera imediatamente os bloqueios.

```
map1 = session1.getMap("Order");
session1.setTransactionIsolation(Session.TRANSACTION_READ_COMMITTED);
session1.begin();

// An S lock is requested but immediately released and
//the value is copied into the transactional cache.

Order order = (Order) map1.get("100");
```

```
// The entry is evicted from the transactional cache.
map1.invalidate("100", false);

// A second transaction updates the same order.
// It acquires a U lock, updates the value, and commits.
// The ObjectGrid successfully acquires the X lock during
// commit since the first transaction is using read
// committed isolation.

Map orderMap2 = session2.getMap("Order");
session2.begin();
order2 = (Order) orderMap2.getForUpdate("100");
order2.quantity=2;
orderMap2.update("100", order2);
session2.commit();

// The same value is requested again.  This time, they
// want to update the value, but it now reflects
// the new value
Order order1Copy (Order) = map1.getForUpdate("100");
```

Leitura não Consolidada com Bloqueio Pessimistic

O nível de isolamento de transação não consolidado de leitura pode ser usado com o eXtreme Scale, o que é um nível que permite leituras sujas, leituras não repetitivas e leituras fantasmas.

Exceção de Colisão Otimista

É possível receber uma OptimisticCollisionException diretamente ou recebê-la com uma ObjectGridException.

O código a seguir é um exemplo de como capturar a exceção e, em seguida, exibir sua mensagem:

```
try {
...
} catch (ObjectGridException oe) {
    System.out.println(oe);
}
```

Causa da Exceção

Uma exceção OptimisticCollisionException é criada em uma situação na qual dois clientes diferentes tentam atualizar a mesma entrada do mapa relativamente ao mesmo tempo. Por exemplo, se um cliente tentar executar uma sessão e atualizar a entrada do mapa após outro cliente ler os dados antes da execução, estes dados estarão incorretos. A exceção é criada quando o outro cliente tenta cometer os dados incorretos.

Recuperando a Chave que Acionou a Exceção

Pode ser útil, durante a resolução de problemas como uma exceção, recuperar a chave correspondente à entrada que acionou a exceção. O benefício da OptimisticCollisionException é que ela contém o método getKey, que retorna o objeto que representa essa chave. A seguir está um exemplo de como recuperar e imprimir a chave ao capturar a OptimisticCollisionException:

```
try {
...
} catch (OptimisticCollisionException oce) {
    System.out.println(oce.getKey());
}
```

ObjectGridException Causa uma OptimisticCollisionException

Uma OptimisticCollisionException pode ser a causa de exibição da ObjectGridException. Se este for o caso, será possível utilizar o seguinte código para determinar o tipo de exceção e imprimir a chave. O seguinte código utiliza o método do utilitário findRootCause, conforme descrito na seção abaixo.

```
try {
...
}
catch (ObjectGridException oe) {
    Throwable Root = findRootCause( oe );
    if (Root instanceof OptimisticCollisionException) {
        OptimisticCollisionException oce = (OptimisticCollisionException)Root;
        System.out.println(oce.getKey());
    }
}
```

Técnica Geral de Manipulação de Exceção

Saber a causa-raiz de um objeto Throwable é útil no isolamento da origem de um problema. O exemplo a seguir demonstra como um manipulador de exceções usa um método utilitário para localizar a causa-raiz do objeto Throwable.

Exemplo:

```
static public Throwable findRootCause( Throwable t )
{
     // Inicie com o Throwable que ocorreu como a raiz.
     Throwable root = t;

     // Siga a cadeia de causas até localizar o último Throwable na cadeia.
     Throwable cause = root.getCause();
     while ( cause != null )
     {
        root = cause;
        cause = root.getCause();
     }

     // Retorne o último Throwable na cadeia como a causa raiz.
     return root;
}
```

Executando a Lógica de Negócios em Paralelo na Grade de Dados (API DataGrid)

A API do DataGrid fornece uma interface de programação simples para executar lógica de negócios sobre toda a, ou um subconjunto da, grade de dados em paralelo onde os dados estão localizados.

APIs do DataGrid e Particionamento:

Com as APIs do DataGrid, um cliente pode enviar solicitações a uma partição, a um subconjunto de partições ou a todas as partições em uma grade de dados. O cliente pode especificar uma lista de chaves, e o WebSphere eXtreme Scale determina o conjunto de partições que estão hospedando as chaves. O pedido é então enviado a todas as partições no conjunto em paralelo e o cliente aguarda

pelos resultados. O cliente também pode enviar pedidos sem especificar chaves, portanto, os pedidos são enviados para todas as partições.

Os agentes que são implementados na grade de dados não funcionam no modo cliente. Esses agentes funcionam diretamente contra o shard primário. Funcionando diretamente contra o shard principal resulta em desempenho máximo, permitindo dezenas de milhares de transações ou mais por segundo porque o agente funciona com os dados em velocidades de memória completas. Trabalhar diretamente com o shard primário também significa que um agente possa ver somente dados que estejam dentro deste shard. Isso proporciona oportunidades interessantes que não podem ser feitas em um cliente.

Um cliente típico do eXtreme Scale deve poder determinar a partição da transação, pois o cliente precisa rotear a solicitação. Se um agente estiver ligado diretamente a um shard, então, nenhum roteamento é necessário. Todos os pedidos seguem contra esse shard. Como o agente está diretamente ligado a um shard, os dados em outros mapas no shard podem ser acessados sem a preocupação com chaves de particionamento comum, e assim por diante, porque não ocorre nenhum roteamento.

Agentes do DataGrid e Mapas Baseados em Entidade:

Um mapa contém objetos-chave e objetos de valor. O objeto-chave é uma tupla gerada conforme o valor. Um agente normalmente é fornecido com o aplicativo especificado pelos objetos-chave.

O objeto-chave é uma tupla gerada conforme o valor. Um agente normalmente é fornecido com o aplicativo especificado pelos objetos-chave. Esse agente será os objetos-chave utilizados pelo aplicativo ou Tuplas se for um Mapa de entidade. Um aplicativo que usa Entidades não desejará lidar com Tuplas diretamente e preferirá trabalhar com os objetos Java mapeados para a Entidade.

Portanto, uma classe de Agente pode implementar a interface EntityAgentMixin. Isso força a classe a implementar mais um método, o getClassForEntity(). Isso retorna a classe de entidade para utilizar com o agente no lado do servidor. As chaves são convertidas nessa Entidade antes de chamar os métodos de processo e de redução.

Essa é uma diferença semântica de um agente não EntityAgentMixin em que esses métodos são fornecidas apenas com as chaves. Um agente implementando EntityAgentMixin recebe o objeto de Entidade que inclui as chaves e os valores em um objeto.

Nota: Se a entidade não existir no servidor, as chaves estão no formato de Tupla bruto da chave ao invés da entidade gerenciada.

Exemplo da API do DataGrid:

As APIs do DataGrid suportam dois padrões de programação de grade comuns: mapa de paralelo e redução de paralelo.

Mapa de Paralelos

O mapa de paralelos permite que as entradas de um conjunto de chaves sejam processadas e retorna um resultado para cada entrada processada. O aplicativo faz uma lista de chaves e recebe um Mapa de pares chave/resultado depois de chamar

uma operação do Mapa. O resultado provém da aplicação de uma função à entrada de cada chave. A função é fornecida pelo aplicativo.

Fluxo de chamada do MapGridAgent

Quando o método AgentManager.callMapAgent é chamado com um conjunto de chaves, a instância do MapGridAgent é serializada e enviada para cada partição principal que as chaves resolvem. Isto significa que quaisquer dados de instância armazenados no agente podem ser enviados para o servidor. Portanto, cada partição primária possui uma instância do agente. O método process é chamado para cada instância uma vez para cada chave que resolve a partição. O resultado de cada método process é, então, serializado de volta para o cliente e retornado para o responsável pela chamada em uma instância de Mapa, onde o resultado é representado como o valor no mapa.

Quando o método AgentManager.callMapAgent é chamado sem um conjunto de chaves, a instância do MapGridAgent é serializada e enviada para cada partição principal. Isto significa que quaisquer dados de instância armazenados no agente podem ser enviados para o servidor. Cada partição principal, portanto, tem uma instância (partição) do agente. O método processAllEntries é chamado para cada partição. O resultado de cada método processAllEntries é, então, serializado de volta para o cliente e retornado para o responsável pela chamada em uma instância de Mapa. O exemplo a seguir parte da premissa de que há uma entidade Person com o seguinte formato:

```
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;
}
```

A função fornecida pelo aplicativo é redigida como uma classe que implementa a interface MapAgentGrid. Este é um exemplo de agente que apresenta uma função para retornar a idade de Person multiplicada por dois.

```
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;
 }
}
```

Esse é um exemplo de um agente Mapa que duplica uma entidade Person. Concentre-se, primeiro, nos métodos do processo. O primeiro método do processo é fornecido com a entidade Person que será trabalhada. Simplesmente, ele retorna o dobro da idade dessa entrada. O segundo método do processo é chamado para cada partição e localiza todos os objetos Person com idades entre lowAge e highAge e retorna os dobros das idades.

```
Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

DoublePersonAgeAgent agent = new DoublePersonAgeAgent();

// make a list of keys
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);

// get the results for those entries
Map<Tuple, Object> = amgr.callMapAgent(agent, keyList);
```

Esse exemplo mostra um cliente que obtém um objeto Session e uma referência ao Mapa Person. A operação do agente é executada em um Mapa específico. A interface AgentManager é recuperada a partir de tal Mapa. Uma instância do agente a ser chamado é criada e qualquer estado necessário incluído no objeto pelos atributos da configuração; nesse caso, não há nenhum. Uma lista de chaves é construída. Um Mapa com os valores para pessoa 1 dobrados, e os mesmos valores para pessoa 2 são retornados.

Em seguida, o agente é chamado para esse conjunto de chaves. O método de processamento dos agentes é chamado em cada partição com algumas chaves especificadas na grade em paralelo. Um Mapa é retornado, fornecendo os resultados combinados para a chave especificada. Nesse caso, será retornado um Mapa com os valores, mantendo a idade da pessoa 1 dobrada e a mesma idade para a pessoa 2.

Mesmo que a chave não exista, o agente será chamado. Isso concede ao agente a oportunidade de criar a entrada do mapa. Se estiver utilizando EntityAgentMixin, a chave a ser processada não será a entidade, mas o valor real da chave Tupla da entidade. Se as chaves são desconhecidas, há a opção de consultar todas as partições para localizar objetos Person de um determinado formato e retornar suas idades em dobro. Veja um exemplo a seguir:

```
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);
```

O exemplo anterior mostra o AgentManager sendo obtido para o Mapa da Pessoa, e o agente construído e inicializado com as idades mínima e máxima para Pessoa de interesse. O agente é chamado, utilizando o método callMapAgent. Observe que nenhuma chave é fornecida. Em virtude disso, o ObjectGrid chama o agente em cada partição na grade em paralelo e, em seguida, retorna os resultados combinados para o cliente. Com isso, todos os objetos Person na grade com idades entre a menor e maior idades serão localizados, e as idades dos objetos Person calculadas em dobro. Isso mostra como que as APIs na grade podem ser utilizadas para executar uma consulta a fim de localizar as entidades que correspondem à determinada consulta. O agente é serializado de modo simples e transportado pelo ObjectGrid para as partições com as entradas necessárias. Os resultados são similarmente serializados para o transporte de volta ao cliente. É necessário ter cuidado com as APIs do Mapa. Se o ObjectGrid estiver hospedando tera bytes de objetos e executando em uma grande quantidade de servidores, as APIs podem sobrecarregar todos eles, menos as maiores máquinas executando o cliente. Isso deve ser utilizado para processar um subconjunto pequeno. Se for necessário processar um grande subconjunto, recomendamos utilizar um agente de redução para executar o processamento fora da grade em vez de num cliente.

Redução de Paralelo ou agentes de agregação

Este estilo de programação processa um subconjunto das entradas e calcula um único resultado para o grupo de entradas. Os exemplos de resultados podem ser:

- valor mínimo
- valor máximo
- alguma outra função específica do negócio

Um agente de redução é codificado e chamado de modo muito similar aos agentes Map.

Fluxo de chamada ReduceGridAgent

Quando o método AgentManager.callReduceAgent é chamado com um conjunto de chaves, a instância do ReduceGridAgent é serializada e enviada para cada partição principal que as chaves resolvem. Isto significa que quaisquer dados de instância armazenados no agente podem ser enviados para o servidor. Portanto, cada partição primária possui uma instância do agente. O método reduce(Session s, ObjectMap map, Collection keys) é chamado uma vez por instância (partição) com o subconjunto de chaves que resolve a partição. O resultado de cada método de redução é, então, serializado de volta para o cliente. O método reduceResults é chamado na instância do ReduceGridAgent do cliente com o conjunto de cada resultado de cada chamada de redução remota. O resultado do método reduceResults é retornado para o responsável pela chamada do método callReduceAgent.

Quando o método AgentManager.callReduceAgent é chamado sem um conjunto de chaves, o ReduceGridAgentinstance é serializado e enviado para cada partição principal. Isto significa que quaisquer dados de instância armazenados no agente podem ser enviados para o servidor. Portanto, cada partição primária possui uma instância do agente. O método reduce(Session s, ObjectMap map) é chamado uma vez por instância (partição). O resultado de cada método de redução é, então, serializado de volta para o cliente. O método reduceResults é chamado na instância do ReduceGridAgent do cliente com o conjunto de cada resultado de cada chamada de redução remota. O resultado do método reduceResults é retornado

para o responsável pela chamada do método callReduceAgent. Este é um exemplo de um agente de redução que simplesmente inclui as idades das entradas compatíveis.

```
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;
 }
}
```

O exemplo anterior mostra o agente. O agente tem três partes importantes. A primeira permite que um conjunto específico de entradas seja processado sem uma consulta. Ela simplesmente é repetida sobre o conjunto de entradas que inclui as idades. A soma é retornada do método. A segunda utiliza uma consulta para selecionar as entradas a serem agregadas. Em seguida, ela soma todas as idades Person correspondentes. O terceiro método é utilizado para agregar os resultados de cada partição a um único resultado. O ObjectGrid executa a agregação de entradas em paralelo por meio do grade. Cada partição produz um resultado intermediário que deve ser agregado aos resultados intermediários de outra partição. Esse terceiro método executa essa tarefa. No exemplo a seguir, o agente é chamado e as idades de todas as Pessoas com idades entre 10 e 20 exclusivamente são agregadas:

```
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);
```

Funções do agente

O agente é livre para fazer operações de ObjectMap ou EntityManager dentro do shard local onde está executando. O agente recebe uma Sessão e pode incluir, atualizar, consultar, ler ou remover dados da partição que a Sessão representa. Alguns aplicativos somente consultarão dados da grade, mas também é possível gravar um agente para aumentar todas as idades da Pessoa em 1 que correspondam a uma determinada consulta. Existe uma transação na Sessão quando o agente é chamado, e é consolidado quando o agente retorna, a menos que uma exceção seja lançada

Manipulação de erros

Se um agente de mapa for chamado com uma chave desconhecida, o valor retornado será um objeto de erro de implementação da interface EntryErrorValue.

Transações

Um agente de mapas é executado numa transação separada do cliente. As chamadas do agente podem ser agrupadas numa única transação. Se o agente falhar (emitir uma exceção), a transação será recuperada. Quaisquer agentes que executaram com êxito em uma transação retrocederão com o agente falho. O AgentManager executará novamente os agentes retrocedidos que executaram com êxito em uma nova transação.

Para obter mais informações, consulte a documentação da API do DataGrid.

Configurando Clientes Programaticamente

É possível configurar um cliente do WebSphere eXtreme Scale com base em seus requisitos, como a necessidade de substituir configurações.

Substituir plug-ins

É possível substituir os seguintes plug-ins em um cliente:

- **Plug-ins do ObjectGrid**
  - Plug-in TransactionCallback
  - Plug-in ObjectGridEventListener
- **Plug-ins do BackingMap**
  - Plug-in Evictor
  - Plug-in MapEventListener
  - Atributo numberOfBuckets
  - Atributo ttlEvictorType
  - Atributo timeToLive

Configurar o Cliente Programaticamente

Também é possível substituir as configurações do ObjectGrid do lado do cliente programaticamente. Crie um objeto ObjectGridConfiguration que seja semelhante em estrutura à instância ObjectGrid do lado do servidor. O código a seguir cria uma instância ObjectGrid do lado do cliente funcionalmente equivalente à substituição do cliente na seção anterior que utiliza um arquivo XML.

Substituição do lado do cliente programaticamente

```
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);
```

A instância ogManager da interface do ObjectGridManager verifica se há substituições apenas nos objetos ObjectGridConfiguration e BackingMapConfiguration que você inclui no Mapa overrideMap. Por exemplo, o código anterior substitui o número de depósitos no mapa OrderLine. Entretanto, o mapa Order permanece inalterado no lado do cliente porque nenhuma configuração para esse mapa é incluída.

Desativar o Cache Local do Cliente

O cache local é ativado por padrão quando um bloqueio é configurado como otimista ou nenhum. Os clientes não mantêm um cache local quando a configuração do bloqueio é configurada como pessimista. Para desativar o cache local, configure o atributo numberOfBuckets para 0 no arquivo descritor do ObjectGrid de substituição do cliente.

Ativando a Replicação de Mapas do Lado do Cliente

Você também pode ativar a replicação de mapas no lado do cliente para disponibilizar os dados mais rápido.

Com o eXtreme Scale, é possível replicar um mapa de servidor em um ou mais clientes utilizando a replicação assíncrona. Um cliente pode pedir uma cópia local somente leitura de um mapa de lado do servidor utilizando o método ClientReplicableMap.enableClientReplication.

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

O primeiro parâmetro é o modo de replicação. Esse modo pode ser uma replicação contínua ou uma replicação de captura instantânea. O segundo parâmetro é uma matriz de IDs de partição que representa as partições a partir das quais replicar os dados. Se o valor for nulo ou uma matriz vazia, os dados são replicados a partir de todas as partições. O último parâmetro é um listener para receber eventos de replicação de cliente. Consulte ClientReplicableMap e ReplicationMapListener na documentação da API para obter detalhes.

Depois de ativada a replicação, então o servidor começa a replicar o mapa para o cliente. O cliente eventualmente está apenas algumas transações atrás do servidor em questão de tempo.

Acessando Dados com o Serviço de Dados REST

Desenvolva aplicativos que executam operações usando protocolos do serviço de dados REST.

Conceitos relacionados

"Operações com o Serviço de Dados REST"
Após iniciar o serviço de dados REST do eXtreme Scale, é possível usar qualquer cliente HTTP para interagir com ele. Um navegador da Web, um cliente PHP, um cliente Java ou um cliente WCF Data Services podem ser utilizados para emitir quaisquer operações de pedido suportadas.

"Visão Geral do Serviço de Dados REST" na página 117
O serviço de dados REST WebSphere eXtreme Scale é um serviço HTTP Java compatível com Microsoft WCF Data Services (formalmente, ADO.NET Data Services) e implementa o Open Data Protocol (OData). O Microsoft WCF Data Services é compatível com essa especificação quando utiliza Visual Studio 2008 SP1 e .NET Framework 3.5 SP1.

Referências relacionadas

"Simultaneidade Otimista no Serviço de Dados REST" na página 273
O serviço de dados REST do eXtreme Scale usa um modelo de bloqueio otimista ao usar cabeçalhos HTTP nativos: If-Match, If-None-Match e ETag. Esses cabeçalhos são enviados em mensagens de pedido e de resposta para retransmitir informações da versão da entidade do servidor para o cliente e do cliente para o servidor.

"Protocolos de Pedido para o Serviço de Dados REST" na página 274
No geral, os protocolos para interação com o serviço REST são os mesmos que os descritos no protocolo AtomPub de Serviços de Dados WCF. No entanto, o eXtreme Scale fornece detalhes adicionais, da perspectiva Modelo de Entidade do eXtreme Scale. Os usuários devem estar familiarizados com os protocolos WCF Data Services antes de lerem esta seção. Alternativamente, os usuários podem ler esta seção com a seção do protocolo WCF Data Services.

"Recuperar Pedidos com Serviço de Dados REST" na página 275
Um Pedido RetrieveEntity é usado por um cliente para recuperar uma entidade do eXtreme Scale. A carga útil da resposta contém os dados da entidade no formato AtomPub ou JSON. Além disso, o operador do sistema $expand pode ser utilizado para expandir as relações. As relações são representadas em sequência dentro da resposta do serviço de dados como um Atom Feed Document, que é uma relação para-muitos, ou um Atom Entry Document que é uma relação para-um.

"Recuperando Não Entidades com Serviços de Dados REST" na página 282
O serviço de dados REST permite recuperar mais que apenas entidades, como coletas e propriedades das entidades.

"Pedidos de Inserção com Serviço de Dados REST" na página 288
Um Pedido InsertEntity pode ser usado para inserir uma nova instância de entidade do eXtreme Scale, potencialmente com novas entidades relacionadas, no serviço de dados REST do eXtreme Scale.

"Pedidos de Atualização com Serviço de Dados REST" na página 292
O serviço de dados REST do WebSphere eXtreme Scale suporta pedidos de atualização para entidades, propriedades de primitivas de entidades e assim por diante.

"Pedidos de Exclusão com Serviços de Dados REST" na página 297
O serviço de dados REST do WebSphere eXtreme Scale pode excluir entidades, valores da propriedade e links.

Operações com o Serviço de Dados REST

Após iniciar o serviço de dados REST do eXtreme Scale, é possível usar qualquer cliente HTTP para interagir com ele. Um navegador da Web, um cliente PHP, um cliente Java ou um cliente WCF Data Services podem ser utilizados para emitir quaisquer operações de pedido suportadas.

O serviço REST implementa um subconjunto da especificação Microsoft Atom Publishing Protocol: Data Services URI and Payload Extensions, Versão 1.0, que faz parte do protocolo OData. Este tópico descreve quais dos recursos da especificação são suportados e como eles são mapeados para o eXtreme Scale.

URI da Raiz do Serviço

O Microsoft WCF Data Services normalmente define um serviço por origem de dados ou modelo de entidade. O serviço de dados REST do eXtreme Scale define um serviço por ObjectGrid definido. Cada ObjectGrid que é definido no arquivo XML de substituição de cliente do ObjectGrid do eXtreme Scale é automaticamente exposto como uma raiz de serviço REST separada.

O URI para a raiz do serviço é:

`http://`*`host`*`:`*`port`*`/`*`contextroot`*`/restservice/`*`gridname`*

Em que:

- *contextroot* é definido quando você implementa o aplicativo de serviço de dados REST e depende do servidor de aplicativos
- *gridname* é o nome do ObjectGrid

Tipos de Pedidos

A lista a seguir descreve os tipos de pedidos do Microsoft WCF Data Services suportados pelo serviço de dados REST do eXtreme Scale. Para obter detalhes sobre cada tipo de pedido suportado pelo WCF Data Services, consulte: MSDN: Tipos de Pedidos.

Tipos de pedido de inserção

Clientes podem inserir recursos utilizando o verbo POST HTTP com as seguintes limitações:

- Pedido InsertEntity: Suportado.
- Pedido InsertLink: Suportado.
- Pedido InsertMediaResource: Não suportado devido à restrição de suporte do recurso de mídia.

Para obter informações adicionais, consulte MSDN: Inserir Tipos de Pedidos.

Tipos de pedido de atualização

Clientes podem atualizar recursos utilizando os verbos PUT e MERGE HTTP com as seguintes limitações:

- Pedido UpdateEntity: Suportado.
- Pedido UpdateComplexType: Não suportado devido à restrição de tipo complexo.
- Pedido UpdatePrimitiveProperty: Suportado.
- Pedido UpdateValue: Suportado.
- Pedido UpdateLink: Suportado.
- Pedido UpdateMediaResource: Não suportado devido à restrição de suporte do recurso de mídia.

Para obter informações adicionais, consulte: MSDN: Inserir Tipos de Pedidos.

Tipos de pedido de exclusão

Clientes podem excluir recursos utilizando o verbo DELETE HTTP com as seguintes limitações:

- Pedido DeleteEntity: Suportado.
- Pedido DeleteLink: Suportado.
- Pedido DeleteValue: Suportado.

Para obter informações adicionais, consulte: MSDN: Excluir Tipos de Pedidos.

Tipos de pedido de recuperação

Clientes podem recuperar recursos utilizando o verbo GET HTTP com as seguintes limitações:

- Pedido RetrieveEntitySet: Suportado.
- Pedido RetrieveEntity: Suportado.
- Pedido RetrieveComplexType: Não suportado devido à restrição de tipo complexo.
- Pedido RetrievePrimitiveProperty: Suportado.
- Pedido RetrieveValue: Suportado.
- Pedido RetrieveServiceMetadata: Suportado.
- Pedido RetrieveServiceDocument: Suportado.
- Pedido RetrieveLink: Suportado.
- Pedido Retrieve Contendo um Mapeamento de Feed Customizável: Não Suportado.
- RetrieveMediaResource: Não suportado devido à restrição de recurso de mídia.

Para obter informações adicionais, consulte: MSDN: Recuperar Tipos de Pedidos.

Opções de consulta do sistema

São suportadas consultas que permitem que clientes identifiquem uma coleta de entidades ou uma única entidade. As opções de consulta do sistema são especificadas em um URI de serviço de dados e são suportadas com as seguintes limitações:

- $expand: Suportada.
- $filter: Suportada.
- $orderby: Suportada.
- $format: Não suportada. O formato aceitável é identificado no cabeçalho do pedido HTTP Accept.
- $skip: Suportada.
- $top: Suportada.

Para obter informações adicionais, consulte: MSDN: Opções de Consulta do Sistema.

Roteamento de partição

O roteamento de partição é baseado na entidade raiz. Um URI de pedido infere uma entidade raiz se o caminho do recurso começar com uma entidade raiz ou com uma entidade que tenha uma associação direta ou indireta com a entidade. Em um ambiente particionado, qualquer pedido que não possa inferir uma entidade raiz será rejeitado. Qualquer pedido que infira uma entidade raiz será roteado para a partição correta.

Para obter informações adicionais sobre como definir um esquema com associações e entidades-raiz, consulte o Modelo de Dados Escalável no eXtreme Scale e o Particionamento.

Pedido de Chamada

Pedidos de chamada não são suportados. Para obter informações adicionais, consulte MSDN: Pedido de Chamada.

Pedido em Lote

Clientes podem criar lotes de vários Conjuntos de Mudanças ou Operações de Consulta dentro de um único pedido. Isso pode reduzir o número de roundtrips para o servidor e permite que vários pedidos participem de uma única transação. Para obter informações adicionais, consulte MSDN: Pedido em Lote.

Pedidos em Túnel

Pedidos em túnel não são suportados. Para obter informações adicionais, consulte MSDN: Pedidos em Túnel.

Tarefas relacionadas

"Acessando Dados com o Serviço de Dados REST" na página 268
Desenvolva aplicativos que executam operações usando protocolos do serviço de dados REST.

Referências relacionadas

"Simultaneidade Otimista no Serviço de Dados REST"
O serviço de dados REST do eXtreme Scale usa um modelo de bloqueio otimista ao usar cabeçalhos HTTP nativos: If-Match, If-None-Match e ETag. Esses cabeçalhos são enviados em mensagens de pedido e de resposta para retransmitir informações da versão da entidade do servidor para o cliente e do cliente para o servidor.

"Protocolos de Pedido para o Serviço de Dados REST" na página 274
No geral, os protocolos para interação com o serviço REST são os mesmos que os descritos no protocolo AtomPub de Serviços de Dados WCF. No entanto, o eXtreme Scale fornece detalhes adicionais, da perspectiva Modelo de Entidade do eXtreme Scale. Os usuários devem estar familiarizados com os protocolos WCF Data Services antes de lerem esta seção. Alternativamente, os usuários podem ler esta seção com a seção do protocolo WCF Data Services.

"Recuperar Pedidos com Serviço de Dados REST" na página 275
Um Pedido RetrieveEntity é usado por um cliente para recuperar uma entidade do eXtreme Scale. A carga útil da resposta contém os dados da entidade no formato AtomPub ou JSON. Além disso, o operador do sistema $expand pode ser utilizado para expandir as relações. As relações são representadas em sequência dentro da resposta do serviço de dados como um Atom Feed Document, que é uma relação para-muitos, ou um Atom Entry Document que é uma relação para-um.

"Recuperando Não Entidades com Serviços de Dados REST" na página 282
O serviço de dados REST permite recuperar mais que apenas entidades, como coletas e propriedades das entidades.

"Pedidos de Inserção com Serviço de Dados REST" na página 288
Um Pedido InsertEntity pode ser usado para inserir uma nova instância de entidade do eXtreme Scale, potencialmente com novas entidades relacionadas, no serviço de dados REST do eXtreme Scale.

"Pedidos de Atualização com Serviço de Dados REST" na página 292
O serviço de dados REST do WebSphere eXtreme Scale suporta pedidos de atualização para entidades, propriedades de primitivas de entidades e assim por diante.

"Pedidos de Exclusão com Serviços de Dados REST" na página 297
O serviço de dados REST do WebSphere eXtreme Scale pode excluir entidades, valores da propriedade e links.

Simultaneidade Otimista no Serviço de Dados REST

O serviço de dados REST do eXtreme Scale usa um modelo de bloqueio otimista ao usar cabeçalhos HTTP nativos: If-Match, If-None-Match e ETag. Esses cabeçalhos são enviados em mensagens de pedido e de resposta para retransmitir informações da versão da entidade do servidor para o cliente e do cliente para o servidor.

Para obter mais detalhes sobre a simultaneidade otimista, consulte Biblioteca MSDN: Simultaneidade Otimista (ADO.NET).

O serviço de dados REST do eXtreme Scale ativará a simultaneidade otimista para uma entidade se um atributo de versão for definido no esquema de entidade para essa entidade. Uma propriedade de versão pode ser definida no esquema da

entidade por uma anotação @Version para classes Java ou um atributo <version/> para entidades definidas com o uso de um arquivo XML do descritor de entidade. O serviço de dados REST do eXtreme Scale propaga automaticamente o valor da propriedade da versão para o cliente no cabeçalho ETag para respostas únicas da entidade usando um atributo m:etag na carga útil para várias respostas XML da entidade e um atributo etag na carga útil para várias respostas JSON da entidade.

Para obter mais detalhes sobre como definir um esquema de entidade do eXtreme Scale, consulte "Definindo um Esquema de Entidade" na página 169.

Conceitos relacionados

"Operações com o Serviço de Dados REST" na página 269
Após iniciar o serviço de dados REST do eXtreme Scale, é possível usar qualquer cliente HTTP para interagir com ele. Um navegador da Web, um cliente PHP, um cliente Java ou um cliente WCF Data Services podem ser utilizados para emitir quaisquer operações de pedido suportadas.

"Visão Geral do Serviço de Dados REST" na página 117
O serviço de dados REST WebSphere eXtreme Scale é um serviço HTTP Java compatível com Microsoft WCF Data Services (formalmente, ADO.NET Data Services) e implementa o Open Data Protocol (OData). O Microsoft WCF Data Services é compatível com essa especificação quando utiliza Visual Studio 2008 SP1 e .NET Framework 3.5 SP1.

Tarefas relacionadas

"Acessando Dados com o Serviço de Dados REST" na página 268
Desenvolva aplicativos que executam operações usando protocolos do serviço de dados REST.

Protocolos de Pedido para o Serviço de Dados REST

No geral, os protocolos para interação com o serviço REST são os mesmos que os descritos no protocolo AtomPub de Serviços de Dados WCF. No entanto, o eXtreme Scale fornece detalhes adicionais, da perspectiva Modelo de Entidade do eXtreme Scale. Os usuários devem estar familiarizados com os protocolos WCF Data Services antes de lerem esta seção. Alternativamente, os usuários podem ler esta seção com a seção do protocolo WCF Data Services.

São fornecidos exemplos para ilustrar o pedido e a resposta. Esses exemplos aplicam-se ao serviço de dados REST e aos Serviços de Dados WCF do eXtreme Scale. Como os navegadores da Web podem apenas recuperar dados, as operações CUD (create, update and delete) devem ser executadas por outro cliente, como Java, JavaScript, RUBY ou PHP.

Conceitos relacionados

"Operações com o Serviço de Dados REST" na página 269
Após iniciar o serviço de dados REST do eXtreme Scale, é possível usar qualquer cliente HTTP para interagir com ele. Um navegador da Web, um cliente PHP, um cliente Java ou um cliente WCF Data Services podem ser utilizados para emitir quaisquer operações de pedido suportadas.

"Visão Geral do Serviço de Dados REST" na página 117
O serviço de dados REST WebSphere eXtreme Scale é um serviço HTTP Java compatível com Microsoft WCF Data Services (formalmente, ADO.NET Data Services) e implementa o Open Data Protocol (OData). O Microsoft WCF Data Services é compatível com essa especificação quando utiliza Visual Studio 2008 SP1 e .NET Framework 3.5 SP1.

Tarefas relacionadas

"Acessando Dados com o Serviço de Dados REST" na página 268
Desenvolva aplicativos que executam operações usando protocolos do serviço de dados REST.

Recuperar Pedidos com Serviço de Dados REST

Um Pedido RetrieveEntity é usado por um cliente para recuperar uma entidade do eXtreme Scale. A carga útil da resposta contém os dados da entidade no formato AtomPub ou JSON. Além disso, o operador do sistema $expand pode ser utilizado para expandir as relações. As relações são representadas em sequência dentro da resposta do serviço de dados como um Atom Feed Document, que é uma relação para-muitos, ou um Atom Entry Document que é uma relação para-um.

Dica: Para obter mais detalhes sobre o protocolo RetrieveEntity definido no WCF Data Services, consulte MSDN: Solicitação RetrieveEntity.

Recuperando uma Entidade

O exemplo de RetrieveEntity a seguir recupera uma entidade Customer com uma chave.

AtomPub

- Método

  GET
- URI do Pedido:

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')
- Cabeçalho do Pedido:

  Aceitar: application/atom+xml
- Carga Útil de Pedido:

  Nenhum
- Cabeçalho da Resposta:

  Tipo de Conteúdo: application/atom+xml
- Carga Útil de Resposta:

```
<?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 Resposta:

  200 OK

JSON

- Método

  GET

- URI do Pedido:

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')

- Cabeçalho do Pedido:

  Aceitar: application/json

- Carga Útil de Pedido:

  Nenhum

- Cabeçalho da Resposta:

  Tipo de Conteúdo: application/json

- Carga Útil de Resposta:

```
{"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 Resposta:

  200 OK

Consultas

Uma consulta também pode ser utilizada com um pedido RetrieveEntitySet ou RetrieveEntity. Uma consulta é especificada pelo operador $filter do sistema.

Para obter detalhes sobre o operador $filter, consulte: MSDN: Opção de Consulta do Sistema de Filtros ($filter)

O protocolo OData suporta várias expressões comuns. O serviço de dados REST do eXtreme Scale suporta um subconjunto de expressões definidas na especificação:

- Expressões Booleanas:
  - eq, ne, lt, le, gt, ge
  - negate
  - not
  - parenthesis
  - and, or
- Expressões Aritméticas:
  - add
  - sub
  - mul
  - div
- Literais de Primitivas
  - Seqüência de Caracteres
  - date-time
  - decimal
  - único
  - double
  - int16
  - int32
  - int64
  - binário
  - null
  - byte

As expressões a seguir *não* estão disponíveis:

- Expressões Booleanas:
  - isof
  - cast
- Expressões de Chamada de Método
- Expressões Aritméticas:
  - mod
- Literais de primitivas:
  - Guid
- Expressões de Membro

Para obter uma lista e uma descrição completas das expressões que estão disponíveis no Microsoft WCF Data Services, consulte a seção 2.2.3.6.1.1: Sintaxe de Expressão Comum.

O exemplo a seguir demonstra um pedido RetrieveEntity com uma consulta. Neste exemplo, todos os clientes cujo nome do contato é “RoadRunner” são recuperados. O único cliente que corresponde a esse filtro é Customer('ACME'), conforme mostrado na carga útil da resposta.

Restrição: Essa consulta só funcionará para entidades não particionadas. Se Customer for particionada, a chave pertencente ao cliente será necessária.

AtomPub

- Método: GET
- URI de Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?$filter=contactName eq 'RoadRunner'
- Cabeçalho do Pedido: Aceitar: application/atom+xml
- Carga Útil de Entrada: Nenhuma
- Cabeçalho de Resposta: Conteúdo-Tipo: application/atom+xml
- Carga Útil de Resposta:

```
<?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 Resposta: 200 OK

JSON

- Método: GET
- URI do Pedido:

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?$filter=contactName eq 'RoadRunner'
- Cabeçalho do Pedido: Aceitar: application/json
- Carga Útil de Pedido: Nenhuma
- Cabeçalho de Resposta: Conteúdo-Tipo: application/json
- Carga Útil de Resposta:

```
{"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 Resposta: 200 OK

Operador do Sistema $expand

O operador do sistema $expand pode ser utilizado para expandir associações. As associações são representadas em linha na resposta do serviço de dados. As associações com diversos valores (para-muitos) são representadas como um Atom Feed Document ou matriz JSON. As associações com valor único (para-um) são representadas como um Atom Entry Document ou objeto JSON.

Para obter mais detalhes sobre o operador do sistema $expand, consulte Expandir Opção de Consulta do Sistema ($expand).

Veja aqui um exemplo de uso do operador do sistema $expand. Neste exemplo, recuperamos a entidade Customer('IBM') com Orders 5000, 5001 e outras associadas a ela. A cláusula $expand está configurada como “orders”, portanto, a coleta de pedidos será expandida como sequencial na carga útil da resposta. Apenas os pedidos 5000 e 5001 são exibidos aqui.

AtomPub

- Método: GET
- URI do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders
- Cabeçalho do Pedido: Aceitar: application/atom+xml
- Carga Útil de Pedido: Nenhuma
- Cabeçalho de Resposta: Conteúdo-Tipo: application/atom+xml
- Carga Útil de Resposta:

```
<?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 Resposta: 200 OK

JSON

- Método: GET
- URI do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders
- Cabeçalho do Pedido: Aceitar: application/json
- Carga Útil de Pedido: Nenhuma
- Cabeçalho de Resposta: Conteúdo-Tipo: application/json
- Carga Útil de Resposta:

```
{"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 Resposta: 200 OK

Conceitos relacionados

"Operações com o Serviço de Dados REST" na página 269
Após iniciar o serviço de dados REST do eXtreme Scale, é possível usar qualquer cliente HTTP para interagir com ele. Um navegador da Web, um cliente PHP, um cliente Java ou um cliente WCF Data Services podem ser utilizados para emitir quaisquer operações de pedido suportadas.

"Visão Geral do Serviço de Dados REST" na página 117
O serviço de dados REST WebSphere eXtreme Scale é um serviço HTTP Java compatível com Microsoft WCF Data Services (formalmente, ADO.NET Data Services) e implementa o Open Data Protocol (OData). O Microsoft WCF Data Services é compatível com essa especificação quando utiliza Visual Studio 2008 SP1 e .NET Framework 3.5 SP1.

Tarefas relacionadas

"Acessando Dados com o Serviço de Dados REST" na página 268
Desenvolva aplicativos que executam operações usando protocolos do serviço de dados REST.

Recuperando Não Entidades com Serviços de Dados REST

O serviço de dados REST permite recuperar mais que apenas entidades, como coletas e propriedades das entidades.

Recuperar uma Coleta de Entidades

Um Pedido RetrieveEntitySet pode ser usado por um cliente para recuperar um conjunto de entidades do eXtreme Scale. As entidades são representadas como um Atom Feed Document ou uma matriz JSON na carga útil da resposta. Para obter mais detalhes sobre o protocolo RetrieveEntitySet definido em Serviços de Dados WCF, consulte MSDN: Pedido RetrieveEntitySet.

O exemplo de pedido RetrieveEntitySet a seguir recupera todas as entidades Order associadas à entidade Customer('IBM'). Apenas os pedidos 5000 e 5001 são exibidos aqui.

AtomPub

- Método: GET
- URI de Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders
- Cabeçalho do Pedido: Aceitar: application/atom+xml
- Carga Útil de Pedido: Nenhuma
- Cabeçalho de Resposta: Conteúdo-Tipo: application/atom+xml
- Carga Útil de Resposta:

```
<?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 Resposta: 200 OK

JSON

- Método: GET
- URI de Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders
- Cabeçalho do Pedido: Aceitar: application/json
- Carga Útil de Pedido: Nenhuma
- Cabeçalho de Resposta: Conteúdo-Tipo: application/json
- Carga Útil de Resposta:

```
{"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 Resposta: 200 OK

Recuperar uma Propriedade

Um pedido RetrievePrimitiveProperty pode ser usado para obter o valor de uma propriedade de uma instância de entidade do eXtreme Scale. O valor da propriedade é representado como formato XML para pedidos AtomPub e objeto JSON para pedidos JSON na carga útil de resposta. Para obter mais detalhes sobre o pedido RetrievePrimitiveProperty, consulte MSDN: Pedido RetrievePrimitiveProperty.

O exemplo de pedido RetrievePrimitiveProperty a seguir recupera a propriedade contactName da entidade Customer('IBM').

AtomPub

- Método: GET
- URI do pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Cabeçalho do Pedido: Aceitar: application/xml
- Carga Útil de Pedido: Nenhuma
- Cabeçalho de Resposta: Conteúdo-Tipo: application/atom+xml
- Carga Útil de Resposta:

```
<contactName
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
    John Doe
</contactName>
```

- Código de Resposta: 200 OK

JSON

- Método: GET
- URI do pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Cabeçalho do Pedido: Aceitar: application/json
- Carga Útil de Pedido: Nenhuma
- Cabeçalho de Resposta: Conteúdo-Tipo: application/json
- Carga Útil de Resposta: `{"d":{"contactName":"John Doe"}}`
- Código de Resposta: 200 OK

Recuperar um Valor da Propriedade

Um pedido RetrieveValue pode ser usado para obter o valor bruto de uma propriedade em uma instância de entidade do eXtreme Scale. O valor da propriedade é representado como um valor bruto na carga útil da resposta. Se o tipo de entidade for um dos seguintes, o tipo de mídia da resposta será “text/plain." Caso contrário, o tipo de mídia da resposta será “application/octet-stream." Esses tipos são:

- Tipos primitivos Java e seus respectivos wrappers
- 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

Para obter mais detalhes sobre o pedido RetrieveValue, consulte MSDN: Pedido RetrieveValue.

O seguinte exemplo de pedido RetrieveValue recupera o valor bruto da propriedade contactName da entidade Customer('IBM').

- Método de Pedido: GET
- URI do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName/$value
- Cabeçalho do Pedido: Aceitar: text/plain
- Carga Útil de Pedido: Nenhuma
- Cabeçalho da Resposta: Tipo de Conteúdo: text/plain
- Carga Útil da Resposta: John Doe
- Código de Resposta: 200 OK

Recuperar um Link

Um pedido RetrieveLink pode ser utilizado para obter o(s) link(s) representando uma associação para-um ou uma associação para-muitos. Para a associação to-one, o link é de uma instância de Entidade do eXtreme Scale para outra e o link é representado na carga útil da resposta. Para a associação to-many, os links são de uma instância de Entidade do eXtreme Scale para todas as outras em uma coleta de entidades do eXtreme Scale especificadas e a resposta é representada como um conjunto de links na carga útil da resposta. Para obter mais detalhes sobre o pedido RetrieveLink, consulte MSDN: Pedido RetrieveLink.

Veja aqui um exemplo de pedido RetrieveLink. Neste exemplo, recuperamos a associação entre a entidade Order(orderId=5000,customer_customerId='IBM') e seu cliente. A resposta mostra o URI da entidade Customer.

AtomPub

- Método: GET
- URI do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- Cabeçalho do Pedido: Aceitar: application/xml
- Carga Útil de Pedido: Nenhuma
- Cabeçalho da Resposta: Tipo de Conteúdo: application/xml
- Carga Útil de Resposta:

```
<?xml version="1.0" encoding="utf-8"?>
<uri>http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('IBM')</uri>
```

- Código de Resposta: 200 OK

JSON

- Método: GET
- URI do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- Cabeçalho do Pedido: Aceitar: application/json
- Carga Útil de Pedido: Nenhuma
- Cabeçalho de Resposta: Conteúdo-Tipo: application/json
- Carga Útil da Resposta: {"d":{"uri":"http:\/\/localhost:8080\/wxsrestservice\/restservice\/NorthwindGrid\/Customer('IBM')"}}

Recuperar Metadados do Serviço

Um Pedido RetrieveServiceMetadata pode ser usado para obter o documento Conceptual Schema Definition Language (CSDL), que descreve o modelo de dados associado ao serviço de dados REST do eXtreme Scale. Para obter mais detalhes sobre o pedido RetrieveServiceMetadata, consulte MSDN: Pedido RetrieveServiceMetadata.

Recuperar Documento de Serviço

Um Pedido RetrieveServiceDocument pode ser usado para recuperar o Documento de Serviço que descreve a coleta de recursos expostos pelo serviço de dados REST do eXtreme Scale. Para obter mais detalhes sobre o pedido RetrieveServiceDocument, consulte MSDN: Pedido RetrieveServiceDocument.

Conceitos relacionados

"Operações com o Serviço de Dados REST" na página 269
Após iniciar o serviço de dados REST do eXtreme Scale, é possível usar qualquer cliente HTTP para interagir com ele. Um navegador da Web, um cliente PHP, um cliente Java ou um cliente WCF Data Services podem ser utilizados para emitir quaisquer operações de pedido suportadas.

"Visão Geral do Serviço de Dados REST" na página 117
O serviço de dados REST WebSphere eXtreme Scale é um serviço HTTP Java compatível com Microsoft WCF Data Services (formalmente, ADO.NET Data Services) e implementa o Open Data Protocol (OData). O Microsoft WCF Data Services é compatível com essa especificação quando utiliza Visual Studio 2008 SP1 e .NET Framework 3.5 SP1.

Tarefas relacionadas

"Acessando Dados com o Serviço de Dados REST" na página 268
Desenvolva aplicativos que executam operações usando protocolos do serviço de dados REST.

Pedidos de Inserção com Serviço de Dados REST

Um Pedido InsertEntity pode ser usado para inserir uma nova instância de entidade do eXtreme Scale, potencialmente com novas entidades relacionadas, no serviço de dados REST do eXtreme Scale.

Pedido Inserir Entidade

Um Pedido InsertEntity pode ser usado para inserir uma nova instância de entidade do eXtreme Scale, potencialmente com novas entidades relacionadas, no serviço de dados REST do eXtreme Scale. Ao inserir uma entidade, o cliente pode especificar se o recurso ou a entidade devem ser vinculados automaticamente a outras entidades existentes no serviço de dados.

O cliente deve incluir as informações sobre ligação necessárias na representação da relação associada na carga útil do pedido.

Além do suporte à inserção de uma nova instância EntityType (E1), o pedido InsertEntity também permite a inserção de novas entidades relacionadas à E1 (descritas por qualquer relação de entidade) em um único Pedido. Por exemplo, ao inserir um Customer('IBM'), podemos inserir todos os pedidos com Customer('IBM'). Esta forma de um Pedido InsertEntity também é conhecida como *inserção profunda*. Com uma inserção profunda, as entidades relacionadas devem ser representadas usando a representação sequencial da relação associada com E1 que identifica o link para as entidades relacionadas a serem inseridas.

As propriedades da entidade a ser inserida são especificadas na carga útil do pedido. As propriedades são analisadas pelo serviço de dados REST e configuradas para a propriedade correspondente na instância da entidade. Para o formato AtomPub, a propriedade é especificada como um elemento XML <d:PROPERTY_NAME>. Para JSON, a propriedade é especificada como uma propriedade de um objeto JSON.

Se uma propriedade estiver ausente na carga útil do pedido, o serviço de dados REST irá configurar o valor da propriedade da entidade para o valor padrão java. Entretanto, o banco de dados backend deve rejeitar um valor padrão, por exemplo, se a coluna não for anulável no banco de dados. Um código de resposta 500 será retornado para indicar um erro do Servidor Interno.

Se houver propriedades duplicadas especificadas na carga útil, a última propriedade será utilizada. Todos os valores anteriores para o mesmo nome da propriedade serão ignorados pelo serviço de dados REST.

Se a carga útil contiver uma propriedade não existente, o serviço de dados REST retornará um código de resposta 400 (Pedido Inválido) para indicar que o pedido enviado pelo cliente estava sintaticamente incorreto.

Se as propriedades-chave estiverem ausentes, o serviço de dados REST retornará um código de resposta 400 (Pedido Inválido) para indicar uma propriedade-chave ausente.

Se a carga útil contiver um link para uma entidade relacionada com uma chave não existente, o serviço de dados REST retornará um código de resposta 404 (Não Localizado) para indicar que a entidade vinculada não pode ser localizada.

Se a carga útil contiver um link para uma entidade relacionada com um nome de associação incorreto, o serviço de dados REST retornará um código de resposta 400 (Pedido Inválido) para indicar que o link não pode ser localizado.

Se a carga útil contiver mais de um link para uma relação para-um, o último link será utilizado. Todos os links anteriores para a mesma associação serão ignorados.

Para obter mais detalhes sobre o pedido InsertEntity, consulte Biblioteca MSDN: Pedido InsertEntity.

Um pedido InsertEntity insere uma entidade Customer com a chave 'IBM'.

AtomPub

- Método: POST
- URI do pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Cabeçalho do pedido: Aceitar: application/atom+xml Content-Type: application/atom+xml
- Carga Útil de Pedido:

```
<?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>
```

- Cabeçalho de Resposta: Conteúdo-Tipo: application/atom+xml
- Carga Útil de Resposta:

```
<?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>
Cabeçalho da Resposta:
Tipo de Conteúdo: application/atom+xml
Carga
Útil de Resposta:
<?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 Resposta: 201 Criado

JSON

- Método: POST
- URI do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer
- Cabeçalho do Pedido: Aceitar: application/json Content-Type: application/json

- Carga Útil de Pedido:

```
{"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country": "USA",}
```

- Cabeçalho de Resposta: Conteúdo-Tipo: application/json
- Carga Útil de Resposta:

```
{"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 Resposta: 201 Criado

Pedido Inserir Link

Um Pedido InsertLink pode ser usado para criar um novo Link entre duas instâncias de entidade do eXtreme Scale. A URI do pedido deve resolver para uma associação to-many do eXtreme Scale. A carga útil do pedido contém um único link que aponta para a entidade de destino da associação para-muitos.

Se o URI do pedido InsertLink representar uma associação para-um, o serviço de dados REST retornará uma resposta 400 (Pedido Inválido).

Se o URI do pedido InsertLink apontar para uma associação que não existe, o serviço de dados REST retornará uma resposta 404 (Não Localizado) para indicar que o link não foi localizado.

Se a carga útil contiver um link com uma chave que não existe, o serviço de dados REST retornará uma resposta 404 (Não Localizado) para indicar que a entidade vinculada não pode ser localizada.

Se a carga útil contiver mais de um link, o Serviço de Dados REST do eXtreme Scale analisará o primeiro link. Os links restantes serão ignorados.

Para obter mais detalhes sobre o pedido InsertLink, consulte: Biblioteca MSDN: Pedido InsertLink.

O seguinte exemplo de pedido InsertLink cria um link de Customer('IBM') para Order(orderId=5000,customer_customerId='IBM').

AtomPub

- Método: POST
- URI do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$link/orders
- Cabeçalho do Pedido: Tipo de Conteúdo: application/xml
- Carga Útil de Pedido:

```
<?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 Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

JSON

- Método: POST
- URI do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$links/orders
- Cabeçalho do Pedido: Tipo de Conteúdo: application/json
- Carga Útil de Pedido:

```
{"uri":
"http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId
    =5000,customer_customerId='IBM')"}
```

- Carga Útil de Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

Conceitos relacionados

"Operações com o Serviço de Dados REST" na página 269
Após iniciar o serviço de dados REST do eXtreme Scale, é possível usar qualquer cliente HTTP para interagir com ele. Um navegador da Web, um cliente PHP, um cliente Java ou um cliente WCF Data Services podem ser utilizados para emitir quaisquer operações de pedido suportadas.

"Visão Geral do Serviço de Dados REST" na página 117
O serviço de dados REST WebSphere eXtreme Scale é um serviço HTTP Java compatível com Microsoft WCF Data Services (formalmente, ADO.NET Data Services) e implementa o Open Data Protocol (OData). O Microsoft WCF Data Services é compatível com essa especificação quando utiliza Visual Studio 2008 SP1 e .NET Framework 3.5 SP1.

Tarefas relacionadas

"Acessando Dados com o Serviço de Dados REST" na página 268
Desenvolva aplicativos que executam operações usando protocolos do serviço de dados REST.

Pedidos de Atualização com Serviço de Dados REST

O serviço de dados REST do WebSphere eXtreme Scale suporta pedidos de atualização para entidades, propriedades de primitivas de entidades e assim por diante.

Atualizar uma Entidade

Um Pedido UpdateEntity pode ser usado para atualizar uma entidade existente do eXtreme Scale. O cliente pode usar um método HTTP PUT para substituir ume entidade existente do eXtreme Scale ou usar um método HTTP MERGE para mesclar as mudanças em uma entidade existente do eXtreme Scale.

Ao atualizar a entidade, o cliente pode especificar se a entidade, além de ser atualizada, deve ser vinculada automaticamente a outras entidades existentes no serviço de dados que estão relacionadas por meio de associações com valor único (para um).

A propriedade da entidade a ser atualizada está na carga útil do pedido. A propriedade é analisada pelo serviço de dados REST e configurada para a

propriedade correspondente na entidade. Para o formato AtomPub, a propriedade é especificada como um elemento XML <d:PROPERTY_NAME>. Para JSON, a propriedade é especificada como uma propriedade de um objeto JSON.

Se uma propriedade estiver ausente na carga útil de solicitação, o serviço de dados REST configurará o valor de propriedade da entidade com o valor padrão Java para o método HTTP PUT. Entretanto, o banco de dados backend deve rejeitar um valor padrão, por exemplo, se a coluna não for anulável no banco de dados. Então, um código de resposta 500 (Erro do Servidor Interno) é retornado para indicar um Erro do Servidor Interno. Se uma propriedade estiver ausente na carga útil de solicitação HTTP MERGE, o serviço de dados REST não alterará o valor da propriedade existente.

Se houver propriedades duplicadas especificadas na carga útil, a última propriedade será usada. Todos os valores anteriores com o mesmo nome da propriedade serão ignorados pelo serviço de dados REST.

Se a carga útil contiver uma propriedade não existente, o serviço de dados REST irá retornar um código de resposta 400 (Pedido Inválido) para indicar que o pedido enviado pelo cliente estava sintaticamente incorreto.

Como parte da serialização de um recurso, se a carga útil de um pedido de Atualização contiver alguma das propriedades-chave para a entidade, o serviço de dados REST irá ignorar esses valores de chave já que as chaves de entidades são imutáveis.

Para obter detalhes sobre o pedido UpdateEntity, consulte: Biblioteca MSDN: Pedido UpdateEntity.

Um pedido UpdateEntity atualiza o nome da cidade de Customer('IBM') para 'Raleigh'.

AtomPub

- Método: PUT
- URI do pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Cabeçalho do Pedido: Tipo de Conteúdo: application/atom+xml
- Carga Útil de Pedido:

```
<?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 Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

JSON

- Método: PUT
- URI do pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Cabeçalho do Pedido: Tipo de Conteúdo: application/json
- Carga Útil de Pedido:

```
{"customerId":"IBM",
"city":"Raleigh",
"companyName":"IBM Corporation",
"contactName":"Big Blue",
"country": "USA",}
```

- Carga Útil de Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

Atualizar uma Propriedade Primitiva da Entidade

O Pedido UpdatePrimitiveProperty pode atualizar um valor da propriedade de uma entidade do eXtreme Scale. A propriedade e o valor a serem atualizados estão na carga útil do pedido. A propriedade não pode ser uma propriedade-chave uma vez que o eXtreme Scale não permite que os clientes alterem as chaves de entidades.

Para obter mais detalhes sobre o pedido UpdatePrimitiveProperty, consulte: Biblioteca MSDN: Pedido UpdatePrimitiveProperty.

Veja aqui um exemplo de pedido UpdatePrimitiveProperty. Neste exemplo, atualizamos o nome da cidade de Customer('IBM') para 'Raleigh'.

AtomPub

- Método: PUT
- URI do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city
- Cabeçalho do Pedido: Tipo de Conteúdo: application/xml
- Carga Útil de Pedido:

```
<?xml version="1.0"
encoding="ISO-8859-1"?>
<city
xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
 Raleigh
</city>
```

- Carga Útil de Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

JSON

- Método: PUT

- URI do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city
- Cabeçalho do Pedido: Tipo de Conteúdo: application/json
- Carga Útil do Pedido: {"city":"Raleigh"}
- Carga Útil de Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

Atualizar um Valor de Propriedade Primitiva da Entidade

O Pedido UpdateValue pode atualizar um valor bruto da propriedade de uma entidade do eXtreme Scale. O valor a ser atualizado é representado como um valor bruto na carga útil do pedido. A propriedade não pode ser uma propriedade-chave uma vez que o eXtreme Scale não permite que os clientes alterem as chaves de entidades.

O tipo de conteúdo do pedido pode ser “text/plain” ou “application/octet-stream”, dependendo do tipo de propriedade. Para obter informações adicionais, consulte “Recuperando Não Entidades com Serviços de Dados REST” na página 282.

Para obter mais detalhes sobre o pedido UpdateValue, consulte: Biblioteca MSDN: Pedido UpdateValue

Veja aqui um exemplo do pedido UpdateValue. Neste exemplo, atualize o nome da cidade de Customer('IBM') para 'Raleigh'.

- Método: PUT
- URI do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city/$value
- Cabeçalho do Pedido: Tipo de Conteúdo: text/plain
- Carga Útil de Pedido: Raleigh
- Carga Útil de Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

Atualizar um Link

O pedido UpdateLink pode ser usado para estabelecer uma associação entre duas instâncias de entidade do eXtreme Scale. A associação pode ser uma relação com valor único (para um) ou uma relação com diversos valores (para muitos).

A atualização de um link entre duas instâncias de entidade do eXtreme Scale pode estabelecer associações ou remover associações. Por exemplo, se o cliente estabelecer uma associação "para um" entre uma entidade `Order(orderId=5000,customer_customerId='IBM')` e a instância `Customer('ALFKI')`, ele precisará dessociar a entidade `Order(orderId=5000,customer_customerId='IBM')` e a entidade de sua instância Customer associada atualmente.

Se as instâncias de entidade especificadas no pedido UpdateLink não puderem ser localizadas, o serviço de dados REST retornará uma resposta 404 (Não Localizado).

Se o URI do pedido UpdateLink especificar uma associação não existente, o serviço de dados REST retornará uma resposta 404 (Não Localizado) para indicar que o link não pode ser localizado.

Se a URI especificada na carga útil do pedido UpdateLink não resolver para a mesma entidade ou a mesma chave conforme especificado na URI, se existir, o Serviço de Dados REST do eXtreme Scale retornará uma resposta 400 (Pedido Inválido).

Se a carga útil de solicitação de UpdateLink contiver diversos links, o serviço de dados REST analisará somente o primeiro link. O restante dos links será ignorado.

Para obter mais detalhes sobre o pedido UpdateLink, consulte: Biblioteca MSDN: Pedido UpdateLink.

Veja aqui um exemplo de pedido UpdateLink. Neste exemplo, atualizamos a relação do cliente da entidade `Order(orderId=5000,customer_customerId='IBM')` e de Customer('IBM') para Customer('IBM').

Lembre-se: O exemplo anterior é apenas ilustrativo. Como todas as associações normalmente são associações chave para uma grade particionada, o link não poderá ser alterado.

AtomPub

- Método: PUT
- URI de Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer
- Cabeçalho do Pedido: Tipo de Conteúdo: application/xml
- Carga Útil de Pedido:

```
<?xml version="1.0"
encoding="ISO-8859-1"?>
<uri>
 http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
</uri>
```

- Carga Útil de Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

JSON

- Método: PUT
- URI da Solicitação: `http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer`
- Cabeçalho do Pedido: Tipo de Conteúdo: application/xml
- Carga Útil da Solicitação: `{"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')"}`
- Carga Útil de Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

Conceitos relacionados

"Operações com o Serviço de Dados REST" na página 269
Após iniciar o serviço de dados REST do eXtreme Scale, é possível usar qualquer cliente HTTP para interagir com ele. Um navegador da Web, um cliente PHP, um cliente Java ou um cliente WCF Data Services podem ser utilizados para emitir quaisquer operações de pedido suportadas.

"Visão Geral do Serviço de Dados REST" na página 117
O serviço de dados REST WebSphere eXtreme Scale é um serviço HTTP Java compatível com Microsoft WCF Data Services (formalmente, ADO.NET Data Services) e implementa o Open Data Protocol (OData). O Microsoft WCF Data Services é compatível com essa especificação quando utiliza Visual Studio 2008 SP1 e .NET Framework 3.5 SP1.

Tarefas relacionadas

"Acessando Dados com o Serviço de Dados REST" na página 268
Desenvolva aplicativos que executam operações usando protocolos do serviço de dados REST.

Pedidos de Exclusão com Serviços de Dados REST

O serviço de dados REST do WebSphere eXtreme Scale pode excluir entidades, valores da propriedade e links.

Excluir uma Entidade

O Pedido DeleteEntity pode excluir uma entidade do eXtreme Scale do serviço de dados REST.

Se alguma relação com a entidade a ser excluída tiver a exclusão em cascata configurada, o serviço de dados REST do eXtreme Scale excluirá a entidade ou entidades relacionadas. Para obter mais detalhes sobre o pedido DeleteEntity, consulte Biblioteca MSDN: Pedido DeleteEntity.

O pedido DeleteEntity a seguir exclui o cliente com a chave 'IBM'.

- Método: DELETE
- URI do pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Carga Útil de Pedido: Nenhuma
- Carga Útil de Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

Excluir um Valor da Propriedade

O Pedido DeleteValue configura uma propriedade de entidade do eXtreme Scale para nula.

Qualquer propriedade de uma entidade do eXtreme Scale pode ser configurada para nula com um pedido DeleteValue. Para configurar uma propriedade como nula, certifique-se do seguinte:

- Para qualquer tipo de número de primitiva e seu wrapper, BigInteger ou BigDecimal, o valor da propriedade será configurado como 0.
- Para o tipo Boolean ou boolean, o valor da propriedade será configurado como false.
- Para o tipo char ou Character, o valor da propriedade será configurado como character #X1 (NIL).

- Para o tipo enum, o valor da propriedade será configurado para o valor numérico com ordinal 0.
- Para todos os outros tipos, o valor da propriedade será configurado como nulo.

Entretanto, um pedido de exclusão pode ser rejeitado pelo banco de dados backend se, por exemplo, a propriedade não for anulável no banco de dados. Nesse caso, o serviço de dados REST retorna uma resposta 500 (Erro do Servidor Interno). Para obter mais detalhes sobre o pedido DeleteValue, consulte: Biblioteca MSDN: Pedido DeleteValue.

Aqui está um exemplo de pedido DeleteValue. Neste exemplo, configuramos o nome do contato de Customer('IBM') como nulo.

- Método: DELETE
- URI do pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Carga Útil de Pedido: Nenhuma
- Carga Útil de Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

Excluir um Link

O pedido DeleteLink pode remover uma associação entre duas instâncias de entidade do eXtreme Scale. A associação pode ser uma relação para-um ou uma relação para-muitos. Entretanto, um pedido de exclusão pode ser rejeitado pelo banco de dados backend se, por exemplo, a restrição de chave estrangeira estiver configurada. Nesse caso, o serviço de dados REST retorna uma resposta 500 (Erro do Servidor Interno). Para obter mais detalhes sobre o pedido DeleteLink, consulte: Biblioteca MSDN: Pedido DeleteLink.

O pedido DeleteLink a seguir remove a associação entre Order(101) e seu Customer associado.

- Método: DELETE
- URI de Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer
- Carga Útil de Pedido: Nenhuma
- Carga Útil de Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

Conceitos relacionados

"Operações com o Serviço de Dados REST" na página 269
Após iniciar o serviço de dados REST do eXtreme Scale, é possível usar qualquer cliente HTTP para interagir com ele. Um navegador da Web, um cliente PHP, um cliente Java ou um cliente WCF Data Services podem ser utilizados para emitir quaisquer operações de pedido suportadas.

"Visão Geral do Serviço de Dados REST" na página 117
O serviço de dados REST WebSphere eXtreme Scale é um serviço HTTP Java compatível com Microsoft WCF Data Services (formalmente, ADO.NET Data Services) e implementa o Open Data Protocol (OData). O Microsoft WCF Data Services é compatível com essa especificação quando utiliza Visual Studio 2008 SP1 e .NET Framework 3.5 SP1.

Tarefas relacionadas

"Acessando Dados com o Serviço de Dados REST" na página 268
Desenvolva aplicativos que executam operações usando protocolos do serviço de dados REST.

APIs e Plug-ins do Sistema

Um plug-in é um componente que fornece uma função aos componentes conectáveis que incluem ObjectGrid e BackingMap. Para usar de maneira mais eficiente o eXtreme Scale como um espaço de processamento de grade de dados ou de banco de dados de memória, é necessário determinar cuidadosamente a melhor maneira de maximizar o desempenho com os plug-ins disponíveis.

Gerenciando Ciclos de Vida de Plug-in

É possível gerenciar ciclos de vida de plug-in com métodos especializados de cada plug-in, que estão disponíveis para serem chamados em pontos funcionais designados. Ambos os métodos initialize e destroy definem o ciclo de vida de plug-ins, que são controlados pelos seus objetos *proprietário*. Um objeto proprietário é o objeto que realmente usa o plug-in fornecido. Um proprietário pode ser um cliente de grade, um servidor ou um mapa de apoio.

Sobre Esta Tarefa

Da mesma forma, todos os plug-ins podem implementar interfaces combinadas opcionais adequadas para seu objeto proprietário. Qualquer plug-in ObjectGrid pode implementar o ObjectGridPlugin de interface combinada opcional. Qualquer plug-in BackingMap pode implementar o BackingMapPlugin de interface combinada opcional. As interfaces combinadas opcionais requerem a implementação de vários métodos adicionais além dos métodos initialize() e destroy() para os plug-ins básicos. Para obter mais informações sobre essas interfaces, consulte a documentação da API.

Quando os objetos proprietários estão inicializando, esses objetos configuram atributos no plug-in e, em seguida, chamam o método initialize dos plug-ins proprietários. Durante o ciclo de destruição dos objetos proprietários, o método destroy dos plug-ins também será chamado consequentemente. Para obter detalhes sobre os métodos initialize e destroy específicos, junto com outros métodos aptos para cada plug-in, consulte os tópicos relevantes para cada plug-in.

Como exemplo, considere um ambiente distribuído. Ambos os ObjectGrids do lado do cliente e do lado do servidor podem ter seus próprios plug-ins. O ciclo de vida

de um ObjectGrid do lado do cliente e, portanto, suas instâncias de plug-in são independentes de todas as instâncias ObjectGrids e de plug-in do lado do servidor.

Nessa topologia distribuída, suponha que você tenha um ObjectGrid denominado `myGrid` definida no arquivo `objectGrid.xml` e configurado com um ObjectGridEventListener customizado chamado myObjectGridEventListener. O arquivo `objectGridDeployment.xml` define a política de implementação para o ObjectGrid `myGrid`. Ambos os arquivos `objectGrid.xml` e `objectGridDeployment.xml` são usados para iniciar servidores de contêiner. Durante a inicialização do servidor de contêiner, a instância do ObjectGrid `myGrid` do lado do servidor é inicializada. Enquanto isso, o método initialize da instância myObjectGridEventListener de propriedade da instância myObjectGrid é chamada. Depois que o servidor de contêiner for iniciado, o aplicativo poderá se conectar à instância ObjectGrid `myGrid` do lado do servidor e obter uma instância do lado do cliente.

Ao obter a instância myGrid do ObjectGrid do lado do cliente, a instância `myGrid` do lado do cliente passa pelo seu próprio ciclo de inicialização e chama o método initialize da sua própria instância myObjectGridEventListener do lado do cliente. Essa instância myObjectGridEventListener do lado do cliente é independente da instância myObjectGridEventListener do lado do servidor. O ciclo de vida é controlado pelo proprietário, que é a instância `myGrid` do ObjectGrid do lado do cliente.

Se o aplicativo desconectar ou destruir a instância `myGrid` do ObjectGrip do lado do cliente, o método destroy que pertence à instância myObjectGridEventListener do lado do cliente será chamado automaticamente. No entanto, esse processo não afeta a instância myObjectGridEventListener no lado do servidor. O método destroy da instância myObjectGridEventListener do lado do servidor pode ser chamado apenas durante o ciclo de vida de destruição da instância `myGrid` do ObjectGrid do lado do servidor ao parar um servidor de contêiner. Especificamente, ao parar um servidor de contêiner, as instâncias ObjectGrid contidas são destruídas e o método destroy de todos os plug-ins proprietários é chamado.

Embora o exemplo anterior seja aplicado especificamente para o caso de uma instância do cliente e do servidor de um ObjectGrid, o proprietário de um plug-in também pode ser uma interface BackingMap. Além disso, tenha cuidado ao determinar suas configurações para os plug-ins que você pode gravar, com base nessas considerações do ciclo de vida. Use os tópicos a seguir para gravar os plug-ins que fornecem eventos de gerenciamento de ciclo de vida estendido que podem ser usados para configurar ou remover recursos em seu ambiente:

Conceitos relacionados

"Visão Geral da Estrutura do OSGi" na página 37
O OSGi define um sistema módulo dinâmico para Java. A plataforma de serviço OSGi possui uma arquitetura em camadas e é projetada para ser executada em vários perfis padrão Java. É possível iniciar servidores e clientes do WebSphere eXtreme Scale em um contêiner OSGi.

Informações relacionadas

Documentação da API

Gravando um Plug-in do ObjectGridPlugin

Um ObjectGridPlugin é uma interface combinada opcional que pode ser usada para fornecer eventos de gerenciamento de ciclo de vida estendidos para todos os outros plug-ins do ObjectGrid.

Sobre Esta Tarefa

Qualquer plug-in do ObjectGrid que implementa o ObjectGridPlugin recebe o conjunto de eventos de ciclo de vida estendido que, além de fornecer mais controle, pode ser usado para configurar ou remover os recursos. Em um contêiner para uma grade de dados particionados, haverá uma instância de ObjectGrid (o proprietário de plugin) para cada partição gerenciada pelo contêiner.Quando partições individuais são removidas, os recursos usados por essa instância de ObjectGrid também devem ser removidos. Portanto, pode ser necessário fechar ou finalizar um recurso, como um arquivo de configuração aberto ou um encadeamento em execução gerenciado por um plug-in, quando a partição proprietária para esse recurso for removida.

A interface ObjectGridPlugin fornece métodos para configurar ou modificar o estado do plug-in, além de métodos para examinar o estado atual do plug-in. Todos os métodos devem ser implementados corretamente e o ambiente de tempo de execução do WebSphere eXtreme Scale verifica o comportamento do método sob determinadas circunstâncias. Por exemplo, depois de chamar o método initialize(), o ambiente de tempo de execução do eXtreme Scale chama o método isInitialized() para assegurar que o método concluiu com êxito a inicialização apropriada.

Procedimento

1. Implemente a interface ObjectGridPlugin para que o plug-in ObjectGridPlugin receba notificações sobre eventos significativos do eXtreme Scale. Há três categorias principais de métodos a seguir:

Métodos de propriedades	**Propósito**
setObjectGrid()	Chamado para configurar a instância do ObjectGrid usada pelo plug-in.
getObjectGrid()	Chamado para confirmar a instância do ObjectGrid usada pelo plug-in.

Métodos de inicialização	**Propósito**
initialize()	Chamado para inicializar o ObjectGridPlugin.
isInitialized()	Chamado para obter ou confirmar o status de inicialização do plug-in.

Métodos de destruição	**Propósito**
destroy()	Chamado para destruir o ObjectGridPlugin.
isDestroyed()	Chamado para obter ou confirme o status destruído do plug-in.

   Consulte a Documentação da API para obter mais informações sobre essas interfaces.

2. Configure um Plug-in ObjectGridPlugin com o XML. Use a classe com.company.org.MyObjectGridPluginTxCallback, que implementa a interface TransactionCallback e a interface ObjectGridPlugin.

   No exemplo de código a seguir, o retorno de chamada de transação customizada, que receberá definitivamente os eventos de ciclo de vida estendidos, é gerado e incluído em um ObjectGrid.

   Importante: A interface TransactionCallback já possui um método initialize, e um novo método de initialize é incluído bem como o método destroy e outros métodos do ObjectGridPlugin. Cada método é utilizado e os métodos initialize executam apenas uma inicialização por vez. O XML a seguir cria uma configuração que usa a interface TransactionCallback aprimorada.

O texto a seguir deve estar no arquivo 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>
```

Observe que as declarações de bean aparecem antes das declarações de backingMap.

3. Forneça o arquivo myGrid.xml para o plug-in ObjectGridManager a fim de facilitar a criação desta configuração.

Tarefas relacionadas

"Gravando um Plug-in BackingMapPlugin"
Um plug-in BackingMap implementa a interface combinada do BackingMapPlugin,que pode ser usada para receber os recursos estendidos para gerenciar seu ciclo de vida.

Informações relacionadas

../com.ibm.websphere.extremescale.javadoc.doc/topics/com/ibm/websphere/objectgrid/management/package-summary.html

Gravando um Plug-in BackingMapPlugin

Um plug-in BackingMap implementa a interface combinada do BackingMapPlugin,que pode ser usada para receber os recursos estendidos para gerenciar seu ciclo de vida.

Sobre Esta Tarefa

Qualquer plug-in BackingMap existente que também implementar a interface BackingMapPlugin receberá automaticamente o conjunto estendido de eventos de ciclo de vida durante a construção e uso.

A interface BackingMapPlugin fornece métodos para configurar ou modificar o estado do plug-in, bem como métodos para examinar o estado atual do plug-in.

Todos os métodos devem ser implementados corretamente e o ambiente de tempo de execução do WebSphere eXtreme Scale verifica o comportamento do método sob determinadas circunstâncias. Por exemplo, depois de chamar o método initialize(), o ambiente de tempo de execução do eXtreme Scale chama o método isInitialized() para assegurar que o método concluiu com êxito a inicialização apropriada.

Procedimento

1. Implemente a interface BackingMapPlugin para que o plug-in BackingMapPlugin receba notificações sobre eventos significativos do eXtreme Scale. Há três categorias principais de métodos a seguir:

Métodos de propriedades	Propósito
setBackingMap()	Chamado para configurar a instância do BackingMap usada pelo plug-in.
getBackingMap()	Chamado para obter ou confirmar a instância do BackingMap usada pelo plug-in.

Métodos de inicialização	Propósito
initialize()	Chamado para inicializar o plug-in BackingMapPlugin.
isInitialized()	Chamado para obter ou confirmar o status de inicialização do plug-in.

Métodos de destruição	Propósito
destroy()	Chamado para destruir o plug-in BackingMapPlugin.
isDestroyed()	Chamado para obter ou confirme o status destruído do plug-in.

Consulte a Documentação da API para obter mais informações sobre essas interfaces.

2. Configurar um Plug-in BackingMapPlugin com XML Suponha que o nome da classe de um plug-in Loader do eXtreme Scale seja a classe com.company.org.MyBackingMapPluginLoader, que implementa a interface Loader e a interface BackingMapPlugin.

   No exemplo de código a seguir, o retorno de chamada de transação customizada, que definitivamente receberá eventos do ciclo de vida estendido, é gerado e incluído em um BackingMap.

   Também é possível configurar um plug-in BackingMapPlugin usando XML. O texto a seguir deve estar no arquivo `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. Forneça o arquivo `myGrid.xml` para o plug-in ObjectGridManager para facilitar a criação desta configuração.

Resultados

A instância BackingMap criada tem um Loader que recebe eventos de ciclo de vida do BackingMapPlugin.

Tarefas relacionadas

"Gravando um Plug-in do ObjectGridPlugin" na página 300
Um ObjectGridPlugin é uma interface combinada opcional que pode ser usada para fornecer eventos de gerenciamento de ciclo de vida estendidos para todos os outros plug-ins do ObjectGrid.

Informações relacionadas

../com.ibm.websphere.extremescale.javadoc.doc/topics/com/ibm/websphere/objectgrid/management/package-summary.html

Plug-ins para Replicação Multimestre

Considere a possibilidade de transformar objetos armazenados em cache para aumentar o desempenho do seu cache. É possível usar o plug-in ObjectTransformer

quando seu uso do processador estiver alto. Até 60-70 por cento do tempo total do processador é gasto serializando e copiando entradas. Ao implementar o plug-in ObjectTransformer, é possível serializar e desserializar objetos com sua própria implementação. É possível usar um plug-in CollisionArbiter para definir como as colisões de mudanças são tratadas em seus domínios.

Desenvolvendo Árbitros Customizados para a Replicação Multimestre

Poderão ocorrer colisões de mudanças se os mesmos registros puderem ser alterados simultaneamente em dois locais. Em uma topologia de replicação multimestre, os domínios do serviço de catálogo detectam colisões automaticamente. Quando um domínio de serviço de catálogo detecta uma colisão, ele chama um árbitro. Geralmente, as colisões são resolvidas com o árbitro de colisão padrão. No entanto, um aplicativo pode fornecer um árbitro de colisão customizado.

Antes de Iniciar

- Consulte "Planejando Diversas Topologias do Datacenter" na página 99 para obter mais informações sobre como planejar e projetar a topologia de replicação multimestre.
- Consulte Configurando Diversas Topologias do Datacenter para obter mais informações sobre como configurar links entre os domínios de serviço de catálogo.

Sobre Esta Tarefa

Se um domínio de serviço de catálogos receber uma entrada replicada que colide com um registro de colisão, o árbitro padrão usará as mudanças do domínio de serviço de catálogo nomeado mais baixo de maneira lexical. Por exemplo, se os domínios A e B gerarem um conflito para um registro, a mudança do domínio B será ignorada. O domínio A mantém sua versão e o registro no domínio B é alterado para que corresponda ao registro do domínio A. Os nomes de domínio são convertidos em maiúsculas para comparação.

Uma alternativa é a topologia de replicação multimestre chamar um plug-in de colisão customizado para decidir o resultado. Essas instruções esboçam como desenvolver um árbitro de colisão customizado e configurar uma topologia de replicação multimestre para usá-lo.

Procedimento

1. Desenvolva um árbitro de colisão customizado e integre-o em seu aplicativo.

   A classe deve implementar a interface:

   ```
   com.ibm.websphere.objectgrid.revision.CollisionArbiter
   ```

   Um plug-in de colisão tem três opções para decidir o resultado de uma colisão. Ele pode escolher a cópia local ou a cópia remota ou pode fornecer uma versão revisada da entrada. Um domínio de serviço de catálogo fornece as seguintes informações para um árbitro de colisão customizado:

   - A versão existente do registro
   - A versão da colisão do registro
   - Um Objeto de sessão que deve ser usado para criar a versão revisada da entrada colidida

   O método de plug-in retorna um objeto que indica sua decisão. O método chamado pelo domínio para chamar o plug-in deve retornar verdadeiro ou

falso, em que falso significa ignorar a colisão. Quando a colisão é ignorada, a versão local permanece inalterada e o árbitro esquece que já ter visto a versão existente. O método retornará um valor real se tiver usado a sessão fornecida para criar uma versão nova, mesclada do registro, reconciliando a mudança.

2. No arquivo `objectgrid.xml`, especifique o plug-in do árbitro customizado.

   O ID deve 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>
```

Conceitos relacionados

"Planejando Diversas Topologias do Datacenter" na página 99
Ao usar a replicação assíncrona multimestre, duas ou mais grades de dados podem se tornar cópias exatas de uns dos outros. Cada grade de dados é hospedada em um domínio do serviço de catálogo independente, com seu próprio serviço de catálogo, servidores de contêiner e um nome exclusivo. Com a replicação assíncrona multimestre, é possível usar links para conectar uma coleção de domínios do serviço de catálogo. Os domínios do serviço de catálogo são então sincronizados usando a replicação sobre os links. É possível construir quase qualquer topologia por meio da definição de links entre os domínios de serviço de catálogo.

"Topologias de Replicação Multimestre" na página 100
Há várias opções diferentes quando escolher a topologia para sua implementação que incorpora a replicação multimestre.

"Considerações de Configuração para Topologias Multimestre" na página 105
Considere os seguintes problemas ao decidir se e como usar as topologias de replicação multimestre.

"Considerações de Design para Replicação Multimestre" na página 108
Ao implementar da replicação multimestre, você deve considerar aspectos de design, como arbitragem, vinculação e desempenho.

"Considerações Sobre o Carregador em uma Topologia Multimestre" na página 106
Quando estiver usando os carregadores em uma topologia multimestre, você deve considerar a possibilidade de colisão e desafios de manutenção das informações de revisão. A grade de dados mantém as informações de revisão sobre os itens nela para que colisões possam ser detectadas quando outros shards primários na configuração gravarem entradas na grade de dados. Quando as entradas são incluídas a partir de um carregador, essas informações de revisão não são incluídas e a entrada assume uma nova revisão. Como a revisão da entrada parece ser uma nova inserção, uma colisão false poderá ocorrer se outro shard primário também alterar esse estado ou obtiver as mesmas informações a partir de um carregador.

Plug-ins para Versão e Comparação de Objetos de Cache

Use o plug-in OptimisticCallback para customizar as operações de versão e de comparação de objetos do cache ao usar a estratégia de bloqueio otimista.

É possível fornecer um objeto de retorno de chamada otimista conectável que implementa a interface com.ibm.websphere.objectgrid.plugins.OptimisticCallback. Para mapas de entidade, um plug-in OptimisticCallback de alto desempenho é automaticamente configurado.

Propósito

Utilize a interface OptimisticCallback para fornecer operações de comparação otimistas para os valores de um mapa. Uma implementação OptimisticCallback é necessária ao utilizar a estratégia de bloqueio otimista. O produto fornece uma implementação de OptimisticCallback padrão. No entanto, geralmente o aplicativo deve conectar sua própria implementação da interface OptimisticCallback.

Implementação Padrão

A estrutura do eXtreme Scale fornece uma implementação padrão da interface OptimisticCallback que é usada se o aplicativo não for conectado a um objeto OptimisticCallback fornecido pelo aplicativo. A implementação padrão sempre retorna o valor especial de NULL_OPTIMISTIC_VERSION como o objeto de versão para o valor e nunca atualiza o objeto de versão. Esta ação faz uma comparação otimista de uma função "no operation". Na maioria dos casos, você não deseja que a função "no operation" ocorra, quando estiver utilizando a estratégia de bloqueio otimista. Seus aplicativos devem implementar a interface OptimisticCallback e conectar suas próprias implementações de OptimisticCallback para que a implementação padrão não seja utilizada. No entanto, existe pelo menos um cenário no qual a implementação de OptimisticCallback fornecida padrão é útil. Considere a seguinte situação:

- Um utilitário de carga é conectado para o mapa de suporte.
- O utilitário de carga sabe como desempenhar a comparação otimista sem assistência de um plug-in OptimisticCallback.

Como o utilitário de carga pode executar a versão otimista sem assistência de um objeto OptimisticCallback? O utilitário de carga conhece o objeto de classe de valor e sabe qual campo de objeto de valor é utilizado como um valor de versão otimista. Por exemplo, suponha que a seguinte interface seja utilizada para o objeto de valor para o mapa employees:

```
public interface Employee
{
    // Sequential sequence number used for optimistic versioning.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Other get/set methods for other fields of Employee object.
}
```

Neste exemplo, o utilitário de carga sabe que pode utilizar o método getSequenceNumber para obter as informações de versão atuais para um objeto de valor Employee. O utilitário de carga incrementa o valor retornado para gerar um novo número de versão antes de atualizar o armazenamento persistente com o novo valor Employee. Para um utilitário de carga Java Database Connectivity (JDBC), o número de sequência atual na cláusula WHERE de uma instrução UPDATE SQL superqualificada é usado e ele usa o novo número de sequência gerado para configurar a coluna do número de sequência para o novo valor de número de sequência. Outra possibilidade é que o utilitário de carga faça uso de alguma função fornecida por backend que atualiza automaticamente uma coluna oculta que pode ser utilizada para versões otimistas.

Em alguns casos, um procedimento armazenado ou acionador possivelmente pode ser utilizado para ajudar a manter uma coluna que contém as informações de controle de versões. Se o utilitário de carga estiver utilizando uma destas técnicas para a manutenção de informações de versões otimistas, então, o aplicativo não precisa fornecer uma implementação do OptimisticCallback. A implementação OptimisticCallback padrão pode ser utilizada neste cenário porque o utilitário de

carga consegue identificar versões otimistas sem nenhuma assistência de um objeto OptimisticCallback.

Implementação Padrão para Entidades

As entidades são armazenadas no ObjectGrid utilizando objetos de tupla. A implementação OptimisticCallback padrão se comporta da mesma maneira que se comporta com mapas de não-entidade. Entretanto, o campo de versão na entidade é identificado utilizando a anotação @Version ou o atributo version no arquivo XML descritor da entidade.

O atributo version pode ser de um dos seguintes tipos: int, Integer, short, Short, long, Long ou java.sql.Timestamp. Uma entidade deve ter apenas um atributo version definido. O atributo version deve ser configurado apenas durante a construção. Depois de a entidade ser persistida, o valor do atributo de versão não deve ser modificado.

Se um atributo version não estiver configurado e a estratégia de bloqueio otimista for utilizada, então, a tupla inteira será implicitamente versionada utilizando o estado inteiro da tupla, o que é mais custoso

No exemplo a seguir, a entidade Employee possui um atributo de versão longa denominado SequenceNumber:

```
@Entity
public class Employee {
private long sequence;
    // Sequential sequence number used for optimistic versioning.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Other get/set methods for other fields of Employee object.
}
```

Gravando um Plug-in OptimisticCallback

Um plug-in OptimisticCallback precisa implementar a interface OptimisticCallback e seguir as convenções comuns do plug-in ObjectGrid. Consulte a interface OptimisticCallback na documentação da API para obter mais informações.

A lista a seguir fornece uma descrição ou consideração para cada um dos métodos na interface OptimisticCallback:

NULL_OPTIMISTIC_VERSION

Este valor especial será retornado pelo método getVersionedObjectForValue se a implementação OptimisticCallback não requerer uma verificação de versão. A implementação de plug-in integrada da classe com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback usa esse valor porque a versão é desativada ao especificar essa implementação de plug-in.

Método getVersionedObjectForValue

O método getVersionedObjectForValue pode retornar uma cópia do valor ou um atributo do valor que pode ser utilizado para fins de versão. Este método é chamado sempre que um objeto é associado a uma transação. Quando nenhum Utilitário de Carga estiver conectado a um mapa de suporte, o mapa de suporte utilizará este valor no tempo de confirmação para desempenhar uma comparação de versão otimista. A comparação de versão otimista é utilizada pelo mapa de apoio para assegurar que a versão não tenha sido alterada depois que a primeira transação acessou pela primeira vez a entrada do mapa que foi modificada por esta transação. Se outra transação já tiver modificado a versão desta entrada do mapa, a comparação de versão falhará e o mapa de apoio exibirá uma exceção OptimisticCollisionException para forçar o retrocesso da transação. Se um Utilitário de Carga estiver conectado, o mapa de suporte não utilizará as informações de controle de versões otimista. Em vez disso, o Utilitário de Carga é responsável por desempenhar a comparação de controle de versões otimista e por atualizar as informações de controle de versões quando necessário. O Utilitário de Carga geralmente obtém o objeto de versão inicial do LogElement transmitido para o método batchUpdate no utilitário de carga, que é chamado quando ocorre uma operação de limpeza ou quando uma transação é confirmada.

O código a seguir mostra a implementação utilizada pelo objeto EmployeeOptimisticCallbackImpl:

```
public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}
```

Conforme demonstrado no exemplo anterior, o atributo sequenceNumber é retornado em um objeto java.lang.Long conforme esperado pelo Utilitário de Carga, que significa que a mesma pessoa que gravou o Utilitário de Carga gravou a implementação de EmployeeOptimisticCallbackImpl ou trabalhou junto com a pessoa que implementou o EmployeeOptimisticCallbackImpl - por exemplo, concordou com o valor retornado pelo método getVersionedObjectForValue. O plug-in OptimisticCallback padrão retorna o valor especial NULL_OPTIMISTIC_VERSION como o objeto de versão.

Método updateVersionedObjectForValue

Este método é chamado sempre que uma transação tiver atualizado um valor e um novo objeto de versão for requerido. Se o método getVersionedObjectForValue retornar um atributo do valor, este método geralmente atualizará o valor de atributo com um novo objeto de versão. Se o método getVersionedObjectForValue retornar uma cópia do valor, este método normalmente não executa nenhuma ação. O plug-in OptimisticCallback padrão não executa nenhuma ação com esse método pois a implementação padrão de getVersionedObjectForValue sempre retorna o valor especial NULL_OPTIMISTIC_VERSION como o objeto de versão. O seguinte exemplo mostra a implementação usada pelo objeto EmployeeOptimisticCallbackImpl que é usado na seção OptimisticCallback:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Conforme demonstrado no exemplo anterior, o atributo sequenceNumber é incrementado em um para que na próxima vez em que o método getVersionedObjectForValue for chamado, o valor java.lang.Long retornado tenha um valor longo que é o valor do número de sequência original mais um, por exemplo, é o próximo valor de versão para esta instância employee. Este exemplo significa que a pessoa que gravou o Utilitário de Carga gravou o EmployeeOptimisticCallbackImpl ou trabalhou junto com a pessoa que implementou o EmployeeOptimisticCallbackImpl.

Método serializeVersionedValue

Este método grava o valor com versão no fluxo especificado. Dependendo da implementação, o valor com versão pode ser utilizado para identificar colisões de atualização otimistas. Em algumas implementações, o valor com versão é uma cópia do valor original. Outras implementações podem ter um número de seqüência ou algum outro objeto para indicar a versão do valor. Como a implementação real é desconhecida, este método é fornecido para executar a serialização apropriada. A implementação padrão faz uma chamada writeObject.

Método inflateVersionedValue

Este método utiliza a versão serializada do valor com versão e retorna o objeto de valor com versão real. Dependendo da implementação, o valor com versão pode ser utilizado para identificar colisões de atualização otimistas. Em algumas implementações, o valor com versão é uma cópia do valor original. Outras implementações podem ter um número de seqüência ou algum outro objeto para indicar a versão do valor. Como a implementação real é desconhecida, este método é fornecido para executar a desserialização apropriada. A implementação padrão chama o método readObject.

Utilizando o Objeto OptimisticCallback Fornecido pelo Aplicativo

Há duas abordagens para incluir um objeto OptimisticCallback fornecido pelo aplicativo na configuração de BackingMap: configuração programática e configuração XML.

Conectar Programaticamente um Objeto OptimisticCallback

O exemplo a seguir demonstra como um aplicativo pode conectar programaticamente um objeto OptimisticCallback para o mapa de apoio de funcionários na instância grid1 do ObjectGrid local:

```
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 );
```

Abordagem de configuração XML para conectar um objeto OptimisticCallback

O aplicativo pode utilizar um arquivo XML para conectar seu objeto OptimisticCallback, conforme mostrado no seguinte exemplo:

```
<?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 em Cache

O WebSphere eXtreme Scale usa diversos processos Java serializar os dados, ao converter as instâncias de objeto Java em bytes e em objetos novamente, conforme necessário, para mover os dados entre processos do cliente e do servidor.

Para serializar dados no eXtreme Scale, é possível usar a serialização Java, o plug-in ObjectTransformer ou os plug-ins DataSerializer.

 A interface ObjectTransformer foi substituída pelos plug-ins DataSerializer, que podem ser usados para armazenar dados arbitrários com eficiência no WebSphere eXtreme Scale para que as APIs do produto existentes possam interagir eficientemente com seus dados.

Conceitos relacionados

Visão Geral da Serialização
Os dados são sempre expressos, porém não necessariamente armazenados, como objetos Java na grade de dados. O WebSphere eXtreme Scale usa diversos processos Java para serializar os dados, ao converter as instâncias de objetos Java em bytes e retornar para os objetos novamente, conforme necessário, para mover os dados entre os processos do cliente e do servidor.

Visão Geral da Programação do Serialziador

É possível usar os plug-ins do DataSerializer para gravar serializadores otimizados para armazenar objetos Java e outros dados no formato binário na grade. O plug-in também fornece métodos que você pode usar para atributos de consulta dentro dos dados binários sem exigir que o objeto de dados inteiro seja expandido.

Os plug-ins DataSerializer incluem três plug-ins principais e várias interfaces combinadas opcionais. O plug-in MapSerializerPlugin identifica e inclui metadados sobre o relacionamento entre um mapa e outros mapas. Ele também inclui uma referência a um KeySerializerPlugin e um ValueSerializerPlugin. Os plug-ins do serializador de chave e valor incluem metadados e código de serialização responsável pela interação com os respectivos dados de chave e valor para um mapa. Um plug-in MapSerializerPlugin deve incluir um ou ambos os serializadores de chave e valor.

O plug-in KeySerializerPlugin fornece métodos e metadados para serializar, aumentar e examinar internamente as chaves. O plug-in ValueSerializer fornece métodos e metadados para serializar, aumentar e examinar internamente os valores. Ambas as interfaces possuem requisitos diferentes. Para obter detalhes sobre quais métodos estão disponíveis nos plug-inss DataSerializer, consulte a documentação da API para o pacote com.ibm.websphere.objectgrid.plugins.io.

Plug-in MapSerializerPlugin
O MapSerializerPlugin é o ponto de plug-in principal para a interface BackingMap e inclui dois plug-ins aninhados: os plug-ins KeySerializerPlugin e ValueSerializerPlugin. Como o eXtreme Scale não suporta plug-ins aninhados ou conectados, o plug-in BasicMapSerializerPlugin acessa estes plug-ins aninhados artificialmente. Quando você usa esses plug-ins com a estrutura do OSGi, o único proxy é o plug-in MapSerializerPlugin. Todos os plug-ins aninhados não devem ser armazenados em cache em outros plug-ins dependentes, como os carregadores, a menos que esses plug-ins também atendam aos eventos do ciclo de vida do BackingMap. Isto é importante ao executar em uma estrutura do OSGi, porque as referências com esses plug-ins podem continuar sendo atualizadas.

Plug-in KeySerializerPlugin
O plug-in KeySerializerPlugin estende a interface DataSerializer e inclui outras interfaces e metadados combinados que descrevem a chave. Use este plug-in para serializar e aumentar os objetos de dados e atributos de chave.

Plug-in ValueSerializerPlugin
O plug-in ValueSerializerPlugin estende a interface DataSerializer, porém não expõe nenhum método adicional. Use este plug-in para serializar e aumentar objetos de dados e atributos de valor.

Interfaces Opcionais e Combinadas

As interfaces opcionais e combinadas fornecem capacidades adicionais, como:

Versão otimista
A interface Provida de Versões permite que o plug-in ValueSerializerPlugin manipule a verificação de versão e as atualizações de versão ao usar bloqueio otimista. Se a Versão não estiver implementada e o bloqueio otimista estiver ativado, a versão será o formulário serializado inteiro do valor de objeto de dados.

Roteamento não baseado em hashCode
A interface Particionável permite que as implementações de KeySerializerPlugin roteiem solicitações para partições explícitas. Isso é equivalente à interface PartitionableKey, quando usado com a API do ObjectMap sem KeySerializerPlugin. Sem esse recurso, a chave é roteada para a partição com base no hashCode resultante.

Interface UserReadable (toString)
A interface UserReadable (toString) permite que todas as implementações de DataSerializer forneçam um método alternativa para exibir dados em arquivos de log e depuradores. Com esse recurso, é possível ocultar dados sensíveis, como senhas. Se implementações de DataSerializer não implementarem esta interface, o ambiente de tempo de execução poderá chamar toString() diretamente no objeto ou incluir representações alternativas, se apropriado.

Suporte à evolução

A interface Mesclável pode ser implementada em implementações de plug-in ValueSerializerPlugin para permitir a interoperabilidade entre diversas versões de objetos quando houver versões diferentes do DataSerializer atualizando dados na grade durante seu tempo de vida. Os métodos Mescláveis permitem que o DataSerializer tenha a opção de reter qualquer dado que não possa entender de outra forma.

Tarefas relacionadas

"Evitando Aumento de Objeto para Recuperar Atributos a partir dos Dados Serializados"
É possível usar os plug-ins DataSerializer para ignorar o aumento de objetos automático e recuperar manualmente os atributos dos dados que já foram serializados.

"Programando para Usar a Estrutura do OSGi" na página 392
Os servidores e clientes do eXtreme Scale podem ser iniciados em um contêiner OSGi para poder incluir e atualizar dinamicamente plug-ins do eXtreme Scale no ambiente de tempo de execução.

Informações relacionadas

Documentação da API do DataSerializer

Evitando Aumento de Objeto para Recuperar Atributos a partir dos Dados Serializados

É possível usar os plug-ins DataSerializer para ignorar o aumento de objetos automático e recuperar manualmente os atributos dos dados que já foram serializados.

Sobre Esta Tarefa

Este tópico é sobre como evitar um aumento do objeto inteiro para recuperar atributos. No entanto, é possível aumentar o objeto inteiro ao aumentar seus objetos Java para uma representação dos dados do tipo POJO. Para aumentar o objeto inteiro, altere a última linha do exemplo neste tópico para a seguinte linha de código:

```
Order order = (Order) sa.getMapSerializerPlugin().getValueSerializerPlugin()
.inflateDataObject(serValue.getContext(), bufValue);
```

Esta tarefa usa o modo de cópia COPY_TO_BYTES_RAW com os plug-ins MapSerializerPlugin e ValueSerializerPlugin. O MapSerializer é o ponto de plug-in principal para a interface BackingMap. Dois plug-ins aninhados são incluídos, o KeyDataSerializer e o ValueDataSerializer. Como o produto não suporta plug-ins aninhados, o plug-in BaseMapSerializer suporta artificialmente estes plug-ins aninhados ou ligados. Portanto, quando usar essas APIs no contêiner OSGi, o MapSerializer será o único proxy. Todos os plug-ins aninhados não devem ser armazenados em cache dentro de outros plug-ins dependentes, como um carregador, a menos que ele também atenda aos eventos de ciclo de vida do BackingMap, de modo que suas referências de suporte possam ser atualizadas.

Procedimento

1. Recupere a instância ObjectMap.
2. Configure o modo de cópia para COPY_TO_BYTES_RAW.
3. Use o método get para recuperar o objeto SerializedValue.
4. Use o método SerializedValue.getByteBuffers() para recuperar a forma serializada do valor.

5. Chame o plug-in ValueSerializerPlugin para aumentar atributos individuais a partir dos buffers de bytes.

Exemplo

Use o exemplo de código a seguir para recuperar atributos a partir dos dados serializados sem aumentar o objeto Java inteiro.

```
// The BackingMap is configured with COPY_TO_BYTES and a MapSerializerPlugin with a ValueSerializerPlugin

Session session = objectGrid.getSession();
ObjectMap orderMap = session.getMap("OrderMap");

// Automatically inflate to a POJO like normal
Order order = (Order) orderMap.get(1234);

// Override the CopyMode to retrieve the bytes.  This process affects all API methods from this point on
// for the life of the Session.
// Note:  The byte array has an eXtreme Scale-specific header.
orderMap.setCopyMode(CopyMode.COPY_TO_BYTES_RAW);
SerializedValue serValue = (SerializedValue) orderMap.get(1234);

// Get the byte buffers
XsByteBuffer[] bufValue= serValue.getByteBuffers();

// Convert/get the byte array
Byte[] bytesValue = ByteBufferUtils.asByteArray(bufValue);

// Retrieve a single attribute from the byte buffer.
String name = (String) sa.getMapSerializerPlugin().getValueSerializerPlugin()
.inflateDataObjectAttributes(serValue.getContext(),
 bufValue, new String[]{"name"};
```

Conceitos relacionados

"Visão Geral da Programação do Serialziador" na página 310
É possível usar os plug-ins do DataSerializer para gravar serializadores otimizados para armazenar objetos Java e outros dados no formato binário na grade. O plug-in também fornece métodos que você pode usar para atributos de consulta dentro dos dados binários sem exigir que o objeto de dados inteiro seja expandido.

Visão Geral da Serialização
Os dados são sempre expressos, porém não necessariamente armazenados, como objetos Java na grade de dados. O WebSphere eXtreme Scale usa diversos processos Java para serializar os dados, ao converter as instâncias de objetos Java em bytes e retornar para os objetos novamente, conforme necessário, para mover os dados entre os processos do cliente e do servidor.

Informações relacionadas

Documentação da API do DataSerializer

Plug-in ObjectTransformer

Com o plug-in ObjectTransformer, é possível serializar, desserializar e copiar objetos no cache para aumentar o desempenho.

 A interface ObjectTransformer foi substituída pelos plug-ins DataSerializer, que podem ser usados para armazenar dados arbitrários eficientemente no WebSphere eXtreme Scale para que as APIs do produto existentes possam ser interagir de modo eficiente com seus dados.

Se você tiver problemas de desempenho com o uso do processador, inclua um plug-in ObjectTransformer em cada mapa. Se um plug-in ObjectTransformer não for fornecido, o processador gastará de 60 a 70% de seu tempo total só serializando e copiando entradas.

Propósito

O plug-in ObjectTransformer permite que os aplicativos forneçam métodos customizados para as seguintes operações:

- Serializar ou desserializar a chave para uma entrada
- Serializar ou desserializar o valor para uma entrada
- Copiar uma chave ou valor para uma entrada

Se nenhum plug-in ObjectTransformer for fornecido, será necessário serializar as chaves e valores, porque o ObjectGrid utiliza a seqüência serializar e desserializar para copiar os objetos. Este método é caro, portanto, utilize um plug-in ObjectTransformer quando o desempenho for importante. A cópia ocorre quando um aplicativo consulta um objeto em uma transação pela primeira vez. É possível evitar essa cópia configurando o modo de cópia como COPY_ON_READ ou reduzir a cópia configurando o modo de cópia como COPY_ON_READ. Otimize a operação de cópia quando requerido pelo aplicativo, fornecendo um método de cópia customizado neste plug-in. Esse plug-in pode reduzir a sobrecarga de cópia de 65 a 70% para 2 a 3% do tempo total do processador.

As implementações dos métodos padrão copyKey e copyValue primeiro tentam utilizar o método clone, se este for fornecido. Se nenhuma implementação do método clone for fornecida, a implementação será padronizada como serialização.

A serialização do objeto também é utilizada diretamente quando o eXtreme Scale estiver em execução no modo distribuído. LogSequence utiliza o plug-in ObjectTransformer para ajudá-lo a serializar chaves e valores antes de transmitir as alterações para os equivalentes no ObjectGrid. Cuidado ao fornecer um método de serialização customizado em vez de utilizar a serialização do Java developer kit integrada. O controle de versões do objeto é um assunto complexo e é possível encontrar problemas com a compatibilidade de versões se você não assegurar que seus métodos customizados foram projetados para controle de versões.

A lista a seguir descreve como o eXtreme Scale tenta serializar chaves e valores:

- Se um plug-in ObjectTransformer customizado for gravado e conectado, o eXtreme Scale chamará os métodos nos métodos na interface ObjectTransformer para serializar chaves e valores e obter cópias de chaves e valores do objeto.
- Se um plug-in ObjectTransformer customizado não for usado, o eXtreme Scale serializa e desserializa os valores de acordo com o padrão. Se o plug-in padrão for utilizado, cada objeto será implementado como externalizável ou implementado como serializável.
  - Se o objeto suportar a interface Externalizável, o método writeExternal será chamado. Os objetos que são implementados como externalizáveis geram melhor desempenho.
  - Se o objeto não suportar a interface Externalizável e implementar a interface Serializável, o objeto será salvo usando o método ObjectOutputStream.

Utilizando a Interface ObjectTransformer

Um objeto ObjectTransformer precisa implementar a interface ObjectTransformer e seguir as convenções comuns do plug-in ObjectGrid.

Duas abordagens, configuração programática e configuração XML, são utilizadas para incluir um objeto ObjectTransformer na configuração BackingMap da seguinte forma.

Conectando um Objeto ObjectTransformer Programaticamente

O fragmento de código a seguir cria o objeto ObjectTransformer customizado e o inclui em um BackingMap:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap backingMap = myGrid.getMap("myMap");
MyObjectTransformer myObjectTransformer = new MyObjectTransformer();
backingMap.setObjectTransformer(myObjectTransformer);
```

Abordagem de Configuração XML para Conectar um ObjectTransformer

Suponha que o nome da classe da implementação ObjectTransformer seja a classe com.company.org.MyObjectTransformer. Essa classe implementa a interface ObjectTransformer. Uma implementação ObjectTransformer pode ser configurada usando o seguinte 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>
```

Cenários de Uso do ObjectTransformer

É possível utilizar o plug-in ObjectTransformer nas seguintes situações:

- Objeto não-serializável
- Objeto serializável mas aprimorando o desempenho da serialização
- Cópia de chave ou valor

No exemplo a seguir, o ObjectGrid é utilizado para armazenar a classe Stock:

```
/**
* Objeto Stock para demo do ObjectGrid
*
*
*/
public class Stock implements Cloneable {
    String ticket;
    double price;
    String company;
    String description;
    int serialNumber;
    long lastTransactionTime;
    /**
    * @return Retorna a descrição.
    */
    public String getDescription()  {
        return description;
    }
    /**
    * @param description A descrição a ser configurada.
    */
    public void setDescription(String description) {
        this.description = description;
    }
    /**
    * @return Retorna lastTransactionTime.
    */
```

```
    public long getLastTransactionTime() {
        return lastTransactionTime;
    }
    /**
    * @param lastTransactionTime O último lastTransactionTime a ser configurado.
    */
    public void setLastTransactionTime(long lastTransactionTime) {
        this.lastTransactionTime = lastTransactionTime;
    }
    /**
    * @return Retorna o preço.
    */
    public double getPrice() {
        return price;
    }
    /**
    * @param price O preço a ser configurado.
    */
    public void setPrice(double price) {
        this.price = price;
    }
    /**
    * @return Retorna um serialNumber.
    */
    public int getSerialNumber() {
        return serialNumber;
    }
    /**
    * @param serialNumber O serialNumber a ser configurado.
    */
    public void setSerialNumber(int serialNumber) {
        this.serialNumber = serialNumber;
    }
    /**
    * @return Retorna o registro.
    */
    public String getTicket() {
        return ticket;
    }
    /**
    * @param ticket O registro a ser configurado.
    */
    public void setTicket(String ticket) {
        this.ticket = ticket;
    }
    /**
    * @return Retorna a empresa.
    */
    public String getCompany() {
        return company;
    }
    /**
    * @param company A empresa a ser configurada.
    */
    public void setCompany(String company) {
        this.company = company;
    }
    //clone
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}
```

É possível gravar uma classe do transformador do objeto customizado para a classe Stock:

```
/**
* Implementação customizada do ObjectTransformer do ObjectGrid para objeto stock
*
*/
public class MyStockObjectTransformer implements ObjectTransformer {
/* (non-Javadoc)
* @see
* com.ibm.websphere.objectgrid.plugins.ObjectTransformer#serializeKey
* (java.lang.Object,
* java.io.ObjectOutputStream)
*/
public void serializeKey(Object key, ObjectOutputStream stream) throws IOException {
    String ticket= (String) key;
    stream.writeUTF(ticket);
```

```
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#serializeValue(java.lang.Object,
java.io.ObjectOutputStream)
*/
public void serializeValue(Object value, ObjectOutputStream stream) throws IOException {
    Stock stock= (Stock) value;
    stream.writeUTF(stock.getTicket());
    stream.writeUTF(stock.getCompany());
    stream.writeUTF(stock.getDescription());
    stream.writeDouble(stock.getPrice());
    stream.writeLong(stock.getLastTransactionTime());
    stream.writeInt(stock.getSerialNumber());
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateKey(java.io.ObjectInputStream)
*/
public Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException {
    String ticket=stream.readUTF();
    return ticket;
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateValue(java.io.ObjectInputStream)
*/

public Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException {
    Stock stock=new Stock();
    stock.setTicket(stream.readUTF());
    stock.setCompany(stream.readUTF());
    stock.setDescription(stream.readUTF());
    stock.setPrice(stream.readDouble());
    stock.setLastTransactionTime(stream.readLong());
    stock.setSerialNumber(stream.readInt());
    return stock;
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyValue(java.lang.Object)
*/
public Object copyValue(Object value) {
    Stock stock = (Stock) value;
    try {
        return stock.clone();
    }
    catch (CloneNotSupportedException e)
    {
        // display exception message    }
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyKey(java.lang.Object)
*/
public Object copyKey(Object key) {
    String ticket=(String) key;
    String ticketCopy= new String (ticket);
    return ticketCopy;
}
}
```

Em seguida, conecte esta classe MyStockObjectTransformer customizada ao 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 Fornecer Listeners de Eventos

Os plug-ins ObjectGridEventListener, MapEventListener, ObjectGridLifecycleListener e BackingMapLifecycleListener podem ser usados para configurar notificações para vários eventos no cache eXtreme Scale. Os plug-ins de listener são registrados com uma instância ObjectGrid ou BackingMap como outros plug-ins do eXtreme Scale e incluem pontos de integração e de customização para aplicativos e provedores de cache.

Plug-in do ObjectGridEventListener

Um plug-in ObjectGridEventListener fornece eventos de ciclo de vida do eXtreme Scale para a instância do ObjectGrid, shards e transações. Utilize um plug-in ObjectGridEventListener para receber notificações quando ocorrerem eventos significativos em um ObjectGrid. Esses eventos incluem inicialização do ObjectGrid, o início de uma transação, o encerramento de uma transação e destruição de um ObjectGrid. Para atender estes eventos, crie uma classe que implementa a interface ObjectGridEventListener e inclua-a no eXtreme Scale.

Para obter mais informações sobre a gravação de um plug-in do ObjectGridEventListener, consulte "Plug-in ObjectGridEventListener" na página 320. Também é possível consultar a Documentação da API para obter mais informações.

Incluindo e removendo instâncias do ObjectGridEventListener

Um ObjectGrid pode ter vários listeners ObjectGridEventListener. Inclua e remova os listeners usando os métodos addEventListener e removeEventListener na interface ObjectGrid. Também é possível registrar de modo declarativo os plug-ins ObjectGridEventListener com o arquivo descritor ObjectGrid. Para obter exemplos, consulte "Plug-in ObjectGridEventListener" na página 320.

Plug-in do MapEventListener

Um plug-in do MapEventListener fornece notificações de callback e alterações de estado de cache significativas que ocorrem para uma instância do BackingMap. Para obter detalhes sobre a gravação de um MapEventListener, consulte "Plug-in MapEventListener" na página 319. Também é possível consultar a Documentação da API para obter mais informações.

Incluindo e Removendo Instâncias do MapEventListener

Um eXtreme Scale pode ter vários listeners ObjectGridEventListener. Inclua e remova listeners com os métodos addMapEventListener e removeMapEventListener na interface BackingMap. Também é possível registrar de modo declarativo os plug-ins MapEventListener com o arquivo descritor ObjectGrid. Para obter exemplos, consulte "Plug-in MapEventListener" na página 319.

Plug-in BackingMapLifecycleListener

Um plug-in BackingMapLifecycleListener fornece notificações de retorno de chamada para as mudanças no estado do ciclo de vida que ocorrem para uma instância do BackingMap. A instância BackingMap continua por meio de um conjunto predefinido de estados durante seu tempo de vida.

Incluindo e removendo instâncias do BackingMapLifecycleListener

Um servidor eXtreme Scale pode ter vários listeners BackingMapLifecycleListener. Inclua e remova listeners com os métodosaddMapEventListener e removeMapEventListener na interface BackingMap. Quaisquer plug-ins do BackingMap que implementam a interface BackingMapLifecycleListener também são incluídos automaticamente como um BackingMapLifecycleListener na instância do ObjectGrid com a qual eles estão registrados. Também é possível registrar declaradamente os listeners BackingMapLifecycleListener com o arquivo descritor

ObjectGrid. Para obter exemplos, consulte Plug-in BackingMapLifecycleListener.

Plug-in ObjectGridLifecycleListener

Um plug-in ObjectGridLifecycleListener fornece notificações de retorno de chamada para as mudanças no estado do ciclo de vida que ocorrem para uma instância do ObjectGrid. A instância do ObjectGrid continua por meio de um conjunto predefinido de estados durante seu tempo de vida.

Incluindo e removendo instâncias do ObjectGridLifecycleListener

Um eXtreme Scale pode ter vários listeners ObjectGridLifecycleListener. Inclua e remova os listeners usando os métodos addEventListener e removeEventListener na interface ObjectGrid. Todos os plug-ins do ObjectGrid que implementam a interface ObjectGridLifecycleListener são automaticamente incluídos como um ObjectGridLifecycleListener na instância do ObjectGrid com a qual eles estão registrados. Também é possível registrar declaradamente os listeners ObjectGridLifecycleListener com o arquivo descritor de implementação ObjectGrid. Para obter exemplos, consulte Plug-in ObjectGridLifecycleListener.

Plug-in MapEventListener

Um plug-in MapEventListener fornece notificações de retorno de chamada e mudanças de estado de cache significativas que ocorrem para um objeto BackingMap: quando um mapa termina o pré-carregamento ou quando uma entrada é despejada do mapa. Um plug-in MapEventListener específico é uma classe customizada que você grava implementando a interface MapEventListener.

Convenções de Plug-in do MapEventListener

Ao desenvolver um plug-in do MapEventListener, é necessário seguir convenções comuns do plug-in. Para obter mais informações sobre convenções de plug-in, consulte “Visão Geral de Plug-ins” na página 115. Para os outros tipos de plug-ins de listener, consulte “Plug-ins para Fornecer Listeners de Eventos” na página 317.

Depois de gravar uma implementação do MapEventListener, será possível conectá-la à configuração de BackingMap programaticamente ou com uma configuração XML.

Gravar uma Implementação do MapEventListener

Seu aplicativo pode incluir uma implementação do plug-in do MapEventListener. O plug-in deve implementar a interface MapEventListener para receber eventos significativos sobre um mapa. Os eventos são enviados para o plug-in do MapEventListener quando uma entrada é despejada do mapa e quando o pré-carregamento de um mapa é concluído.

Conexão Programática em uma Implementação do MapEventListener

O nome da classe para o MapEventListener customizado é a classe com.company.org.MyMapEventListener. Esta classe implementa a interface MapEventListener. O trecho de código a seguir cria o objeto MapEventListener customizado e o inclui em um 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 uma Implementação do MapEventListener Utilizando XML

Uma implementação do MapEventListner pode ser configurada utilizando XML. O XML a seguir deve estar no arquivo 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>
```

Fornecer este arquivo para a instância do ObjectGridManager facilita a criação desta configuração. O fragmento de código a seguir mostra como criar um ObjectGrid utilizando este arquivo XML. A instância ObjectGrid recém-criada tem um MapEventListener configurado no BackingMap myMap.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
 objectGridManager.createObjectGrid("myGrid", new URL("file:etc/test/myGrid.xml"),
  true, false);
```

Plug-in ObjectGridEventListener

Um plug-in ObjectGridEventListener fornece eventos de ciclo de vida do WebSphere eXtreme Scale para o ObjectGrid, shards e transações. Um plug-in ObjectGridEventListener fornece notificações quando um ObjectGrid é inicializado ou destruído, e quando uma transação é iniciada ou encerrada. Plug-ins ObjectGridEventListener são classes customizadas que você grava implementando a interface ObjectGridEventListener. Opcionalmente, a implementação inclui subinterfaces do ObjectGridEventGroup e segue as convenções comuns do plug-in do eXtreme Scale.

Visão Geral

Um plug-in ObjectGridEventListener é útil quando um plug-in Loader estiver disponível e for necessário inicializar uma ou mais conexões do Java Database Connectivity (JDBC) para um backend quando as transações forem iniciadas e encerradas. Normalmente, um plug-in ObjectGridEventListener e um plug-in Loader são gravados juntos.

Gravando um plug-in ObjectGridEventListener

Um plug-in ObjectGridEventListener deve implementar a interface ObjectGridEventListener para receber notificações sobre eventos significativos do

eXtreme Scale. Para receber notificações de eventos adicionais, é possível implementar as interfaces a seguir. Estas subinterfaces são incluídas na interface ObjectGridEventGroup:

- Interface ShardEvents
- Interface ShardLifecycle
- Interface TransactionEvents

Para obter mais informações sobre essas interfaces, consulte a Documentação da API.

Eventos do Shard

Quando o serviço de catálogo colocar shards primários ou de réplica na partição em uma Java virtual machine (JVM), uma nova instância do ObjectGrid será criada nessa JVM para hospedar esse shard. Alguns aplicativos que precisam iniciar os encadeamentos no JVM, hospedam a notificação de necessidade primária desses eventos. A interface ObjectGridEventGroup.ShardEvents declara os métodos shardActivate e shardDeactivate. Esses métodos são chamados apenas quando uma parte é ativada como primária e quando a parte é desativada a partir da primária. Esses dois eventos permitem que o aplicativo inicie encadeamentos adicionais quando o shard for primário e pare os encadeamentos quando o shard voltar a ser uma réplica ou for retirado de serviço.

Um aplicativo pode determinar qual partição foi ativada ao procurar por um BackingMap específico na referência ObjectGrid fornecida para o método shardActivate usando o método ObjectGrid#getMap. O aplicativo pode visualizar, em seguida, o número de partição usando o método BackingMap#getPartitionId(). As partições são numeradas de 0 ao número de partições no descritor de implementação menos um.

Eventos de Ciclo de Vida do Shard

Os eventos dos métodos ObjectGridEventListener.initialize e ObjectGridEventListener.destroy são entregues utilizando a interface ObjectGridEventGroup.ShardLifecycle.

Eventos de Transação

Os métodos ObjectGridEventListener.transactionBegin e ObjectGridEventListener.transactionEnd são entregues através da interface ObjectGridEventGroup.TransactionEvents.

Se um plug-in ObjectGridEventListener implementa as interfaces ObjectGridEventListener e ShardLifecycle, então, os eventos de ciclo de vida do shard serão os únicos eventos a serem entregues para o listener. Após implementar qualquer uma das novas interfaces ObjectGridEventGroup internas, o eXtreme Scale entrega apenas esses eventos específicos através de novas interfaces. Com essa implementação, o código pode ser compatível com as versões anteriores. Se você estiver utilizando as novas interfaces internas, poderá agora receber apenas os eventos específicos necessários.

Utilizando o Plug-in ObjectGridEventListener

Para utilizar um plug-in ObjectGridEventListener customizado, primeiro crie uma classe que implementa a interface ObjectGridEventListener e quaisquer

subinterfaces do ObjectGridEventGroup opcionais. Inclua o listener customizado em um ObjectGrid para receber notificação de eventos importantes. Você tem duas abordagens para incluir um plug-in ObjectGridEventListener na configuração do eXtreme Scale: configuração programática e configuração XML.

Configurar um Plug-in do ObjectGridEventListener Programaticamente

Suponha que o nome de classe do listener de eventos do eXtreme Scale seja a classe com.company.org.MyObjectGridEventListener. Esta classe implementa a interface ObjectGridEventListener. O fragmento de código a seguir cria um ObjectGridEventListener customizado e o inclui em um ObjectGrid.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
MyObjectGridEventListener myListener = new MyObjectGridEventListener();
myGrid.addEventListener(myListener);
```

Configurar um Plug-in do ObjectGridEventListener com XML

Também é possível configurar um plug-in ObjectGridEventListener utilizando XML. O XML a seguir cria uma configuração que é equivalente ao listener de eventos do ObjectGrid programaticamente criado e descrito. O texto a seguir deve estar no arquivo `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>
```

Observe que as declarações de bean aparecem antes das declarações do backingMap. Forneça este arquivo para o plug-in ObjectGridManager para facilitar a criação desta configuração. O fragmento de código a seguir demonstra como criar uma instância do ObjectGrid utilizando este arquivo XML. A instância do ObjectGrid criada possui um listener ObjectGridEventListener configurado no ObjectGrid myGrid.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", new URL("file:etc/test/myGrid.xml"), true, false);
```

Plug-in BackingMapLifecycleListener

Um plug-in BackingMapLifecycleListener recebe a notificação de eventos de mudança de estado do ciclo de vida do WebSphere eXtreme Scale para o mapa de apoio.

O plug-in BackingMapLifecycleListener recebe um evento contendo um objeto BackingMapLifecycleListener.State para cada mudança de estado do mapa de apoio. Qualquer plug-in BackingMap que também implementa a interface BackingMapLifecycleListener será automaticamente ser incluído como um listener na instância do BackingMap na qual o plug-in está registrado.

Visão Geral

Um plug-in BackingMapLifecycleListener é útil quando um plug-in BackingMap existente precisa executar atividades referentes às atividades em um plug-in

relacionado. Como exemplo, um plug-in do carregador precisar recuperar a configuração a partir de um plug-in MapIndexPlugin ou DataSerializer cooperantes.

Ao implementar a interface BackingMapLifecycleListener e detectar o evento BackingMapLifecycleListener.State.INITIALIZED, o carregador pode saber sobre o estado de outros plug-ins na instância do BackingMap. O carregador pode recuperar informações com segurança a partir do plug-in MapIndexPlugin ou DataSerializer cooperante, desde que o BackingMap esteja no estado INITIALIZED, significando que o método initialize() do outro plug-in foi chamado.

Um BackingMapLifecycleListener pode ser incluídos ou removidos a qualquer momento, seja antes ou depois do ObjectGrid e seus BackingMaps são inicializados.

Gravar o Plug-in BackingMapLifecycleListener

Um plug-in BackingMapLifecycleListener deve implementar a interface BackingMapLifecycleListener para receber notificações sobre eventos significativos do eXtreme Scale . Qualquer plug-in BackingMap pode implementar a interface BackingMapLifecycleListener e ser automaticamente incluído como um listener quando ele também for incluído no mapa de apoio.

Para obter mais informações sobre essas interfaces, consulte a Documentação da API.

Evento de Ciclo de Vida e Relacionamentos de Plug-in

O BackingMapLifecycleListener recupera o estado do ciclo de vida a partir do evento no método backingMapStateChanged, por exemplo:

```
public void backingMapStateChanged(BackingMap map,
                                   LifecycleEvent event)
throws LifecycleFailedException {
  switch(event.getState()) {
    case INITIALIZED: // All other plug-ins are initialized.
      // Retrieve reference to plug-in X for use from map.
      break;
    case DESTROYING:  // Destroy phase is starting
      // Eliminate reference to plug-in X it may be destroyed before this plug-in
      break;
  }
}
```

A tabela a seguir descreve o relacionamento entre os eventos de ciclo de vida enviados para um plug-in BackingMapLifecycleListener e os estados do BackingMap e de outros objetos do plug-in.

Valor de BackingMapLifecycleListener.State	Descrição
INITIALIZING	A fase de inicialização do BackingMap está começando. O BackingMap e os plug-ins do BackingMap estão prestes a serem inicializados.
INITIALIZED	A fase de inicialização do BackingMap foi concluída. Todos os plug-ins do BackingMap foram inicializados. O estado INITIALIZED pode recorrer quando as atividades de posicionamento de shard (promoção ou rebaixamento) ocorrem.

Valor de BackingMapLifecycleListener.State	Descrição
STARTING	A instância do BackingMap está sendo ativada para uso como uma instância local, como uma instância do cliente ou como uma instância em um shard primário ou de réplica no servidor. Todos os plug-ins do ObjectGrid na instância do ObjectGrid que possuem esta instância do BackingMap foram inicializados. O estado STARTING pode recorrer quando as atividades de posicionamento de shard (promoção ou rebaixamento) ocorrem.
PRELOAD	A instância do BackingMap está configurada para o estado PRELOAD pela API StateManager para pré-carregamento ou o carregador configurado está pré-carregando dados no mapa de apoio.
ONLINE	A instância do BackingMap está pronta para funcionar como uma instância local, como uma instância do cliente ou como uma instância em um shard primário ou de réplica no servidor. Todos os plug-ins do ObjectGrid na instância do ObjectGrid que possuem esta instância do BackingMap foram inicializados. Esse estado estável é típico do BackingMap. O estado ONLINE pode recorrer quando as atividades de posicionamento de shard (promoção ou rebaixamento) ocorrem.
QUIESCE	O trabalho está parando no BackingMap como resultado da API StateManager ou de outro evento. Nenhum trabalho novo é permitido. Seu plug-in termina qualquer trabalho existente o mais breve possível.
OFFLINE	Todo o trabalho foi interrompido no BackingMap como resultado da API StateManager ou de outro evento. Nenhum trabalho novo é permitido.
DESTROYING	A instância do BackinMap está iniciando a fase de destruição. Os plug-ins do BackingMap para a instância estão prestes a serem destruídos.
DESTROYED	A instância do BackingMap e todos os plug-ins de BackingMap foram destruídos.

Configurar um Plug-in BackingMapLifecycleListener com XML

Suponha que o nome da classe do listener de eventos do eXtreme Scale seja a classe com.company.org.MyBackingMapLifecycleListener. Essa classe implementa a interface BackingMapLifecycleListener.

É possível configurar um plug-in BackingMapLifecycleListener usando XML. O texto a seguir deve estar no arquivo XML da grade de objeto:

```
<?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>
```

Forneça este arquivo para o plug-in ObjectGridManager para facilitar a criação desta configuração. A instância BackingMap criada possui um listener BackingMapLifecycleListener configurado no ObjectGrid myGrid.

Assim como o BackingMapLifecycleListener, outros plug-ins BackingMap, como Loader ou MapIndexPlugin, que são especificados usando XML e que também implementam a interface BackingMapLifecycleListener, serão incluídos automaticamente como listeners de ciclo de vida.

Referências relacionadas

"Plug-in ObjectGridLifecycleListener"
Um plug-in ObjectGridLifecycleListener recebe a notificação de ciclo de vida do WebSphere eXtreme Scale, os eventos de mudança de estado para a grade de dados.

Plug-in ObjectGridLifecycleListener

Um plug-in ObjectGridLifecycleListener recebe a notificação de ciclo de vida do WebSphere eXtreme Scale, os eventos de mudança de estado para a grade de dados.

O plug-in ObjectGridLifecycleListener recebe um evento contendo um objeto ObjectGridLifecycleListener.State para cada mudança de estado do ObjectGrid. Qualquer plug-in ObjectGrid que também implementa a interface ObjectGridLifecycleListener é incluído automaticamente como um listener na instância do ObjectGrid na qual o plug-in está registrado.

Visão Geral

Um plug-in ObjectGridLifecycleListener é útil quando um plug-in ObjectGrid existente precisa executar atividades ligadas às atividades de um plug-in relacionado. Como exemplo, um plug-in TransactionCallback pode precisar recuperar a configuração a partir de um plug-in ObjectGridEventListener ou ShardListener cooperantes.

Ao implementar a interface ObjectGridLifecycleListener e detectar o evento ObjectGridLifecycleListener.State.INITIALIZED, o plug-in TransactionCallback pode detectar o estado de outros plug-ins na instância do ObjectGrid. O plug-in TransactionCallback pode recuperar com segurança informações a partir do plug-in ObjectGridEventListener ou do plug-in ShardListener cooperante, desde que o ObjectGrid esteja em um estado INICIALIZADO, significando que o método initialize() do outro plug-in foi chamado.

É possível incluir um plug-in ObjectGridLifecycleListener a qualquer momento, antes ou depois de inicializar o ObjectGrid.

Gravar um Plug-in ObjectGridLifecycleListener

Um plug-in ObjectGridLifecycleListener deve implementar a interface ObjectGridLifecycleListener para receber notificações sobre eventos significativos do eXtreme Scale. Qualquer plug-in ObjectGrid pode implementar a interface ObjectGridLifecycleListener e ser incluído automaticamente como um listener quando ele também for incluído no ObjectGrid.

Para obter mais informações sobre essas interfaces, consulte a Documentação da API.

Evento de Ciclo de Vida e Relacionamentos de Plug-in

O ObjectGridLifecycleListener recupera o estado do ciclo de vida a partir do evento no método objectgridStateChanged, por exemplo:

```
public void objectGridStateChanged(ObjectGrid grid,
                                   LifecycleEvent event)
throws LifecycleFailedException {
  switch(event.getState()) {
    case INITIALIZED: // All other plug-ins are initialized.
      // Retrieve reference to plug-in X for use from grid.
      break;
    case DESTROYING:  // Destroy phase is starting
      // Eliminate reference to plug-in X it may be destroyed before this plug-in
      break;
  }
```

A tabela a seguir descreve a relação entre os eventos de ciclo de vida enviados para um ObjectGridLifecycleListener e os estados do ObjectGrid e de outros objetos do plug.

Valor de ObjectGridLifecycleListener.State	Descrição
INITIALIZING	A fase de inicialização do ObjectGrid está começando. O ObjectGrid e os plug-ins do ObjectGrid estão prestes a serem inicializados.
INITIALIZED	A fase de inicialização do ObjectGrid foi concluída. Todos os plug-ins do ObjectGrid foram inicializados. O estado INITIALIZED pode recorrer quando as atividades de posicionamento de shard (promoção ou rebaixamento) ocorrem. Todos os plug-ins de BackingMap nas instâncias do BackingMap de propriedade desta instância do ObjectGrid foram inicializados.
STARTING	A instância do ObjectGrid está sendo ativada para uso como uma instância local, como uma instância do cliente ou como uma instância em um shard primário ou de réplica no servidor. O estado STARTING pode recorrer quando as atividades de posicionamento de shard (promoção ou rebaixamento) ocorrem.
PRELOAD	A instância do ObjectGrid está configurada para o estado PRELOAD pela API StateManager ou para outra configuração.
ONLINE	A instância do ObjectGrid está pronta para funcionar como uma instância local, como uma instância do cliente ou como uma instância em um shard primário ou de réplica no servidor. Esse estado estável é típico do ObjectGrid. O estado ONLINE pode recorrer quando as atividades de posicionamento de shard (promoção ou rebaixamento) ocorrem.
QUIESCE	O trabalho está parando no ObjectGrid como resultado da API StateManager ou de outro evento. Nenhum trabalho novo é permitido. Termine qualquer trabalho existente o mais breve possível.
OFFLINE	Todo o trabalho é interrompido no ObjectGrid como resultado da API StateManager ou de outro evento. Nenhum trabalho novo é permitido.
DESTROYING	A instância do ObjectGrid está iniciando a fase de destruição. Os plug-ins do ObjectGrid para a instância estão prestes a serem destruídos. Durante a fase de destruição, todas as instâncias do BackingMap possuídas por esta instância do ObjectGrid também serão destruídas.
DESTROYED	A instância do ObjectGrid, suas instâncias do BackingMap e todos os plug-ins do ObjectGrid forem destruídos.

Configurar um Plug-in ObjectGridLifecycleListener com XML

Suponha que o nome da classe do listener de eventos do eXtreme Scale seja a classe com.company.org.MyObjectGridLifecycleListener. Essa classe implementa a interface ObjectGridLifecycleListener.

É possível configurar um plug-in ObjectGridLifecycleListener usando XML. O XML a seguir cria uma configuração usando a interface ObjectGridLifecycleListener. O texto a seguir deve estar no arquivo xml da grade de objeto:

```
<?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>
```

Observe que as declarações de bean aparecem antes das declarações do backingMap. Forneça este arquivo para o plug-in ObjectGridManager para facilitar a criação desta configuração.

Tal como o ObjectGridLifecycleListener registrado no exemplo anterior, outros plug-ins do ObjectGrid, CollisionArbiter ou TransactionCallback por exemplo, que são especificados usando XML que também implementa a interface ObjectGridLifecycleListener, serão incluídos automaticamente como listeners de ciclo de vida.

Referências relacionadas

"Plug-in BackingMapLifecycleListener" na página 322
Um plug-in BackingMapLifecycleListener recebe a notificação de eventos de mudança de estado do ciclo de vida do WebSphere eXtreme Scale para o mapa de apoio.

Plug-ins para Indexar Dados

O HashIndex integrado, a classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, é um plug-in MapIndexPlugin que pode ser incluído no BackingMap para construir índices estáticos ou dinâmicos. Essa classe suporta ambas as interfaces MapIndex e MapRangeIndex. A definição e a implementação de índices podem aprimorar significativamente o desempenho da consulta.

Tarefas relacionadas

"Configurando o Plug-in HashIndex"
É possível configurar o HashIndex integrado, a classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, com um arquivo XML, programaticamente ou com uma anotação de entidade em um mapa de entidade.

"Acessando Dados com Índices (API de Índice)" na página 144
Use indexação para acesso a dados mais eficiente.

Referências relacionadas

"Atributos do Plug-in HashIndex" na página 330
É possível usar os seguintes atributos para configurar o plug-in HashIndex. Esses atributos definem propriedades, como se você estiver usando um atributo ou HashIndex composto ou se a indexação do intervalo estiver ativada.

Configurando o Plug-in HashIndex

É possível configurar o HashIndex integrado, a classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, com um arquivo XML, programaticamente ou com uma anotação de entidade em um mapa de entidade.

Sobre Esta Tarefa

Configurar um índice composto é o mesmo que configurar um índice regular com XML, exceto para o valor da propriedade **attributeName**. Em um índice composto, o valor da propriedade **attributeName** é uma lista de atributos delimitados por vírgulas. Por exemplo, a classe de valor Endereço tem três atributos: cidade, estado e CEP. Um índice composto pode ser definido com o valor da propriedade **attributeName** como `"city,state,zipcode"` indicando que a cidade, estado e CEP são incluídos no índice composto.

Também, note que o HashIndexes composto não suporta consultas de intervalo e, portanto, não pode ter a propriedade RangeIndex configurada para true.

Procedimento

- Configure um índice composto no arquivo descritor XML do ObjectGrid.

  Use o elemento backingMapPluginCollections para definir o 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 um índice composto programaticamente.

  O seguinte código de exemplo cria o mesmo índice composto:

  ```
  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 um índice composto com notações de entidade.

  Se você estiver usando mapas de entidade, será possível usar uma abordagem de anotação para definir um índice composto. É possível definir uma lista de CompositeIndex dentro da anotação CompositeIndexes no nível da classe de entidade. O CompositeIndex possui um nome e a propriedade **attributeNames**. Cada CompositeIndex é associado a uma instância do HashIndex aplicada ao mapa de apoio que é associado à entidade. O HashIndex é configurado como um índice de não-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;
  }
  ```

  A propriedade nomeada para cada índice composto deve ser exclusiva dentro do mapa de entidade e de apoio. Se o nome não for especificado, um nome gerado será usado. A propriedade **attributeName** é usada para preencher o HashIndex attributeName com a lista de atributos delimitados por vírgulas. Os nomes de atributos coincidem com os nomes de campo persistente quando as entidades são configuradas para usar acesso à campo, ou o nome da propriedade como definida para as convenções de nomenclatura JavaBeans para entidades de acesso à propriedade. Por exemplo: Se o nome do atributo for `street`, o método getter da propriedade será denominado getStreet.

Exemplo: Incluindo HashIndex no BackingMap

No exemplo a seguir, o plug-in HashIndex é configurado ao incluir plug-ins de índice estáticos no arquivo 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>
```

Neste exemplo de configuração XML, a classe HashIndex integrada é usada como o plug-in de índice. O HashIndex suporta propriedades que os usuários podem configurar, como Name, RangeIndex e AttributeName.

- A propriedade **Name** é configurada como `CODE`, uma sequência que identifica este plug-in de índice. O valor de propriedade **Name** deve ser exclusivo dentro do escopo do mapa de apoio. O nome pode ser usado para recuperar o objeto do índice pelo nome a partir da instância do ObjectMap do BackingMap.
- A propriedade **RangeIndex** é configurada como `true`, o que significa que o aplicativo pode efetuar cast do objeto do índice recuperado para a interface MapRangeIndex. Se a propriedade RangeIndex for configurada como `false`, o aplicativo poderá apenas efetuar cast do objeto do índice recuperado para a interface MapIndex. Um MapRangeIndex suporta funções para localizar dados usando funções de intervalo, como maior que, menor que, ou ambas, enquanto um MapIndex suporta apenas funções iguais. Se o índice for por consulta, a propriedade **RangeIndex** deverá ser configurada para `true` em índices de atributo único. Para um índice de relacionamento e um índice composto, a propriedade **RangeIndex** deverá ser configurada para `false`.
- A propriedade **AttributeName** é configurada como `employeeCode`, o que significa que o atributo employeeCode do objeto em cache é usado para construir um índice de atributo único. Se um aplicativo precisar procurar por objetos em cache com diversos atributos, a propriedade **AttributeName** poderá ser configurada para uma lista de atributos delimitados por vírgula, rendendo um índice composto.

Em resumo, o exemplo anterior define um HashIndex do intervalo de atributo único. Ele é um HashIndex de atributo único porque o valor da propriedade **AttributeName** `employeeCode` inclui apenas um nome de atributo. É também um HashIndex de intervalo.

Conceitos relacionados

"Plug-ins para Indexar Dados" na página 327
O HashIndex integrado, a classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, é um plug-in MapIndexPlugin que pode ser incluído no BackingMap para construir índices estáticos ou dinâmicos. Essa classe suporta ambas as interfaces MapIndex e MapRangeIndex. A definição e a implementação de índices podem aprimorar significativamente o desempenho da consulta.

"Plug-ins para Indexação Customizada de Objetos de Cache" na página 333
Com um plug-in MapIndexPlugin, ou índice, é possível gravar estratégias de indexação customizadas que vão além de índices integrados fornecidos pelo eXtreme Scale.

"Usando um Índice Composto" na página 336
O HashIndex composto aprimora o desempenho da consulta e evita a custosa varredura de mapa. O recurso também fornece uma maneira conveniente para a API HashIndex localizar objetos em cache quando os critérios de busca envolvem muitos atributos.

"Indexação" na página 97
Use o plug-in MapIndexPlugin para construir um índice ou vários índices em um BackingMap para suportar acesso a dados sem chave.

Referências relacionadas

"Atributos do Plug-in HashIndex"
É possível usar os seguintes atributos para configurar o plug-in HashIndex. Esses atributos definem propriedades, como se você estiver usando um atributo ou HashIndex composto ou se a indexação do intervalo estiver ativada.

Atributos do Plug-in HashIndex:

É possível usar os seguintes atributos para configurar o plug-in HashIndex. Esses atributos definem propriedades, como se você estiver usando um atributo ou HashIndex composto ou se a indexação do intervalo estiver ativada.

Atributos

Nome Especifica o nome do índice. O nome deve ser exclusivo para cada mapa. O nome é usado para recuperar o objeto do da instância de mapa de objetos para o mapa de apoio.

AttributeName
Especifica os nomes delimitados por vírgula dos atributos a serem indexados. Para índices acessados por campo, os nomes de atributos são equivalentes aos nomes de campo. Para índices acessados por propriedade, os nomes de atributos são os nomes da propriedade compatíveis com JavaBean. Se apenas um nome de atributo existir, o HashIndex será um índice de atributo único. Se este atributo for um relacionamento, ele também será um índice de relacionamento. Se múltiplos nomes de atributo são incluídos nos nomes de atributo, o HashIndex é um índice composto.

FieldAccessAttribute
Usado para mapas sem entidade. Se `true`, o objeto será acessado usando os campos diretamente. Se não for especificado ou se especificar `false`, o método getter para o atributo será usado para acessar os dados.

POJOKeyIndex
Usado para mapas sem entidade. Se especificar `true`, o índice examinará o objeto na parte da chave do mapa. Esta configuração é útil quando a chave

é uma chave composta e o valor não tem a chave integrada dentro dele. Se não for especificado ou se especificar `false`, o índice examinará o objeto na parte do valor do mapa.

RangeIndex
Se especificar `true`, a indexação do intervalo será ativada e o aplicativo poderá efetuar cast do objeto do índice recuperado para a interface MapRangeIndex. Se a propriedade **`RangeIndex`** for configurada como `false`, o aplicativo poderá efetuar cast do objeto do índice recuperado apenas para a interface MapIndex.

HashIndex de Atributo Único Versus HashIndex Composto

Quando a propriedade **`AttributeName`** de HashIndex inclui diversos nomes de atributos, o HashIndex será um índice composto. Caso contrário, se ela incluir somente um nome de atributo, ela é um índice de atributo único. Por exemplo, o valor da propriedade AttributeName de um HashIndex composto pode ser `city,state,zipcode`. Ela inclui três atributos delimitados por vírgulas. Se a propriedade valor **`AttributeName`** for apenas `zipcode` que tem apenas um atributo, ele será um HashIndex de atributo único.

O HashIndex composto fornece uma maneira eficiente de consultar objetos em cache quando os critérios de busca envolvem muitos atributos. No entanto, ele não suporta o índice de intervalo e sua propriedade RangeIndex deve ser configurada para `false`.

Consulte o tópico sobre HashIndex composto no *Guia de Administração*.

HashIndex de Relacionamento

Se o atributo indexado de um HashIndex de atributo único for um relacionamento, tanto com valor único ou múltiplos valores, o HashIndex é um HashIndex de relacionamento. Para HashIndex de relacionamento, a propriedade RangeIndex de HashIndex deve ser definida para “false”.

O HashIndex de relacionamento pode acelerar as consultas que usam referências cíclicas ou usam os filtros de consulta IS NULL, IS EMPTY, SIZE e MEMBER OF. Para obter mais informações, consulte o “Otimização de Consulta Utilizando Índices” na página 456informações sobre a otimização da consulta com índices no *Guia de Programação*.

HashIndex Principal

Para mapas sem entidade, quando a propriedade **`POJOKeyIndex`** do HashIndex é configurada para `true`, o HashIndex é um HashIndex principal e a parte da chave da entrada é usada para indexação. Quando a propriedade AttributeName do HashIndex não é especificada, a chave inteira é indexada; caso contrário, o HashIndex principal poderá ser apenas um HashIndex de atributo único.

Por exemplo, a inclusão da seguinte propriedade na amostra precedente faz com que o HashIndex se torne o HashIndex principal porque o valor da propriedade POJOKeyIndex é `true`.

```
<property name="POJOKeyIndex" type="boolean" value="true"
description="indicates if POJO key HashIndex" />
```

No exemplo do índice de chave anterior, como o valor da propriedade **AttributeName** é especificado como `employeeCode`, o atributo indexado é o campo **employeeCode** da parte da chave da entrada do mapa. Se você desejar construir o índice principal na parte da chave inteira da entrada do mapa, remova a propriedade **AttributeName**.

HashIndex de Intervalo

Quando a propriedade RangeIndex do HashIndex é configurada para `true`, o HashIndex é um índice de intervalo e pode suportar a interface MapRangeIndex. Uma implementação MapRangeIndex suporta funções para localizar dados usando funções de intervalo, como maior que, menor que, ou ambas, enquanto que um MapIndex suporta apenas funções iguais. Para um índice de atributo único, a propriedade **RangeIndex** pode ser configurada para `true` apenas se o atributo indexado for do tipo Comparable. Se o índice de atributo único for usado pela consulta, a propriedade RangeIndex deverá ser configurada para `true` e o atributo indexado deverá ser do tipo Comparable. Para o HashIndex de relacionamento e o HashIndex composto, a propriedade RangeIndex deve ser configurada para `false`.

A amostra precedente é um HashIndex de intervalo porque o valor da propriedade RangeIndex é `true`.

A tabela a seguir fornece um resumo do uso do índice de intervalo.

Tabela 5. Suporte para Índice de Intervalo. Define se os tipos de HashIndex suportam o índice de intervalo.

Tipo HashIndex	Suporta índice de intervalo
HashIndex de atributo único: chave ou atributo indexado é do tipo Comparable	Sim
HashIndex de atributo único: chave ou atributo indexado não é do tipo Comparable	Não
HashIndex Composto	Não
HashIndex de Relacionamento	Não

Otimização de Consulta com Plug-ins HashIndex

Definir índices pode significativamente melhorar o desempenho da consulta. As consultas do WebSphere eXtreme Scale podem usar plug-insHashIndex integrados para melhorar o desempenho das consultas. Embora o uso de índices possa aprimorar significativamente o desempenho da consulta, isso pode ter um impacto no desempenho nas operações do mapa transacional.

Conceitos relacionados

"Plug-ins para Indexar Dados" na página 327
O HashIndex integrado, a classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, é um plug-in MapIndexPlugin que pode ser incluído no BackingMap para construir índices estáticos ou dinâmicos. Essa classe suporta ambas as interfaces MapIndex e MapRangeIndex. A definição e a implementação de índices podem aprimorar significativamente o desempenho da consulta.

"Plug-ins para Indexação Customizada de Objetos de Cache"
Com um plug-in MapIndexPlugin, ou índice, é possível gravar estratégias de indexação customizadas que vão além de índices integrados fornecidos pelo eXtreme Scale.

"Usando um Índice Composto" na página 336
O HashIndex composto aprimora o desempenho da consulta e evita a custosa varredura de mapa. O recurso também fornece uma maneira conveniente para a API HashIndex localizar objetos em cache quando os critérios de busca envolvem muitos atributos.

"Indexação" na página 97
Use o plug-in MapIndexPlugin para construir um índice ou vários índices em um BackingMap para suportar acesso a dados sem chave.

Tarefas relacionadas

"Configurando o Plug-in HashIndex" na página 327
É possível configurar o HashIndex integrado, a classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, com um arquivo XML, programaticamente ou com uma anotação de entidade em um mapa de entidade.

"Acessando Dados com Índices (API de Índice)" na página 144
Use indexação para acesso a dados mais eficiente.

Plug-ins para Indexação Customizada de Objetos de Cache:

Com um plug-in MapIndexPlugin, ou índice, é possível gravar estratégias de indexação customizadas que vão além de índices integrados fornecidos pelo eXtreme Scale.

As implementações MapIndexPlugin devem usar a interface MapIndexPlugin e seguir as convenções comuns do plug-in do eXtreme Scale.

As seções a seguir incluem alguns dos métodos importantes da interface de índice.

Método setProperties

Use o método setProperties para inicializar programaticamente do plug-in de índice. O parâmetro Objeto de propriedades transmitido para o método deve conter as informações necessárias sobre configuração para inicializar o plug-in de índice adequadamente. A implementação do método setProperties, junto com a do método getProperties, são necessárias em um ambiente distribuído pois a configuração do plug-in de índice se move entre os processos do cliente e do servidor. A seguir está um exemplo de implementação deste método.

```
setProperties(Properties properties)

// setProperties method sample code
    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

O método getProperties extrai a configuração do plug-in de índice de uma instância MapIndexPlugin. É possível usar as propriedades extraídas para inicializar outra instância do MapIndexPlugin para ter os mesmos estados internos. Os implementações dos métodos getProperties e setProperties são necessárias em um ambiente distribuído. A seguir há um exemplo de implementação do método getProperties.

```
getProperties()

// getProperties method sample code
    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

O método setEntityMetadata é chamado pelo tempo de execução do WebSphere eXtreme Scale durante a inicialização para configurar EntityMetadata do BackingMap associado na instância MapIndexPlugin. O EntityMetadata é necessário para suportar a indexação de objetos de tupla. Uma tupla é um conjunto de dados que representa um objeto da entidade ou sua chave. Se o BackingMap for para uma entidade, então, é necessário implementar este método.

O código de amostra a seguir implementa o método setEntityMetadata.

```
setEntityMetadata(EntityMetadata entityMetadata)

// setEntityMetadata method sample code
    public void setEntityMetadata(EntityMetadata entityMetadata) {
        ivEntityMetadata = entityMetadata;
        if (ivEntityMetadata != null) {
            // este é um mapa de tupla
            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) {
                // did not find the attribute in value tuple, try to find it on key tuple.
                // if found on key tuple, implies key indexing on one of tuple key attributes.
                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) {
            // if entityMetadata is not null and we could not find the
   // attributeName in entityMetadata, this is an
            // error
            throw new ObjectGridRuntimeException("Invalid attributeName.
Entity: " + ivEntityMetadata.getName());
        }
    }
}
```

Métodos de Nome do Atributo

O método setAttributeName configura o nome do atributo a ser indexado. A classe de objeto de cache deve fornecer o método get para o atributo indexado. Por exemplo, se o objeto possuir um atributo employeeName ou EmployeeName, o índice chama o método getEmployeeName no objeto para extrair o valor de atributo. O nome do atributo deve ser o mesmo nome no método get e o atributo deve implementar a interface Comparable. Se o atributo for do tipo booleano, também é possível utilizar o método padrão isAttributeName.

O método getAttributeName retorna o nome do atributo indexado.

Método getAttribute

O método getAttribute retorna o valor de atributo indexado do objeto especificado. Por exemplo, se um objeto Employee possui um atributo denominado employeeName que é indexado, o método getAttribute pode ser utilizado para extrair o valor de atributo employeeName de um objeto Employee especificado. Este método é necessário em um ambiente WebSphere eXtreme Scale distribuído.

```
getAttribute(Object value)

// getAttribute method sample code
    public Object getAttribute(Object value) throws ObjectGridRuntimeException {
        if (ivPOJOKeyIndex) {
            // In the POJO key indexing case, no need to get attribute from value object.
            // The key itself is the attribute value used to build the index.
            return null;
        }

        try {
            Object attribute = null;
            if (value != null) {
                // handle Tuple value if ivTupleValueIndex != -1
                if (ivTupleValueIndex == -1) {
                    // regular value
                    if (ivFieldAccessAttribute) {
                        attribute = this.getAttributeField(value).get(value);
                    } else {
                        attribute = getAttributeMethod(value).invoke(value, emptyArray);
                    }
                } else {
                    // Tuple value
                    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);
        }
    }
```

Tarefas relacionadas

"Configurando o Plug-in HashIndex" na página 327
É possível configurar o HashIndex integrado, a classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, com um arquivo XML, programaticamente ou com uma anotação de entidade em um mapa de entidade.

"Acessando Dados com Índices (API de Índice)" na página 144
Use indexação para acesso a dados mais eficiente.

Referências relacionadas

"Atributos do Plug-in HashIndex" na página 330
É possível usar os seguintes atributos para configurar o plug-in HashIndex. Esses atributos definem propriedades, como se você estiver usando um atributo ou HashIndex composto ou se a indexação do intervalo estiver ativada.

Usando um Índice Composto:

O HashIndex composto aprimora o desempenho da consulta e evita a custosa varredura de mapa. O recurso também fornece uma maneira conveniente para a API HashIndex localizar objetos em cache quando os critérios de busca envolvem muitos atributos.

Desempenho Melhorado

Um HashIndex composto fornecer uma maneira rápida e conveniente de buscar objetos em cache com múltiplos atributos em critérios de busca de correspondência. O índice composto suporta todas as buscas de correspondência de atributo, mas não suportam buscas de intervalo.

Nota: Índices compostos não suportam o operador BETWEEN no idioma de consulta do ObjectGrid porque BETWEEN necessitaria de suporte de intervalo. Os condicionais maior que (>) e menor que (<) também não funcionam porque necessitam de índices de intervalo.

Um índice composto pode melhorar o desempenho de consultas se o índice composto apropriado estiver disponível para a condição WHERE. Isso significa que o índice composto tem exatamente os mesmos atributos envolvidos na condição WHERE com atributos integrais correspondidos.

Uma consulta pode ter muitos atributos envolvidos em uma condição como no exemplo a seguir.

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND
a.zipcode='55901'
```

O índice composto pode melhorar o desempenho da consulta evitando a varredura de mapa ou unindo vários resultados de índice de atributo único. No exemplo, se o índice composto for definido com atributos (city,state,zipcode), o mecanismo de consulta poderá utilizar o índice composto para localizar a entrada com `city='Rochester'`, `state='MN'` e `zipcode='55901'`. Sem índice composto e índice de atributo nos atributos cidade, estado e código postal, o mecanismo de consulta terá de varrer o mapa ou juntar-se a múltiplas buscas de atributo único, o que geralmente tem gasto adicional caro. Também, a consulta de índice composto somente suporta um padrão integralmente correspondido.

Configurando um Índice Composto

É possível configurar a indexação composta de três maneiras: usando XML, programaticamente, e com anotações de entidade apenas para mapas de entidade.

Configuração Programática

O código de exemplo programático abaixo criará o mesmo índice composto do XML precedente.

```
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 a configuração de um índice composto é a mesma que a configuração de um índice regular com XML, exceto para o valor da propriedade attributeName. No caso de um índice composto, o valor de attributeName é uma lista de atributos delimitados por vírgulas. Por exemplo, a classe de valor Endereço tem 3 atributos: cidade, estado e código postal. Um índice composto pode ser definido com o valor da propriedade attributeName como `"city,state,zipcode"` indicando a cidade, estado e código postal são incluídos no índice composto.

Também, note que o HashIndexes composto não suporta consultas de intervalo e, portanto, não pode ter a propriedade RangeIndex configurada para true.

Utilizando XML

Para configurar um índice composto com XML, inclua código como o abaixo no elemento backingMapPluginCollections do arquivo de configuração.

```
Composite index - XML configuration approach
<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>
```

Com anotações de entidade

No caso do mapa de entidade, a abordagem de anotação pode ser utilizada para definir um índice composto. É possível definir uma lista de CompositeIndex sem a anotaçãoCompositeIndexes no nível da classe de entidade. O CompositeIndex possui uma propriedade name e attributeNames. Cada CompositeIndex é associado com uma instância HashIndex aplicada à BackingMap associada da entidade. O HashIndex é configurado como um índice de não-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;
}
```

A propriedade nomeada para cada índice composto deve ser única dentro da entidade e do BackingMap. Se o nome não for especificado, um nome gerado será utilizado. A propriedade attributeNames é utilizada para preencher o HashIndex attributeName com a lista de atributos delimitados por vírgulas. Os nomes de

atributos coincidem com os nomes de campo persistente quando as entidades são configuradas para usar acesso à campo, ou o nome da propriedade como definida para as convenções de nomenclatura JavaBeans para entidades de acesso à propriedade. Por exemplo: Se o nome do atributo for "street", o método getter da propriedade é denominado getStreet.

Executando Consultas de Índice Composto

Após um índice composto ser configurado, um aplicativo pode utilizar o método findAll(Object) da interface MapIndex para executar consultas, conforme abaixo.

```
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);
```

O MapIndex.EMPTY_VALUE é designado para o compositeValue[0] que indica que o atributo cidade é excluído da avaliação. Somente objetos com atributo estado igual a "MN" e atributo código postal igual a "55901" serão incluídos no resultado.

As seguintes consultas se beneficiam da configuração do índice composto 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'
```

O mecanismo de consulta localizará o índice composto apropriado e o usará para melhorar o desempenho da consulta em casos de correspondência de atributo integral.

Em alguns cenários, o aplicativo pode precisar definir múltiplos índices compostos com atributos sobrepostos para satisfazer todas as consultas com atributos integrais correspondidos. Uma desvantagem de aumentar o número de índices é o possível gasto adicional de desempenho em operações de mapa.

Migração e Interoperabilidade

A única restrição para o uso de um índice composto é que um aplicativo não pode configurá-lo em um ambiente distribuído com contêineres heterogêneos. Contêineres antigos e novos não podem ser combinados, já que contêineres antigos não reconhecerão uma configuração de índice composto. O índice composto é exatamente igual ao índice de atributo regular existente, exceto que o anterior permite a indexação sobre vários atributos. Ao utilizar apenas o índice de atributo regular, um ambiente de contêineres combinados ainda é viável.

Tarefas relacionadas

"Configurando o Plug-in HashIndex" na página 327
É possível configurar o HashIndex integrado, a classe com.ibm.websphere.objectgrid.plugins.index.HashIndex, com um arquivo XML, programaticamente ou com uma anotação de entidade em um mapa de entidade.

"Acessando Dados com Índices (API de Índice)" na página 144
Use indexação para acesso a dados mais eficiente.

Referências relacionadas

"Atributos do Plug-in HashIndex" na página 330
É possível usar os seguintes atributos para configurar o plug-in HashIndex. Esses atributos definem propriedades, como se você estiver usando um atributo ou HashIndex composto ou se a indexação do intervalo estiver ativada.

Plug-ins para a Comunicação com os Bancos de Dados

Com um plug-in Loader, um mapa ObjectGrid pode se comportar como um cache de memória para dados que são normalmente mantidos em um armazenamento persistente no mesmo sistema ou em algum outro sistema. Geralmente, um banco de dados ou sistema de arquivos é utilizado como o armazenamento persistente. Uma JVM (Java Virtual Machine) também pode ser usada como a origem de dados, permitindo que caches baseados em hub sejam construídos usando ObjectGrid. Um utilitário de carga possui a lógica para leitura e gravação de dados para um armazenamento persistente e a partir dele.

Os utilitários de carga são plug-ins de mapa de apoio que são chamados quando são feitas alterações no mapa de apoio ou quando o mapa de apoio não pode atender a um pedido de dados (um erro de cache).

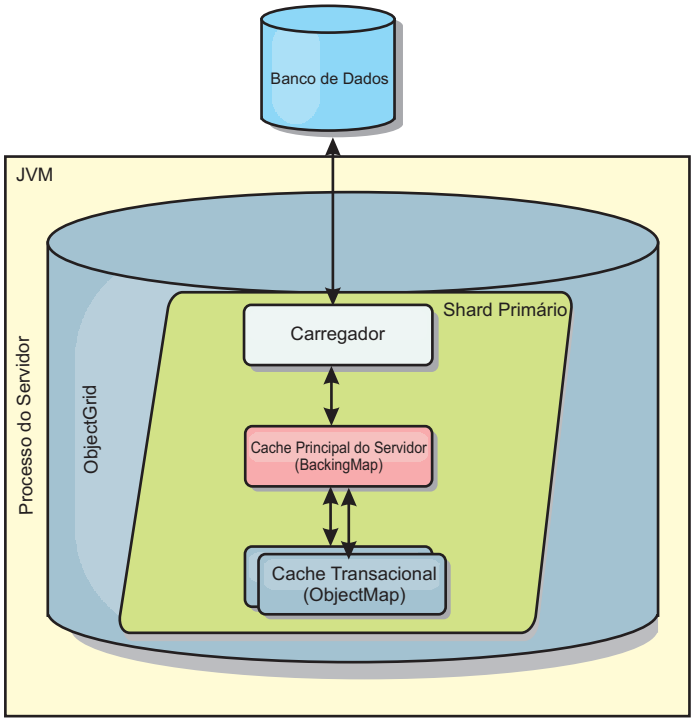


Figura 26. Utilitário de Carga

O WebSphere eXtreme Scale inclui dois utilitários de carga integrados para interagir com os backends de banco de dados relacional. Os utilitários de carga Java Persistence API (JPA) utilizam os recursos Object-Relational Mapping (ORM) das duas implementações OpenJPA e Hibernate da especificação JPA.

Utilizando um Utilitário de Carga

Para incluir um utilitário de carga na configuração do BackingMap, é possível utilizar a configuração programática ou a configuração do XML. Um utilitário de carga possui o seguinte relacionamento com um mapa de apoio:

- Um mapa de apoio pode ter apenas um utilitário de carga.
- Um mapa de apoio de cliente (cache local) não pode ter um utilitário de carga.
- Uma definição de utilitário de carga pode ser aplicado a múltiplos mapas de apoio, mas cada mapa de apoio possui sua própria instância do utilitário de carga.

Carregadores em Configurações Multimestre

Para obter considerações sobre como usar os carregadores em configurações multimestre, consulte "Considerações Sobre o Carregador em uma Topologia Multimestre" na página 106.

Conectando um Utilitário de Carga Programaticamente

O trecho de código a seguir demonstra como conectar o Utilitário de Carga fornecido pelo aplicativo ao mapa de apoio para map1, por meio da API do 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 assume que a classe MyLoader é a classe fornecida pelo aplicativo que implementa a interface com.ibm.websphere.objectgrid.plugins.Loader. Como a associação de um Utilitário de Carga com um mapa de apoio não pode ser alterada após a inicialização de um ObjectGrid, o código deverá ser executado antes de chamar o método initialize da interface ObjectGrid que está sendo chamada. Uma exceção IllegalStateException ocorre em uma chamada de método setLoader se ela for chamada depois de a inicialização ocorrer.

O Utilitário de Carga fornecido pelo aplicativo pode ter propriedades configuradas. No exemplo, o utilitário de carga MyLoader é utilizado para ler e gravar dados de uma tabela em um banco de dados relacional. O utilitário de carga deve especificar o nome do banco de dados e o nível de isolamento do SQL. O loader MyLoader possui os métodos setDataBaseName e setIsolationLevel que permitem que o aplicativo configure estas duas propriedades do Loader.

Abordagem da Configuração XML para Conectar um Utilitário de Carga

Um Utilitário de Carga fornecido pelo aplicativo também pode ser conectado utilizando um arquivo XML. O exemplo a seguir demonstra como conectar o carregador MyLoader ao mapa de apoio map1 com o mesmo nome de banco de dados e propriedades do Carregador de nível de isolamento. É necessário especificar o className para seu utilitário de carga, os detalhes de nome e conexão do banco de dados e as propriedades do nível de isolamento. A mesma estrutura XML poderá ser usada se você estiver usando apenas um pré-carregador ao especificar o nome da classe do pré-carregador em vez de um nome da classe do carregador completo:

```
<?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>
```

Referências relacionadas

"Considerações sobre a Programação do Utilitário de Carga do JPA" na página 366
Um Utilitário de Carga do Java Persistence API (JPA) é uma implementação do plug-in do utilitário de carga que usa o JPA para interagir com o banco de dados. Use as seguintes considerações ao desenvolver um aplicativo que usa um utilitário de carga do JPA.

Configurando Carregadores de Banco de Dados

Os utilitários de carga são plug-ins de mapa de apoio que são chamados quando são feitas alterações no mapa de apoio ou quando o mapa de apoio não pode atender a um pedido de dados (um erro de cache).

Considerações sobre Pré-carregamento

Os utilitários de carga são plug-ins de mapa de apoio que são chamados quando são feitas alterações no mapa de apoio ou quando o mapa de apoio não pode atender a um pedido de dados (um erro de cache). Para obter uma visão geral de como o eXtreme Scale interage com um carregador, consulte o "Cache Sequencial" na página 83.

Cada mapa de apoio tem um atributo preloadMode booleano que é configurado para indicar se o pré-carregamento de um mapa é executado de forma assíncrona. Por padrão, o atributo preloadMode está configurado como false, o que indica que a inicialização do mapa de suporte não será concluída até que o pré-carregamento do mapa esteja concluído. Por exemplo, a inicialização do mapa de apoio não será concluída até que o método preloadMap seja retornado. Se o método preloadMap

ler uma grande quantidade de dados no seu back end e carregá-los para o mapa, o tempo de conclusão desse procedimento pode ser relativamente longo. Neste caso, é possível configurar um mapa de suporte para utilizar o pré-carregamento assíncrono do mapa, configurando o atributo preloadMode como true. Essa configuração faz com que o código de inicialização do mapa de apoio inicie um encadeamento que chama o método preloadMap, permitindo que a inicialização de um mapa de apoio seja concluída enquanto o pré-carregamento do mapa ainda está em andamento.

Em um cenário eXtreme Scale distribuído, um dos padrões de pré-carregamento é o pré-carregamento de cliente. No padrão de pré-carregamento do cliente, um cliente eXtreme Scale é responsável por recuperar dados do backend e, em seguida, inserir os dados no servidor de contêiner distribuído usando agentes DataGrid. Além disso, o pré-carregamento de cliente poderia ser executado no método Loader.preloadMap em uma, e apenas uma, partição específica. Nesse caso, o carregamento assíncrono de dados para a grade se tornaria muito importante. Se o pré-carregamento do cliente fosse executado no mesmo encadeamento, o mapa de apoio nunca seria inicializado, assim, a partição na qual ele reside nunca ficaria ON-LINE. Portanto, o cliente eXtreme Scale não poderia enviar o pedido para a partição e, eventualmente, isso causaria uma exceção.

Se um cliente do eXtreme Scale for usado no método preloadMap, você deverá definir o atributo **preloadMode** como true. A alternativa é iniciar um encadeamento no código de pré-carregamento do cliente.

O trecho de código a seguir ilustra como o atributo preloadMode é configurado para ativar o pré-carregamento assíncrono:

```
BackingMap bm = og.defineMap( "map1" );
bm.setPreloadMode( true );
```

O atributo preloadMode também pode ser configurado utilizando um arquivo XML conforme ilustrado no seguinte exemplo:

```
<backingMap name="map1" preloadMode="true" pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
```

TxID e Utilização da Interface TransactionCallback

Tanto o método get quanto o método batchUpdate na interface do Loader são transmitidos para um objeto TxID que representa a transação Session que requer que a operação get ou batchUpdate seja executada. Os métodos get e batchUpdate podem ser chamados mais de uma vez por transação. Portanto, os objetos com escopo definido pela transação requeridos pelo Loader geralmente são mantidos em um slot do objeto TxID. Um utilitário de carga JDBC (Java Database Connectivity) é usado para ilustrar como um utilitário de carga usa as interfaces TxID e TransactionCallback.

Vários mapas do ObjectGrid podem ser armazenados no mesmo banco de dados. Cada mapa possui seu próprio carregador, e cada carregador pode precisar se conectar ao mesmo banco de dados. Quando os carregadores se conectam com o banco de dados, eles devem usar a mesma conexão JDBC. Usar a mesma conexão confirma as mudanças em cada tabela como parte da mesma transação do banco de dados. Geralmente, a mesma pessoa que grava a implementação Loader também grava a implementação TransactionCallback. O melhor método é quando a interface TransactionCallback é estendida para incluir os métodos que o Loader precisa para obter uma conexão com o banco de dados e para armazenar em cache

as instruções preparadas. O motivo para esta metodologia torna-se aparente à medida que você visualiza como as interfaces TransactionCallback e TxID são utilizadas pelo utilitário de carga.

Como um exemplo, o utilitário de carga pode precisar da interface TransactionCallback para ser estendido conforme a seguir:

```
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 );
}
```

Com tais novos métodos, os métodos get e batchUpdate do Loader podem obter uma conexão da seguinte forma:

```
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;
}
```

No exemplo anterior e nos exemplos que seguem, vTcb e ivOcb são variáveis da instância do Carregador que foram inicializadas conforme descrito na seção Considerações de Pré-carregamento. A variável ivTcb é uma referência à instância MyTransactionCallback e ivOcb é uma referência à instância MyOptimisticCallback. A variável databaseName é uma variável da instância do Utilitário de Carga que foi configurada como uma propriedade do Utilitário de Carga durante a inicialização do mapa de suporte. O argumento isolationLevel é uma das constantes da Conexão JDBC que estão definidas para os diversos níveis de isolamento suportados pelo JDBC. Se o Utilitário de Carga estiver utilizando uma implementação otimista, o método get geralmente utilizará uma conexão de autoconfirmação JDBC para buscar os dados do banco de dados. Nesse caso, o Utilitário de Carga pode ter um método getAutoCommitConnection que seja implementado da seguinte forma:

```
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;
}
```

Lembre-se de que o método batchUpdate possui a seguinte instrução 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 um dos métodos buildBatchSQL utiliza a interface MyTransactionCallback para obter uma instrução preparada. Este é um trecho de código que mostra o método buildBatchSQLUpdate construindo uma instrução SQL update para atualizar uma entrada EmployeeRecord e incluindo-a na atualização de batch:

```
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();
}
```

Quando o loop batchUpdate tiver construído todas as instruções preparadas, ele chamará o método getPreparedStatementCollection. Esse método é implementado como segue:

```
private Collection getPreparedStatementCollection( TxID tx, Connection conn )
{
    return ( ivTcb.getPreparedStatementCollection( tx, conn, "employee" ) );
}
```

Quando o aplicativo chama o método commit no objeto Session, o código do Session chamará o método commit, no método TransactionCallback, depois de enviar todas as alterações feitas pela transação fora do Utilitário de Carga para cada mapa alterado pela transação. Como todos os Loaders utilizaram o método MyTransactionCallback para obter todas as conexões e instruções preparadas necessárias, o método TransactionCallback sabe qual conexão utilizar para solicitar que o back end confirme as alterações. Portanto, estender a interface TransactionCallback com métodos requeridos por cada um dos Loaders tem as seguintes vantagens:

- O objeto TransactionCallback encapsula a utilização de slots TxID para dados com escopo definido pela transação e o Loader não requer informações sobre os slots TxID. O Loader apenas precisa saber sobre os métodos que foram incluídos no TransactionCallback utilizando a interface MyTransactionCallback para as funções de suporte requeridas pelo Loader.
- O objeto TransactionCallback pode assegurar que o compartilhamento da conexão ocorra entre cada Loader que se conecta ao mesmo backend para que um protocolo de confirmação de duas fases seja evitado.
- O objeto TransactionCallback pode assegurar que a conexão com o backend seja orientada para conclusão por meio de uma confirmação ou rollback chamado na conexão quando apropriado.
- O TransactionCallback garante a limpeza dos recursos do banco de dados após a conclusão de uma transação.

- O TransactionCallback fica oculto se ele estiver obtendo uma conexão gerenciada a partir de um ambiente gerenciado como WebSphere Application Server ou algum outro servidor de aplicativos compatível com Java 2 Platform, Enterprise Edition (J2EE). Esta vantagem permite que o mesmo código do Loader seja utilizado em ambientes gerenciados e não gerenciados. Apenas o plug-in TransactionCallback deve ser alterado.
- Para obter informações detalhadas sobre como a implementação do TransactionCallback utiliza os slots TxID para dados dentro do escopo da transação, consulte Plug-in do TransactionCallback.

OptimisticCallback

Conforme mencionado anteriormente, o Utilitário de Carga pode utilizar uma abordagem otimista para controle de simultaneidade. Nesse caso, o exemplo do método buildBatchSQLUpdate precisará ser modificado ligeiramente para implementar uma abordagem otimista. Existem várias maneiras possíveis para utilizar uma abordagem otimista. Uma maneira típica é ter uma coluna de time stamp ou uma coluna do contador de números de seqüência para o controle de versões de cada atualização da linha. Suponha que a tabela de funcionários tenha uma coluna de números de seqüência que aumenta sempre que a linha é atualizada. Em seguida, você modifica a assinatura do método buildBatchSQLUpdate para que ela seja transmitida para o objeto LogElement em vez do par chave e valor. Ele também precisa utilizar o objeto OptimisticCallback que está conectado ao mapa de suporte para obter o objeto da versão inicial e para atualizar o objeto da versão. Este é um exemplo de método buildBatchSQLUpdate modificado que utiliza a variável da instância ivOcb inicializada conforme descrito na seção preloadMap:

exemplo de código do método batch-update modificado

```
private void buildBatchSQLUpdate( TxID tx, LogElement le, Connection conn )
 throws SQLException, LoaderException
{
    // Obter o objeto da versão inicial quando esta entrada do
 mapa foi lida pela última vez ou
    // atualizada no banco de dados.
    Employee emp = (Employee) le.getCurrentValue();
    long initialVersion = ((Long) le.getVersionedValue()).longValue();
    // Obter o objeto da versão de Employee atualizado para a operação SQL
    //update.
    Long currentVersion = (Long)ivOcb.getVersionedObjectForValue( emp );
    long nextVersion = currentVersion.longValue();
    // Agora construa o SQL update que inclui o objeto de versão na cláusula where
    // para verificação otimista.
    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();
}
```

O exemplo mostra que o LogElement é utilizado para obter o valor de versão inicial. Quando a transação acessa pela primeira vez a entrada do mapa, é criado um LogElement com o objeto Employee inicial obtido do mapa. O objeto Employee inicial também é transmitido para o método getVersionedObjectForValue na

interface OptimisticCallback e o resultado é salvo no LogElement. Este processamento ocorre antes de um aplicativo receber uma referência ao objeto Employee inicial e de chamar algum método que altere o estado do objeto Employee inicial.

O exemplo mostra que o Loader utiliza o método getVersiondObjectForValue para obter o objeto de versão para o objeto Employee atual atualizado. Antes de chamar o método batchUpdate na interface do utilitário de carga, eXtreme Scale chama o método updateVersionedObjectForValue na interface OptimisticCallback para gerar um novo objeto de versão a ser gerado para o objeto Employee atualizado. Quando o método batchUpdate retornar ao ObjectGrid, o LogElement será atualizado com o objeto de versão atual e se tornará o novo objeto de versão inicial. Esta etapa é necessária porque o aplicativo pode ter chamado o método flush no mapa em vez do método commit na Session. É possível que o Loader seja chamado várias vezes por uma única transação para a mesma chave. Por este motivo, o eXtreme Scale assegura que o LogElement seja atualizado com o novo objeto de versão toda vez que a linha for atualizada na tabela de funcionários.

Agora que o Utilitário de Carga tem o objeto de versão inicial e o próximo objeto de versão, ele pode executar uma instrução SQL update que configura a coluna SEQNO para o próximo valor do objeto de versão e utiliza o valor do objeto de versão inicial na cláusula where. Essa abordagem, às vezes, é referida como uma instrução update super qualificada. A utilização da instrução update super qualificada permite que o banco de dados relacional verifique se a linha não foi alterada por alguma outra transação entre o tempo de leitura do banco de dados por parte da transação e o momento em que esta o atualizou. Se outra transação modificou a linha, a matriz de contagem retornada pela atualização de batch indica que zero linhas foram atualizadas para esta chave. O Utilitário de Carga é responsável por verificar se a operação SQL update atualizou a linha. Se isso não ocorreu, ele exibirá uma exceção com.ibm.websphere.objectgrid.plugins.OptimisticCollisionException para informar o objeto Session que o método batchUpdate falhou porque mais de uma transação simultânea está tentando atualizar a mesma linha na tabela de banco de dados. Esta exceção faz a Session efetuar rollback e o aplicativo deve tentar novamente a transação inteira. O fundamento lógico é que a nova tentativa será bem-sucedida, motivo pelo qual esta abordagem é chamada de otimista. A abordagem otimista apresenta um desempenho melhor quando os dados não sofrem alterações freqüentes ou transações simultâneas raramente tentam atualizar a mesma linha.

É importante que o Utilitário de Carga utilize o parâmetro key do construtor OptimisticCollisionException para identificar qual chave ou conjunto de chaves causou a falha do método batchUpdate otimista. O parâmetro de chave pode ser o próprio objeto de chave ou uma matriz de objetos de chave se mais de uma chave resultar em uma falha de atualização otimista. E o eXtreme Scale usa o método getKey do construtor OptimisticCollisionException para determinar quais entradas de mapa contêm dados desatualizados e causaram a exceção. Parte do processamento de rollback é liberar cada entrada do mapa stale do mapa. A liberação de entradas stale é necessária para que qualquer transação subseqüente que acessa a mesma chave ou chaves resulte no método get da interface do Loader que está sendo chamada para atualizar as entradas do mapa com os dados atuais do banco de dados.

Outras maneiras para um Loader implementar uma abordagem otimista incluem:

- Não existe nenhuma coluna de time stamp ou de número de seqüência. Neste caso, o método getVersionObjectForValue na interface OptimisticCallback apenas retorna o próprio objeto de valor como a versão. Com essa abordagem, o

Utilitário de Carga precisa construir uma cláusula WHERE que inclua cada um dos campos do objeto de versão inicial. Essa abordagem não é eficiente e nem todos os tipos de colunas estão qualificados para serem utilizados na cláusula WHERE de uma instrução SQL update super qualificada. Esta abordagem geralmente não é utilizada.

- Não existe nenhuma coluna de time stamp ou de número de seqüência. No entanto, diferente da abordagem anterior, a cláusula WHERE contém apenas os campos de valores modificados pela transação. Um método para detectar quais campos foram modificados é configurar o modo de cópia no mapa de suporte como CopyMode.COPY_ON_WRITE. Esse modo de cópia requer que uma interface de valor seja transmitida para o método setCopyMode na interface BackingMap. O BackingMap cria objetos de proxy dinâmicos que implementam a interface de valor fornecida. Com este modo de cópia, o Loader pode lançar cada valor em um objeto com.ibm.websphere.objectgrid.plugins.ValueProxyInfo. A interface ValueProxyInfo possui um método que permite que o Loader obtenha a Lista de nomes de atributos que foram alterados pela transação. Esse método permite que o Utilitário de Carga chame os métodos get na interface de valor para os nomes de atributos a fim de obter os dados alterados e construa uma instrução SQL update que configure apenas os atributos alterados. A cláusula WHERE agora pode ser construída de modo a ter a coluna-chave primária e cada uma das colunas de atributos alteradas. Esta abordagem é mais eficiente do que a abordagem anterior, mas requer que mais código seja gravado no Loader e gera a possibilidade de que o cache de instrução preparado precise ser maior para manipular as diferentes permutações. No entanto, se as transações geralmente modificarem apenas alguns dos atributos, esta limitação pode não ser um problema.
- Alguns bancos de dados relacionais podem ter uma API para ajudar na manutenção automática de dados da coluna que são úteis para o controle de versões otimista. Consulte a documentação do banco de dados para determinar se existe esta possibilidade.

Criando um Utilitário de Carga

É possível gravar sua própria implementação de plug-in de utilitário de carga em seus aplicativos, que deve seguir as convenções de plug-in do WebSphere eXtreme Scale comuns.

Incluindo um Plug-in do Utilitário de Carga

A interface do Utilitário de Carga possui a seguinte definição:

```
public interface Loader
{
    static final SpecialValue KEY_NOT_FOUND;
    List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException;
    void batchUpdate(TxID txid, LogSequence sequence) throws LoaderException, OptimisticCollisionException;
    void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
}
```

Consulte o “Utilitários de Carga” na página 90 para obter informações adicionais.

Método get

O mapa de apoio chama o método get do utilitário de carga para obter os valores associados a uma lista de chaves, que é transmitida como o argumento keyList. O método get é necessário para retornar uma lista de valores java.lang.util.List, um valor para cada chave que estiver na lista de chaves. O primeiro valor retornado na lista de valores corresponde à primeira chave na lista de chaves, o segundo valor retornado na lista de valores corresponde à segunda chave na lista de chaves e assim por diante. Se o utilitário de carga não localizar o valor para uma chave na

lista de chaves, ele precisará retornar o objeto de valor especial KEY_NOT_FOUND definido na interface do Utilitário de Carga. Como um mapa de apoio pode ser configurado para permitir `null` como um valor válido, é muito importante para o utilitário de carga retornar o objeto especial KEY_NOT_FOUND quando ele não puder localizar a chave. Este valor especial permite que o mapa de apoio faça a distinção entre um valor null e um valor inexistente porque a chave não foi localizada. Se um mapa de suporte não suportar valores null, um utilitário de carga que retorna um valor nulo em vez do objeto KEY_NOT_FOUND para uma chave que não existe resultará em uma exceção.

O argumento forUpdate informa o utilitário de carga se o aplicativo chamou um método get no mapa ou um método getForUpdate no mapa. Consulte a interface ObjectMap na documentação de API para obter mais informações. O Loader é responsável por implementar uma política de controle de simultaneidade que controla o acesso simultâneo ao armazenamento persistente. Por exemplo, muitos sistemas de gerenciamento de banco de dados relacional suportam a sintaxe for update na instrução SQL select utilizada para ler dados a partir de uma tabela relacional. O utilitário de carga pode optar por utilizar a sintaxe for update na instrução SQL select com base se um `true booleano` foi transmitido como o valor de argumento para o parâmetro forUpdate deste método. Geralmente, o Utilitário de Carga utiliza a sintaxe for update apenas quando a política de controle de simultaneidade pessimista for utilizada. Para um controle de simultaneidade otimista, o Utilitário de Carga nunca utiliza a sintaxe `for update` na instrução SQL select. O utilitário de carga é responsável por decidir utilizar o argumento forUpdate com base na política de controle de simultaneidade que está sendo utilizada pelo utilitário de carga.

Para obter uma explicação do parâmetro txid, consulte “Plug-ins para o Gerenciamento de Eventos de Ciclo de Vida da Transação” na página 382.

Método batchUpdate

O método batchUpdate é importante na interface Loader. Este método é chamado sempre que o eXtreme Scale precisa aplicar todas as alterações atuais no Utilitário de Carga. O Utilitário de Carga recebe uma lista de alterações para o Mapa selecionado. As alterações são iteradas e aplicadas ao backend. O método recebe o valor TxID atual e as alterações a serem aplicadas. A amostra a seguir interage sobre o conjunto de alterações e três instruções JDBC (Java Database Connectivity) em lote, uma com insert, outra com update e uma com 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 {
        // Obter uma conexão SQL para utilizar.
        Connection conn = getConnection(tx);
        try {
            // Processar a lista de alterações e construir um conjunto de
   instruções preparadas
            // para executar uma operação SQL update, insert ou delete
            // de batch.
            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;
                }
            }
            // Executar as instruções de batch que foram construídas pelo loop acima.
            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;
        }
    }
```

A amostra anterior ilustra a lógica de alto nível de processamento do argumento LogSequence, mas os detalhes de como uma instrução SQL insert, update ou delete é construída não são ilustrados. Alguns dos pontos-chave que estão ilustrados incluem:

- O método getPendingChanges é chamado no argumento LogSequence para obter um iterador sobre a lista de LogElements que o Utilitário de Carga precisa processar.
- O método LogElement.getType().getCode() é utilizado para determinar se o LogElement serve para uma operação SQL insert, update ou delete.
- Uma exceção SQLException é capturada e encadeada em uma exceção LoaderException exibida para reportar que ocorreu uma exceção durante a atualização do batch.
- O suporte à atualização do batch JDBC é utilizado para reduzir o número de consultas para o backend que devem ser feitas.

Método preloadMap

Durante a inicialização do eXtreme Scale, cada mapa de apoio que é definido é inicializado. Se um Utilitário de Carga for conectado a um mapa de apoio, o mapa de apoio chamará o método preloadMap na interface do Utilitário de Carga para permitir que o utilitário de carga faça a pré-busca de dados de seu backend e carregue os dados no mapa. A amostra a seguir assume que as primeiras 100 linhas de uma tabela Employee são lidas a partir do banco de dados e carregadas no mapa. A classe EmployeeRecord é uma classe fornecida pelo aplicativo que contém os dados de funcionários lidos a partir da tabela de funcionários.

Nota: Essa amostra busca todos os dados do banco de dados e depois os insere no mapa base de uma partição. Em um cenário de implementação do eXtreme Scale distribuído real, os dados devem ser distribuídos em todas as partições. Consulte “Desenvolvendo Carregadores JPA Baseados em Cliente” na página 399 para obter informações adicionais.

```
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();
```

```
            // Obter uma conexão de autoconfirmação para utilização que esteja configurada para
            // um nível de isolamento confirmado por leitura.
            conn = getAutoCommitConnection(tx);
            // Pré-carregar o Mapa Employee com objetos EmployeeRecord.
            // Ler todos os Funcionários a partir da tabela, mas
            // limitar o pré-carregamento às primeiras 100 linhas.
            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 a transação.
            session.commit();
            tranActive = false;
        } catch (Throwable t) {
            throw new LoaderException("preload failure: " + t, t);
        } finally {
            if (tranActive) {
                try {
                    session.rollback();
                } catch (Throwable t2) {
                    // Tolerar falhas de rollback e
                    // permitir que o Throwable original seja emitido.
                }
            }
            // Certificar-se de limpar outros recursos do banco de dados aqui,
            // bem como instruções de fechamento, conjuntos de resultados, etc.
        }
    }
```

Esta amostra ilustra os seguintes pontos-chave:

- O mapa de suporte preloadMap utiliza o objeto Session transmitido para ele como o argumento de sessão.
- O método Session.beginNoWriteThrough é utilizado para iniciar a transação em vez do método begin.
- O Utilitário de Carga não pode ser chamado para cada operação put que ocorrer neste método para carregar o mapa.
- O utilitário de carga pode mapear colunas da tabela de funcionários em um campo no objeto Java EmployeeRecord. O Utilitário de Carga captura todas as exceções que podem ser emitidas que ocorrem e emite uma exceção LoaderException com a exceção que pode ser emitida capturada encadeada a ele.
- O bloco finally assegura que qualquer exceção que possa ser emitida, e que ocorre entre o tempo em que os métodos beginNoWriteThrough e commit são chamados, faça o bloco finally recuperar a transação ativa. Esta ação é importante para assegurar que qualquer transação que tenha sido iniciada pelo método preloadMap seja concluída antes de retornar ao responsável pela chamada. O bloco finally é um bom local para você executar outras ações de limpeza que podem ser necessárias, como o fechamento da conexão Java Database Connectivity (JDBC) e de outros objetos JDBC.

A amostra preloadMap está utilizando uma instrução SQL select que seleciona todas as linhas da tabela. Em seu Utilitário de Carga fornecido pelo aplicativo, pode ser necessário configurar uma ou mais propriedades do Utilitário de Carga para controlar a quantidade da tabela que precisa ser pré-carregada no mapa.

Como o método preloadMap é chamado apenas uma vez durante a inicialização de BackingMap, ele também é um bom local para executar o código de inicialização do Utilitário de Carga em uma etapa. Mesmo que o Utilitário de Carga opte por não fazer a pré-busca de dados do backend e carregar os dados no mapa,

provavelmente, ele precisará desempenhar alguma outra inicialização em uma etapa para tornar outros métodos do Utilitário de Carga mais eficientes. O exemplo a seguir ilustra o armazenamento em cache do objeto TransactionCallback e do objeto OptimisticCallback como variáveis da instância do Utilitário de Carga para que os outros métodos do Utilitário de Carga não precisem fazer chamadas de método para obter acesso a estes objetos. Este armazenamento em cache de valores de plug-in do ObjectGrid pode ser desempenhado pois, após a inicialização do BackingMap, os objetos TransactionCallback e OptimisticCallback não podem ser alterados ou substituídos. É aceitável armazenar em cache estas referências do objeto como variáveis da instância do Loader.

```
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;

    // Variáveis da instância do Loader.
    MyTransactionCallback ivTcb; // MyTransactionCallback

    // estende TransactionCallback
    MyOptimisticCallback ivOcb; // MyOptimisticCallback

    // implementa OptimisticCallback
    // ...
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException [Replication programming]
        // Armazenar em cache os objetos TransactionCallback e OptimisticCallback
        // em variáveis da instância deste Loader.
        ivTcb = (MyTransactionCallback) session.getObjectGrid().getTransactionCallback();
        ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
        // O restante do código preloadMap (conforme mostrado no exemplo anterior).
    }
```

Para obter informações sobre o pré-carregamento e o pré-carregamento recuperável pertencentes ao failover de replicação, consulte o Replicação para Disponibilidadeinformações sobre a replicação no *Visão Geral do Produto*.

Utilitários de Carga com Mapas de Entidade

Se o utilitário de carga for conectado a um mapa de entidade, o utilitário de carga deverá lidar com os objetos de tupla. Os objetos de tupla são um formato de dados de entidade especial. O utilitário de carga deve converter os dados entre a tupla e outros formatos de dados. Por exemplo, o método get retorna uma lista de valores que correspondem ao conjunto de chaves que são transmitidas para o método. As chaves transmitidas estão no tipo de Tupla, chamadas de tuplas de chaves. Assumindo que o utilitário de carga mantém o mapa com um banco de dados utilizando JDBC, o método get deve converter cada tupla de chave em uma lista de valores de atributos que corresponda às colunas de chaves primárias da tabela que é mapeada para o mapa de entidade, executar a instrução SELECT com a cláusula WHERE que utiliza os valores de atributos convertidos como critérios para procurar dados no banco de dados e, em seguida, converter os dados retornados em tuplas de valores. O método get obtém dados do banco de dados e converte os dados em tuplas de valores para as tuplas de chaves transmitidas e, em seguida, retorna uma lista de tuplas de valores que corresponde ao conjunto de chaves de tuplas que são transmitidas para o responsável pela chamada. O método get pode executar uma instrução SELECT para procurar todos os dados de uma vez ou executar uma instrução SELECT para cada tupla de chave. Para obter detalhes sobre a programação que mostra como usar o utilitário de carga quando os dados são armazenados usando um gerenciador de entidades, consulte “Utilizando um Utilitário de Carga com Mapas de Entidade e Tuplas” na página 372.

Referências relacionadas

"Considerações sobre a Programação do Utilitário de Carga do JPA" na página 366
Um Utilitário de Carga do Java Persistence API (JPA) é uma implementação do plug-in do utilitário de carga que usa o JPA para interagir com o banco de dados. Use as seguintes considerações ao desenvolver um aplicativo que usa um utilitário de carga do JPA.

Pré-carregamento de Mapa

Mapas podem ser associados aos utilitários de carga. Um utilitário de carga é utilizado para buscar objetos quando eles não podem ser localizados no mapa (uma ocorrência de cache) e também para gravar alterações em um backend quando ocorre o commit de uma transação. Os Utilitários de Carga também podem ser utilizados para pré-carregar dados para um mapa. O método preloadMap da interface do Utilitário de Carga é chamado em cada mapa quando sua partição correspondente no MapSet torna-se um primário. O método preloadMap não é chamado nas réplicas. Ele tenta carregar todos os dados referenciados destinados a partir do backend no mapa utilizando a sessão fornecida. O mapa relevante é identificado pelo argumento BackingMap que é passado para o método preloadMap.

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

Pré-carregando no MapSet particionado

Os mapas podem ser particionados em N partições. Portanto, os mapas podem ser divididos em vários servidores, com cada entrada identifica por uma chave que é armazenada apenas em um destes servidores. Mapas muitos grandes podem ser mantidos em um eXtreme Scale porque o aplicativo não está mais limitado pelo tamanho de heap de uma JVM única para conter todas as entradas de um Mapa. Aplicativos que desejam pré-carregar com o método preloadMap da interface do Utilitário de Carga deve identificar o subconjunto de dados que ele pré-carrega. Sempre existe um número fixo de partições. É possível determinar este número utilizando o seguinte exemplo de código:

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

Este exemplo de código mostra como um aplicativo pode identificar o subconjunto de dados para pré-carregar a partir do banco de dados. Os aplicativos sempre devem utilizar estes métodos mesmo quando o mapa não é inicialmente particionado. Estes métodos permitem flexibilidade: Se o mapa for posteriormente particionado pelos administradores, então, o utilitário de carga continua a funcionar corretamente.

O aplicativo deve emitir consultas para recuperar o subconjunto *myPartition* a partir do backend. Se um banco de dados for utilizado, então, pode ser mais fácil ter uma coluna com o identificador de partições para um determinado registro, a menos que haja alguma consulta natural que permita que os dados na tabela sejam particionados facilmente.

Desempenho

A implementação de pré-carregamento copia dados do backend para o mapa, armazenando vários objetos no mapa em uma única transação. O número ideal de registros a serem armazenados por transação depende de vários fatores, incluindo complexidade e tamanho. Por exemplo, após a transação incluir blocos de mais de 100 entradas, o benefício do desempenho diminui conforme você aumenta o número de entradas. Para determinar o número ideal, comece com 100 entradas e,

em seguida, aumente o número até que não sejam mais percebidos ganhos de desempenho. Transações maiores resultam em melhor desempenho de replicação. Lembre-se, apenas o primário executa o código de pré-carregamento. Os dados pré-carregados são replicados do primário para quaisquer réplicas que estão on-line.

Pré-carregando MapSets

Se o aplicativo utiliza um MapSet com vários mapas, então, cada mapa possui seu próprio utilitário de carga. Cada utilitário de carga possui um método preload. Cada mapa é carregado serialmente pelo eXtreme Scale. Pode ser mais eficiente pré-carregar todos os mapas, projetando um único mapa como o mapa de pré-carregamento. Esse processo é uma convenção do aplicativo. Por exemplo, dois mapas, department e employee, podem utilizar o Utilitário de Carga de department para pré-carregar os mapas department e employee. Isto assegura que, transacionalmente, se um aplicativo desejar um departamento, os funcionários desse departamento estarão no cache. Quando o Utilitário de Carga do departamento pré-carregar um departamento do backend, ele também buscará os funcionários para esse departamento. O objeto department e seus objetos employee associados são, então, incluídos no mapa utilizando uma transação única.

Pré-carregamento Recuperável

Alguns clientes têm conjuntos de dados muito grandes que precisam ser armazenados em cache. O pré-carregamento de dados pode consumir muito tempo. Às vezes, o pré-carregamento deve ser concluído antes de o aplicativo ficar on-line. É possível beneficiar-se ao tornar o pré-carregamento recuperável. Supunha que haja um milhão de registros para pré-carregar. O primário está pré-carregando-os e falha no registro de número 800.000. Normalmente, a réplica escolhida para ser o novo primário limpa qualquer estado replicado e começa do início. O eXtreme Scale pode utilizar uma interface ReplicaPreloadController. O utilitário de carga para o aplicativo também precisa implementar a interface ReplicaPreloadController. Este exemplo inclui um método único no Utilitário de Carga: `Status checkPreloadStatus(Session session, BackingMap bmap);`. Este método é chamado pelo tempo de execução do eXtreme Scale antes do método preload da interface do Utilitário de Carga ser chamada normalmente. O eXtreme Scale testa o resultado deste método (Status) para determinar seu comportamento sempre que uma réplica é promovida para um primário.

Tabela 6. Valor de Status e Resposta

Valor do Status Retornado	Resposta do eXtreme Scale
Status.PRELOADED_ALREADY	O eXtreme Scale não chama o método preload porque este valor do status indica que o mapa foi totalmente pré-carregado.
Status.FULL_PRELOAD_NEEDED	O eXtreme Scale limpa o mapa e chama o método preload normalmente.
Status.PARTIAL_PRELOAD_NEEDED	O eXtreme Scale deixa o mapa no estado em que se encontra e chama o pré-carregamento. Essa estratégia permite que o Utilitário de Carga do aplicativo continue o pré-carregamento desse ponto em diante.

Claramente, enquanto um primário está pré-carregando o mapa, ele deve deixar algum estado em um mapa no MapSet que está sendo replicado de forma que a réplica determine qual status retornar. É possível utilizar um mapa extra denominado, por exemplo, RecoveryMap. Este RecoveryMap deve fazer parte do MapSet que está sendo pré-carregado para garantir que o mapa seja replicado consistentemente com os dados que estão sendo pré-carregados. A seguir, está uma implementação sugerida.

À medida que ocorre o commit de cada bloco de registros, o processo também atualiza um contador ou valor no RecoveryMap como parte de tal transação. Os dados pré-carregados e os dados de RecoveryMap são replicados atomicamente para as réplicas. Quando a réplica é promovida para o primário, ela pode verificar o RecoveryMap para saber o que aconteceu.

O RecoveryMap pode conter uma única entrada com a chave de estado. Se não existir nenhum objeto para esta chave, será necessário um pré-carregamento completo (checkPreloadStatus retorna FULL_PRELOAD_NEEDED). Se existir um objeto para esta chave de estado e o valor for COMPLETE, o pré-carregamento será concluído e o método checkPreloadStatus retornará PRELOADED_ALREADY. Caso contrário, o objeto de valor indicará de onde o pré-carregamento deve ser reiniciado e o método checkPreloadStatus retornará PARTIAL_PRELOAD_NEEDED. O utilitário de carga pode armazenar o ponto de recuperação em uma variável de instância para o utilitário de carga para que, quando o pré-carregamento for chamado, ele saiba o ponto de partida. O RecoveryMap também pode conter uma entrada por mapa se cada mapa for pré-carregado de maneira independente.

Manipulando a recuperação no modo de replicação síncrono com um Utilitário de Carga

O tempo de execução doeXtreme Scale é projetado para não perder dados com commit quando o primário falha. A seção a seguir mostra os algoritmos utilizados. Estes algoritmos se aplicam apenas quando um grupo de replicação utiliza a replicação síncrona. Um utilitário de carga é opcional.

O tempo de execução do eXtreme Scale pode ser configurado para replicar todas as alterações a partir de um primário para as réplicas de maneira síncrona. Quando uma réplica síncrona é posicionada ela recebe uma cópia dos dados existentes no shard primário. Durante este período, o primário continua a receber transações e copiá-las assincronamente para a réplica. A réplica não é considerada como estando on-line neste período.

Depois de a réplica capturar o primário, ela entre no modo peer e começa a replicação síncrona. Cada transação consolidada no primário é enviada às réplicas síncronas e o primário aguarda por uma resposta de cada réplica. Uma sequência de consolidação síncrona com um utilitário de carga no primário se parece com o conjunto e etapas a seguir:

Tabela 7. Sequência de Commit no Primário

Etapa com o Utilitário de Carga	Etapa sem o Utilitário de Carga
Obter bloqueios para entradas	igual
Limpar alterações no utilitário de carga	no-op
Salvar alterações no cache	igual
Enviar alterações para réplicas e esperar confirmação	igual
Confirmar para o utilitário de carga por meio do Plug-in TransactionCallback	commit do plug-in chamado, mas não faz nada
Liberar bloqueios para entradas	igual

Observe que as alterações são enviadas para a réplica antes de serem confirmadas para o utilitário de carga. Para determinar quando ocorre o commit das alterações na réplica, revise esta sequência: No momento da inicialização, inicialize as listas tx no primário, conforme abaixo.

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

Durante o processamento de confirmação síncrona, utilize a seguinte seqüência:

Tabela 8. Processamento de Commit Síncrono

Etapa com o Utilitário de Carga	**Etapa sem o Utilitário de Carga**
Obter bloqueios para entradas	igual
Limpar alterações no utilitário de carga	no-op
Salvar alterações no cache	igual
Enviar alterações com uma transação confirmada, efetuar rollback da transação para a réplica e esperar confirmação	igual
Limpar lista de transações confirmadas e de transações que receberam rollback	igual
Confirmar o utilitário de carga por meio do plug-in TransactionCallBack	A confirmação do plug-in TransactionCallBack ainda é chamada mas, geralmente, não faz nada
Se a confirmação for bem-sucedida, inclua a transação nas transações confirmadas; caso contrário, inclua nas transações que receberam rollback	no-op
Liberar bloqueios para entradas	igual

Para processamento de réplica, utilize a seguinte seqüência:

1. Receber alterações
2. Confirmar todas as transações recebidas na lista de transações confirmadas
3. Efetuar rollback de todas as transações recebidas na lista de transações que receberam rollback
4. Iniciar uma transação ou sessão
5. Aplicar alterações à transação ou sessão
6. Salvar a transação ou sessão na lista pendente
7. Retornar resposta

Observe que, na réplica, não existem interações do Utilitário de Carga enquanto ele está no modo de réplica. O primário deve enviar todas as alterações por meio do Utilitário de Carga. A réplica não faz nenhuma mudança. Um efeito secundário deste algoritmo é que a réplica sempre tem as transações, mas elas não são confirmadas, até que a próxima transação primária envie o status de confirmação destas transações. Elas são então confirmadas ou recebem rollback na réplica. Mas, até então, as transações não são confirmadas. Podemos incluir um cronômetro no primário que enviará o resultado da transação após um breve período de tempo (alguns segundos). Esse cronômetro limita, mas não elimina, qualquer deterioração desse espaço de tempo. Este staleness é um problemas apenas ao utilizar o modo de leitura de réplica. Do contrário, a deterioração não tem impacto sobre o aplicativo.

Quando o primário falha, é provável que poucos commits ou rollback tenham ocorrido nas transações no primário, mas a mensagem nunca fez isto para a réplica com estas saídas. Quando uma réplica for promovida para o novo primário, uma de suas primeiras ações será manipular esta condição. Cada transação pendente é processada novamente junto ao novo conjunto de mapas do primário. Se houver um Utilitário de Carga, então, cada transação é fornecida para o Utilitário de Carga. Estas transações são aplicadas na ordem FIFO (primeiro a entrar, primeiro a sair) estrita. Se uma transação falhar, ela será ignorada. Se três transações estiverem pendentes, A, B e C, então A poderá confirmar, B poderá efetuar rollback e C também poderá confirmar. Nenhuma transação tem impacto sobre as outras. Suponha que elas sejam independentes.

Um utilitário de carga talvez queira utilizar uma lógica um pouco diferente quando no modo recuperação de failover versus modo normal. O utilitário de carga pode saber facilmente quando está em modo de recuperação de failover, implementando a interface ReplicaPreloadController. O método checkPreloadStatus é chamado apenas quando a recuperação de failover é concluída. Portanto, se o método apply da interface do Utilitário de Carga for chamado antes do checkPreloadStatus, ele será uma transação de recuperação. Depois que o método checkPreloadStatus for chamado, a recuperação de failover estará concluída.

Configurando o Suporte do Carregador Write-behind

Você pode ativar o suporte write-behind usando o arquivo XML descritor do ObjectGrid ou programaticamente usando a interface BackingMap.

Use o arquivo XML descritor do ObjectGrid para ativar o suporte write-behind ou programaticamente usando a interface BackingMap.

Arquivo XML descritor do ObjectGrid

Ao configurar um ObjectGrid usando o arquivo XML descritor do ObjectGrid, o utilitário de carga write-behind é ativado configurando-se o atributo writeBehind na tag backingMap. Este é um exemplo:

```
<objectGrid name="library" >
     <backingMap name="book" writeBehind="T300;C900" pluginCollectionRef="bookPlugins"/>
```

No exemplo anterior, o suporte para write-behind do mapa de apoio book está ativado com o parâmetro T300;C900. O atributo write-behind especifica a duração da atualização máxima e/ou uma contagem máxima de atualização de chave. O formato do parâmetro write-behind é:

```
write-behind attribute  ::= <defaults> | <update time> | <update key count> | <update time> ";" <update key count>
update time  ::= "T" <positive integer>
update key count  ::= "C" <positive integer>
defaults  ::= "" {table}
```

Ocorrem atualizações no utilitário de carga quando um dos seguintes eventos ocorre:

1. O tempo máximo de atualização em segundos decorreu desde a última atualização.
2. O número de chaves atualizadas no mapa de fila alcançou a contagem de chaves de atualização.

Estes parâmetros são apenas dicas. A contagem de atualização real e a duração da atualização estarão dentro do intervalo próximo dos parâmetros. Entretanto, não garantimos que a contagem de atualização real ou a duração da atualização sejam as mesmas que as definidas nos parâmetros. Além disso, a primeira atualização behind pode ocorrer após até o dobro da duração da atualização. Isto ocorre

porque ObjectGrid escolhe aleatoriamente o momento de início da atualização para que todas as partições não cheguem no banco de dados simultaneamente.

No exemplo anterior T300;C900, o utilitário de carga grava os dados no backend quando 300 segundos decorreram desde a última atualização ou quando 900 chaves estão pendentes para serem atualizadas. A duração da atualização padrão é 300 segundos e a contagem de chaves de atualização padrão é de 1000.

Tabela 9. Algumas Opções de write-behind

Valor do Atributo	Tempo
T100	A duração da atualização é 100 segundos e a contagem de chaves de atualização é de 1000 (o valor padrão)
C2000	A duração da atualização é de 300 segundos (o valor padrão) e a contagem de chaves de atualização é de 2000.
T300;C900	A duração da atualização é de 300 segundos e a contagem de chaves de atualização é de 900.
""	A duração da atualização é de 300 segundos (o valor padrão) e a contagem de chaves de atualização é de 1000 (o valor padrão). **Nota:** Se você configurar o utilitário de carga write-behind como uma cadeia vazia: `writeBehind=""`, o utilitário de carga write-behind é ativado usando os valores padrão. Portanto, não especifique o atributo writeBehind se não desejar que o suporte write-behind seja ativado.

Ativando o Suporte Write-behind Programaticamente

Quando estiver criando um mapa de apoio programaticamente para um eXtreme Scale em memória local, o método a seguir pode ser usado na interface BackingMap para ativar e desativar o suporte para write-behind.

```
public void setWriteBehind(String writeBehindParam);
```

Para obter mais detalhes sobre como usar o método setWriteBehind, consulte o informações sobre a interface BackingMap no *Guia de Programação*.

Referências relacionadas

"Exemplo: Gravando uma Classe Dumper no Modo write-behind" na página 363
Essa amostra de código de origem mostra como gravar um watcher (dumper) para manipular atualizações write-behind com falhas.

Armazenamento em Cache Write-behind:

É possível utilizar armazenamento em cache write-behind para reduzir o gasto adicional que ocorre durante a atualização de um banco de dados que você está utilizando como back end.

Visão Geral do Armazenamento em Cache Write-Behind

O armazenamento em cache write-behind enfileira assincronamente as atualizações no plug-in do Utilitário de Carga. É possível melhorar o desempenho desconectando atualizações, inserções e remoções para um mapa, a sobrecarga de atualização do banco de dados de backend. A atualização assíncrona é executada após um atraso baseado em tempo (por exemplo, cinco minutos) ou um atraso baseado em entradas (1000 entradas).

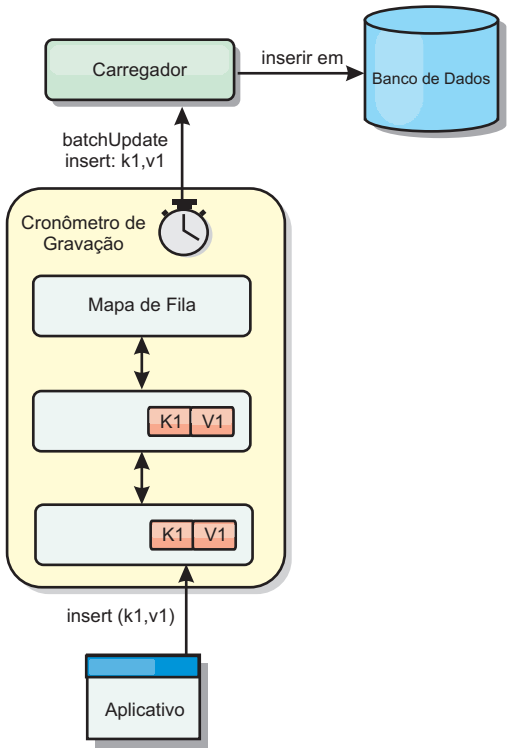


Figura 27. Armazenamento em Cache Write-behind

A configuração write-behind em um BackingMap cria um encadeamento entre o utilitário de carga e o mapa. O utilitário de carga então delega pedidos de dados através do encadeamento de acordo com as definições da configuração no método BackingMap.setWriteBehind. Quando uma transação do eXtreme Scale insere, atualiza ou remove uma entrada de um mapa, um objeto LogElement é criado para cada um destes registros. Estes elementos são enviados para o utilitário de carga write-behind e enfileirados em um ObjectMap especial denominado mapa de fila. Cada mapa de apoio com a configuração write-behind ativada possui seus próprios mapas de fila. Um encadeamento write-behind remove periodicamente os dados enfileirados dos mapas de fila e executa o push deles para o utilitário de carga de backend real.

O utilitário de carga write-behind enviará apenas os tipos insert, update e delete dos objetos LogElement para o utilitário de carga real. Todos os outros tipos de objetos LogElement, por exemplo, o tipo EVICT, são ignorados.

O suporte write-behind *é* uma extensão do plug-in do Carregador, que você usa para integrar o eXtreme Scale ao banco de dados. Por exemplo, consulte as informações do Configurando Utilitários de Carga do JPA sobre como configurar um carregador JPA.

Benefícios

Ativar o suporte write-behind possui os seguintes benefícios:

- **Isolamento de falha de backend:** O armazenamento em cache write-behind fornece uma camada de isolamento das falhas de backend. Quando o banco de dados de backend falha, as atualizações são enfileiradas no mapa de fila. Os

aplicativos podem continuar a conduzir transações para o eXtreme Scale. Quando o backend se recupera, os dados no mapa de fila são enviados para o backend.

- **Carga de backend reduzida:** O utilitário de carga write-behind mescla as atualizações em uma base de chave, portanto, apenas uma atualização mesclada por chave existe no mapa de fila. Esta mesclagem diminui o número de atualizações no backend.
- **Desempenho de transação aprimorado:** Tempos de transação do eXtreme Scale individuais são reduzidos porque a transação não precisa aguardar até que os dados sejam sincronizados com o backend.

Considerações de Design do Aplicativo

Ativar o suporte write-behind é simples, mas o design de um aplicativo para trabalhar com o suporte write-behind precisa de consideração cuidadosa. Sem o suporte de write-behind, a transação de ObjectGrid engloba a transação de backend. A transação do ObjectGrid inicia antes da transação de backend iniciar e termina após a transação de backend terminar.

Com suporte write-behind ativado, a transação do ObjectGrid é concluída antes que a transação de backend inicie. A transação do ObjectGrid e a transação de backend não estão acopladas.

Limitadores de Integridade Referencial

Cada mapa de apoio que é configurado com suporte write-behind possui seu próprio encadeamento write-behind para enviar os dados para o backend. Portanto, os dados que são atualizados em diferentes mapas em uma transação do ObjectGrid são atualizadas no backend em diferentes transações de backend. Por exemplo, a transação T1 atualiza a chave key1 no mapa Map1 e a chave key2 no mapa Map2. A atualização da key1 para o mapa Map1 é atualizada no backend em uma transação de backend e a key2 atualizada para o mapa Map2 é atualizada no backend em outra transação de backend por encadeamentos write-behind diferentes. Se os dados armazenados no Map1 e Map2 possuírem relações, tais como limitadores de chave estrangeira no backend, as atualizações podem falhar.

Ao projetar os limitadores de integridade referencial em seu banco de dados de backend, certifique-se de que atualizações fora de ordem sejam permitidas.

Comportamento do Bloqueio de Mapa de Fila

Outra grande diferença de comportamento da transação é o comportamento do bloqueio. O ObjectGrid suporta três diferentes estratégias de bloqueio: PESSIMISTIC, OPTIMISITIC e NONE. Os mapas de fila write-behind utilizam a estratégia de bloqueio pessimista não importando qual estratégia de bloqueio está configurada para seu mapa de apoio. Existem dois diferentes tipos de operações que adquirem um bloqueio no mapa de fila:

- Quando uma transação do ObjectGrid é confirmada ou um flush (flush de mapa ou flush de sessão) acontece, a transação lê a chave no mapa de fila e coloca um bloqueio S na chave.
- Quando uma transação do ObjectGrid é confirmada, a transação tenta atualizar o bloqueio S para o bloqueio X na chave.

Devido a este comportamento do mapa de fila extra, é possível visualizar algumas diferenças de comportamento de bloqueio.

- Se o mapa do usuário for configurado como a estratégia de bloqueio PESSIMISTIC, não há muita diferença no comportamento de bloqueio. Sempre que um flush ou commit é chamado, um bloqueio S é colocado na mesma chave no mapa de fila. Durante o momento do commit, um bloqueio X não é adquirido apenas para a chave no mapa do usuário, ele também é adquirido para a chave no mapa de fila.
- Se o mapa do usuário for configurado com a estratégia de bloqueio OPTIMISTIC ou NONE, a transação do usuário seguirá o padrão de estratégia de bloqueio PESSIMISTIC. Sempre que um flush ou commit é chamado, um bloqueio S é adquirido para a mesma chave no mapa de fila. Durante o momento do commit, um bloqueio X é adquirido para a chave no mapa de fila utilizando a mesma transação.

Novas Tentativas de Transações do Utilitário de Carga

O ObjectGrid não suporta transações 2-phase ou XA. O encadeamento write-behind remove registros do mapa de fila e atualiza os registros no backend. Se o servidor falhar no meio da transação, algumas atualizações de backend podem ser perdidas.

O utilitário de carga write-behind automaticamente tentará gravar novamente transações falhas e enviará uma LogSequence duvidosa para o backend para evitar a perda de dados. Esta ação requer que o utilitário de carga seja idempotente, o que significa que o Loader.batchUpdate(TxId, LogSequence) é chamado duas vezes com o mesmo valor, ele fornece o mesmo resultado como se tivesse sido aplicado uma vez. As implementações do utilitário de carga devem implementar a interface RetryableLoader para ativar este recurso. Consulte a documentação da API para obter mais detalhes.

Falha do Utilitário de Carga

O plug-in do utilitário de carga pode falhar quando não consegue se comunicar com o back end do banco de dados. Isto pode acontecer se o servidor de banco de dados ou a conexão de rede estiver inativa. O utilitário de carga write-behind irá enfileirar as atualizações e tentará executar o push das alterações de dados para o utilitário de carga periodicamente. O utilitário de carga deve notificar o tempo de execução do ObjectGrid que há um problema de conectividade do banco de dados lançando uma exceção LoaderNotAvailableException.

Portanto, a implementação do Utilitário de Carga deve poder distinguir uma falha de dados ou uma falha de utilitário de carga físico. A falha de dados deve ser lançada ou relançada como uma LoaderException ou uma OptimisticCollisionException, mas uma falha de utilitário de carga físico deve ser lançada ou relançada como uma LoaderNotAvailableException. O ObjectGrid manipula estas exceções de maneira diferente:

- Se uma LoaderException for capturada pelo utilitário de carga write-behind, o utilitário de carga write-behind a considerará falha devido a alguma falha de dados, tal como uma falha de chave duplicada. O utilitário de carga write-behind irá remover a atualização do lote e tentará atualizar um registro em um momento para isolar a falha de dados. Se uma {{LoaderException}} for capturada durante uma atualização de registro, um registro de atualização falho é criado e registrado no mapa de atualização falho.
- Se uma LoaderNotAvailableException for capturada pelo utilitário de carga write-behind, o utilitário de carga write-behind a considerará falha porque não pode se conectar ao final do banco de dados, por exemplo, porque o backend do

banco de dados estiver inativo, uma conexão com o banco de dados não estiver disponível ou a rede estiver inativa. O utilitário de carga write-behind aguardará por 15 segundo e, em seguida, tentará novamente executar uma atualização de lote no banco de dados.

O erro comum é lançar uma LoaderException enquanto uma LoaderNotAvailableException deve ser lançada. Todos os registros enfileirados no utilitário de carga write-behind se tornarão atualizações de registro falhas, o que frustra o propósito do isolamento de falha do backend.

Considerações sobre Desempenho

O suporte ao armazenamento em cache write-behind aumenta o tempo de resposta removendo a atualização do utilitário de carga da transação. Ele também aumenta o rendimento do banco de dados porque as atualizações de banco de dados são combinadas. É importante compreender o gasto adicional introduzido pelo encadeamento write-behind, que executa o pull dos dados da mapa de fila e executa o push para o utilitário de carga.

A contagem máxima de atualização ou o tempo máximo de atualização necessário a ser ajustado com base nos padrões de uso e no ambiente esperados. Se o valor da contagem máxima de atualização ou o tempo máximo de atualização for muito pequeno, o gasto adicional do encadeamento write-behind pode exceder os benefícios. Configurar um valor maior para estes dois parâmetros também pode aumentar o uso da memória para enfileirar os dados e aumentar o tempo de envelhecimento dos registros do banco de dados.

Para obter um melhor desempenho, ajuste os parâmetros write-behind com base nos seguintes fatores:

- Proporção de transações de leitura e gravação
- Mesma frequência de atualização de registro
- Latência de atualização de banco de dados.

Referências relacionadas

"Exemplo: Gravando uma Classe Dumper no Modo write-behind" na página 363
Essa amostra de código de origem mostra como gravar um watcher (dumper) para manipular atualizações write-behind com falhas.

Manipulando Atualizações Write-Behind Falhas:

Como a transação do WebSphere eXtreme Scale termina antes de a transação de backend iniciar, é possível que ocorra um falso sucesso da transação.

Se você tentar inserir uma entrada em uma transação do eXtreme Scale que não exista no mapa de apoio mas existe no banco de dados backend, causando uma chave duplicada, a transação do eXtreme Scale será bem-sucedida. Entretanto, a transação na qual o encadeamento write-behind insere o objeto no banco de dados backend falha com uma exceção de chave duplicada.

Manipulando Atualizações write-behind com Falha: Lado do Cliente

Uma atualização deste tipo, ou qualquer outra atualização de backend falha, é uma atualização write-behind falha. Atualizações write-behind falhas são armazenadas em um mapa de atualização write-behind falho. Este mapa funciona como uma fila de eventos para atualizações falhas. A chave da atualização é um objeto Integer exclusivo e o valor é uma instância do FailedUpdateElement. O mapa de

atualização write-behind com falha é configurado com um evictor, que despeja os registros uma hora depois de terem sido inseridos. Assim, os registros de atualização com falha serão perdidos se eles não forem recuperados dentro de 1 hora.

A API do ObjectMap pode ser utilizada para recuperar as entradas do mapa de atualização write-behind falho. O nome do mapa de atualização write-behind com falha é: IBM_WB_FAILED_UPDATES_<map name>. Consulte a documentação da API do WriteBehindLoaderConstants para os nomes de prefixo de cada um dos mapas do sistema write-behind. A seguir há um exemplo.

processo com falha - código de exemplo

```
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);
    // Faça algo interessante com a chave, o valor ou a exceção.

}
session.commit();
```

Uma chamada de método getNextKey funciona com uma partição específica para cada transação eXtreme Scale. Em um ambiente distribuído, para obter chaves de todas as partições, você deve iniciar diversas transações, como mostrado no exemplo a seguir :

obtendo chaves de todas as partições - código de exemplo

```
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);
        // Faça algo interessante com a chave, o valor ou a exceção.

    }
    Session.commit();
}
```

Nota: O mapa de atualização com falha fornece uma maneira de monitorar o funcionamento do aplicativo. Se um sistema produzir muitos registros no mapa de atualização com falha, significa que é necessário revisar o aplicativo ou a arquitetura para usar o suporte write-behind. É possível usar o comando **xscmd -showMapSizes** para ver o tamanho da entrada do mapa de atualização com falha.

Manipulando atualizações write-behind com Falha: Listener do Shard

É importante detectar e registrar quando uma transação write-behind falha. Todo aplicativo utilizando write-behind precisa implementar um watcher para manipular atualizações write-behind falhas. Isto evita potencialmente ficar sem

memória já os registros no Mapa de atualização inválido não são despejados porque o aplicativo é esperado para manipulá-los.

O código a seguir mostra como conectar tal watcher, ou "dumper ", que deve ser incluído no descritor XML do ObjectGrid como no fragmento.

```
<objectGrid name="Grid">
 <bean id="ObjectGridEventListener" className="utils.WriteBehindDumper"/>
```

É possível visualizar o bean ObjectGridEventListener que foi incluído, que é o watcher write-behind referido acima. O watcher interage nos Mapas para todos os shards principais em um JVM procurando por aqueles com write-behind ativado. Se ele localizar um, então, ele tenta registrar até 100 atualizações inválidas. Ele continua observando um shard principal até que o shard seja movido para um JVM diferente. Todos os aplicativos que usam write-behind devem usar um watcher semelhante a este. Caso contrário, o Java Virtual Machines fica sem memória porque este mapa de erro nunca é despejado.

Consulte o “Exemplo: Gravando uma Classe Dumper no Modo write-behind” para obter informações adicionais.

Referências relacionadas

“Exemplo: Gravando uma Classe Dumper no Modo write-behind”
Essa amostra de código de origem mostra como gravar um watcher (dumper) para manipular atualizações write-behind com falhas.

Exemplo: Gravando uma Classe Dumper no Modo write-behind:

Essa amostra de código de origem mostra como gravar um watcher (dumper) para manipular atualizações write-behind com falhas.

```
//
//This sample program is provided AS IS and may be used, executed, copied and
//modified without royalty payment by customer (a) for its own instruction and
//study, (b) in order to develop applications designed to run with an IBM
//WebSphere product, either for customer's own internal use or for redistribution
//by customer, as part of such an application, in customer's own products. "
//
//5724-J34 (C) COPYRIGHT International Business Machines Corp. 2009
//All Rights Reserved * Licensed Materials - Property of 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;

/**
 * Write behind expects transactions to the Loader to succeed. If a transaction for a key fails then
 * it inserts an entry in a Map called PREFIX + mapName. The application should be checking this
 * map for entries to dump out write behind transaction failures. The application is responsible for
 * analyzing and then removing these entries. These entries can be large as they include the key, before
 * and after images of the value and the exception itself. Exceptions can easily be 20k on their own.
 *
 * The class is registered with the grid and an instance is created per primary shard in a JVM. Isso cria
 * um único encadeamento
 * e depois esse encadeamento verifica cada mapa de erro write behind
para o shard, imprime o problema e
```

```
 * depois remove a entrada.
 *
 * Isso significa que existirá um encadeamento por shard. Se o shard for
movido para outra JVM, o método deactivate
 * irá parar o encadeamento.
 * method stops the thread.
 * @author bnewport
 *
 */
public class WriteBehindDumper implements ObjectGridEventListener,
ObjectGridEventGroup.ShardEvents, Callable<Boolean>
{
 static Logger logger = Logger.getLogger(WriteBehindDumper.class.getName());

 ObjectGrid grid;

 /**
  * Thread pool to handle table checkers. If the application has it's own pool
  * then change this to reuse the existing pool
  */
 static ScheduledExecutorService pool = new ScheduledThreadPoolExecutor(2);
 // two threads to dump records

 // the future for this shard
 ScheduledFuture<Boolean> future;

 // true if this shard is active
 volatile boolean isShardActive;

 /**
  * Normal time between checking Maps for write behind errors
  */
 final long BLOCKTIME_SECS = 20L;

 /**
  * An allocated session for this shard. No point in allocating them again and again
  */
 Session session;
 /**
  * When a primary shard is activated then schedule the checks to periodically check
  * the write behind error maps and print out any problems
  */
 public void shardActivated(ObjectGrid grid)
 {
  try
  {
   this.grid = grid;
   session = grid.getSession();

   isShardActive = true;
   future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS); // check every BLOCKTIME_SECS seconds initially
  }
  catch(ObjectGridException e)
  {
   throw new ObjectGridRuntimeException("Exception activating write dumper", e);
  }
 }

 /**
  * Mark shard as inactive and then cancel the checker
  */
 public void shardDeactivate(ObjectGrid arg0)
 {
  isShardActive = false;
  // if it's cancelled then cancel returns true
  if(future.cancel(false) == false)
  {
   // otherwise just block until the checker completes
   while(future.isDone() == false) // wait for the task to finish one way or the other
   {
    try
    {
     Thread.sleep(1000L); // check every second
    }
    catch (InterruptedException e)
    {
    }
   }
  }
 }

 /**
  * Simple test to see if the map has write behind enabled and if so then return
  * the name of the error map for it.
  * @param mapName The map to test
  * @return The name of the write behind error map if it exists otherwise null
  */
 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;
}

/**
 * This runs for each shard. It checks if each map has write behind enabled and if it does
  * then it prints out any write behind
 * transaction errors and then removes the record.
 */
public Boolean call()
{
 logger.fine("Called for " + grid.toString());
 try
 {
  // while the primary shard is present in this JVM
  // only user defined maps are returned here, no system maps like write behind maps are in
  // this list.
  Iterator<String> iter = grid.getListOfMapNames().iterator();
  boolean foundErrors = false;
  // iterate over all the current Maps
  while(iter.hasNext() && isShardActive)
  {
   String origName = iter.next();

   // if it's a write behind error map
   String name = getWriteBehindNameIfPossible(grid, origName);
   if(name != null)
   {
    // try to remove blocks of N errors at a time
    ObjectMap errorMap = null;
    try
    {
     errorMap = session.getMap(name);
    }
    catch(UndefinedMapException e)
    {
     // at startup, the error maps may not exist yet, patience...
     continue;
    }
    // try to dump out up to N records at once
    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);
      //
      // Your application should log the problem here
      logger.info("WriteBehindDumper ( " + origName + ") for key (" + elem.getKey() + ") Exception: " +
       elem.getThrowable().toString());
      //
      //
      errorMap.remove(seqKey);
     }
     else
      break;
    }
    session.commit();
   }
  } // do next map
  // loop faster if there are errors
  if(isShardActive)
  {
   // reschedule after one second if there were bad records
   // otherwise, wait 20 seconds.
   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();

  //don't leave a transaction on the session.
  if(session.isTransactionActive())
  {
   try { session.rollback(); } catch(Exception e2) {}
  }
 }
 return true;
}

public void destroy() {
```

```
  // TODO Auto-generated method stub

 }

 public void initialize(Session arg0) {
  // TODO Auto-generated method stub

 }

 public void transactionBegin(String arg0, boolean arg1) {
  // TODO Auto-generated method stub

 }

 public void transactionEnd(String arg0, boolean arg1, boolean arg2,
   Collection arg3) {
  // TODO Auto-generated method stub

 }
}
```

Conceitos relacionados

"Configurando o Suporte do Carregador Write-behind" na página 356
Você pode ativar o suporte write-behind usando o arquivo XML descritor do ObjectGrid ou programaticamente usando a interface BackingMap.

"Armazenamento em Cache Write-behind" na página 86
É possível utilizar armazenamento em cache write-behind para reduzir o gasto adicional que ocorre durante a atualização de um banco de dados que você está utilizando como back end.

"Manipulando Atualizações Write-Behind Falhas" na página 361
Como a transação do WebSphere eXtreme Scale termina antes de a transação de backend iniciar, é possível que ocorra um falso sucesso da transação.

Considerações sobre a Programação do Utilitário de Carga do JPA

Um Utilitário de Carga do Java Persistence API (JPA) é uma implementação do plug-in do utilitário de carga que usa o JPA para interagir com o banco de dados. Use as seguintes considerações ao desenvolver um aplicativo que usa um utilitário de carga do JPA.

Entidade eXtreme Scale e Entidade do JPA

É possível designar qualquer classe POJO como uma entidade eXtreme Scale usando as anotações de entidade eXtreme Scale, uma configuração XML ou ambos. Também é possível designar a mesma classe POJO como uma entidade do JPA utilizando anotações de entidades JPA, a configuração XML ou ambas.

Entidade eXtreme Scale : Uma entidade eXtreme Scale representa dados persistentes que são armazenados em mapas ObjectGrid. Um objeto de entidade é transformado em uma tupla de chave e em uma tupla de valor, que então são armazenadas como pares chave-valor nos mapas. Uma tupla é uma matriz de atributos primitivos.

Entidade do JPA: Uma entidade do JPA representa dados persistentes que são armazenados em um banco de dados relacional automaticamente usando uma persistência gerenciada por contêiner. Os dados são persistidos em algum tipo sistema de armazenamento de dados no formato apropriado, como tuplas de banco de dados em um banco de dados.

Quando uma entidade do eXtreme Scale é persistida, suas relações serão armazenadas em outros mapas de entidade. Por exemplo, se você estiver persistindo uma entidade Consumer com uma relação de um para muitos em uma entidade ShippingAddress, se a persistência em cascata for ativada, a entidade ShippingAddress será armazenada no mapa shippingAddress em formato de tupla.

Se você estiver persistindo uma entidade JPA, as entidades do JPA relacionadas também serão persistidas para as tabelas de banco de dados se a persistência em cascata estiver ativada. Quando uma classe POJO é designada como entidade do eXtreme Scale e entidade do JPA, os dados podem ser persistidos para os mapas e bancos de dados da entidade ObjectGrid. Os usos comuns são:

- **Cenário de Pré-Carregamento**: Uma entidade é carregada de um banco de dados usando um provedor do JPA e é persistida nos mapas de entidade do ObjectGrid.
- **Cenário do Utilitário de Carga**: Uma implementação do Utilitário de Carga é conectada aos mapas de entidade ObjectGrid para que uma entidade armazenada nos mapas de entidade ObjectGrid possa ser persistida ou carregada a partir de um banco de dados usando provedores do JPA.

Também é comum que uma classe POJO seja designada apenas como uma entidade JPA. Neste caso, o que é armazenado nos mapas do ObjectGrid são as instâncias do POJO, versus as tuplas de entidade no caso da entidade do ObjectGrid.

Considerações de Design do Aplicativo para Mapas de Entidade

Ao conectar uma instância do JPALoader, as instâncias de objeto serão diretamente armazenadas nos mapas do ObjectGrid.

Porém, ao conectar a um JPAEntityLoader, a classe de entidade é designada como uma entidade do eXtreme Scale e uma entidade do JPA. Neste caso, trate esta entidade como se ela tivesse dois armazenamentos de persistência: os mapas de entidade do ObjectGrid e o armazenamento de persistência do JPA. A arquitetura torna-se mais complicado do que o caso do JPALoader.

Para obter mais informações sobre o plug-in JPAEntityLoader e considerações sobre o design de aplicativo, consulte o informações sobre o plug-in JPAEntityLoader no *Guia de Administração*. Essas informações também podem ajudar se você planejar implementar seu próprio utilitário de carga para os mapas de entidade.

Considerações sobre Desempenho

Certifique-se de configurar o tipo de busca ávido ou lento adequado para os relacionamentos. Por exemplo, um relacionamento um-para-muitos bidirecional de Consumer com ShippingAddress, com o OpenJPA para ajudar a explicar as diferenças de desempenho. Neste exemplo, uma consulta JPA tenta `select o from Consumer o where . . .` efetuar um carregamento em massa e também carregar todos os objetos ShippingAddress relacionados. Um relacionamento um para muitos definido na classe Consumer é o seguinte:

```
@Entity
public class Consumer implements Serializable {

    @OneToMany(mappedBy="consumer",cascade=CascadeType.ALL, fetch =FetchType.EAGER)
    ArrayList <ShippingAddress> addresses;
```

A seguir há o consumidor da relação muitos-para-um definido na classe do ShippingAddress:

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.EAGER)
    Consumer consumer;
}
```

Se os tipos de busca de ambos os relacionamentos estiverem configurados como ávido, o OpenJPA utilizará consultas N+1+1 para obter todos os objetos Consumer e objetos ShippingAddress, em que N é o número de objetos ShippingAddress. Entretanto, se ShippingAddress for alterado para utilizar o tipo de busca lento conforme a seguir, ele fará apenas duas consultas para obter todos os dados.

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.LAZY)
    Consumer consumer;
}
```

Embora a consulta retorne os mesmos resultados, ter um número baixo de consultas diminui significativamente a interação com o banco de dados, o que pode aumentar o desempenho do aplicativo.

Conceitos relacionados

"Plug-ins para a Comunicação com os Bancos de Dados" na página 339
Com um plug-in Loader, um mapa ObjectGrid pode se comportar como um cache de memória para dados que são normalmente mantidos em um armazenamento persistente no mesmo sistema ou em algum outro sistema. Geralmente, um banco de dados ou sistema de arquivos é utilizado como o armazenamento persistente. Uma JVM (Java Virtual Machine) também pode ser usada como a origem de dados, permitindo que caches baseados em hub sejam construídos usando ObjectGrid. Um utilitário de carga possui a lógica para leitura e gravação de dados para um armazenamento persistente e a partir dele.

"Criando um Utilitário de Carga" na página 347
É possível gravar sua própria implementação de plug-in de utilitário de carga em seus aplicativos, que deve seguir as convenções de plug-in do WebSphere eXtreme Scale comuns.

"Plug-in JPAEntityLoader"
O plug-in JPAEntityLoader é uma implementação Loader integrada que usa o Java Persistence API (JPA) para se comunicar com o banco de dados quando usar a API EntityManager. Ao utilizar a API do ObjectMap, utilize o utilitário de carga JPALoader.

"Utilizando um Utilitário de Carga com Mapas de Entidade e Tuplas" na página 372
O gerenciador de entidades converte todos os objetos de entidade em objetos de tupla antes que eles sejam armazenados em um mapa do WebSphere eXtreme Scale. Cada entidade tem uma tupla de chave e uma tupla de valor. Este par chave-valor é armazenado no mapa do eXtreme Scale associado à entidade. Ao usar um mapa do eXtreme Scale com um utilitário de carga, o utilitário de carga deve interagir com os objetos da tupla.

"Gravando um Utilitário de Carga com um Controlador de Pré-carregamento de Réplica" na página 377
Um utilitário de carga com um controlador de pré-carregamento de réplica é um utilitário de carga que implementa a interface ReplicaPreloadController além da interface do utilitário de carga.

"Utilitários de Carga" na página 90
Com um plug-in Carregador, uma grade de dados pode se comportar como um cache de memória para dados que normalmente são mantidos em um armazenamento persistente no mesmo sistema ou em outro sistema. Geralmente, um banco de dados ou sistema de arquivos é utilizado como o armazenamento persistente. Uma JVM (Java Virtual Machine) também pode ser usada como a origem de dados, permitindo que caches baseados em hub seja construído usando o eXtreme Scale. Um utilitário de carga possui a lógica para leitura e gravação de dados para um armazenamento persistente e a partir dele.

Plug-in JPAEntityLoader:

O plug-in JPAEntityLoader é uma implementação Loader integrada que usa o Java Persistence API (JPA) para se comunicar com o banco de dados quando usar a API EntityManager. Ao utilizar a API do ObjectMap, utilize o utilitário de carga JPALoader.

Detalhes do Utilitário de Carga

Use o plug-in do JPAEntityLoader ao armazenar dados usando a API ObjectMap.
Use o plug-in do JPAEntityLoader ao armazenar dados usando a API EntityManager.

Os Utilitários de Carga fornecem duas funções principais:

1. **get**: No método get, o primeiro plug-in JPAEntityLoader chama o método javax.persistence.EntityManager.find(Class entityClass, Object key) para localizar a entidade JPA. Em seguida, o plug-in projeta esta entidade do JPA nas tuplas de entidades. Durante a projeção, ambos os atributos de tuplas e as chaves de associação são armazenados na tupla de valor. Após o processamento de cada chave, o método get retorna uma lista de tuplas de valor de entidades.
2. **batchUpdate**: O método batchUpdate usa um objeto LogSequence que contém uma lista de objetos LogElement. Cada objeto LogElement contém uma tupla de chave uma tupla de valor. Para interagir com o provedor JPA, primeiro é necessário localizar a entidade eXtreme Scale com base na tupla de chave. Com base no tipo LogElement, execute as seguintes chamadas JPA:
   - **insert**: javax.persistence.EntityManager.persist(Object o)
   - **update**: javax.persistence.EntityManager.merge(Object o)
   - **remove**: javax.persistence.EntityManager.remove(Object o)

Um LogElement com tipo **update** faz o JPAEntityLoader chamar javax.persistence.EntityManager.merge(Object o) para mesclar a entidade. Além disso, um LogElement com tipo **update** pode ser o resultado de uma chamada com.ibm.websphere.objectgrid.em.EntityManager.merge(object o) ou uma mudança de atributo da instância gerenciada EntityManager de eXtreme Scale. Consulte o seguinte exemplo:

```
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();
```

Neste exemplo, um LogElement de tipo atualizar é enviado para o JPAEntityLoader do consumidor do mapa. O método javax.persistence.EntityManager.merge(Object o) será chamado para o gerenciador de entidade do JPA em vez de um atributo atualizar para a entidade gerenciada do JPA. Devido a esta mudança de comportamento, há algumas limitações na utilização deste modelo de programação.

Regras de Design do Aplicativo

Entidades possuem relacionamentos com outras entidades. Projetar um aplicativo com relacionamentos envolvidos e com JPAEntityLoader conectado requer considerações adicionais. O aplicativo deve seguir as quatro regras a seguir, descritas nas seções a seguir.

Suporte à Profundidade de Relacionamentos Limitado

O JPAEntityLoader é suportado apenas ao utilizar entidades sem nenhum relacionamento ou entidades com relacionamentos de nível único. Relacionamentos com mais de um nível, como por exemplo, `Company > Department > Employee` não são suportados.

Um Utilitário de Carga por Mapa

Utilizando os relacionamentos da entidade Consumer-ShippingAddress como um exemplo, quando você carrega um consumidor com a busca ávida ativada, é possível carregar todos os objetos ShippingAddress relacionados. Quando você persiste ou funde um objeto Consumer, é possível persistir ou fundir objetos ShippingAddress relacionados se cascade-persist ou cascade-merge estiver ativado.

Não é possível conectar um utilitário de carga para o mapa da entidade-raiz que armazena as tuplas da entidade Consumer. É necessário configurar um utilitário de carga para cada mapa de entidade.

Mesmo tipo de cascata para JPA e eXtreme Scale

Considere novamente o cenário no qual a entidade Consumer possui um relacionamento um-para-muitos com ShippingAddress. É possível examinar o cenário no qual cascade-persist está ativado para este relacionamento. Quando um objeto Consumer é persistido no eXtreme Scale, o número N associado de objetos ShippingAddress será também será persistido no eXtreme Scale.

Uma chamada de persistência do objeto Consumer com um relacionamento cascade-persist com ShippingAddress converte-se em um método javax.persistence.EntityManager.persist(consumer) e *N* chamadas javax.persistence.EntityManager.persist(shippingAddress) pela camada JPAEntityLoader. Entretanto, estas N chamadas de persistência extra para ShippingAddress são desnecessárias devido à configuração cascade-persist do ponto de vista do provedor JPA. Para solucionar este problema, o eXtreme Scale fornece um novo método isCascaded na instância LogElement. O método isCascaded indica se LogElement é um resultado de uma operação em cascata EntityManager do eXtreme Scale. Neste exemplo, o JPAEntityLoader do mapa ShippingAddress recebe *N* objetos LogElement devido às chamadas de persistência em cascata. O JPAEntityLoader descobre que o método isCascaded retorna true e, em seguida, ignora-os sem fazer nenhuma chamada JPA. Portanto, a partir de um ponto de vista do JPA, apenas uma chamada javax.persistence.EntityManager.persist(consumer) é recebida.

O mesmo comportamento é exibido se você fundir uma entidade ou remover uma entidade com cascata ativada. As operações em cascata são ignoradas pelo plug-in JPAEntityLoader.

A estrutura do suporte de cascata é para reproduzir as operações EntityManager do eXtreme Scale para os provedores JPA. Estas operações incluem operações persist, merge e remove. Para ativar o suporte de cascata, verifique se a configuração de cascata para o JPA e o EntityManager do eXtreme Scale sejam os mesmos.

Utilize a Atualização de Entidades com Cuidado

Como descrito anteriormente, a estrutura do suporte em cascata é para reproduzir as operações EntityManager do eXtreme Scale para os provedores do JPA. Se o aplicativo chamar o método ogEM.persist(consumer) para o EntityManager do eXtreme Scale, mesmos os objetos ShippingAddress associados serão persistidos devido à configuração cascade-persist e o JPAEntityLoader chama apenas o método jpAEM.persist(consumer) para os provedores JPA.

Entretanto, se o aplicativo atualizar uma entidade gerenciada, essa atualização será convertida em uma chamada de mesclagem JPA pelo plug-in JPAEntityLoader. Neste cenário, o suporte para vários níveis de relacionamentos e associações-chave não é garantido. Neste caso, a boa prática é utilizar o método javax.persistence.EntityManager.merge(o) em vez de atualizar uma entidade gerenciada.

Referências relacionadas

"Considerações sobre a Programação do Utilitário de Carga do JPA" na página 366
Um Utilitário de Carga do Java Persistence API (JPA) é uma implementação do plug-in do utilitário de carga que usa o JPA para interagir com o banco de dados. Use as seguintes considerações ao desenvolver um aplicativo que usa um utilitário de carga do JPA.

Utilizando um Utilitário de Carga com Mapas de Entidade e Tuplas

O gerenciador de entidades converte todos os objetos de entidade em objetos de tupla antes que eles sejam armazenados em um mapa do WebSphere eXtreme Scale. Cada entidade tem uma tupla de chave e uma tupla de valor. Este par chave-valor é armazenado no mapa do eXtreme Scale associado à entidade. Ao usar um mapa do eXtreme Scale com um utilitário de carga, o utilitário de carga deve interagir com os objetos da tupla.

O eXtreme Scale inclui plug-ins do utilitário de carga que simplificam a integração com bancos de dados relacionais. Os Utilitários de Carga da Java Persistence API (JPA) usam uma Java Persistence API para interagir com o banco de dados e criar os objetos de entidade. Os utilitários de carga JPA são compatíveis com as entidades doeXtreme Scale.

Tuplas

Uma tupla contém informações sobre os atributos e associações de uma entidade. Os valores primitivos são armazenados utilizando seus wrappers primitivos. Outros tipos de objeto suportados são armazenados em seu formato nativo. A associações com outras entidades são armazenadas como uma coleta de objetos de tupla de chave que representam as chaves das entidades de destino.

Cada atributo ou associação é armazenado utilizado um índice baseado em zero. É possível recuperar o índice de cada atributo usando os métodos getAttributePosition ou getAssociationPosition. Depois que a posição ser recuperada, ela permanecerá inalterada durante o ciclo de vida do eXtreme Scale. A posição poderá ser alterada quando o eXtreme Scale for reiniciado. Os métodos setAttribute, setAssociation e setAssociations são usados para atualizar os elementos na tupla.

Atenção: Quando você criar ou atualizar objetos da tupla, atualize cada campo de primitiva com um valor não-nulo. Os valores de primitivas como `int` não devem ser nulos. Se você não alterar o valor para um padrão, poderão ocorrer problemas de desempenho ruim, afetando também campos marcados com a anotação @Version ou atributo de versão no arquivo XML descritor da entidade.

O exemplo a seguir explica como processar as tuplas. Para obter mais informações sobre a definição de entidades para esse exemplo, consulte as informações sobre o esquema de ordem de entidade que se encontram no tutorial do gerenciador de entidade no *Visão Geral do Produto*. O WebSphere eXtreme Scale é configurado para usar os utilitários de carga em cada uma das entidades. Além disso, apenas a entidade Order será usada e esta entidade específica possui um relacionamento muitos-para-um com a entidade Customer. O nome do atributo é `customer` e ele possui um relacionamento um-para-muitos com a entidade OrderLine.

Utilize o Projector para criar objetos de Tupla automaticamente a partir das entidades. A utilização do Projector simplifica os utilitários de carga ao usar um utilitário de mapeamento relacional de objeto, como Hibernate ou 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;
}
```

Uma classe OrderLoader que implementa a interface do utilitário de carga é mostrada no código a seguir. O seguinte exemplo assume que um plug-in TransactionCallback associado esteja definido.

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();
 }

}
```

A variável da instância entityMetaData é inicializada durante a chamada de método preLoadMap a partir do eXtreme Scale. A variável *entityMetaData* não será nula se o Mapa for configurado para usar entidades. Caso contrário, o valor será nulo.

Método batchUpdate

O método batchUpdate fornece a habilidade de saber que ação o aplicativo pretende executar. Com base em uma operação de inserção, atualização ou exclusão, uma conexão pode ser aberta com o banco de dados e o trabalho ser executado. Como a chave e os valores são do tipo Tupla, eles devem ser transformados para que os valores façam sentido na instrução SQL.

A tabela ORDER foi criada com a definição de DDL (Data Definition Language) mostrada no código a seguir:

```
CREATE TABLE ORDER (ORDERNUMBER VARCHAR(250) NOT NULL, DATE TIMESTAMP, CUSTOMER_ID VARCHAR(250))
ALTER TABLE ORDER ADD CONSTRAINT PK_ORDER PRIMARY KEY (ORDERNUMBER)
```

O código a seguir demonstra como converter uma Tupla em um 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) {

// The order has just one key orderNumber
2)                  String ORDERNUMBER=(String) getKeyAttribute("orderNumber", (Tuple) key);
// Get the value of date
3)                  java.util.Date unFormattedDate = (java.util.Date) getValueAttribute("date",(Tuple)value);
// The values are 2 associations. Lets process customer because
// the our table contains customer.id as primary key
4)                  Object[] keys= getForeignKeyForValueAssociation("customer","id",(Tuple) value);
                    //Order to Customer is M to 1. There can only be 1 key
5)                  String CUSTOMER_ID=(String)keys[0];
// parse variable unFormattedDate and format it for the database as formattedDate
6)                   String formattedDate = "2007-05-08-14.01.59.780272"; // formatted for DB2
// Finally, the following SQL statement to insert the record
7) //INSERT INTO ORDER (ORDERNUMBER, DATE, CUSTOMER_ID) VALUES(ORDERNUMBER,formattedDate, CUSTOMER_ID)
                }
                break;
            case LogElement.CODE_UPDATE:
                break;
            case LogElement.CODE_DELETE:
                break;
            }
        }

    }
// returns the value to attribute as stored in the key Tuple
 private Object getKeyAttribute(String attr, Tuple key) {
        //get key metadata
        TupleMetadata keyMD = entityMetaData.getKeyMetadata();
        //get position of the attribute
        int keyAt = keyMD.getAttributePosition(attr);
        if (keyAt > -1) {
            return key.getAttribute(keyAt);
        } else { // attribute undefined
            throw new IllegalArgumentException("Invalid position index for  "+attr);
        }

    }
// returns the value to attribute as stored in the value Tuple
    private Object getValueAttribute(String attr, Tuple value) {
        //similar to above, except we work with value metadata instead
        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);
        }
    }
// returns an array of keys that refer to association.
    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. Certifique-se de que entityMetaData não seja nulo, o que implica na entradas do cache de chave e valor serem do tipo Tuple. A partir de entityMetaData, Key TupleMetaData é recuperado, o que realmente reflete apenas a parte principal dos metadados Order.
2. Processe o KeyTuple e obtenha o valor do Atributo-chave orderNumber
3. Processe o ValueTuple e obtenha o valor da data do atributo
4. Processe o ValueTuple e obtenha o valor das chaves do consumidor da associação
5. Extraia CUSTOMER_ID. Com base no relacionamento, uma ordem pode ter apenas um consumidor; portanto, teremos apenas uma chave. Por isso, o tamanho das chaves é 1. Por simplicidade, ignoramos a análise da data para verificar se o formato está correto.
6. Como esta é uma operação insert, a instrução SQL é transmitida para a conexão da origem de dados para concluir a operação insert.

A demarcação da transação e o acesso ao banco de dados são abordados em "Criando um Utilitário de Carga" na página 347.

Método get

Se a chave não for localizada no cache, chame o método get no plug-in do Utilitário de Carga para localizar a chave.

A chave é uma Tupla. A primeira etapa é converter a Tupla para valores primitivos que possam ser transmitidos na instrução SELECT SQL. Depois de todos os atributos serem recuperados do banco de dados, converta-os em Tuplas. O código a seguir demonstra a classe 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();

    // The order has just one key orderNumber
3)    String ORDERNUMBERKEY = (String) getKeyAttribute("orderNumber",orderKeyTuple);
    //We need to run a query to get values of
4)    // SELECT CUSTOMER_ID, date FROM ORDER WHERE ORDERNUMBER='ORDERNUMBERKEY'

5)    //1) Foreign key: CUSTOMER_ID
6)    //2) date
    // Assuming those two are returned as
7)                              String  CUSTOMER_ID = "C001"; // Assuming Retrieved and initialized
8)    java.util.Date retrievedDate = new java.util.Date();
                                // Assuming this date reflects the one in database

    // We now need to convert this data into a tuple before returning

    //create a value tuple
9)    TupleMetadata valueMD = entityMetaData.getValueMetadata();
    Tuple valueTuple=valueMD.createTuple();


    //add retrievedDate object to Tuple
    int datePosition = valueMD.getAttributePosition("date");
10)    valueTuple.setAttribute(datePosition, retrievedDate);

    //Next need to add the Association
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'
    // Assuming two rows of line number are returned with values 1 and 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 {
   // does not support tuples
  }
  return returnList;
 }
```

1. O método get é chamado quando o cache do ObjectGrid não consegue localizar a chave e solicita que o utilitário de carga faça a busca. Teste o valor para entityMetaData e continue se ele não for nulo.
2. A keyList contém Tuplas.
3. Recupere o valor de atributo orderNumber.
4. Execute a consulta para recuperar a data (valor) e o ID do cliente (chave estrangeira).
5. CUSTOMER_ID é uma chave estrangeira que deve ser configurada na tupla de associação.
6. A data é um valor e já deverá estar configurada.
7. Como este exemplo não executa chamadas JDBC, CUSTOMER_ID é assumido.
8. Como este exemplo não executa chamadas JDBC, a data é assumida.
9. Crie a Tupla de valor.
10. Configure o valor da data na Tupla, com base em sua posição.
11. O pedido possui duas associações. Inicie com o atributo customer que faz referência à entidade do cliente. Você deve ter o valor do ID para configurar na Tupla.
12. Localize a posição do ID na entidade do cliente.
13. Configure os valores apenas das chaves de associação.
14. Além disso, as linhas são uma associação que devem ser configuradas como grupo de chaves de associação, da mesma forma como é feito para a associação do cliente.
15. Como é necessário configurar as chaves para o lineNumber associado a este pedido, execute o SQL para recuperar os valores de lineNumber.

16. Configure as chaves de associação no valueTuple. Isto conclui a criação da Tupla que é retornada ao BackingMap

Este tópico oferece as etapas para criação de tuplas, e uma descrição apenas para a entidade Order. Execute etapas semelhantes para as outras entidades e todo o processo que está ligado ao plug-in TransactionCallback. Consulte “Plug-ins para o Gerenciamento de Eventos de Ciclo de Vida da Transação” na página 382 para obter detalhes.

Referências relacionadas

“Considerações sobre a Programação do Utilitário de Carga do JPA” na página 366
Um Utilitário de Carga do Java Persistence API (JPA) é uma implementação do plug-in do utilitário de carga que usa o JPA para interagir com o banco de dados. Use as seguintes considerações ao desenvolver um aplicativo que usa um utilitário de carga do JPA.

Gravando um Utilitário de Carga com um Controlador de Pré-carregamento de Réplica

Um utilitário de carga com um controlador de pré-carregamento de réplica é um utilitário de carga que implementa a interface ReplicaPreloadController além da interface do utilitário de carga.

A interface ReplicaPreloadController é projetada para fornecer uma maneira para uma réplica que se torna o shard primário saber se o shard primário anterior concluiu o processo de pré-carregamento. Se o pré-carregamento estiver parcialmente concluído, as informações para continuar onde o primário anterior parou são fornecidas. Com a implementação da interface ReplicaPreloadController, uma réplica que se torna o shard primário continua o processo de pré-carregamento onde o shard primário anterior parou e continua a conclusão do pré-carregamento geral.

Em um ambiente distribuído de WebSphere eXtreme Scale, um mapa pode ter réplicas e pode pré-carregar grandes volumes de dados durante a inicialização. O pré-carregamento é uma atividade do utilitário de carga e pode ocorrer apenas no mapa primário durante a inicialização. O pré-carregamento pode demorar muito para concluir, se um grande volume de dados for pré-carregado. Se o mapa primário concluiu uma grande parte dos dados pré-carregados, mas for parado por motivos desconhecidos durante a inicialização, uma réplica torna-se primária. Nessa situação, os dados de pré-carregamento que foram concluídos pela primária anterior são perdidos, pois a nova primária, em geral, desempenha um pré-carregamento incondicional. Com um pré-carregamento incondicional, a nova primária inicia o processo de pré-carregamento do início e os dados pré-carregados anteriormente são ignorados. Se desejar que o novo shard primário continue onde o shard primário anterior parou durante o processo de pré-carregamento, forneça um Utilitário de carga que implemente a interface ReplicaPreloadController. Para obter mais informações, consulte a documentação da API.

Para obter mais informações sobre os Loaders, consulte “Utilitários de Carga” na página 90informações sobre os carregadores no *Visão Geral do Produto*. Se você estiver interessado em gravar um plug-in do Utilitário de Carga comum, consulte “Criando um Utilitário de Carga” na página 347.

A interface ReplicaPreloadController possui a seguinte definição:

```
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);
}
```

As seções a seguir abordam alguns dos métodos da interface Utilitário de Carga e ReplicaPreloadController.

Método checkPreloadStatus

Quando um Utilitário de Carga implementa a interface ReplicaPreloadController, o método checkPreloadStatus é chamado antes do método preloadMap durante a inicialização do mapa. O status de retorno deste método determina se o método preloadMap é chamado. Se este método retornar `Status#PRELOADED_ALREADY`, o método de pré-carregamento não é chamado. Caso contrário, o método preload será executado. Devido a este comportamento, este método deve servir como o método de inicialização do Utilitário de Carga. Você deve inicializar as propriedades do Utilitário de Carga neste método. Este método deve retornar o status correto, ou o pré-carregamento pode não funcionar conforme esperado.

```
public Status checkPreloadStatus(Session session, BackingMap backingMap) {
        // When a loader implements ReplicaPreloadController interface,
    // this method will be called before preloadMap method during
    // map initialization. Whether the preloadMap method will be
    // called depends on teh returned status of this method. So, this
    // method also serve as Loader's initialization method. This method
    // has to return the right staus, otherwise the preload may not
    // work as expected.

        // Note: must initialize this loader instance here.
        ivOptimisticCallback = backingMap.getOptimisticCallback();
        ivBackingMapName = backingMap.getName();
        ivPartitionId = backingMap.getPartitionId();
        ivPartitionManager = backingMap.getPartitionManager();
        ivTransformer = backingMap.getObjectTransformer();
        preloadStatusKey = ivBackingMapName + "_" + ivPartitionId;

        try {
            // get the preloadStatusMap to retrieve preload status that
      // could be set by other JVMs.
            ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);

            // retrieve last recorded preload data chunk index.
            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

A execução do método preloadMap depende do resultado retornado do método checkPreloadStatus. Se o método preloadMap for chamado, ele geralmente deve recuperar as informações de status do pré-carregamento a partir do mapa de status do pré-carregamento designado e determinar como continuar. A forma ideal seria o método preloadMap saber se o pré-carregamento foi parcialmente concluído e onde exatamente deve iniciar. Durante o pré-carregamento de dados, o método preloadMap deve atualizar o status do pré-carregamento no mapa de status do pré-carregamento designado. O status do pré-carregamento que é armazenado no mapa de status de pré-carregamento é recuperado pelo método checkPreloadStatus quando ele precisar verificar o status de pré-carregamento.

```
public void preloadMap(Session session, BackingMap backingMap)
     throws LoaderException {
        EntityMetadata emd = backingMap.getEntityMetadata();
        if (emd != null && tupleHeapPreloadData != null) {
            // The getPreLoadData method is similar to fetching data
      // from database. These data will be push into cache as
      // preload process.
            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);

                    // obter o preloadStatusMap para registrar o status pré-carregado.
                    ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);

                    // Note: when this preloadMap method is invoked, the
         // checkPreloadStatus has been called, Both preloadStatus
         // and preloadedLastDataChunkIndex have been set. And the
         // preloadStatus must be either PARTIAL_PRELOAD_NEEDED
         // or FULL_PRELOAD_NEEDED that will require a preload again.

                    // If large amount of data will be preloaded, the data usually
         // is divided into few chunks and the preload process will
         // process each chunk within its own tran. This sample only
         // preload few entries and assuming each entry represent a chunk.
                    // so that the preload process an entry in a tran to simulate
         // chunk preloading.

                    Set entrySet = ivPreloadData.entrySet();
                    preloadMap = new HashMap();
                    ivMap = preloadMap;

                    // The dataChunkIndex represent the data chunk that is in
         // processing
                    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 = " + whileCo

                        dataChunkIndex++;

                        // if the current dataChunkIndex <= preloadedLastDataChunkIndex
                        // no need to process, becasue it has been preloaded by
           // other JVM before. only need to process dataChunkIndex
           // > preloadedLastDataChunkIndex
                        if (dataChunkIndex <= preloadedLastDataChunkIndex) {
                            System.out.println("ignore current dataChunkIndex =
               " + dataChunkIndex + " that has been previously
```

```
    preloaded.");
                continue;
            }

            // Note: This sample simulate data chunk as an entry.
            // each key represent a data chunk for simplicity.
            // If the primary server or shard stopped for unknown
 // reason, the preload status that indicates the progress
 // of preload should be available in preloadStatusMap. A
// replica that become a primary can get the preload status
// and determine how to preload again.
            // Note: recording preload status should be in the same
 // tran as putting data into cache; so that if tran
// rollback or error, the recorded preload status is the
// actual status.

            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) {
                    // if just only 1 partition, no need to deal with
    // partition.
                    // just push data into cache
                    map.put(key, value);
                    preloadMap.put(key, value);
                    shouldRecordPreloadStatus = true;
                } else if (ivPartitionManager.getPartition(key) == ivPartitionId) {
                    // if map is partitioned, need to consider the
    // partition key only preload data that belongs
    // to this partition.
                    map.put(key, value);
                    preloadMap.put(key, value);
                    shouldRecordPreloadStatus = true;
                } else {
                    // ignore this entry, because it does not belong to
    // this partition.
                }

                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);
                        // means we are at the lastest data chunk, if commit
     // successfully, record preload complete.
     // at this point, the preload is considered to be done
                        // use -99 as special mark for preload complete status.

                        preloadStatusMap.get(preloadStatusKey);

                        // a put follow a get will become update if the get
      // return an object, otherwise, it will be insert.
                        preloadStatusMap.put(preloadStatusKey, new Integer(preloadCompleteMark));

                    } else {
                        // record preloaded current dataChunkIndex into
      // preloadStatusMap a put follow a get will become
      // update if teh get return an object, otherwise, it
      // will be insert.
                        preloadStatusMap.get(preloadStatusKey);
                        preloadStatusMap.put(preloadStatusKey, new Integer(dataChunkIndex));
                    }
                }
```

```
                        session.commit();
                        tranActive = false;

                        // to simulate preloading large amount of data
                        // put this thread into sleep for 30 secs.
                        // The real app should NOT put this thread to sleep
                        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) {
                                // preload ignoring exception from rollback
                            }
                        }
                    }
                }

                // at this point, the preload is considered to be done for sure
                // use -99 as special mark for preload complete status.
                // this is a insurance to make sure the complete mark is set.
                // besides, when partitioning, each partition does not know when
        // is its last data chunk. so the following block serves as the
        // overall preload status complete reporting.
                System.out.println("Overall preload status complete -> record preload status. map = " + this.ivBa
             preloadStatusKey = " + preloadStatusKey + ", mark complete =" +
            preloadCompleteMark);
                session.begin();
                preloadStatusMap.get(preloadStatusKey);
                // a put follow a get will become update if teh get return an object,
                // caso contrário, ele será inserido.
                preloadStatusMap.put(preloadStatusKey, new Integer(preloadCompleteMark));
                session.commit();

                ivMap = preloadMap;
              } catch (Throwable e) {
                e.printStackTrace();
                throw new LoaderException("preload failed with exception: " + e, e);
              }
            }
        }
    }
```

Mapa de Status do Pré-carregamento

É necessário utilizar um mapa de status do pré-carregamento para suportar a implementação da interface ReplicaPreloadController. O método preloadMap deve sempre verificar primeiro o status do pré-carregamento armazenado no mapa de status do pré-carregamento e atualizar o status do pré-carregamento no mapa de status do pré-carregamento sempre que ele enviar os dados para o cache. O método checkPreloadStatus pode recuperar o status do pré-carregamento a partir do mapa de status do pré-carregamento, determinar o status de pré-carregamento e retornar o status para o responsável pela chamada. O mapa de status do pré-carregamento deve estar no mesmo mapSet que outros mapas que possuem utilitários de carga do controlador de pré-carregamento de réplica.

Referências relacionadas

"Considerações sobre a Programação do Utilitário de Carga do JPA" na página 366
Um Utilitário de Carga do Java Persistence API (JPA) é uma implementação do plug-in do utilitário de carga que usa o JPA para interagir com o banco de dados. Use as seguintes considerações ao desenvolver um aplicativo que usa um utilitário de carga do JPA.

Plug-ins para o Gerenciamento de Eventos de Ciclo de Vida da Transação

Use o plug-in TransactionCallback para customizar as operações de versão e de comparação de objetos do cache ao usar a estratégia de bloqueio otimista.

É possível fornecer um objeto de retorno de chamada otimista conectável que implementa a interface com.ibm.websphere.objectgrid.plugins.OptimisticCallback. Para os mapas de entidade, um plug-in OptimisticCallback de alto desempenho é automaticamente configurado.

Propósito

Utilize a interface OptimisticCallback para fornecer operações de comparação otimistas para os valores de um mapa. Uma implementação OptimisticCallback é necessária ao utilizar a estratégia de bloqueio otimista. O WebSphere eXtreme Scale fornece uma implementação OptimisticCallback padrão. No entanto, geralmente o aplicativo deve conectar sua própria implementação da interface OptimisticCallback. Consulte o "Estratégias de Bloqueio" na página 235informações sobre as estratégias de bloqueio no *Visão Geral do Produto* para obter informações adicionais.

Implementação Padrão

A estrutura do eXtreme Scale fornece uma implementação padrão da interface OptimisticCallback que é usada se o aplicativo não for conectado a um objeto OptimisticCallback fornecido pelo aplicativo, como demonstrado na seção anterior. A implementação padrão sempre retorna o valor especial de NULL_OPTIMISTIC_VERSION como o objeto de versão para o valor e nunca atualiza o objeto de versão. Esta ação transforma a comparação optimistic em uma função no operation. Na maioria dos casos, você não quer que a função no operation ocorra quando você estiver usando a estratégia de bloqueio optimistic. Seus aplicativos devem implementar a interface OptimisticCallback e conectar suas próprias implementações de OptimisticCallback para que a implementação padrão não seja utilizada. No entanto, existe pelo menos um cenário no qual a implementação OptimisticCallback fornecida padrão é útil. Considere a seguinte situação:

- Um utilitário de carga está conectado para o mapa de apoio.
- O utilitário de carga sabe como desempenhar a comparação otimista sem assistência de um plug-in OptimisticCallback.

Como o utilitário de carga pode saber como lidar com a versão otimista sem assistência de um objeto OptimisticCallback? O utilitário de carga conhece o objeto de classe de valor e sabe qual campo de objeto de valor é utilizado como um valor de versão otimista. Por exemplo, suponha que a seguinte interface seja utilizada para o objeto de valor para o mapa employees:

```
public interface Employee
{
    // Sequential sequence number used for optimistic versioning.
```

```
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Other get/set methods for other fields of Employee object.
}
```

Neste caso, o utilitário de carga sabe que pode utilizar o método getSequenceNumber para obter as informações de versão atuais para um objeto de valor Employee. O utilitário de carga incrementa o valor retornado para gerar um novo número de versão antes de atualizar o armazenamento persistente com o novo valor Employee. Para um utilitário de carga do JDBC (Java Database Connectivity), o número de sequência atual na cláusula where de uma instrução update SQL super qualificada é usado, e ele usa o novo número de sequência gerado para configurar a coluna de número de sequência ao novo valor de número de sequência.

Outra possibilidade é que o utilitário de carga faça uso de alguma função fornecida por backend que atualiza automaticamente uma coluna oculta que pode ser utilizada para versões otimistas. Em alguns casos, um procedimento armazenado ou acionador possivelmente pode ser utilizado para ajudar a manter uma coluna que contém informações de controle de versões. Se o utilitário de carga estiver utilizando uma destas técnicas para a manutenção de informações de versões otimistas, então, o aplicativo não precisa fornecer uma implementação do OptimisticCallback. A implementação padrão OptimisticCallback pode ser utilizada porque o utilitário de carga consegue identificar versões otimistas sem nenhuma assistência de um objeto OptimisticCallback.

Implementação Padrão para Entidades

As entidades são armazenadas no ObjectGrid utilizando objetos de tupla. A implementação padrão do OptimisticCallback se comporta da mesma maneira que se comporta com mapas de não-entidade. Entretanto, o campo de versão na entidade é identificado utilizando a anotação @Version ou o atributo version no arquivo XML descritor da entidade.

O atributo version pode ser de um dos seguintes tipos: int, Integer, short, Short, long, Long ou java.sql.Timestamp. Uma entidade deve ter somente um atributo de versão definido. O atributo de versão deve ser configurado somente durante a construção. Depois de a entidade ser persistida, o valor do atributo de versão não deve ser modificado.

Se um atributo de versão não estiver configurado e a estratégia de bloqueio optimistic for usada, então toda a tupla assume implicitamente a versão usando o estado completo da tupla.

No exemplo a seguir, a entidade Employee possui um atributo de versão longa denominado SequenceNumber:

```
@Entity
public class Employee {
private long sequence;
    // Sequential sequence number used for optimistic versioning.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {
```

```
        this.sequence = newSequenceNumber;
    }
    // Other get/set methods for other fields of Employee object.
}
```

Gravando uma Implementação OptimisticCallback

Uma implementação OptimisticCallback precisa implementar a interface OptimisticCallback e seguir as convenções comuns do plug-in ObjectGrid.

A lista a seguir fornece uma descrição ou consideração para cada um dos métodos na interface OptimisticCallback:

NULL_OPTIMISTIC_VERSION

Este valor especial será retornado pelo método getVersionedObjectForValue se a implementação OptimisticCallback padrão for utilizada em vez de uma implementação OptimisticCallback fornecida pelo aplicativo.

Método getVersionedObjectForValue

O método getVersionedObjectForValue pode retornar uma cópia do valor ou pode retornar um atributo do valor que pode ser utilizado para fins de controle de versões. Este método é chamado sempre que um objeto é associado a uma transação. Quando nenhum utilitário de carga é configurado em um mapa de apoio, o mapa de apoio usa este valor no momento da consolidação para executar uma comparação de versão optimistic. A comparação de versão otimista é utilizada pelo mapa de suporte para assegurar que a versão não tenha sido alterada desde que esta transação acessou pela primeira vez a entrada do mapa que foi modificada por esta transação. Se outra transação já tiver modificado a versão desta entrada do mapa, a comparação de versão falhará e o mapa de suporte exibirá uma exceção OptimisticCollisionException para forçar o rollback da transação. Se um Utilitário de Carga estiver conectado, o mapa de suporte não utilizará as informações de controle de versões otimista. Em vez disso, o Utilitário de Carga é responsável por desempenhar a comparação de controle de versões otimista e por atualizar as informações de controle de versões quando necessário. O Utilitário de Carga geralmente obtém o objeto de controle de versões inicial do LogElement transmitido para o método batchUpdate no Utilitário de Carga, que é chamado quando ocorre uma operação de limpeza ou uma transação é confirmada.

O código a seguir mostra a implementação utilizada pelo objeto EmployeeOptimisticCallbackImpl:

```
public Object getVersionedObjectForValue(Object value)
{
    if (value == null)
    {
        return null;
    }
    else
    {
        Employee emp = (Employee) value;
        return new Long( emp.getSequenceNumber() );
    }
}
```

Conforme demonstrado no exemplo anterior, o atributo sequenceNumber é retornado em um objeto java.lang.Long conforme esperado pelo Utilitário de Carga, que significa que a mesma pessoa que gravou o Utilitário de Carga gravou

a implementação de EmployeeOptimisticCallbackImpl ou trabalhou junto com a pessoa que implementou EmployeeOptimisticCallbackImpl. Por exemplo, essas pessoas concordam com o valor que é retornado pelo método getVersionedObjectForValue. Conforme descrito anteriormente, a implementação OptimisticCallback padrão retorna o valor especial NULL_OPTIMISTIC_VERSION como o objeto de versão.

Método updateVersionedObjectForValue

O método updateVersionedObjectForValue method é chamado sempre que uma transação tiver atualizado um valor e um novo objeto de versão for necessário. Se o método getVersionedObjectForValue retornar um atributo do valor, este método geralmente atualizará o valor de atributo com um novo objeto de versão. Se o método getVersionedObjectForValue retornar uma cópia do valor, este método normalmente não será atualizado. O método OptimisticCallback padrão não atualiza pois a implementação padrão de getVersionedObjectForValue sempre retorna o valor especial NULL_OPTIMISTIC_VERSION como o objeto de versão. O seguinte exemplo mostra a implementação usada pelo objeto EmployeeOptimisticCallbackImpl que é usado na seção OptimisticCallback:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Conforme demonstrado no exemplo anterior, o atributo sequenceNumber é incrementado em um para que, na próxima vez o método getVersionedObjectForValue for chamado, o valor java.lang.Long retornado tenha um valor longo que seja o valor do número de sequência original. Mais um, por exemplo, é o próximo valor de versão para esta instância de funcionário. Novamente, este exemplo significa que a mesma pessoa que gravou o Utilitário de Carga gravou a implementação EmployeeOptimisticCallbackImpl ou trabalhou junto com a pessoa que implementou o EmployeeOptimisticCallbackImpl.

Método serializeVersionedValue

Este método grava o valor com versão no fluxo especificado. Dependendo da implementação, o valor com versão pode ser utilizado para identificar colisões de atualização otimistas. Em algumas implementações, o valor com versão é uma cópia do valor original. Outras implementações podem ter um número de seqüência ou algum outro objeto para indicar a versão do valor. Como a implementação real é desconhecida, este método é fornecido para executar a serialização apropriada. A implementação padrão faz uma chamada writeObject.

Método inflateVersionedValue

Este método utiliza a versão serializada do valor com versão e retorna o objeto de valor com versão real. Dependendo da implementação, o valor com versão pode ser utilizado para identificar colisões de atualização otimistas. Em algumas implementações, o valor com versão é uma cópia do valor original. Outras implementações podem ter um número de seqüência ou algum outro objeto para indicar a versão do valor. Como a implementação real é desconhecida, este método

é fornecido para desempenhar a desserialização apropriada. A implementação padrão chama o método readObject.

Utilizando uma Implementação OptimisticCallback Fornecida pelo Aplicativo

Há duas abordagens para incluir uma implementação OptimisticCallback fornecido pelo aplicativo na configuração de BackingMap: configuração programática e configuração XML.

Conectar Programaticamente em uma Implementação OptimisticCallback

O exemplo a seguir demonstra como um aplicativo pode conectar programaticamente um objeto OptimisticCallback para o mapa de apoio de funcionários na instância grid1 do 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( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

Abordagem de configuração XML para conectar uma implementação OptimisticCallback

O objeto EmployeeOptimisticCallbackImpl no exemplo anterior deve implementar a interface OptimisticCallback. O aplicativo também pode utilizar um arquivo XML para conectar seu objeto OptimisticCallback conforme mostrado no seguinte exemplo:

```
<?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>
```

Visão Geral do Processamento de Transações

O WebSphere eXtreme Scale usa transações de acordo com seu mecanismo de interação com os dados.

Para interagir com os dados, o encadeamento em seu aplicativo precisa de sua própria sessão. Quando o aplicativo desejar usar o ObjectGrid em um encadeamento, chame um dos métodos ObjectGrid.getSession para obter um encadeamento. Com a sessão, o aplicativo pode trabalhar com dados que são armazenados nos mapas ObjectGrid.

Quando um aplicativo usa um objeto de Sessão, a sessão deve estar no contexto de uma transação. Uma transação inicia e é consolidada ou inicia e é recuperada usando os métodos begin, commit e rollback no objeto de Sessão. Os aplicativos também podem trabalhar em modo de auto-consolidação, no qual a Sessão inicia e consolida automaticamente uma transação sempre que uma operação é executada

no mapa. O modo de auto-confirmação não pode agrupar várias operações em uma única transação, assim, ele é a opção mais lenta se você estiver criando um lote de várias operações em uma única transação. Porém, para transações que contêm uma operação, a auto-consolidação é a opção mais rápida.

Introdução aos Slots de Plug-in

Um slot de plug-in é um espaço de armazenamento transacional reservado para os plug-ins que compartilham o contexto transacional. Estes slots fornecem uma forma para os plug-ins do eXtreme Scale se comunicarem um com o outro, compartilhar contexto transacional e assegurar que os recursos transacionais sejam usados correta e consistentemente dentro de uma transação.

Um plug-in pode armazenar o contexto transacional, como a conexão com o banco de dados, a conexão com o JMS (Java Message Service), e assim por diante, em um slot de plug-ins. O contexto transacional armazenado pode ser recuperado por qualquer plug-in que conheça o número do slot do plug-in, o qual serve como chave para recuperar o contexto transacional.

Utilizando Slots de Plug-in

Os slots de plug-in são parte da Interface TxID. Consulte a Documentação da API para obter mais informações sobre a interface.Os slots são entradas em uma matriz ArrayList. Os plug-ins podem reservar uma entrada na matriz ArrayList ao chamar o método ObjectGrid.reserveSlot e indicar que requer um slot em todos os objetos TxID. Após reservar os slots, os plug-ins podem inserir um contexto transacional nos slots de cada objeto TxID e recuperá-lo posteriormente. As operações put e get são coordenadas pelos números de slot que são retornados pelo método ObjectGrid.reserveSlot.

Um plug-in normalmente tem um ciclo de vida. A utilização dos slots de plug-in deve se adequar ao ciclo de vida do plug-in. Normalmente, o plug-in deve reservar slots de plug-in durante o estágio de inicialização e obter os números de cada slot. Durante o tempo de execução normal, o plug-in insere contexto transacional no slot reservado no objeto TxID no ponto apropriado. Normalmente, esse ponto apropriado é o início da transação. O plug-in ou outros plug-ins podem, desse modo, obter o contexto provisório armazenado pelo número de slot do TxID dentro da transação.

O plug-in tipicamente executa uma limpeza por meio da remoção do contexto transacional e dos slots. O fragmento de código a seguir ilustra como utilizar os slots de plug-in em um 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 {
        // In initialization stage, reserve desired plug-in slots by calling the
        //reserveSlot method of ObjectGrid and
        // passing in the designated slot name, TxID.SLOT_NAME.
        // Note: you have to pass in this TxID.SLOT_NAME that is designated
        // for application.
        try {
            // cache the returned slot numbers
            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 {
        // put transactional contexts into the reserved slots at the
        // beginning of the transaction.
```

```
    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 {
    // get the stored transactional contexts and use them
    // then, clean up all transactional resources.
    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) {
    }

}

/**
 * Close connection and swallow any Throwable that occurs.
 *
 * @param connection
 */
private void closeConnection(Connection connection) {
    try {
        connection.close();
    } catch (Throwable e1) {
    }
}

public void rollback(TxID id) throws TransactionCallbackException
    // get the stored transactional contexts and use them
    // then, clean up all transactional resources.
    try {
        Connection conn = (Connection) id.getSlot(connectionSlot);
        conn.rollback();
        cleanUpSlots(id);
    } catch (SQLException e) {
    }
}

public boolean isExternalTransactionActive(Session session) {
    return false;
}

// Getter methods for the slot numbers, other plug-in can obtain the slot numbers
// from these getter methods.

public int getConnectionSlot() {
    return connectionSlot;
}
public int getAutoCommitConnectionSlot() {
    return autoCommitConnectionSlot;
}
public int getPreparedStatementSlot() {
    return psCacheSlot;
}
```

O fragmento de código a seguir ilustra como um Utilitário de Carga pode obter o contexto transacional armazenado inserido pelo exemplo de plug-in TransactionCallback anterior:

```
public class DatabaseLoader implements Loader
{
    DatabaseTransactionCallback tcb;
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException {
        // The preload method is the initialization method of the Loader.
        // Obtain interested plug-in from Session or ObjectGrid instance.
        tcb = (DatabaseTransactionCallback)session.getObjectGrid().getTransactionCallback();
    }
    public List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException
    {
        // get the stored transactional contexts that is put by tcb's begin method.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implement get here
        return null;
    }
    public void batchUpdate(TxID txid, LogSequence sequence) throws LoaderException, OptimisticCollisionException
    {
        // get the stored transactional contexts that is put by tcb's begin method.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implement batch update here ...
    }
}
```

Gerenciadores de Transações Externas

Normalmente, as transações do eXtreme Scale começam com o método Session.begin e terminam com o método Session.commit. Entretanto, quando um ObjectGrid é integrado, um coordenador de transação externa pode iniciar e terminar transações. Nesse caso, você não precisa chamar os métodos begin ou commit.

Coordenação de Transação Externa

O plug-in TransactionCallback é estendido com o método isExternalTransactionActive(Session session) que associa a sessão do eXtreme Scale com uma transação externa. O cabeçalho do método é o seguinte:

```
public synchronized boolean isExternalTransactionActive(Session session)
```

Por exemplo, o eXtreme Scale pode ser configurado para se integrar com o WebSphere Application Server e WebSphere Extended Deployment.

Além disso, o eXtreme Scale oferece um plug-in integrado chamado WebSphere "Plug-ins para o Gerenciamento de Eventos de Ciclo de Vida da Transação" na página 382, que descreve como construir o plug-in para ambientes do WebSphere Application Server, e também adaptar o plug-in para outras estruturas.

A chave para esta integração total é a utilização da API ExtendedJTATransaction no WebSphere Application Server Versão 5.x e Versão 6.x. Porém, se você estiver usando o WebSphere Application Server Versão 6.0.2, aplique a APAR PK07848 para suportar este método. Use o código de amostra a seguir para associar uma sessão do ObjectGrid com um ID da transação do WebSphere Application Server:

```
/**
* This method is required to associate an objectGrid session with a WebSphere
* Application Server transaction ID.
*/
Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
    // lembre-se de que este localid significa que a sessão é salva posteriormente.
    localIdToSession.put(new Integer(jta.getLocalId()), session);
    return true;
}
```

Recuperar uma Transação Externa

Algumas vezes você pode precisar recuperar um objeto de serviço de transação externa para o plug-in TransactionCallback usar. No servidor do WebSphere Application Server, busque o objeto ExtendedJTATransaction de seu espaço de nomes como mostrado no exemplo a seguir:

```
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 outros produtos, é possível utilizar uma abordagem semelhante para recuperar o objeto de serviço de transações.

Controlar Confirmação por Retorno de Chamada Externo

O plug-in TransactionCallback precisa receber um sinal externo para confirmar ou recuperar a sessão do eXtreme Scale. Para receber este sinal externo, utilize o retorno de chamada do serviço de transações externas. Implemente a interface de retorno de chamada externa e registre-a no serviço de transações externas. Por exemplo, com WebSphere Application Server implemente a interface SynchronizationCallback como mostrado no exemplo a seguir:

```
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);
      // find the Session for the localId
      Session session = (Session)localIdToSession.get(lid);
      if(session != null) {
          try {
            // if WebSphere Application Server is committed when
            // hardening the transaction to backingMap.
            // We already did a flush in beforeCompletion
            if(didCommit) {
               session.commit();
            } else {
               // otherwise rollback
               session.rollback();
            }
         } catch(NoActiveTransactionException e) {
            // impossible in theory
         } catch(TransactionException e) {
```

```
            // given that we already did a flush, this should not fail
         } finally {
            // always clear the session from the mapping map.
            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  does not formally define
            // a way to signal the
            // transaction has failed so do this
            throw new RuntimeException("Cache flush failed", e);
         }
      }
   }
}
```

Use as APIs do eXtreme Scale com o plug-in TransactionCallback

O plug-in TransactionCallback desativa a auto-consolidação no eXtreme Scale. O padrão de uso normal para um eXtreme Scale é o seguinte:

```
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
ogSession.begin();
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
ogSession.commit();
```

Quando este plug-in TransactionCallback está em uso, o eXtreme Scale assume que o aplicativo usa o eXtreme Scale quando uma transação gerenciada pelo contêiner está presente. O trecho de código anterior muda o seguinte código neste ambiente:

```
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();
}
```

O método myMethod é semelhante a um cenário de aplicativo da Web. O aplicativo usa a interface UserTransaction normal para iniciar, consolidar e recuperar transações. O eXtreme Scale automaticam ente inicia e consolida a transação do contêiner. Se o método for um método EJB (Enterprise JavaBeans) que usa o atributo TX_REQUIRES, então remova a referência UserTransaction e as chamadas para iniciar e consolidar transações e o método funciona da mesma forma. Neste caso, o contêiner é responsável por iniciar e encerrar a transação.

Plug-in WebSphereTransactionCallback

Ao usar o plug-in WebSphereTransactionCallback, os aplicativos corporativos que estão em execução em um ambiente do WebSphere Application Server podem gerenciar as transações do ObjectGrid.

Quando você está usando uma sessão do ObjectGrid dentro de um método que está configurado para usar transações gerenciadas por contêiner, o contêiner corporativo automaticamente inicia, consolida ou recupera a transação do ObjectGrid. Ao usar os objetos UserTransaction do Java Transaction API (JTA), a transação ObjectGrid será gerenciada automaticamente pelo objeto UserTransaction.

Para uma discussão detalhada da implementação deste plug-in, consulte "Gerenciadores de Transações Externas" na página 389.

Nota: O ObjectGrid não suporta transações XA, 2-phase. Este plug-in não relaciona a transação do ObjectGrid com o gerenciador de transações. Assim, se o ObjectGrid falhar ao consolidar, todos os outros recursos que são gerenciados pela transação XA não são recuperados.

Conectar Programaticamente no Objeto WebSphereTransactionCallback

É possível ativar o WebSphereTransactionCallback na configuração ObjectGrid com a configuração programática ou XML. O fragmento de código a seguir usa o aplicativo para criar o objeto WebSphereTransactionCallback e o inclui em um ObjectGrid:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
WebSphereTransactionCallback wsTxCallback= new WebSphereTransactionCallback ();
myGrid.setTransactionCallback(wsTxCallback);
```

Abordagem de configuração XML para conectar o objeto WebSphereTransactionCallback

A seguinte configuração XML cria o objeto WebSphereTransactionCallback e o inclui em um ObjectGrid. O texto a seguir deve estar no arquivo `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>
```

Programando para Usar a Estrutura do OSGi

Os servidores e clientes do eXtreme Scale podem ser iniciados em um contêiner OSGi para poder incluir e atualizar dinamicamente plug-ins do eXtreme Scale no ambiente de tempo de execução.

Conceitos relacionados

"Visão Geral da Programação do Serialziador" na página 310
É possível usar os plug-ins do DataSerializer para gravar serializadores otimizados para armazenar objetos Java e outros dados no formato binário na grade. O plug-in também fornece métodos que você pode usar para atributos de consulta dentro dos dados binários sem exigir que o objeto de dados inteiro seja expandido.

Visão Geral da Serialização
Os dados são sempre expressos, porém não necessariamente armazenados, como objetos Java na grade de dados. O WebSphere eXtreme Scale usa diversos processos Java para serializar os dados, ao converter as instâncias de objetos Java em bytes e retornar para os objetos novamente, conforme necessário, para mover os dados entre os processos do cliente e do servidor.

Informações relacionadas

Documentação da API do DataSerializer

Construindo Plug-ins Dinâmicos do eXtreme Scale

O WebSphere eXtreme Scale inclui plug-ins ObjectGrid e BackingMap. Estes plug-ins são implementados em Java e são configurados usando o arquivo XML do descritor do ObjectGrid. Para criar um plug-in dinâmico que pode ser dinamicamente atualizado, eles precisam estar cientes dos eventos de ciclo de vida de ObjectGrid e BackingMap porque eles podem precisar concluir algumas ações durante uma atualização. Aprimorar um pacote configurável de plug-in com métodos de retorno de chamada, listeners de eventos, ou ambos, do ciclo de vida permite que o plug-in conclua essas ações em momentos apropriados.

Antes de Iniciar

Este tópico supõe que você construiu o plug-in apropriado. Para obter informações adicionais sobre como desenvolver plug-ins do eXtreme Scale, consulte o tópico APIs e Plug-ins do Sistema.

Sobre Esta Tarefa

Todos os plug-ins do eXtreme Scale se aplicam a uma instância de BackingMap ou de ObjectGrid. Muitos plug-ins também interagem com outros plug-ins. Por exemplo, um plug-in Loader e TransactionCallback trabalham juntos para interagir corretamente com uma transação do banco de dados e as várias chamadas de banco de dados JDBC. Alguns plug-ins também pode precisar armazenar em cache dados de configuração a partir de outros plug-ins para melhorar o desempenho.

Os plug-ins BackingMapLifecycleListener e ObjectGridLifecycleListener fornecem operações de ciclo de vida para as respectivas instâncias de BackingMap e ObjectGrid. Este processo permite que plug-ins sejam notificados quando o BackingMap ou ObjectGrid pai e seus respectivos plug-ins podem ser alterados. Os plug-ins BackingMap implementam a interface de BackingMapLifecyleListener e os plug-ins ObjectGrid implementam a interface de ObjectGridLifecycleListener. Estes plug-ins são chamados automaticamente quando o ciclo de vida do BackingMap ou ObjectGrid pai é alterado. Para obter mais informações sobre os plug-ins de ciclo de vida, consulte o tópico "Gerenciando Ciclos de Vida de Plug-in" na página 299.

É possível aprimorar os pacotes configuráveis usando os métodos ou os listeners de evento do ciclo de vida nas seguintes tarefas comuns:

- Iniciar e parar recursos, como encadeamentos ou assinantes de sistema de mensagens.
- Especificar que uma notificação ocorra quando os plug-ins equivalentes forem atualizados, permitindo o acesso direto ao plug-in e a detecção de quaisquer mudanças.

Sempre que outro plug-in for acessado diretamente, acesse esse plug-in por meio do contêiner OSGi para assegurar que todas as partes do sistema referenciem o plug-in correto. Se, por exemplo, algum componente no aplicativo referenciar diretamente, ou armazenar em cache, uma instância de um plug-in, ele manterá sua referência para essa versão do plug-in, mesmo depois que o plug-in tiver sido atualizado dinamicamente. Esse comportamento pode causar problemas relacionados ao aplicativo, bem como fugas de memória. Portanto, grave o código que depende dos plug-ins dinâmicos que obtêm sua referência usando semânticas getService() do OSGi. Se o aplicativo precisar armazenar em cache um ou mais plug-ins, ele atenderá eventos de ciclo de vida usando interfaces ObjectGridLifecycleListener e BackingMapLifecycleListener. O aplicativo também deve poder atualizar seu cache quando necessário, de modo thread safe.

Todos os plug-ins do eXtreme Scale usados com o OSGi também devem implementar as respectivas interfaces BackingMapPlugin ou ObjectGridPlugin. Novos plug-ins, tal como a interface MapSerializerPlugin impingem essa prática. Essas interfaces fornecem ao ambiente de tempo de execução do eXtreme Scale e ao OSGi uma interface consistente para injeção de estado no plug-in e controle de seu ciclo de vida.

Ao usar esta tarefa para especificar que uma notificação ocorre quando os plug-ins equivalentes são atualizados, é possível criar um factory de listener que produz uma instância do listener.

Procedimento

- Atualize a classe de plug-in ObjectGrid para implementar a interface ObjectGridPlugin. Esta interface inclui métodos que permitem que o eXtreme Scale inicialize, configure a instância do ObjectGrid e destrua o plug-in. Consulte o exemplo de código a seguir:

```
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() {
        // Handle any plug-in initialization here.  This is called by
        // eXtreme Scale, and not the OSGi bean manager.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destroy the plug-in and release any resources.  This
        // can be callsed by the OSGi Bean Manager or by eXtreme Scale.
        state = State.DESTROYED;
```

```
    }

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

- Atualize a classe de plug-in do ObjectGrid para implementar a interface ObjectGridLifecycleListener. Consulte o exemplo de código a seguir:

```
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:
                  // Lookup a Loader or MapSerializerPlugin using
                  // OSGi or directly from the ObjectGrid instance.
                  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;
              }
          }
          ...
      }
```

- Atualize um plug-in do BackingMap. Atualize a classe de plug-in do BackingMap para implementar a interface de plug-in do BackingMap. Esta interface inclui métodos que permitem que o eXtreme Scale inicialize, configure a instância do BackingMap e destrua o plug-in. Consulte o exemplo de código a seguir:

```
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() {
        // Handle any plug-in initialization here.  This is called by
        // eXtreme Scale, and not the OSGi bean manager.
        state = State.INITIALIZED;
    }
    boolean isInitialized() {
        return state == State.INITIALIZED;
    }

    public void destroy() {
        // Destroy the plug-in and release any resources.  This
```

```
        // can be callsed by the OSGi Bean Manager or by eXtreme Scale.
        state = State.DESTROYED;
    }

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

- Atualize a classe de plug-in do BackingMap para implementar a interface BackingMapLifecycleListener. Consulte o exemplo de código a seguir:

```
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:
                // Lookup a MapSerializerPlugin using
                // OSGi or directly from the ObjectGrid instance.
                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

Implementando a interface ObjectGridPlugin ou BackingMapPlugin, o eXtreme Scale pode controlar o ciclo de vida de seu plug-in nos momentos certos.

Implementando a interface ObjectGridLifecycleListener ou BackingMapLifecycleListener, o plug-in é automaticamente registrado como um listener dos eventos de ciclo de vida do ObjectGrid ou do BackingMap associados. O evento INITIALIZING é usado para sinalizar que todos os plug-ins do ObjectGrid e do BackingMap foram inicializados e estão disponíveis para consultar e usar. O evento ONLINE é usado para sinalizar que o ObjectGrid está on-line e pronto para iniciar eventos de processamento.

Programação para Integração de JPA

O Java Persistence API (JPA) é uma especificação que permite o mapeamento de objetos Java para bancos de dados relacionais. O JPA contém uma especificação completa de object-relational mapping (ORM) usando anotações de metadados da linguagem Java, descritores XML, ou ambos para definir o mapeamento entre objetos Java e um banco de dados relacional. Inúmeras implementações comerciais e de software livre estão disponíveis.

Para usar o JPA, é necessário ter um provedor JPA suportado, como OpenJPA ou Hibernate, arquivos JAR e um arquivo`META-INF/persistence.xml` no seu caminho da classe.

Tarefas relacionadas

"Resolução de Problemas de Carregadores" na página 517
Use estas informações para resolver problemas com os carregadores de banco de dados.

Configurando Utilitários de Carga do JPA
Um Utilitário de Carga do Java Persistence API (JPA) é uma implementação de plug-in que utiliza JPA para interagir com o banco de dados.

Carregadores JPA

O Java Persistence API (JPA) é uma especificação que permite o mapeamento de objetos Java para bancos de dados relacionais. O JPA contém uma especificação completa de object-relational mapping (ORM) usando anotações de metadados da linguagem Java, descritores XML, ou ambos para definir o mapeamento entre objetos Java e um banco de dados relacional. Inúmeras implementações comerciais e de software livre estão disponíveis.

É possível utilizar uma implementação de plug-in de utilitário de carga do Java Persistence API (JPA) com eXtreme Scale para interagir com qualquer banco de dados suportado por seu utilitário de carga escolhido. Para usar o JPA, é necessário ter um provedor JPA suportado, como OpenJPA ou Hibernate, arquivos JAR e um arquivo`META-INF/persistence.xml` no seu caminho da classe.

Os plug-ins JPALoader com.ibm.websphere.objectgrid.jpa.JPALoader e JPAEntityLoader com.ibm.websphere.objectgrid.jpa.JPAEntityLoader são dois plug-ins do utilitário de carga do JPA integrados que são usados para sincronizar os mapas do ObjectGrid com um banco de dados. É necessário ter uma implementação do JPA, como Hibernate ou OpenJPA, para usar este recurso. O banco de dados pode ser qualquer back end que seja suportado pelo provedor JPA escolhido.

É possível usar o plug-in do JPALoader ao armazenar dados usando a API ObjectMap. Use o plug-in do JPAEntityLoader ao armazenar dados usando a API EntityManager.

Arquitetura do Utilitário de Carga do JPA

O Utilitário de Carga do JPA é usado para mapas do eXtreme Scale que armazenam objetos Java antigos simples (POJO).

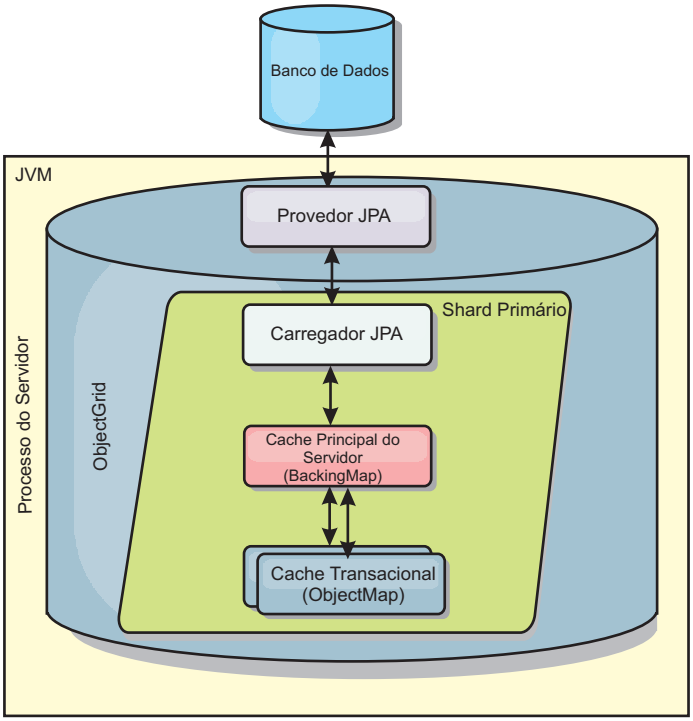


Figura 28. Arquitetura do Utilitário de Carga do JPA

Quando um método ObjectMap.get(Object key) é chamado, o eXtreme Scale executa as primeiras verificações se a entrada está contida na camada do ObjectMap. Se não, o tempo de execução delega a solicitação ao Utilitário de Carga do JPA. Sob solicitação de carregamento da chave, o JPALoader chama o método EntityManager.find(Object key) do JPA para localizar os dados de uma camada do JPA. Se os dados estiverem contidos no gerenciador de entidades JPA, ele serão retornados; caso contrário, o provedor JPA interage com o banco de dados para obter o valor.

Quando uma atualização para o ObjectMap ocorre, por exemplo, usando o método ObjectMap.update(Objectkey, Object value), o tempo de execução do eXtreme Scale cria um LogElement para esta atualização e a envia para o JPALoader. O JPALoader chama o método EntityManager.merge(Object value) do JPA para atualizar o valor no banco de dados.

Para o JPAEntityLoader, as mesmas quatro camadas estão envolvidas. Porém, como o plug-in JPAEntityLoader é usado para mapas que armazenam entidades do eXtreme Scale, as relações entre as entidades poderiam complicar o cenário de uso. Uma entidade do eXtreme Scale é diferenciada de uma entidade do JPA. Para obter mais detalhes, consulte o “Plug-in JPAEntityLoader” na página 369.

Métodos

Utilitários de Carga Fornecem Três Métodos Principais:

1. get: Retorna uma lista de valores que corresponde à lista de chaves que são passadas por meio da recuperação de dados usando o JPA. O método usa o JPA para localizar as entidades no banco de dados. Para o plug-inJPALoader, a lista retornada contém uma lista de entidades JPA diretamente a partir da operação find. Para o plug-in JPAEntityLoader, a lista retornada contém tuplas do valor da entidade de eXtreme Scale convertidas de entidades do JPA.

2. batchUpdate: grava os dados dos mapas do ObjectGrid para o banco de dados. Dependendo dos diferentes tipos de operação (inserir, atualizar ou excluir), o utilitário de carga usa as operações de persistir, mesclar ou remover para atualizar os dados para o banco de dados. Para o JPALoader, os objetos no mapa são utilizados diretamente como entidades JPA. Para o JPAEntityLoader, as tuplas de entidade no mapa são convertidas nos objetos que são utilizados como entidades JPA.
3. preloadMap: Pré-carrega o mapa usando o método do utilitário de carga do ClientLoader.load. Para mapas particionados, o método preloadMap é chamado apenas em uma partição. A partição é especificada na propriedade preloadPartition da classe JPALoader ou JPAEntityLoader. Se o valor de preloadPartition for configurado para menor que zero ou maior que (*total_number_of_partitions* - 1), o pré-carregamento será desativado.

Ambos os plug-ins JPALoader e JPAEntityLoader funcionam com a classe JPATxCallback para coordenar as transações do eXtreme Scale e as transações do JPA. O JPATxCallback precisa ser configurado na instância do ObjectGrid para utilizar estes dois utilitários de carga.

Configuração e Programação

Se você estiver usando os carregadores JPA em um ambiente multimestre, consulte o "Considerações Sobre o Carregador em uma Topologia Multimestre" na página 106. Para obter mais informações sobre como configurar os carregadores do JPA, consulte informações sobre os carregadores do JPA no *Guia de Administração*. Para obter informações adicionais sobre utilitário de carga do JPA de programação, consulte o *Guia de Programação*.

Desenvolvendo Carregadores JPA Baseados em Cliente

É possível implementar o pré-carregamento e o recarregamento de dados no seu aplicativo usando o utilitário Java Persistence API (JPA). Este recurso pode simplificar o carregamento dos mapas quando as consultas ao banco de dados não puderem ser particionadas.

Antes de Iniciar

- Um provedor de JPA deve ser usado com um banco de dados suportado.
- Antes de pré-carregar ou de recarregar os mapas, você deve configurar o estado de disponibilidade de um ObjectGrid para `PRELOAD`. É possível configurar o estado de disponibilidade com o método setObjectGridState da interface StateManager. A interface StateManager impede que outros clientes acessem o ObjectGrid quando ainda não estiver on-line. Depois de pré-carregar ou de recarregar o mapa, o estado pode ser configurado de volta para ONLINE.
- Quando estiver pré-carregando mapas diferentes em um ObjectGrid, configure o estado ObjectGrid para `PRELOAD` uma vez e configure o valor de volta para `ONLINE` depois que todos os mapas concluírem o carregamento dos dados. Esta coordenação pode ser feita pela interface ClientLoadCallback. Configure o estado ObjectGrid para `PRELOAD` após a primeira notificação preStart a partir da instância ObjectGrid e configure-a de volta para `ONLINE` após a última notificação postFinish.
- Se for necessário pré-carregar mapas de diferentes Java Virtual Machines, será necessário coordenar entre várias Java Virtual Machines. Configure o estado de ObjectGrid para `PRELOAD` uma vez antes de o primeiro mapa ser pré-carregado em qualquer uma das Java Virtual Machines e configure o valor de volta para `ONLINE` depois que todos os mapas concluírem o carregamento de dados em

todas as Java Virtual Machines. Para obter informações adicionais, consulte Gerenciando a Disponibilidade do ObjectGrid.

Sobre Esta Tarefa

Quando executar uma operação de pré-carregamento ou de recarregamento no mapa, as seguintes ações ocorrerão:

1. A ação inicial a ser executada depende de se uma operação de pré-carregamento ou de recarregamento está sendo executada.
   - **Operação de pré-carregamento:** O mapa a ser pré-carregado é limpo. Para um mapa de entidade, se qualquer relação for configurada como remoção em cascata, quaisquer mapas relacionados são limpos.
   - **Operação de recarregamento:** A consulta fornecida é executada no mapa e os resultados são invalidados. Para um mapa de entidade, se qualquer relação for configurada com a opção **`CascadeType.INVALIDATE`**, as entidades relacionadas também serão invalidadas a partir dos seus mapas.
2. Execute a consulta ao JPA para as entidades em um lote.
3. Para cada lote, uma lista de chaves e uma lista de valores para cada partição são construídas.
4. Para cada partição, o agente da grade de dados é chamado para inserir ou atualizar os dados no lado do servidor diretamente se ele for um cliente do eXtreme Scale. Se a grade de dados for uma instância local, os dados nos mapas serão inseridos ou atualizados diretamente.

Conceitos relacionados

"Visão Geral do Utilitário de Pré-Carregamento JPA Baseado em Cliente"
O utilitário de pré-carregamento Java Persistence API (JPA) baseado em cliente carrega dados nos mapas de apoio do eXtreme Scale usando uma conexão de cliente para o ObjectGrid.

Referências relacionadas

"Exemplo: Pré-carregando um Mapa com a Interface ClientLoader" na página 402
É possível pré-carregar um mapa para preencher os dados do mapa antes que os clientes comecem a acessar o mapa.

"Exemplo, Recarregando um Mapa com a Interface ClientLoader" na página 403
Recarregar um mapa é o mesmo que pré-carregar um mapa, exceto que o argumento **`isPreload`** é configurado para false no método ClientLoader.load.

"Exemplo: Chamando um Carregador do Cliente" na página 404
É possível usar o método de pré-carregamento na interface Loader para chamar um carregador do cliente.

Informações relacionadas

Interface ClientLoader

Interface StateManager

Visão Geral do Utilitário de Pré-Carregamento JPA Baseado em Cliente

O utilitário de pré-carregamento Java Persistence API (JPA) baseado em cliente carrega dados nos mapas de apoio do eXtreme Scale usando uma conexão de cliente para o ObjectGrid.

Este recurso pode simplificar o carregamento dos mapas quando as consultas ao banco de dados não puderem ser particionadas. Um carregador, como um Carregador JPA também pode ser usado e é ideal quando os dados podem ser carregados em paralelo.

O utilitário de pré-carregamento JPZ baseado em cliente pode usar as implementações de OpenJPA ou Hibernate para carregar o ObjectGrid de um banco de dados. Porque oWebSphere eXtreme Scale não interage diretamente com o banco de dados ou o Java Database Connectivity (JDBC), qualquer banco de dados que o OpenJPA ou o Hibernate suporta pode ser usado para carregar o ObjectGrid.

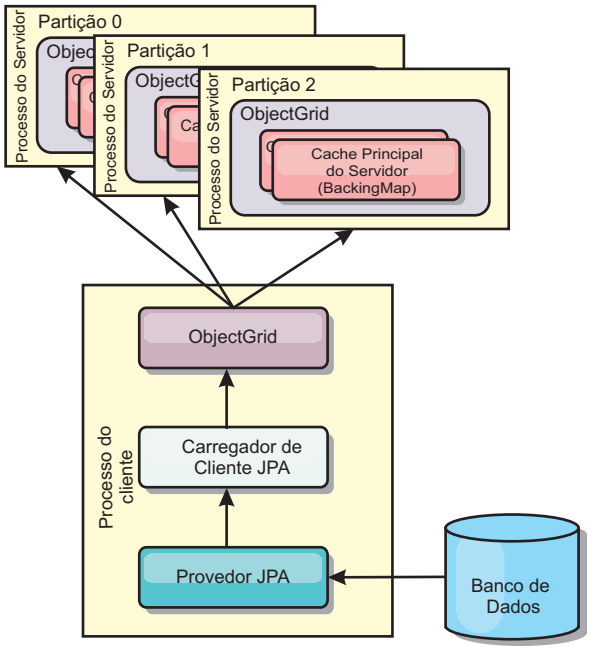


Figura 29. Utilitário de Carga do Cliente que usa Implementação JPA para Carregar o ObjectGrid

Normalmente, um aplicativo de usuário fornece um nome de unidade de persistência, um nome de classe de entidade e uma consulta JPA para o utilitário de carga do cliente. O utilitário de carga do cliente recupera o gerenciador de entidades JPA baseado no nome da unidade de persistência, utiliza o gerenciador de entidades para consultar dados do banco de dados com a classe de entidade fornecida e a consulta JPA e, finalmente, carrega os dados nos mapas distribuídos do ObjectGrid. Quando relações de múltiplos níveis estão envolvidas na consulta, é possível usar uma cadeia de consulta customizada para otimizar o desempenho. Opcionalmente, uma mapa de propriedade de persistência poderia ser fornecido para substituir as propriedades de persistência configuradas.

Um utilitário de carga do cliente pode carregar dados em dois modos diferentes, como exibidos na tabela a seguir:

Tabela 10. Modos do Utilitário de Carga do Cliente

Modo	**Descrição**
Pré-carregar	Limpa e carrega todas as entradas no mapa de apoio. Se o mapa for um mapa de entidade, quaisquer mapas de entidade relacionados também serão limpos se a opção CascadeType.REMOVE do ObjectGrid estiver ativada.

Tabela 10. Modos do Utilitário de Carga do Cliente (continuação)

Modo	Descrição
Recarregar	A consulta JPA é executada junto ao ObjectGrid para invalidar todas as entidades no mapa que correspondem à consulta. Se o mapa for um mapa de entidade, quaisquer mapas de entidade relacionados também serão limpos se a opção CascadeType.INVALIDATE do ObjectGrid estiver ativada.

Em qualquer um dos casos, uma consulta JPA é utilizada para selecionar e carregar as entidades desejadas a partir do banco de dados e armazená-las nos mapas do ObjectGrid. Se o mapa do ObjectGrid for um mapa de não-entidade, as entidades JPA serão separadas e armazenadas diretamente. Se o mapa do ObjectGrid for um mapa de entidade, as entidades JPA serão armazenadas como tuplas de entidade do ObjectGrid. É possível fornecer uma consulta JPA ou utilizar a consulta padrão `select o from EntityName o`.

Para obter mais informações sobre como configurar o utilitário de pré-carregamento do JPA baseado em cliente, consulte o "Desenvolvendo Carregadores JPA Baseados em Cliente" na página 399informações no *Guia de Programação*

Tarefas relacionadas

"Desenvolvendo Carregadores JPA Baseados em Cliente" na página 399
É possível implementar o pré-carregamento e o recarregamento de dados no seu aplicativo usando o utilitário Java Persistence API (JPA). Este recurso pode simplificar o carregamento dos mapas quando as consultas ao banco de dados não puderem ser particionadas.

Referências relacionadas

"Exemplo: Pré-carregando um Mapa com a Interface ClientLoader"
É possível pré-carregar um mapa para preencher os dados do mapa antes que os clientes comecem a acessar o mapa.

"Exemplo, Recarregando um Mapa com a Interface ClientLoader" na página 403
Recarregar um mapa é o mesmo que pré-carregar um mapa, exceto que o argumento **`isPreload`** é configurado para false no método ClientLoader.load.

"Exemplo: Chamando um Carregador do Cliente" na página 404
É possível usar o método de pré-carregamento na interface Loader para chamar um carregador do cliente.

Informações relacionadas

Interface ClientLoader

Interface StateManager

Exemplo: Pré-carregando um Mapa com a Interface ClientLoader

É possível pré-carregar um mapa para preencher os dados do mapa antes que os clientes comecem a acessar o mapa.

Exemplo de Pré-Carregamento Baseado no Cliente

O seguinte trecho de código de amostra mostra um carregamento de cliente simples. Neste exemplo, o mapa CUSTOMER é configurado como um mapa de entidade. A classe de entidade Customer, que é configurada no arquivo descritor XML de metadados da entidade ObjectGrid, possui uma relação de um para

muitos com as entidades Order. A entidade Customer possui a opção CascadeType.ALL ativada na relação com a entidade Order. Antes que o método ClientLoader.load seja chamado, o estado ObjectGrid é configurado para PRELOAD. O parâmetro **isPreload** no método de carregamento é configurado para true.

```
// Get the StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();

// Set ObjectGrid state to PRELOAD before calling ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Load the data
c.load(objectGrid, "CUSTOMER", "customerPU", null,
    null, null, null, true, null);

// Set ObjectGrid state back to ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Conceitos relacionados

"Visão Geral do Utilitário de Pré-Carregamento JPA Baseado em Cliente" na página 400
O utilitário de pré-carregamento Java Persistence API (JPA) baseado em cliente carrega dados nos mapas de apoio do eXtreme Scale usando uma conexão de cliente para o ObjectGrid.

Tarefas relacionadas

"Desenvolvendo Carregadores JPA Baseados em Cliente" na página 399
É possível implementar o pré-carregamento e o recarregamento de dados no seu aplicativo usando o utilitário Java Persistence API (JPA). Este recurso pode simplificar o carregamento dos mapas quando as consultas ao banco de dados não puderem ser particionadas.

Informações relacionadas

Interface ClientLoader

Interface StateManager

Exemplo, Recarregando um Mapa com a Interface ClientLoader

Recarregar um mapa é o mesmo que pré-carregar um mapa, exceto que o argumento **isPreload** é configurado para false no método ClientLoader.load.

Exemplo de Recarregamento Baseado no Cliente

A seguinte amostra ilustra como recarregar mapas. Comparado com a amostra de pré-carregamento, a diferença principal é que o loadSql e os parâmetros são fornecidos. Esta amostra recarrega apenas os dados do Cliente com um ID entre 1000 e 2000. O parâmetro **isPreload** no método de carregamento é configurado para false.

```
// Get the StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();

// Set ObjectGrid state to PRELOAD before calling ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Load the data
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);

// Set ObjectGrid state back to ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Lembre-se: Esta sequência de consulta observa as duas sintaxes de consulta JPA e a sintaxe de consulta da entidade eXtreme Scale . Essa sequência de consulta é importante porque ela é executada duas vezes: para invalidar as entidades ObjectGrid correspondidas e para carregar as entidades do JPA correspondidas.

Conceitos relacionados

"Visão Geral do Utilitário de Pré-Carregamento JPA Baseado em Cliente" na página 400
O utilitário de pré-carregamento Java Persistence API (JPA) baseado em cliente carrega dados nos mapas de apoio do eXtreme Scale usando uma conexão de cliente para o ObjectGrid.

Tarefas relacionadas

"Desenvolvendo Carregadores JPA Baseados em Cliente" na página 399
É possível implementar o pré-carregamento e o recarregamento de dados no seu aplicativo usando o utilitário Java Persistence API (JPA). Este recurso pode simplificar o carregamento dos mapas quando as consultas ao banco de dados não puderem ser particionadas.

Informações relacionadas

Interface ClientLoader

Interface StateManager

Exemplo: Chamando um Carregador do Cliente

É possível usar o método de pré-carregamento na interface Loader para chamar um carregador do cliente.

Use o método de pré-carregamento na interface Loader para chamar um carregador do cliente:

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

Este método sinaliza ao utilitário de carga para pré-carregar os dados no mapa. Uma implementação do utilitário de carga pode utilizar um utilitário de carga do cliente para pré-carregar os dados em todas as suas partições. Por exemplo, o utilitário de carga do JPA usa o utilitário de carga do cliente para pré-carregar os dados no mapa.

Para obter mais informações, consulte o tópico da visão geral dos carregadores do JPA em *Visão Geral do Produto*.

Exemplo: Chamando um Carregador de Cliente com o Método preloadMap

A seguir há um exemplo de como pré-carregar o mapa usando o utilitário de carga do cliente no método preloadMap. O exemplo primeiro verifica se o número da partição atual é o mesmo que o da partição pré-carregada. Se o número da partição não for igual ao da partição pré-carregada, nenhuma ação ocorrerá. Se os números

da partição corresponderem, o utilitário de carga do cliente será chamado para carregar os dados nos mapas. Você deve chamar o carregador do cliente em uma, e somente uma, partição.

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

....
 ObjectGrid objectGrid = session.getObjectGrid();
 int partitionId = backingMap.getPartitionId();
 int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();

 // Only call client loader data in one partition
 if (partitionId == preloadPartition) {
     ClientLoader c = ClientLoaderFactory.getClientLoader();
    // Call the client loader to load the data
    try {
        c.load(objectGrid, "CUSTOMER", "customerPU",
     null, entityClass, null, null, true, null);
    } catch (ObjectGridException e) {
        LoaderException le = new LoaderException("Exception caught in ObjectMap " + ogName + "." +
        le.initCause(e);
        throw le;
    }
 }
}
```

Lembre-se: Configure o atributo de backingMap "preloadMode" para `true`, para que o método de pré-carregamento seja executado de forma assíncrona. Caso contrário, o método de pré-carregamento impedirá que a instância de ObjectGrid seja ativada.

Conceitos relacionados

"Visão Geral do Utilitário de Pré-Carregamento JPA Baseado em Cliente" na página 400
O utilitário de pré-carregamento Java Persistence API (JPA) baseado em cliente carrega dados nos mapas de apoio do eXtreme Scale usando uma conexão de cliente para o ObjectGrid.

Tarefas relacionadas

"Desenvolvendo Carregadores JPA Baseados em Cliente" na página 399
É possível implementar o pré-carregamento e o recarregamento de dados no seu aplicativo usando o utilitário Java Persistence API (JPA). Este recurso pode simplificar o carregamento dos mapas quando as consultas ao banco de dados não puderem ser particionadas.

Informações relacionadas

Interface ClientLoader

Interface StateManager

Exemplo: Criando um Carregador JPA Baseado em Cliente Customizado

O método ClientLoader.load na interface Loader fornece uma função de carregamento do cliente que atende à maioria dos cenários. No entanto, se desejar carregar os dados sem o método ClientLoader.load, poderá implementar seu próprio utilitário de pré-carregamento.

Modelo de Carregador Customizado

Use o seguinte modelo para desenvolver seu carregador:

```
// Get the StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();
```

```
// Set ObjectGrid state to PRELOAD before calling ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);


// Load the data
...<your preload utility implementation>...

// Set ObjectGrid state back to ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Desenvolvendo um Carregador JPA Baseado em Cliente com um Agente DataGrid

Quando estiver carregando os dados no lado do cliente, usar um agente DataGrid poderá aumentar o desempenho. Ao usar o agente DataGrid, todas as leituras e gravações de dados ocorrerão no processo do servidor. Também é possível projetar seu aplicativo para certificar-se de que os agentes DataGrid em várias partições sejam executados em paralelo para aumentar ainda mais o desempenho.

Sobre Esta Tarefa

Para obter mais informações sobre o agente DataGrid, consulte "APIs do DataGrid e Particionamento" na página 260.

Depois de criar a implementação de pré-carregamento de dados, poderá criar um Utilitário de Carga genérico para concluir as seguintes tarefas:

- Consultar os dados do banco de dados nos lotes.
- Criar uma lista de chaves e uma lista de valores para cada partição.
- Para cada partição, chame o método agentMgr.callReduceAgent(agent, aKey) para executar o agente no servidor em um encadeamento. Ao executar um encadeamento, poderá executar os agentes simultaneamente em várias partições.

Exemplo

O fragmento de código a seguir é um exemplo de como carregar usando um agente do 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) {
            // In the POJO case, it is very straightforward,
            // we can just put everything in the
            // map using insert
            insert(m);
        } else {
            // 2. Entity case.
            // In the Entity case, we can persist the entities
            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);
```

```
    }

}

/**
 * Basically this is fresh load.
 * @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 Returns the 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;
    }
}
```

Exemplo: Usando o Plug-in Hibernate para Pré-Carregar Dados no Cache do ObjectGrid

É possível usar o método preload da classe ObjectGridHibernateCacheProvider para pré-carregar dados no cache do ObjectGrid para uma classe de entidade.

Exemplo: Usando a classe EntityManagerFactory

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("testPU");
ObjectGridHibernateCacheProvider.preload("objectGridName", emf, TargetEntity.class, 100, 100);
```

Importante: Por padrão, as entidades não fazem parte do segundo nível de cache. No classes de Entidade que precisam de armazenamento em cache, inclua a anotação @cache. Este é um exemplo:

```
import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;
@Entity
@Cache(usage=CacheConcurrencyStrategy.TRANSACTIONAL)
public class HibernateCacheTest { ... }
```

É possível substituir esse padrão ao configurar o elemento modo de cache compartilhado no arquivo `persistence.xml` ou usando a propriedade javax.persistence.sharedCache.mode.

Exemplo: Usando a classe SessionFactory

```
org.hibernate.cfg.Configuration cfg = new Configuration();
// use o método addResource, addClass e setProperty da Configuração para preparar a
// configuração necessária para criar o SessionFactor
SessionFactory sessionFactory= cfg.buildSessionFactory();
ObjectGridHibernateCacheProvider.preload("objectGridName", sessionFactory, TargetEntity.class, 100, 100);
```

Nota:

1. Em um sistema distribuído, este mecanismo de pré-carregamento somente pode ser chamado de uma Java virtual machine. O mecanismo de pré-carregamento não pode executar simultaneamente a partir de várias Java Virtual Machines.
2. Antes de executar o pré-carregamento, você deve inicializar o cache do eXtreme Scale ao criar o EntityManager usando o EntityManagerFactory para que todos os BackingMaps correspondentes sejam criados, caso contrário, o pré-carregamento forçará o cache a ser inicializado com somente um BackingMap padrão para suportar todas as entidades. Isto significa que um único BackingMap é compartilhado por todas as entidades.

Iniciando o Atualizador Baseado em Tempo do JPA

Ao iniciar um atualizador baseado em tempo do Java Persistence API (JPA), os mapas do ObjectGrid são atualizados com as últimas alterações no banco de dados.

Antes de Iniciar

Configurar o atualizador baseado em tempo. Consulte o Configurando um Atualizador de Dados Baseado em Tempo do JPAinformações sobre a configuração de um atualizador de dados baseado em tempo do JPA no *Guia de Administração*.

Sobre Esta Tarefa

Para obter mais informações sobre como o atualizador de dados baseado em tempo do Java Persistence API (JPA) trabalha, consulte "Atualizador de Dados Baseado em Tempo JPA" na página 413.

Procedimento

- Inicie um atualizador de banco de dados baseado em tempo.
  - **Automaticamente para o eXtreme Scale distribuído:**

    Se você criar uma configuração do timeBasedDBUpdate para o mapa de apoio, o atualizador do banco de dados baseado em tempo será iniciado automaticamente quando um shard primário do ObjectGrid distribuído for ativado. Para um ObjectGrid com várias partições, o atualizador do banco de dados baseado em tempo será iniciado apenas na partição 0.
  - **Automaticamente para o eXtreme Scale local:**

    Se você criar uma configuração do timeBasedDBUpdate para o mapa de apoio, o atualizador do banco de dados baseado em tempo será iniciado automaticamente quando o mapa local for ativado.
  - **Manualmente:**

    Também é possível iniciar ou parar manualmente um atualizador de banco de dados baseado em tempo utilizando a API do TimeBasedDBUpdater.

    ```
    public synchronized void startDBUpdate(ObjectGrid objectGrid, String mapName,
     String punitName, Class entityClass, String timestampField, DBUpdateMode mode) {
    ```

    1. **ObjectGrid**: a instância do ObjectGrid (local ou cliente).
    2. **mapName**: o nome do mapa de apoio para o qual o atualizador de banco de dados baseado em tempo é iniciado.
    3. **punitName**: O nome da unidade de persistência JPA para criar um factory do gerenciador de entidade JPA; o valor padrão é o nome da primeira unidade de pesistência definida no arquivo `persistence.xml`.
    4. **entityClass**: O nome da classe de entidade usado para interagir com o provedor Java Persistence API (JPA); o nome da classe de entidade é usado para obter as entidades do JPA usando as consultas de entidade.

5. **timestampField**: Um campo de registro de data e hora da classe de entidade para identificar a hora ou a sequência quando um registro back end de banco de dados foi atualizado ou inserido pela última vez.
6. **mode**: O modo de atualização de banco de dados baseado em tempo; um tipo INVALIDATE_ONLY faz com que ele invalide as entradas no mapa do ObjectGrid se os registros correspondentes no banco de dados foram atualizados; um tipo UPDATE_ONLY indica para atualizar as entradas existentes no mapa do ObjectGrid com os valores mais recentes do banco de dados; entretanto, todos os registros recentemente inseridos no banco de dados são ignorados; um tipo INSERT_UPDATE indica para atualizar as entradas existentes no mapa do ObjectGrid com os valores mais recentes a partir do banco de dados; além disso, todos os registros recentemente inseridos no banco de dados são inseridos no mapa do ObjectGrid.

Se você desejar parar o atualizador de banco de dados baseado em tempo, poderá chamar o seguinte método para parar o atualizador:

```
public synchronized void stopDBUpdate(ObjectGrid objectGrid, String mapName)
```

Os parâmetros ObjectGrid e mapName devem ser os mesmos que aqueles transmitidos no método startDBUpdate.

- Crie o campo do registro de data e hora no seu banco de dados.
  - **DB2**

    Como parte do recurso de bloqueio otimista, o DB2 9.5 fornece um recurso de registro de data e hora de alteração de linha. É possível definir uma coluna ROWCHGTS utilizando o formato ROW CHANGE TIMESTAMP, conforme a seguir:

    ```
    ROWCHGTS TIMESTAMP NOT NULL
         GENERATED ALWAYS
         FOR EACH ROW ON UPDATE AS
         ROW CHANGE TIMESTAMP
    ```

    Em seguida, é possível indicar o campo de entidade que corresponde à esta coluna como o campo do registro de data e hora pela anotação ou configuração. Este é um exemplo:

    ```
    @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**

  No Oracle, há uma semicoluna ora_rowscn para o número de alteração do sistema do registro. É possível utilizar esta coluna para o mesmo propósito. Um exemplo da entidade que usa o campo ora_rowscn como o campo do registro de data e hora de atualização de banco de dados baseado em tempo é o seguinte:

```
@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;
}
```

– **Outros Bancos de Dados**

  Para outros tipos de bancos de dados, é possível criar uma coluna da tabela para controlar as alterações. Os valores da coluna precisam ser gerenciados manualmente pelo aplicativo que atualiza a tabela.

  Tome o banco de dados Apache Derby como exemplo: É possível criar uma coluna "ROWCHGTS" para controlar os números de alteração. Além disso, um número de alteração mais recente é controlado para esta tabela. Sempre que um registro for inserido ou atualizado, o número de alteração mais recente para a tabela é aumentado e o valor da coluna ROWCHGTS para o registro é atualizado com o número aumentado.

```
@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;
}
```

Atualizador de Dados Baseado em Tempo JPA

Um atualizador de banco de dados baseado em tempo do Java Persistence API (JPA) atualiza os mapas do ObjectGrid com as últimas alterações no banco de dados.

Quando são feitas alterações diretamente em um banco de dados que está sendo confrontado pelo WebSphere eXtreme Scale, essas alterações não são refletidas simultaneamente na grade do eXtreme Scale. Para implementar corretamente o eXtreme Scale como um espaço de processamento de banco de dados de memória, lembre-se de que sua grade pode sair de sincronia com o banco de dados. O atualizar de banco de dados baseado em tempo usa o recurso System Change Number (SCN) no Oracle 10g e a indicação de data e hora da alteração da linha no DB2 9.5 para monitorar as alterações no banco de dados para invalidação e atualização. O atualizador também permite que os aplicativos tenham um campo definido pelo usuário para o mesmo propósito.

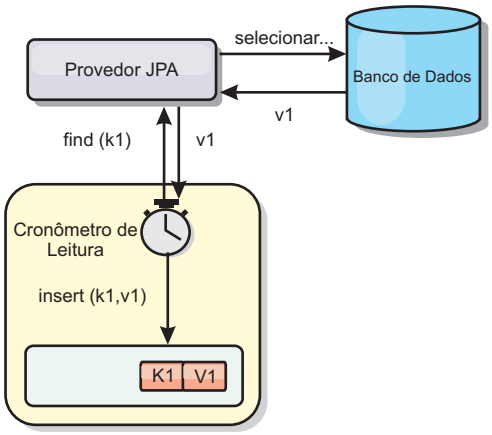


Figura 30. Atualização Periódica

O atualizador de banco de dados baseado em tempo consulta periodicamente o banco de dados usando as interfaces do JPA para obter as entidades do JPA que representam os registros recentemente inseridos e atualizados no banco de dados. Para atualizar periodicamente os registros, cada registro no banco de dados deve ter um registro de data e hora para identificar o tempo ou a sequência em que o registro foi atualizado ou inserido pela última vez. O registro de data e hora não precisa estar no formato do registro de data e hora. O valor do registro de data e hora pode estar em um formato inteiro ou longo, se ele gerar um valor exclusivo e cada vez maior.

Vários bancos de dados comerciais fornecem este recurso.

Por exemplo, no DB2 9.5, é possível definir uma coluna usando o formato ROW CHANGE TIMESTAMP como a seguir:

```
ROWCHGTS TIMESTAMP NOT NULL
      GENERATED ALWAYS
      FOR EACH ROW ON UPDATE AS
      ROW CHANGE TIMESTAMP
```

No Oracle, é possível utilizar a pseudo-coluna **ora_rowscn**, que representa o número de alteração de sistema do registro.

O atualizador do banco de dados baseado em tempo atualiza os mapas do ObjectGrid de três diferentes maneiras:

1. INVALIDATE_ONLY. Invalidar as entradas no mapa ObjectGrid se os registros correspondentes no banco de dados foram alterados.
2. UPDATE_ONLY. Atualizar as entradas no mapa do ObjectGrid se os registros correspondentes no banco de dados foram alterados. Entretanto, todos os registros recentemente inseridos no banco de dados são ignorados.
3. INSERT_UPDATE. Atualizar as entradas existentes no mapa do ObjectGrid com os valores mais recentes do banco de dados. Além disso, todos os registros recentemente inseridos no banco de dados são inseridos no mapa do ObjectGrid.

Para obter informações adicionais sobre como configurar o atualizador de dados baseado em tempo JPA, consulte o informações no *Guia de Administração*.

Desenvolvendo Aplicativos com a Estrutura Spring

Aprenda como integrar seus aplicativos eXtreme Scale com o Spring Framework popular.

Conceitos relacionados

"Visão Geral da Estrutura Spring" na página 120
O Spring é uma estrutura para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie as transações e configure clientes e servidores que compõem a grade de dados em memória implementada.

"Beans de Extensão Spring e Suporte a Espaço de Nomes" na página 421
O WebSphere eXtreme Scale fornece um recurso para declarar Plain Old Java Objects (POJOs) para uso como pontos de extensão no arquivo `objectgrid.xml`, além de uma maneira de nomear os beans e especificar o nome da classe. Normalmente, as instâncias da classe especificada são criadas, e tais objetos são usados como plug-ins. Agora, o eXtreme Scale pode delegar que Spring obtenha instâncias destes objetos plug-in. Se um aplicativo utiliza o Spring, então, normalmente, tais POJOs possuem um requisito de serem conectados ao resto do aplicativo.

Referências relacionadas

"Beans de Extensão Gerenciados pelo Spring" na página 419
É possível declarar que os POJOs sejam usados como pontos de extensão no arquivo `objectgrid.xml`. Se você nomear os beans e, em seguida, especificar o nome de classe, o eXtreme Scale normalmente cria instâncias da classe especificada e usa essas instâncias como o plug-in. O WebSphere eXtreme Scale pode delegar para que o Spring aja como o bean factory para obter instâncias desses objetos de plug-in.

Arquivo XML descritor do Spring
Use um arquivo XML descritor do Spring para configurar e integrar o eXtreme Scale com o Spring.

Arquivo `objectgrid.xsd` Spring
Use o arquivo `objectgrid.xsd` Spring para integrar o eXtreme Scale ao Spring para gerenciar as transações do eXtreme Scale e configurar os clientes e servidores.

Visão Geral da Estrutura Spring

O Spring é uma estrutura para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie as transações e configure clientes e servidores que compõem a grade de dados em memória implementada.

Transações Nativas Gerenciadas do Spring

O Spring fornece transações gerenciadas por contêiner que são similares a um servidor de aplicativos do Java Platform, Enterprise Edition. Porém, o mecanismo do Spring pode usar diferentes implementações. O WebSphere eXtreme Scale fornece a integração do gerenciador de transações que permite ao Spring para gerenciar os ciclos de vida da transação do ObjectGrid. Consulte o informações sobre transações nativas no *Guia de Programação* para obter detalhes.

Beans de Extensão Gerenciados do Spring e Suporte a Espaço de Nomes

Além disso, o eXtreme Scale se integra ao Spring para permitir que os beans de estilo do Spring definidos para pontos de extensão ou plug-ins. Este recurso fornece configurações mais sofisticadas e mais flexíveis para configuração dos pontos de extensão.

Além dos beans de extensão gerenciados do Spring, o eXtreme Scale fornece um espaço de nomes Spring chamado "objectgrid". Beans e implementações integradas são predefinidos neste espaço de nomes, o que facilita aos usuários configurar o eXtreme Scale.

Suporte ao Escopo Shard

Com a configuração do Spring estilo tradicional, um bean ObjectGrid pode se do tipo singleton ou prototype. O ObjectGrid também suporta um novo escopo chamado de escopo "shard". Se um bean for definido como escopo shard, então somente um bean será criado por shard. Todas as solicitações de beans com um ID ou IDs correspondentes a essa definição de bean no mesmo shard resulta nessa instância de bean específica sendo retornada pelo contêiner Spring.

O exemplo a seguir mostra que um bean com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl é definido com o escopo configurado para shard. Portanto, apenas uma instância da classe JPAPropFactoryImpl é criada por shard.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Fluxo da Web do Spring

O Fluxo da Web do Spring armazena seu estado de sessão em uma sessão HTTP por padrão. Se um aplicativo da web usar o eXtreme Scale para gerenciamento de sessões, o Spring armazenará automaticamente o estado com o eXtreme Scale. Além disso, a tolerância a falhas é ativada da mesma forma que a sessão.

Consulte as informações de gerenciamento de sessões HTTP no *Visão Geral do Produto* para obter detalhes adicionais.

compactando

As extensões Spring do eXtreme Scale estão no arquivo `ogspring.jar`. Este arquivo Java archive (JAR) deve estar no caminho de classe para o suporte ao Spring funcionar. Se um aplicativo Java EE que estiver em execução em um WebSphere Application Server Network Deployment aumentado pelo WebSphere Extended Deployment, coloque o arquivo `spring.jar` e seus arquivos associados nos módulos EAR (enterprise archive). Você também deve colocar o arquivo `ogspring.jar` no mesmo local.

Tarefas relacionadas

"Desenvolvendo Aplicativos com a Estrutura Spring" na página 414
Aprenda como integrar seus aplicativos eXtreme Scale com o Spring Framework popular.

"Iniciando um Servidor de Contêiner com o Spring" na página 424
É possível iniciar um servidor de contêiner usando beans de extensão gerenciados pelo Spring e o suporte ao namespace.

"Gerenciando Transações com o Spring"
O Spring é uma estrutura popular para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie transações do eXtreme Scale e configure clientes e servidores eXtreme Scale.

Referências relacionadas

"Beans de Extensão Gerenciados pelo Spring" na página 419
É possível declarar que os POJOs sejam usados como pontos de extensão no arquivo `objectgrid.xml`. Se você nomear os beans e, em seguida, especificar o nome de classe, o eXtreme Scale normalmente cria instâncias da classe especificada e usa essas instâncias como o plug-in. O WebSphere eXtreme Scale pode delegar para que o Spring aja como o bean factory para obter instâncias desses objetos de plug-in.

Arquivo XML descritor do Spring
Use um arquivo XML descritor do Spring para configurar e integrar o eXtreme Scale com o Spring.

Arquivo `objectgrid.xsd` Spring
Use o arquivo `objectgrid.xsd` Spring para integrar o eXtreme Scale ao Spring para gerenciar as transações do eXtreme Scale e configurar os clientes e servidores.

Gerenciando Transações com o Spring

O Spring é uma estrutura popular para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie transações do eXtreme Scale e configure clientes e servidores eXtreme Scale.

Sobre Esta Tarefa

A Estrutura Spring é altamente integrable com o eXtreme Scale, conforme discutido nas seções a seguir.

Procedimento

- **Transações nativas:** O Spring fornece transações gerenciadas por contêiner junto com o estilo de um servidor de aplicativos Java Platform, Enterprise Edition, mas com a vantagem de que o mecanismo Springs pode ter implementações diferentes conectadas. Este tópico descreve um gerenciador eXtreme Scale Platform Transaction que pode ser utilizado com o Spring. Isso permite que os programadores anotem os Plain Old Java Objects (POJOs), façam com que o Spring adquira automaticamente Sessões a partir do eXtreme Scale e comecem, confirmem, recuperem, suspendam e continuem as transações do eXtreme Scale. As transações do Spring são descritas mais completamente no Capítulo 10 da documentação oficial de referência do Spring. O seguinte explica como criar um gerenciador de transações do eXtreme Scale e usá-lo com os POJOs anotados. Ele também explica como utilizar esta abordagem com o eXtreme Scale cliente ou local, bem como com um aplicativo do estilo Data Grid.
- **Gerenciador de Transação:** Para trabalhar com Spring, eXtreme Scale fornece uma implementação de um Spring PlatformTransactionManager. Este gerenciador pode fornecer sessões gerenciadas do eXtreme Scale para POJOs

gerenciados pelo Spring. Através do uso das anotações, o Spring gerencia essas sessões para os POJOs em termos de ciclo de vida da transação. O seguinte snippet XML mostra como criar um Gerenciador de transações:

```
<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>
```

Isto mostra o bean transactionManager que está sendo declarado e vinculado ao bean Service que utilizará transações Spring. Demonstraremos isto utilizando anotações e este é o motivo ara a cláusula tx:annotation no início.

- **Obtendo uma sessão do ObjectGrid:** Um POJO que possui métodos gerenciados pelo Spring agora podem obter a sessão do ObjectGrid para a transação atual utilizando

  ```
  Session s = txManager.getSession();
  ```

  Isto retorna a sessão para o POJO utilizar. Beans participando na mesma transação receberão a mesma sessão quando eles chamam este método. O Spring identificará automaticamente o início para o objeto Session e também chamará automaticamente o commit ou rollback, quando necessário. Também é possível obter um EntityManager do ObjectGrid simplesmente chamando getEntityManager do objeto Session.
- **Configurando a instância do ObjectGrid para um encadeamento:** Uma única Java Virtual Machine (JVM) pode hospedar várias instâncias do ObjectGrid. Cada shard primário colocado em uma JVM possui sua própria instância do ObjectGrid. Uma JVM atuando como um cliente para um ObjectGrid remoto utiliza uma instância do ObjectGrid retornada do ClientClusterContext do método de conexão para interagir com tal Grade. Antes de chamar um método em um POJO utilizando transações Spring para o ObjectGrid, o encadeamento deve ser primed com a instância do ObjectGrid a utilizar. A instância do TransactionManager possui um método permitindo que uma instância específica do ObjectGrid seja especificada. Depois de especificada, todas as chamadas subsequentes do txManager.getSession retornarão Sessões para essa instância de ObjectGrid.

  O exemplo a seguir mostra um principal de amostra para exercitar este recurso:

  ```
  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");
  ```

  Aqui, utilizamos um Spring ApplicationContext. O ApplicationContext é utilizado para obter uma referência para o txManager e especificar um ObjectGrid a utilizar neste encadeamento. O código então obtém uma referência

para o serviço e chama métodos nele. Cada chamada de método neste nível faz com que o Spring crie um objeto Session e faz chamadas begin/commit em torno da chamada de método. Quaisquer exceções causarão um retrocesso.

- **Interface SpringLocalTxManager:** A interface SpringLocalTxManager é implementada pelo ObjectGrid Platform Transaction Manager e possui todas as interfaces públicas. Os métodos nesta interface são utilizados para selecionar a instância do ObjectGrid para utilizar em um encadeamento e obter um objeto Session para o encadeamento. Quaisquer POJOs que utilizam transações locais do ObjectGrid devem ser injetados com uma referência para esta instância do gerenciador e apenas uma única instância deve ser criada, ou seja, seu escopo deve ser um singleton. Esta instância é criada utilizando um método estático no ObjectGridSpringFactory. getLocalPlatformTransactionManager().

  Restrição: O WebSphere eXtreme Scale não suporta o JTA ou two-phase commit por vários motivos, principalmente devido à escalabilidade. Assim, exceto em um último participante single-phase, o ObjectGrid não interage em transações globais do tipo XA ou JTA. Este gerenciador de plataformas é destinado a tornar o uso de transações locais do ObjectGrid o mais fácil possível para os desenvolvedores do Spring.

Conceitos relacionados

"Visão Geral da Estrutura Spring" na página 120
O Spring é uma estrutura para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie as transações e configure clientes e servidores que compõem a grade de dados em memória implementada.

"Beans de Extensão Spring e Suporte a Espaço de Nomes" na página 421
O WebSphere eXtreme Scale fornece um recurso para declarar Plain Old Java Objects (POJOs) para uso como pontos de extensão no arquivo `objectgrid.xml`, além de uma maneira de nomear os beans e especificar o nome da classe. Normalmente, as instâncias da classe especificada são criadas, e tais objetos são usados como plug-ins. Agora, o eXtreme Scale pode delegar que Spring obtenha instâncias destes objetos plug-in. Se um aplicativo utiliza o Spring, então, normalmente, tais POJOs possuem um requisito de serem conectados ao resto do aplicativo.

Referências relacionadas

"Beans de Extensão Gerenciados pelo Spring"
É possível declarar que os POJOs sejam usados como pontos de extensão no arquivo `objectgrid.xml`. Se você nomear os beans e, em seguida, especificar o nome de classe, o eXtreme Scale normalmente cria instâncias da classe especificada e usa essas instâncias como o plug-in. O WebSphere eXtreme Scale pode delegar para que o Spring aja como o bean factory para obter instâncias desses objetos de plug-in.

Arquivo XML descritor do Spring
Use um arquivo XML descritor do Spring para configurar e integrar o eXtreme Scale com o Spring.

Arquivo `objectgrid.xsd` Spring
Use o arquivo `objectgrid.xsd` Spring para integrar o eXtreme Scale ao Spring para gerenciar as transações do eXtreme Scale e configurar os clientes e servidores.

Beans de Extensão Gerenciados pelo Spring

É possível declarar que os POJOs sejam usados como pontos de extensão no arquivo `objectgrid.xml`. Se você nomear os beans e, em seguida, especificar o nome de classe, o eXtreme Scale normalmente cria instâncias da classe especificada

e usa essas instâncias como o plug-in. O WebSphere eXtreme Scale pode delegar para que o Spring aja como o bean factory para obter instâncias desses objetos de plug-in.

Se um aplicativo usar o Spring, os POJOs precisarão estar acessíveis ao restante do aplicativo.

Um aplicativo pode registrar uma instância do Spring Bean Factory a ser usada para um ObjectGrid especificado por nome. O aplicativo cria uma instância de um contexto de aplicativo BeanFactorty ou Spring e, em seguida, registra-a com o ObjectGrid usando o seguinte método estático:

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object springBeanFactory)
```

O método anterior se aplica quando o eXtreme Scale localiza um bean de extensão cujo className inicia com o prefixo {spring}. Tal uma extensão de bean, que pode ser um ObjectTransformer, Loader, TransactionCallback, e assim por diante, o restante do nome é usado como um nome Spring Bean. Em seguida, a instância de bean é obtida usando o Spring Bean Factory.

O ambiente de implementação do eXtreme Scale também pode criar um Spring Bean Factory a partir de um arquivo de configuração XML do Spring padrão. Se nenhum bean factory foi registrado para um determinado ObjectGrid, sua implementação procurará pelo arquivo XML chamado "/<ObjectGridName>_spring.xml". Por exemplo, se sua grade de dados for denominada GRID, o arquivo XML será chamado "/GRID_spring.xml' e aparecerá no caminho de classe no pacote raiz. O ObjectGrid constrói um ApplicationContext usando o arquivo "/<ObjectGridName>_spring.xml e constrói beans a partir do bean factory.

A seguir há um exemplo de nome de classe:

```
"{spring}MyLoaderBean"
```

Usar o nome da classe anterior permite que o eXtreme Scale use o Spring para procurar um bean denominado "MyLoaderBean". É possível especificar POJOs gerenciados pelo Spring em qualquer ponto de extensão se o bean factory foi registrado. As extensões Spring estão no arquivo ogspring.jar. Esse arquivo JAR deve estar no caminho de classe para o suporte ao Spring. Se um aplicativo J2EE executado no WebSphere Application Server Network Deployment foi aumentado com o WebSphere Extended Deployment, você deverá colocar o aplicativo, o arquivo spring.jar e seus arquivos associados nos módulos EAR. O ogspring.jar também deve ser colocado no mesmo local.

Conceitos relacionados

"Visão Geral da Estrutura Spring" na página 120
O Spring é uma estrutura para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie as transações e configure clientes e servidores que compõem a grade de dados em memória implementada.

"Beans de Extensão Spring e Suporte a Espaço de Nomes"
O WebSphere eXtreme Scale fornece um recurso para declarar Plain Old Java Objects (POJOs) para uso como pontos de extensão no arquivo `objectgrid.xml`, além de uma maneira de nomear os beans e especificar o nome da classe. Normalmente, as instâncias da classe especificada são criadas, e tais objetos são usados como plug-ins. Agora, o eXtreme Scale pode delegar que Spring obtenha instâncias destes objetos plug-in. Se um aplicativo utiliza o Spring, então, normalmente, tais POJOs possuem um requisito de serem conectados ao resto do aplicativo.

Tarefas relacionadas

"Desenvolvendo Aplicativos com a Estrutura Spring" na página 414
Aprenda como integrar seus aplicativos eXtreme Scale com o Spring Framework popular.

"Iniciando um Servidor de Contêiner com o Spring" na página 424
É possível iniciar um servidor de contêiner usando beans de extensão gerenciados pelo Spring e o suporte ao namespace.

"Gerenciando Transações com o Spring" na página 417
O Spring é uma estrutura popular para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie transações do eXtreme Scale e configure clientes e servidores eXtreme Scale.

Beans de Extensão Spring e Suporte a Espaço de Nomes

O WebSphere eXtreme Scale fornece um recurso para declarar Plain Old Java Objects (POJOs) para uso como pontos de extensão no arquivo `objectgrid.xml`, além de uma maneira de nomear os beans e especificar o nome da classe. Normalmente, as instâncias da classe especificada são criadas, e tais objetos são usados como plug-ins. Agora, o eXtreme Scale pode delegar que Spring obtenha instâncias destes objetos plug-in. Se um aplicativo utiliza o Spring, então, normalmente, tais POJOs possuem um requisito de serem conectados ao resto do aplicativo.

Em alguns cenários, você deve usar o Spring para configurar um plug-in, como no exemplo a seguir:

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
        <property name="persistenceUnitName" type="java.lang.String"  value="employeePU" />
    </bean>
    ...
</objectGrid>
```

A implementação de TransactionCallback integrada, a classe com.ibm.websphere.objectgrid.jpa.JPATxCallback, é configurada como a classe TransactionCallback. Esta classe é configurada com a propriedade **`persistenceUnitName`**, conforme mostrado no exemplo anterior. A classe JPATxCallback também tem o atributo JPAPropertyFactory, que é do tipo java.lang.Object. A configuração XML do ObjectGrid não pode suportar este tipo de configuração.

A integração Spring do eXtreme Scale soluciona este problema delegando a criação do bean à estrutura do Spring. A configuração revisada é a seguinte:

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="{spring}jpaTxCallback"/>
    ...
</objectGrid>
```

O arquivo Spring para o objeto "Grid" contém as seguintes informações:

```
<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>
```

Aqui, o TransactionCallback está especificado como `{spring}jpaTxCallback`, e os beans jpaTxCallback e jpaPropFactory estão configurados no arquivo Spring como mostrado no exemplo anterior. A configuração Spring torna possível a configuração de um bean JPAPropertyFactory como um parâmetro do objeto JPATxCallback.

Bean factory Spring padrão

Quando o eXtreme Scale encontra um plug-in ou um bean de extensão (como um ObjectTransformer, utilitário de carga, TransactionCallback e assim por diante) com um valor className que inicia com o prefixo `{spring}`, o eXtreme Scale usará o restante do nome como um nome Spring Bean e obterá a instância do bean usando o Spring Bean Factory.

Pelo padrão, se nenhum bean factory tiver sido registrado para um determinado ObjectGrid, então ele tenta localizar um arquivo `ObjectGridName_spring.xml`. Por exemplo, se sua grade de dados for chamada como "Grid", então o arquivo XML será chamado como `/Grid_spring.xml`. Este arquivo deve estar no caminho da classe ou em um diretório `META-INF` que está no caminho da classe. Se este arquivo for encontrado, então o eXtreme Scale construi um ApplicationContext usando tal arquivo e construi beans a partir desse bean factory.

Bean factory Spring customizado

O WebSphere eXtreme Scale também fornece uma API ObjectGridSpringFactory para registrar uma instância do Spring Bean Factory para usar para um ObjectGrid específico nomeado. Esta API registra uma instância de BeanFactory com eXtreme Scale ussando o método estático a seguir:

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object
springBeanFactory)
```

Suporte a Espaço de Nomes

Desde a versão 2.0, o Spring possui um mecanismo para extensão baseado em esquema para o formato XML do Spring básico para definição e configuração de beans. O ObjectGrid usa este novo recurso para definir e configurar beans ObjectGrid. Com a extensão de esquema XML do Spring, algumas das implementações integradas dos plug-ins do eXtreme Scale e alguns beans do ObjectGrid são predefinidos no espaço de nomes "objectgrid". Ao escrever os arquivos de configuração Spring, não é necessário especificar o nome completo de classe das implementações integradas. Em vez disso, é possível referenciar os beans predefinidos.

Além disso, com os atributos dos beans definidos no esquema XML, é menos provável que você forneça um nome de atributo errado. A validação XML baseada no esquema XML pode capturar estes tipos de erros anteriormente no ciclo de desenvolvimento.

Estes beans definidos nas extensões de esquema XML são:

- transactionManager
- registro
- servidor
- catálogo
- catalogServerProperties
- contêiner
- JPALoader
- JPATxCallback
- JPAEntityLoader
- LRUEvictor
- LFUEvictor
- HashIndex

Estes beans são definidos no esquema XML objectgrid.xsd. Este arquivo XSD é enviado como arquivo `com/ibm/ws/objectgrid/spring/namespace/objectgrid.xsd` no arquivo `ogspring.jar`. Para obter descrições detalhadas do arquivo XSD e dos beans definidos no arquivo XSD, consulte o Arquivo XML descritor do Springinformações sobre o arquivo descritor Spring no *Guia de Administração*.

Use o exemplo JPATxCallback da seção anterior. Na seção anterior, o bean JPATxCallback é configurado como o seguinte:

```
<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>
```

Usando este recurso de espaço de nomes, a configuração XML do Spring pode ser escrita da seguinte forma:

```
<objectgrid:JPATxCallback id="jpaTxCallback" persistenceUnitName="employeeEMPU"
 jpaPropertyFactory="jpaPropFactory" />

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl"
 scope="shard">
</bean>
```

Observe que, em vez de especificar a classe com.ibm.websphere.objectgrid.jpa.JPATxCallback como no exemplo anterior, o bean objectgrid:JPATxCallback pré-definido foi usado diretamente. Como pode ser visto, esta configuração é menos detalhada e mais amigável para verificação de erro.

Para obter uma descrição de como trabalhar com beans Spring, consulte o “Iniciando um Servidor de Contêiner com o Spring” na página 424.

Tarefas relacionadas

"Desenvolvendo Aplicativos com a Estrutura Spring" na página 414
Aprenda como integrar seus aplicativos eXtreme Scale com o Spring Framework popular.

"Iniciando um Servidor de Contêiner com o Spring"
É possível iniciar um servidor de contêiner usando beans de extensão gerenciados pelo Spring e o suporte ao namespace.

"Gerenciando Transações com o Spring" na página 417
O Spring é uma estrutura popular para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie transações do eXtreme Scale e configure clientes e servidores eXtreme Scale.

Referências relacionadas

"Beans de Extensão Gerenciados pelo Spring" na página 419
É possível declarar que os POJOs sejam usados como pontos de extensão no arquivo `objectgrid.xml`. Se você nomear os beans e, em seguida, especificar o nome de classe, o eXtreme Scale normalmente cria instâncias da classe especificada e usa essas instâncias como o plug-in. O WebSphere eXtreme Scale pode delegar para que o Spring aja como o bean factory para obter instâncias desses objetos de plug-in.

Arquivo XML descritor do Spring
Use um arquivo XML descritor do Spring para configurar e integrar o eXtreme Scale com o Spring.

Arquivo `objectgrid.xsd` Spring
Use o arquivo `objectgrid.xsd` Spring para integrar o eXtreme Scale ao Spring para gerenciar as transações do eXtreme Scale e configurar os clientes e servidores.

Iniciando um Servidor de Contêiner com o Spring

É possível iniciar um servidor de contêiner usando beans de extensão gerenciados pelo Spring e o suporte ao namespace.

Sobre Esta Tarefa

Com vários arquivos XML configurados para Spring, é possível iniciar os servidores de contêiner eXtreme Scale básicos.

Procedimento

1. **Arquivo XML do ObjectGrid**

   Primeiramente, defina um arquivo XML do ObjectGrid muito simples que contenha um "Grid" do ObjectGrid e uma mapa "Test". O ObjectGrid possui um plug-in ObjectGridEventListener chamado "partitionListener", e o mapa "Test" possui um Evictor conectado chamado "testLRUEvictor". Observe que ambos os plug-ins ObjectGridEventListener e Evictor são configurados usando Spring pois seus nomes contêm "{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. **Arquivo XML de implementação do ObjectGrid:**

   Agora, crie um arquivo XML de implementação simples do ObjectGrid da forma a seguir. Ele particiona o ObjectGrid em 5 partições, e nenhuma réplica é necessária.

```
<?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. **Arquivo XML Spring do ObjectGrid**

   Agora serão usados tanto beans de extensão gerenciado Spring do ObjectGrid e recursos de suporte a espaço de nomes para configurar os beans ObjectGrid. O nome do arquivo XML Spring é `Grid_spring.xml`. Observe que estão incluídos dois esquemas no arquivo XML: `spring-beans-2.0.xsd` é para uso dos beans gerenciados do Spring, e `objectgrid.xsd` é para uso dos beans predefinidos no namespace do 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>
```

   Havia seis beans definidos neste arquivo XML do Spring:

   a. *objectgrid:register*: Isto registra o bean factory padrão para o "Grid" do ObjectGrid.
   b. *objectgrid:server*: Isto define um servidor do ObjectGrid com o nome "server". Este servidor também fornece o serviço de catálogo desde que ele possua um bean objectgrid:catalog aninhado nele.
   c. *objectgrid:catalog*: Isto define um terminal de serviço de catálogo ObjectGrid, que está configurado como "localhost:2809".

d. *objectgrid:container*: Isto define um contêiner ObjectGrid com o arquivo XML objectgrid especificado e o arquivo XML de implementação como discutido anteriormente. A propriedade de servidor especifica em qual servidor este contêiner está hospedado.

e. *objectgrid:LRUEvictor*: Isto define um LRUEvictor com a quantidade de filas LRU para usar configurada como 31.

f. *bean partitionListener*: Isto define um plug-in ShardListener. É necessário fornecer uma implementação para este plug-in, caso contrário, ele não poderá usar os beans predefinidos. Além disso, esse escopo do bean é configurado como "shard", o que significa que existe apenas uma instância desse ShardListener por shard ObjectGrid.

4. **Iniciando o servidor:**

   O fragmento a seguir inicia o servidor ObjectGrid, que hospeda tanto o serviço de contêiner e o serviço de catálogo. Como podemos ver, o único método que precisamos chamar para iniciar o servidor é para obter um "contêiner" de bean do bean factory. Isto simplifica a complexidade de programação pela movimentação da maioria da lógica na configuração do 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();
    }
}
```

Conceitos relacionados

"Visão Geral da Estrutura Spring" na página 120
O Spring é uma estrutura para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie as transações e configure clientes e servidores que compõem a grade de dados em memória implementada.

"Beans de Extensão Spring e Suporte a Espaço de Nomes" na página 421
O WebSphere eXtreme Scale fornece um recurso para declarar Plain Old Java Objects (POJOs) para uso como pontos de extensão no arquivo `objectgrid.xml`, além de uma maneira de nomear os beans e especificar o nome da classe. Normalmente, as instâncias da classe especificada são criadas, e tais objetos são usados como plug-ins. Agora, o eXtreme Scale pode delegar que Spring obtenha instâncias destes objetos plug-in. Se um aplicativo utiliza o Spring, então, normalmente, tais POJOs possuem um requisito de serem conectados ao resto do aplicativo.

Referências relacionadas

"Beans de Extensão Gerenciados pelo Spring" na página 419
É possível declarar que os POJOs sejam usados como pontos de extensão no arquivo `objectgrid.xml`. Se você nomear os beans e, em seguida, especificar o nome de classe, o eXtreme Scale normalmente cria instâncias da classe especificada e usa essas instâncias como o plug-in. O WebSphere eXtreme Scale pode delegar para que o Spring aja como o bean factory para obter instâncias desses objetos de plug-in.

Arquivo XML descritor do Spring
Use um arquivo XML descritor do Spring para configurar e integrar o eXtreme Scale com o Spring.

Arquivo `objectgrid.xsd` Spring
Use o arquivo `objectgrid.xsd` Spring para integrar o eXtreme Scale ao Spring para gerenciar as transações do eXtreme Scale e configurar os clientes e servidores.

Configurando Clientes na Estrutura Spring

É possível substituir as configurações do ObjectGrid do lado do cliente com a Estrutura Spring

Sobre Esta Tarefa

. O arquivo XML de exemplo a seguir mostra como construir um elemento ObjectGridConfiguration e utilizá-lo para substituir algumas configurações do lado do cliente. Uma configuração semelhante pode ser criada usando a configuração programática ou configurando o arquivo XML do descritor do ObjectGrid.

Procedimento

1. Crie um arquivo XML para configurar os clientes com a estrutura 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. Carregue o arquivo XML criado e construa o ObjectGrid.

```
BeanFactory beanFactory = new XmlBeanFactory(newUrlResource
  ("file:test/companyGridSpring.xml"));
ObjectGrid companyGrid = (ObjectGrid) beanFactory.getBean("companyGrid");
```

Consulte o “Visão Geral da Estrutura Spring” na página 120 para obter mais informações sobre como criar um arquivo descritor XML.

Capítulo 6. Ajuste do desempenho

É possível ajustar as configurações em seu ambiente para aumentar o desempenho geral de seu ambiente do WebSphere eXtreme Scale.

Ajustando o Agente de Dimensionamento de Cache para Estimativas Exatas de Consumo de Memória

O WebSphere eXtreme Scale suporta o dimensionamento do consumo de memória das instâncias BackingMap em grades de dados distribuídas. O dimensionamento do consumo de memória não é suportado para instâncias de grades de dados locais. O valor relatado por WebSphere eXtreme Scale para um determinado mapa é muito próximo do valor relatado pela análise de dump do heap. Se o objeto do mapa for complexo, os dimensionamentos poderão ser menos precisos. A mensagem CWOBJ4543 é exibida no log para qualquer objeto de entrada de cache que não pode ser dimensionada com exatidão porque ela é excessivamente complexa. É possível obter medidas mais precisas para evitar uma complexidade desnecessária no mapa.

Procedimento

- Ative o agente de dimensionamento.

  Se estiver usando uma Java virtual machine Java 5 ou superior, use o agente de dimensionamento. Com o agente de dimensionamento, o WebSphere eXtreme Scale pode obter informações adicionais a partir da JVM para melhorar suas estimativas. O agente pode ser carregado incluindo o seguinte argumento na linha de comandos JVM:

  ```
  -javaagent:WXS lib directory/wxssizeagent.jar
  ```

  Para uma topologia integrada, inclua o argumento na linha de comandos do processo do WebSphere Application Server.

  Para uma topologia distribuída, inclua o argumento na linha de comandos dos processos (contêineres) do eXtreme Scale e do processo do WebSphere Application Server.

  Quando carregada corretamente, a seguinte mensagem é gravada no arquivo `SystemOut.log`.

  ```
  CWOBJ4541I: O dimensionamento de memória de BackingMap aprimorado está ativado.
  ```

- Prefira os tipos de dados Java no lugar de tipos de dados customizados, onde possível.

  O WebSphere eXtreme Scale pode dimensionar com exatidão o custo de memória dos seguintes tipos:

  - java.lang.String e matrizes em que String é a classe do componente (String[])
  - Todos os tipos de wrapper primitivos (Byte, Short, Character, Boolean, Long, Double, Float, Integer) e matrizes em que os wrappers primitivos são o tipo de componente (por exemplo, Integer[], Character[])
  - java.math.BigDecimal e java.math.BigInteger e matrizes em que essas duas classes são o tipo de componente (BigInteger[] e BigDecimal[])
  - Tipos temporais (java.util.Date, java.sql.Date, java.util.Time, java.sql.Timestamp)
  - java.util.Calendar e java.util.GregorianCalendar

- Evite a internação do objeto, quando possível.

  Quando um objeto é inserido em um mapa, o WebSphere eXtreme Scale assume que ele possui referência apenas ao objeto e todos os objetos a todos os objetos aos quais ele referecia diretamente. Se inserir 1000 Objetos customizados em um mapa e cada um tiver uma referência à mesma instância da sequência, o WebSphere eXtreme Scale dimensionará esse instância da sequência em 1.000 vezes, superestimando o tamanho real do mapa no heap. No entanto, o WebSphere eXtreme Scale compensa corretamente para os seguintes cenários de internação comuns:

  – Referências a Enumerações de Java 5
  – Referências a Classes que seguem o Padrão de Enumeração Typesafe. As classes que seguem este padrão possuem apenas construtores privados definidos, possuem pelo menos um campo final estático privado de seu próprio tipo e, se Serializável for implementado, a classe implementará o método readResolve().
  – Internação de wrapper Primitivo Java 5. Por exemplo, use Integer.valueOf(1) em vez do novo Integer(1)

  Se você tiver que usar a internação, use uma das técnicas anteriores para obter estimativas mais exatas.

- Use tipos customizados com atenção.

  Ao usar tipos customizados, prefira tipos de dados primitivos para campos vs tipos de Objeto.

  Além disso, prefira os tipos de Objeto listados na entrada 2 no lugar de suas próprias implementações customizadas.

  Ao usar tipos customizados, mantenha a árvore de Objeto para um nível. Ao inserir um Objeto customizado em um mapa, o WebSphere eXtreme Scale calculará apenas o custo do Objeto inserido, que inclui os campos primitivos e todos os Objetos aos quais ele faz referência diretamente. O WebSphere eXtreme Scale não seguirá referências adicionais na árvore de Objeto. Se inserir um Objeto no mapa e o WebSphere eXtreme Scale detectar referências que não foram seguidas durante o processo de dimensionamento, uma mensagem codificada CWOBJ4543, que inclui o nome da Classe que não pôde ser totalmente dimensionada, é exibida. Quando esse erro ocorrer, trate as estatísticas de tamanho no mapa como dados de tendência, em vez de confiar nas estatísticas de dimensionamento como sendo totalmente exatas.

- Use o modo de cópia CopyMode.COPY_TO_BYTES, se possível.

  Use o modo de cópia de CopyMode.COPY_TO_BYTES para remover qualquer dúvida sobre como dimensionar os Objetos de valor que estão sendo inseridos no mapa, mesmo quando uma árvore de Objeto tiver muitos níveis para serem dimensionados normalmente (resultando na mensagem CWOBJ4543).

Conceitos relacionados

"Dimensionamento do Consumo do Cache de Memória"
O WebSphere eXtreme Scale pode estimar precisamente o uso de memória do heap Java de um determinado BackingMap em bytes. Aproveite este recurso para ajudar a dimensionar corretamente as configurações de heap e as políticas de despejo da Java virtual machine. O comportamento deste recurso varia com a complexidade dos Objects que estão sendo colocados no mapa de retorno e com a forma como o mapa é configurado. Atualmente, este recurso é suportado apenas para grades de dados distribuídas. As instâncias de grade de dados local não suportam o dimensionamento de bytes usados.

Dimensionamento do Consumo do Cache de Memória

O WebSphere eXtreme Scale pode estimar precisamente o uso de memória do heap Java de um determinado BackingMap em bytes. Aproveite este recurso para ajudar a dimensionar corretamente as configurações de heap e as políticas de despejo da Java virtual machine. O comportamento deste recurso varia com a complexidade dos Objects que estão sendo colocados no mapa de retorno e com a forma como o mapa é configurado. Atualmente, este recurso é suportado apenas para grades de dados distribuídas. As instâncias de grade de dados local não suportam o dimensionamento de bytes usados.

Considerações sobre o Consumo de Heap

O eXtreme Scale armazena todos os seus dados dentro do espaço de heap dos processos JVM que compõem a grade de dados. Para um determinado mapa, o espaço de heap que ele consome pode ser dividido nos seguintes componentes:

- O tamanho de todos os objetos principais atualmente no mapa
- O tamanho de todos os objetos de valor atualmente no mapa
- O tamanho de todos os objetos EvictorData que estão em uso pelos plug-ins Evictor no mapa
- A sobrecarga da estrutura de dados subjacente

O número de bytes usados que são relatados pelas estatísticas de dimensionamento é a soma desses quatro componentes. Esses valores são calculados por entrada nas operações de inserção, atualização e remoção de mapa, significando que o mapa eXtreme Scale tem um valor atual para o número de bytes que um determinado mapa de apoio consume.

Quando grades são particionadas, cada partição contém uma parte do mapa de apoio. Como as estatísticas de dimensionamento são calculadas no nível mais baixo do código do eXtreme Scale, cada partição de um mapa de apoio rastreia seu próprio tamanho. É possível usar as APIs de Estatísticas do eXtreme Scale para rastrear o tamanho acumulativo do mapa e também o tamanho de suas partições individuais.

Em geral, use o dimensionamento de dados como uma medida das tendências de dados ao longo do tempo, e não como uma medida exata do espaço de heap usado pelo mapa. Por exemplo, se o tamanho relatado de um mapa dobra de 5 MB para 10 MB, visualize o consumo de memória do mapa como tendo dobrado. A medida real de 10 MB pode ser exata por vários motivos. Se você levar em conta os motivos e seguir as boas práticas, a exatidão das medidas de tamanho se aproximará daquelas do pós-processamento de um dump de heap Java.

O principal problema com a exatidão é que o Modelo de Memória Java não é restritivo o suficiente para permitir medidas de memória que devem ser exatas. O problema fundamental é que um objeto pode estar ativo no heap devido a várias referências. Por exemplo, se a mesma instância de objeto de 5 KB for inserida em três mapas separados, qualquer um desses três mapas impede que o Object seja posto na lixeira. Nesta situação, qualquer uma das medidas a seguir seria justificável:

- O tamanho de cada mapa é aumentado em 5 KB.
- O tamanho do primeiro mapa no qual o Object é colocado é aumentado em 5 KB.
- O tamanho dos outros dois mapas não é aumentado. O tamanho de cada mapa é aumentado em uma fração do tamanho do objeto.

Esta ambiguidade se deve porque essas medidas devem ser consideradas como dados de tendência, a não ser que você tenha removido a ambiguidade por meio das opções de design, melhores práticas e do entendimento das opções de implementação que podem fornecer estatísticas mais precisas.

O eXtreme Scale supõe que um determinado mapa retém a única referência de vida longa para os Objects de chave e de valor que ele contém. Se o mesmo objeto de 5 KB for colocado em três mapas, o tamanho de cada mapa será aumentado em 5 KB. O aumento, geralmente, não é um problema, pois o recurso é suportado apenas para grades de dados distribuídas. Se você inserir o mesmo Object em três mapas diferentes em um cliente remoto, cada mapa recebe sua própria cópia do Object. As configurações COPY MODE transacionais padrão também geralmente garantem que cada mapa tenha sua própria cópia de um determinado Object.

Internalização do Object

A internalização de objeto pode provocar um desafio ao estimar ouso de memória do heap. Ao implementar a internalização de um objeto, o código do aplicativo assegura propositalmente que todas as referências a um determinado valor de objeto realmente apontem para a instância do mesmo objeto no heap e, portanto, o mesmo local na memória. Um exemplo disso pode ser a seguinte classe:

```
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:
    }
}
```

A internalização do Object causa uma superestimativa ao dimensionar as estatísticas porque o eXtreme Scale supõe que os objetos usam locais de memória diferentes. Se um milhão de objetos ShippingOrder existir, as estatísticas de dimensionamento exibirão o custo de um milhão de Sequências que mantêm as informações de estado. Na realidade, apenas três Sequências existentes são membros estáticos da classe. O custo da memória para os membros estáticos da classe nunca deve ser incluído em nenhum mapa do eXtreme Scale. No entanto, essa situação não pode ser detectada no tempo de execução. Há diversas maneiras pelas quais a internalização de objeto semelhante pode ser implementada, o que explica por que é tão difícil de detectar. Não é prático o eXtreme Scale proteger-se contra todas as implementações possíveis. Entretanto, o eXtreme Scale não protege contra os tipos de internalização de objeto mais normalmente usados. Para otimizar o uso da memória com a internalização de Object, implemente a internalização apenas em objetos personalizados que entram nas duas categorias a seguir para aprimorar a exatidão das estatísticas de consumo de memória:

- O eXtreme Scale é ajustado automaticamente para enumerações do Java 5 e para o padrão Typesafe Enum, conforme descrito em Visão Geral do Java 2 Platform Standard Edition 5.0: Enumerações.
- O eXtreme Scale automaticamente leva em consideração a internalização automática de tipos de wrapper primitivos, como Número Inteiro. A internalização automática para tipos de wrapper primitivos foi introduzida no Java 5 por meio da utilização de métodos estáticos do valueOf.

Estatísticas de Consumo de Memória

Use um dos métodos a seguir para acessar as estatísticas de consumo de memória.

API de Estatísticas

Use o método MapStatsModule.getUsedBytes() que fornece estatísticas para um único mapa, incluindo o número de entradas e a taxa de ocorrências.

Para obter detalhes, consulte Módulos Estatísticos.

Beans Gerenciados (MBeans)

Use a estatística MBean gerenciada pelo MapUsedBytes. É possível usar vários tipos diferentes de Java Management Extensions (JMX) MBeans para administrar e monitorar as implementações. Cada MBean faz referência a uma entidade específica, como um mapa, um eXtreme Scale, um servidor, um grupo de replicação ou um membro do grupo de replicação.

Para obter detalhes, consulte Administrando com Beans Gerenciados (MBeans).

Módulos PMI (Performance Monitoring Infrastructure)

É possível monitorar o desempenho de seus aplicativos com os módulos PMI. Especificamente, use o módulo PMI de mapa para contêineres integrados no WebSphere Application Server.

Para obter detalhes, consulte Módulos PMI.

Console do WebSphere eXtreme Scale

Com o console, é possível visualizar as estatísticas de consumo de memória. Consulte o Monitorando com o Console da Web.

Todos esses métodos acessam a mesma medida subjacente do consumo de memória de uma determinada instância BaseMap. O tempo de execução do WebSphere eXtreme Scale tenta, com o melhor esforço, calcular o número de bytes de memória de heap consumidos pelos objetos de chave e de valor que são armazenados no mapa e também a sobrecarga do próprio mapa. É possível ver quanta memória de heap cada mapa consume entre todas as grades de dados distribuídas.

Na maioria dos casos, o valor relatado por WebSphere eXtreme Scale para um determinado mapa é muito próximo do valor relatado pela análise de dump do heap. O WebSphere eXtreme Scale dimensiona exatamente sua própria sobrecarga, mas não pode considerar cada objeto possível que pode ser colocado em um mapa. Seguir as melhores práticas descritas no "Ajustando o Agente de Dimensionamento de Cache para Estimativas Exatas de Consumo de Memória" na página 431 pode melhorar a exatidão do tamanho em medidas de bytes fornecidas pelo WebSphere eXtreme Scale.

Tarefas relacionadas

"Ajustando o Agente de Dimensionamento de Cache para Estimativas Exatas de Consumo de Memória" na página 431
O WebSphere eXtreme Scale suporta o dimensionamento do consumo de memória das instâncias BackingMap em grades de dados distribuídas. O dimensionamento do consumo de memória não é suportado para instâncias de grades de dados locais. O valor relatado por WebSphere eXtreme Scale para um determinado mapa é muito próximo do valor relatado pela análise de dump do heap. Se o objeto do mapa for complexo, os dimensionamentos poderão ser menos precisos. A mensagem CWOBJ4543 é exibida no log para qualquer objeto de entrada de cache que não pode ser dimensionada com exatidão porque ela é excessivamente complexa. É possível obter medidas mais precisas para evitar uma complexidade desnecessária no mapa.

Ajustando o Desempenho para Desenvolvimento de Aplicativos

Para melhorar o desempenho de sua grade de dados em memória ou do espaço de processamento de banco de dados, é possível examinar várias considerações, como o uso das melhores práticas para recursos de produto, como bloqueio, serialização e desempenho de consulta.

Ajustando o Modo de Cópia

O WebSphere eXtreme Scale faz uma cópia do valor com base nas configurações de CopyMode disponíveis. Determine qual configuração funciona melhor para seus requisitos de implementação.

É possível usar o método setCopyMode(CopyMode, valueInterfaceClass) da API BackingMap para configurar o modo de cópia para um dos seguintes campos estáticos finais que estão definidos na classe com.ibm.websphere.objectgrid.CopyMode.

Quando um aplicativo usa a interface ObjectMap para obter uma referência para uma entrada de mapa, use essa referência somente dentro da transação da grade de dados que obteve a referência. O uso da referência em uma transação diferente pode gerar erros. Por exemplo, se você usar a estratégia de bloqueio pessimista para o BackingMap, uma chamada de método get ou getForUpdate adquire um bloqueio S (compartilhado) ou U (atualização), dependendo da transação. O método get retorna a referência ao valor e o bloqueio que foi obtido é liberado quando a transação é concluída. A transação deve chamar o método get ou

getForUpdate para bloquear a entrada do mapa em uma transação diferente. Cada transação deve obter sua própria referência para o valor chamando o método get ou getForUpdate em vez de reutilizar a mesma referência de valor em múltiplas transações.

CopyMode para Mapas de Entidade

Ao usar um mapa associado a uma entidade da API EntityManager, o mapa sempre retorna os objetos Tuple da entidade diretamente sem fazer uma cópia, a menos que o modo de cópia COPY_TO_BYTES esteja sendo usado. É importante que o CopyMode seja atualizado ou que a Tupla seja copiada apropriadamente ao fazer alterações.

COPY_ON_READ_AND_COMMIT

O modo COPY_ON_READ_AND_COMMIT é o modo padrão. O argumento valueInterfaceClass é ignorado quando este modo é utilizado. Esse modo assegura que um aplicativo não contém uma referência ao objeto de valor que está no BackingMap. Em vez disso, o aplicativo está sempre trabalhando com uma cópia do valor que está no BackingMap. O modo COPY_ON_READ_AND_COMMIT assegura que o aplicativo nunca possa danificar os dados que estão em cache no BackingMap. Quando uma transação do aplicativo chama um método ObjectMap.get para uma chave especificada e é o primeiro acesso da entrada do ObjectMap para essa chave, será retornada uma cópia do valor. Quando a transação for confirmada, todas as alterações feitas pelo aplicativo são copiadas no BackingMap para assegurar que o aplicativo não tenha uma referência ao valor confirmado no BackingMap.

COPY_ON_READ

O modo COPY_ON_READ aprimora o desempenho no modo COPY_ON_READ_AND_COMMIT, eliminando a cópia que ocorre quando uma transação é confirmada. O argumento valueInterfaceClass é ignorado quando este modo é utilizado. Para preservar a integridade dos dados do BackingMap, o aplicativo assegura que cada referência que ele possui para uma entrada será destruída após a confirmação da transação. Com esse modo, o método ObjectMap.get retorna uma cópia do valor em vez de retornar uma referência ao valor para assegurar que essas alterações feitas pelo aplicativo no valor não afetem o valor de BackingMap até que a transação seja confirmada. No entanto, quando a transação não é confirmada, não é feita uma cópia de alterações. Em vez disso, a referência à cópia que foi retornada pelo método ObjectMap.get é armazenada no BackingMap. O aplicativo destrói todas as referências de entrada do mapa após a confirmação da transação. Se o aplicativo não destruir as referências de entrada do mapa, o aplicativo pode fazer com que os dados em cache no BackingMap sejam danificados. Se um aplicativo estiver utilizando este modo e tiver problemas, vá para o modo COPY_ON_READ_AND_COMMIT para verificar se o problema ainda existe. Se o problema não existir mais, isto indica que o aplicativo está falhando ao destruir todas as suas referências após a confirmação da transação.

COPY_ON_WRITE

O modo COPY_ON_WRITE aprimora o desempenho no modo COPY_ON_READ_AND_COMMIT, eliminando a cópia que ocorre quando o método ObjectMap.get é chamado pela primeira vez por uma transação para uma chave especificada. O método ObjectMap.get retorna um proxy para o valor em vez de uma referência direta ao objeto de valor. O proxy assegura que não seja

feita uma cópia do valor, a menos que o aplicativo chame um método set na interface de valor especificada pelo argumento valueInterfaceClass. O proxy fornece uma cópia na implementação de gravação. Quando uma transação é confirmada, o BackingMap examina o proxy para determinar se foi feita alguma cópia como resultado da chamada de um método set. Se tiver sido feita uma cópia, a referência a essa cópia será armazenada no BackingMap. A grande vantagem deste modo é que um valor nunca é copiado durante uma leitura ou em uma confirmação quando a transação nunca chama um método set para alterar o valor.

Os modos COPY_ON_READ_AND_COMMIT e COPY_ON_READ fazem uma cópia detalhada quando um valor é recuperado do ObjectMap. Se um aplicativo atualizar apenas alguns dos valores recuperados em uma transação, este modo não será o ideal. O modo COPY_ON_WRITE suporta este comportamento de maneira eficiente, mas requer que o aplicativo utilize um padrão simples. Os objetos de valor devem suportar uma interface. O aplicativo deve usar os métodos nesta interface quando ele estiver interagindo com o valor em uma Sessão do eXtreme Scale. Se este for o caso, então o eXtreme Scale cria proxies para os valores que são retornados ao aplicativo. O proxy possui uma referência ao valor real. Se o aplicativo executa operações de leitura apenas, as operações de leitura sempre executam contra a cópia real. Se o aplicativo modificar um atributo no objeto, o proxy fará uma cópia do objeto real e, em seguida, fará a modificação na cópia. O proxy então utiliza a cópia desse ponto em diante. O uso das cópia permite que a operação de cópia seja completamente evitada para objetos que são apenas de leitura pelo aplicativo. Todas as operações de modificação devem começar com o prefixo configurado. Os Enterprise JavaBeans normalmente são codificados para usarem este estilo de nomenclatura de métodos para métodos que modificam os atributos dos objetos. Esta convenção deve ser seguida. Todos os objetos modificados são copiados no momento em que forem modificados pelo aplicativo. Este cenário de leitura e gravação é o cenário mais eficiente suportado pelo eXtreme Scale. Para configurar um mapa para utilizar o modo COPY_ON_WRITE, utilize o seguinte exemplo: Nesse exemplo, o aplicativo armazena objetos Person que são chaveados utilizando o nome no Mapa. O objeto pessoal é representado no seguinte 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;
    }
}
```

O aplicativo utiliza apenas a interface IPerson quando interage com valores que são recuperados de um ObjectMap. Modifique o objeto para utilizar uma interface como no exemplo a seguir.

```
interface IPerson
{
    void setName(String n);
    String getName();
```

```
    void setAge(int a);
    int getAge();
}
// Modificar Person para implementar a interface IPerson
class Person implements IPerson {
    ...
}
```

O aplicativo precisa então configurar o BackingMap para utilizar modo COPY_ON_WRITE, como no exemplo a seguir:

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// use COPY_ON_WRITE for this Map with
// IPerson as the valueProxyInfo Class
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// The application should then use the following
// pattern when using the PERSON Map.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// the application casts the returned value to IPerson and not Person
IPerson p = (IPerson)person.get("Billy");
p.setAge( p.getAge() + 1 );
...
// make a new Person and add to Map
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// the following snippet WON'T WORK. Will result in ClassCastException
sess.begin();
// the mistake here is that Person is used rather than
// IPerson
Person a = (Person)person.get("Bobby");
sess.commit();
```

A primeira seção mostra o aplicativo recuperando um valor que foi denominado Billy no mapa. O aplicativo lança o valor retornado para o objeto IPerson, não para o objeto Person porque o proxy retornado implementa duas interfaces:

- A interface especificada na chamada de método BackingMap.setCopyMode
- A interface com.ibm.websphere.objectgrid.ValueProxyInfo

É possível lançar o proxy para dois tipos. A última parte do trecho de código anterior demonstra o que não é permitido no modo COPY_ON_WRITE. O aplicativo recupera o registro do Bobby e tenta converter o registro para um objeto Person. Esta ação falha com uma exceção de lançamento de classe, porque o proxy retornado não é um objeto Person. O proxy retornado implementa o objeto IPerson e ValueProxyInfo.

A interface ValueProxyInfo e o suporte de atualização parcial: Esta interface permite que um aplicativo recupere o valor somente leitura consolidado referenciado pelo proxy ou o conjunto de atributos que foram modificados durante esta transação.

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

O método ibmGetRealValue retorna uma cópia de leitura do objeto. O aplicativo não deve modificar este valor. O método ibmGetDirtyAttributes retorna uma lista de cadeias que representam os atributos que foram modificados pelo aplicativo durante esta transação. O principal caso de uso para ibmGetDirtyAttributes está em um JDBC (Java Database Connectivity) o utilitário de carga baseado em CMP. Apenas os atributos que estão denominados na lista precisam ser atualizados na instrução SQL ou no objeto mapeado para a tabela, que resulta em um SQL gerado pelo Loader mais eficiente. Quando uma transação de cópia na gravação é confirmada e, se um utilitário de carga estiver conectado, o utilitário de carga poderá lançar os valores dos objetos modificados na interface ValueProxyInfo para obter estas informações.

A manipulação do método equals ao utilizar COPY_ON_WRITE ou proxies: Por exemplo, o código a seguir constrói um objeto Person e, então, o insere em um ObjectMap. Em seguida, ele recupera o mesmo objeto utilizando o método ObjectMap.get. O valor é lançado para a interface. Se o valor for lançado na interface Person, isto resultará em uma exceção ClassCastException, porque o valor retornado é um proxy que implementa a interface IPerson e não é um objeto Person. A verificação de igualdade falha ao utilizar a operação == porque eles não são o mesmo objeto.

```
session.begin();
// new the Person object
Person p = new Person(...);
personMap.insert(p.getName, p);
// retrieve it again, remember to use the interface for the cast
IPerson p2 = personMap.get(p.getName());
if(p2 == p) {
    // they are the same
} else {
    // they are not
}
```

Outra consideração é quando é necessário substituir o método equals. Conforme ilustrado no trecho de código a seguir, o método equals deve verificar se o argumento é um objeto que implementa a interface IPerson e lança o argumento para ser um IPerson. Como o argumento pode ser um proxy que implementa a interface IPerson, é necessário utilizar os métodos getAge e getName ao comparar variáveis de instância para igualdade.

```
{
    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 configuração ObjectQuery e HashIndex: Ao utilizar COPY_ON_WRITE com o ObjectQuery ou com um plug-in HashIndex, é importante configurar o esquema ObjectQuery e o plug-in HashIndex para acessar os objetos utilizando métodos de propriedade, que é o padrão. Se configurado para utilizar acesso ao campo, o mecanismo de consulta e o índice tentarão acessar os campos no objeto do proxy, que sempre retornará nulo ou 0, já que a instância do objeto será um proxy.

NO_COPY

O modo NO_COPY permite que um aplicativo assegure que ele nunca modifique um objeto de valor que é obtido utilizando o método ObjectMap.get em troca de aprimoramentos de desempenho. O argumento valueInterfaceClass é ignorado quando este modo é utilizado. Se este modo for utilizado, nunca será feita uma cópia do valor. Se o aplicativo modificar valores, então os dados no BackingMap serão danificados. O modo NO_COPY é útil, principalmente para mapas de leitura nos quais os dados nunca são modificados pelo aplicativo. Se o aplicativo estiver utilizando este modo e tiver problemas, vá para o modo COPY_ON_READ_AND_COMMIT para verificar se o problema ainda existe. Se o problema não existir mais, isto indica que o aplicativo está modificando o valor retornado pelo método ObjectMap.get, durante ou após a confirmação da transação. Todos os mapas associados às entidades da API EntityManager usam automaticamente este modo independentemente do que foi especificado na configuração do eXtreme Scale.

Todos os mapas associados às entidades da API EntityManager usam automaticamente este modo independentemente do que foi especificado na configuração do eXtreme Scale.

COPY_TO_BYTES

É possível armazenar objetos em um formato serializado em vez do formato POJO. Ao usar a configuração COPY_TO_BYTES, é possível reduzir a área de cobertura da memória que um gráfico grande de objetos pode consumir. Consulte “Aprimorando o Desempenho com Mapas de Matriz de Byte” na página 442 para obter informações adicionais.

COPY_TO_BYTES_RAW

7.1.1+ Com o modo COPY_TO_BYTES_RAW, é possível acessar diretamente o formulário serializado de seus dados. Este modo de cópia oferece uma maneira eficiente para você interagir com bytes serializados, que permite ignorar o processo de desserialização para acessar objetos na memória.

No arquivo XML descritor do ObjectGrid, é possível configurar o modo de cópia para COPY_TO_BYTES, e configurar programaticamente o modo de cópia para COPY_TO_BYTES_RAW nas instâncias em que você desejar acessar os dados brutos serializados. Configure o modo de cópia COPY_TO_BYTES_RAW no arquivo XML descritor do ObjectGrid apenas quando seu aplicativo usar os dados brutos como parte de um processo de aplicativo principal.

Uso Incorreto do CopyMode

Ocorrem erros quando um aplicativo tenta melhorar o desempenho utilizando o modo de cópia COPY_ON_READ, COPY_ON_WRITE ou NO_COPY, conforme descrito acima. Os erros intermitentes não ocorrem quando você altera o modo de cópia para COPY_ON_READ_AND_COMMIT.

Problema

O problema pode ser devido a dados danificados no mapa do ObjectGrid, que é um resultado de um aplicativo que está violando o contrato de programação do modo de cópia que está sendo utilizado. O dano dos dados pode causar erros imprevisíveis de forma intermitente ou de maneira inexplicada ou inesperada.

Solução

O aplicativo deve estar em conformidade com o contrato de programação estabelecido para o modo de cópia em utilização. Para os modos de cópia COPY_ON_READ e COPY_ON_WRITE, o aplicativo utiliza uma referência a um objeto de valor fora do escopo da transação a partir do qual a referência foi obtida. Para utilizar esses modos, o aplicativo deverá excluir a referência ao objeto de valor depois da conclusão da transação e obter uma nova referência em cada transação que acesse tal objeto. Para o modo de cópia NO_COPY, o aplicativo deve nunca alterar o objeto de valor. Nesse caso, programe o aplicativo de modo que ele não altere o objeto de valor ou configure-o para utilizar um modo de cópia diferente.

Referências relacionadas

Arquivo XML descritor do ObjectGrid
Para configurar o WebSphere eXtreme Scale, utilize um arquivo XML descritor do ObjectGrid e a API do ObjectGrid.

Aprimorando o Desempenho com Mapas de Matriz de Byte

É possível armazenar os valores em seus mapas em uma matriz de byte em vez do formulário POJO, o que reduz a área de cobertura da memória que um grande gráfico de objetos pode consumir.

Vantagens

A quantidade de memória que é consumida aumenta com o número de objetos em um gráfico de objetos. Ao reduzir um gráfico de objetos complicado a uma matriz de bytes, somente um objeto é mantido na pilha em vez de vários objetos. Com esta redução do número de objetos na pilha, o tempo de execução Java tem menos objetos para procurar durante a coleta de lixo.

O mecanismo de cópia padrão usado pelo WebSphere eXtreme Scale é a serialização, que é dispendiosa. Por exemplo, se o modo de cópia padrão de `COPY_ON_READ_AND_COMMIT` é usado, uma cópia é feita no tempo de leitura e no tempo de obtenção. Em vez de fazer uma cópia no tempo de leitura, com matrizes de byte, o valor é aumentado a partir dos bytes, e em vez de fazer uma cópia no tempo de consolidação, o valor é serializado para bytes. Usar matrizes de byte resulta em consistência de dados equivalentes à configuração padrão com uma redução da memória usada.

Ao usar matrizes de byte, note que ter um mecanismo de serialização otimizado é crítico para ver uma redução do consumo de memória. Para obter mais informações, consulte “Ajustando o Desempenho de Serialização” na página 449.

Configurando Mapas de Matriz de Byte

É possível ativar mapas de matriz de byte com o arquivo XML ObjectGrid modificando o atributo CopyMode que é usado por um mapa para a configuração COPY_TO_BYTES, mostrada no exemplo a seguir:

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

Considerações

Você deve considerar se usará ou não os mapas da matriz de byte em um determinado cenário. Embora seja possível reduzir o uso de memória, o uso do processador aumenta quando você usa matrizes de byte.

A seguinte lista destaca vários fatores que devem ser considerados antes de escolher usar da função do mapa de matriz de byte.

Tipo de Objeto

Comparativamente, a redução de memória pode não ser possível com o uso de mapas de matriz de byte para alguns tipos de objeto. Consequentemente, vários tipos de objetos existem para os quais você não deve usar mapas de matriz de byte. Se você estiver usando qualquer um dos wrappers primitivos Java como valores, ou um POJO que não contenha referências a outros objetos (somente campos primitivos de armazenamento), o número de Objetos Java já é o mais baixo possível–há apenas um. Como a quantidade de memória usada pelo objeto já está otimizada, usar um mapa de matriz de byte para esses tipos de objetos não é recomendado. Os mapas de matriz de byte são mais adequados a tipos de objeto que contenham outros objetos ou coletas de objetos nos quais o número total de objetos POJO seja maior que um.

Por exemplo, se você tiver um objeto Cliente que tenha um Endereço comercial e um Endereço residencial, assim como uma coleta de Pedidos, o número de objetos na heap e o número de bytes usados por esses objetos pode ser reduzido usando-se mapas de matriz de byte.

Acesso local

Ao usar outros modos de cópia, os aplicativos poderão ser otimizados quando as cópias forem feitas, se os objetos forem Clonáveis com o ObjectTransformer padrão ou quando um ObjectTransformer customizado for fornecido com um método copyValue otimizado. Comparado com outros modos de cópia, a cópia de operações de leituras, gravações ou consolidações terá um custo adicional ao acessar os objetos localmente. Por exemplo, se você tiver um cache perto em uma topologia distribuída ou estiver acessando diretamente uma instância ObjectGrid local ou de servidor, o tempo de acesso e de confirmação aumentará com o uso de mapas de matriz de bytes devido ao custo de serialização. Você verá um custo similar em uma topologia distribuída se usar agentes da grade de dados ou acessar o servidor primário ao utilizar o plug-in ObjectGridEventGroup.ShardEvents.

Interações de Plug-in

Com mapas de matriz de byte, os objetos não são aumentados durante a comunicação de um cliente com um servidor a menos que o servidor precise do formulário POJO. Os plug-ins que interagem com o valor do mapa experimentarão uma redução no desempenho devido ao requisito para aumentar o valor.

Qualquer plug-in que use o LogElement.getCacheEntry ou LogElement.getCurrentValue verá esse custo adicional. Se você desejar obter a chave, é possível usar LogElement.getKey, que evita o custo adicional associado com o método LogElement.getCacheEntry().getKey. As seções a seguir discutem os plug-ins sob a perspectiva do uso de matrizes de byte.

Índices e consultas

Quando os objetos são armazenados em formato POJO, o custo de fazer indexação e consulta é mínimo porque o objeto não precisa ser aumentado. Ao usar um mapa de matriz de byte, você terá o custo adicional de aumentar o objeto. Em geral, se o seu aplicativo usar índices ou consultas, é recomendado usar mapas de matriz de byte a menos que você execute somente consultas sobre atributos-chave.

Bloqueio Otimista

Ao usar a estratégia de bloqueio otimista, você terá o custo adicional durante atualizações e operações inválidas. Isso advém da necessidade de aumentar o valor no servidor para obter o valor da versão para fazer verificação de colisão otimista. Se você estiver apenas usando o bloqueio otimista para garantir operações de busca e não precisar de verificação de colisão otimista, é possível usar o com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback para desativar a verificação de versão.

Utilitário de carga

Com um Utilitário de Carga, você também terá o custo no tempo de execução do eXtreme Scale de aumentar e reserializar o valor quando ele for usado pelo Utilitário de Carga. Também é possível usar mapas de matriz de byte com Utilitários de Carga, mas considere o custo de fazer alterações no valor em tal cenário. Por exemplo, é possível usar o recurso de matriz de byte no contexto de um cache principalmente de leitura. Neste caso, o benefício de ter menos objetos na heap e menos memória usada excederá o custo incorrido de usar matrizes de byte em operações de inserção e atualização.

ObjectGridEventListener

Ao utilizar o método transactionEnd method no plug-in ObjectGridEventListener, você terá um custo adicional no lado do servidor para pedidos remotos ao acessar um CacheEntry ou o valor atual de LogElement. Se a implementação do método não acessar esses campos, então você não terá o custo adicional.

Referências relacionadas

Arquivo XML descritor do ObjectGrid
Para configurar o WebSphere eXtreme Scale, utilize um arquivo XML descritor do ObjectGrid e a API do ObjectGrid.

Ajustando Operações de Cópia com a Interface ObjectTransformer

A interface ObjectTransformer utiliza retornos de chamada para o aplicativo para fornecer implementações customizadas de operações comuns e caras, como serialização de objeto e cópias detalhadas em objetos.

 A interface ObjectTransformer foi substituída pelos plug-ins DataSerializer, que podem ser usados para armazenar dados arbitrários eficientemente no WebSphere eXtreme Scale para que as APIs do produto existentes possam ser interagir de modo eficiente com seus dados.

Visão Geral

As cópias de valores são sempre feitas, exceto quando o modo NO_COPY é usado. O mecanismo de cópia padrão que é empregado no eXtreme Scale é a serialização, que é conhecida como uma operação cara. A interface ObjectTransformer é usada nesta situação. A interface ObjectTransformer usa retornos de chamadas para o aplicativo fornecer uma implementação customizada de operações comuns e caras, como serialização de objetos e cópias detalhadas em objetos.

Um aplicativo pode oferecer uma implementação da interface ObjectTransformer para um mapa, e o eXtreme Scale então delega os métodos neste objeto e confia no aplicativo para oferecer uma versão otimizada de cada método na interface. A interface ObjectTransformer é a seguinte:

```
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);
}
```

É possível associar uma interface ObjectTransformer com um BackingMap usando o seguinte código de exemplo:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ajustar Operações de Cópia Detalhada

Depois que um aplicativo receber um objeto de um ObjectMap, o eXtreme Scale executará uma cópia detalhada no valor do objeto para assegurar que a cópia no mapa BaseMap mantenha a integridade dos dados. O aplicativo pode então modificar o valor de objeto de maneira segura. Quando a transação for confirmada, a cópia do valor de objeto no mapa BaseMap será atualizada para o novo valor modificado e o aplicativo parará de utilizar o valor desse ponto em diante. Você poderia ter copiado o objeto novamente na fase de confirmação para fazer uma cópia privada. Entretanto, nesse caso, o custo do desempenho desta ação foi equilibrado ao solicitar que o programador do aplicativo não utilize o valor após a confirmação da transação. O ObjectTransformer padrão tenta utilizar um clone ou um par de serialize e inflate para gerar uma cópia. O par de serialize e inflate é o cenário de desempenho de pior caso. Se o traçado de perfil indicar que serialize e inflate são um problema para seu aplicativo, grave um método de clone apropriado para criar uma cópia detalhada. Se você não conseguir alterar a classe, crie um plug-in ObjectTransformer customizado e implemente mais métodos copyValue e copyKey eficientes.

Ajustando Evictors

Se você utilizar evictores de plug-in, eles não ficarão ativos até você criá-los e associá-los a um mapa de apoio. As boas práticas a seguir aumentarão o desempenho para least frequently used (LFU) e menos utilizado recentemente used (LRU).

Evictor LFU (Least Frequently Used)

O conceito de um evictor LFU é remover entradas do mapa que não são utilizadas freqüentemente. As entradas do mapa são distribuídas em uma quantidade de heaps binários configurados. Conforme aumenta o uso de uma determinada entrada de cache, ela ocupa uma posição mais alta no heap. Quando o evictor tenta um conjunto de evicções, ele remove apenas as entradas de cache que estão localizadas abaixo de um ponto específico no heap binário. Por isso, as entradas Least Frequently Used são liberadas.

Evictor LRU (Least Recently Used)

O Evictor LRU segue os mesmos conceitos do Evictor LFU com algumas diferenças. A principal diferença é que o LRU utiliza uma fila PEPS (Primeiro a

Entrar, Primeiro a Sair) em vez de um conjunto de heaps binários. Sempre que uma entrada de cache é acessada, ela é movida para o início da fila. Consequentemente, a frente da fila contém as entradas de mapa recentemente mais usadas e, seu final, as entradas de mapa recentemente menos usadas. Por exemplo, a entrada de cache A é utilizada 50 vezes e a entrada de cache B é utilizada apenas uma vez após a entrada de cache A. Neste caso, a entrada de cache B está no início da fila, porque foi utilizada mais recentemente e a entrada de cache A está no final da fila. O evictor LRU libera as entradas de cache que estão no final da fila, que são as entradas do mapa Least Recently Used.

Propriedades LFU e LRU e Boas Práticas para Aprimorar o Desempenho

Número de heaps

Ao utilizar o evictor LFU, todas as entradas de cache para um determinado mapa são ordenadas sobre o número de heaps especificado, aprimorando o desempenho significativamente e impedindo que todas as evicções sejam sincronizadas em um heap binário que contenha todas as ordenações para o mapa. Uma maior quantidade de heaps também acelera o tempo requerido para reordenação dos heaps, porque cada heap tem menos entradas. Configure o número de heaps como 10% do número de entradas em seu BaseMap.

Número de filas

Ao utilizar o evictor LFU, todas as entradas de cache para um determinado mapa são ordenadas sobre o número de filas LRU especificado, aprimorando o desempenho significativamente e impedindo que todas as evicções sejam sincronizadas em uma fila que contenha todas as ordenações para o mapa. Configure o número de filas como 10% do número de entradas em seu BaseMap.

Propriedade MaxSize

Quando um evictor LFU ou LRU começa a liberar entradas, ele utiliza a propriedade do evictor MaxSize para determinar quantos heaps binários ou elementos de fila LRU serão liberados. Por exemplo, suponha que você tenha configurado o número de heaps ou filas para ter aproximadamente 10 entradas do mapa em cada fila do mapa. Se sua propriedade MaxSize estiver configurada como 7, o evictor liberará 3 entradas de cada heap ou objeto de fila para retornar o tamanho de cada heap ou fila para abaixo de 7. O evictor libera apenas entradas do mapa de um heap ou fila quando esse heap ou fila tiver mais do que o valor da propriedade MaxSize de elementos contidos nele. Configure MaxSize como 70% do tamanho de heap ou de fila. Para este exemplo, o valor é configurado como 7. É possível obter um tamanho aproximado de cada heap ou fila, dividindo o número de entradas BaseMap pelo número de heaps ou filas utilizadas.

Propriedade SleepTime

Um evictor não remove constantemente entradas de seu mapa. Ao invés disso, ele está inativo para uma quantidade de tempo configurada, verificando o mapa apenas a cada n segundos, em que n faz referência à propriedade SleepTime. Esta propriedade também afeta de forma positiva o desempenho: a execução muito freqüente de uma limpeza por evicção reduz o desempenho devido aos recursos necessários para esse processamento. Porém, não usar o evictor frequentemente pode resultar em um mapa que tem três entradas que não são necessárias. Um mapa completo de entradas desnecessárias pode afetar negativamente os requisitos

de memória e os recursos de processamento requeridos para seu mapa. A configuração de limpezas por despejo como quinze segundos é uma boa prática para a maioria dos mapas. Se houver gravações freqüentes no mapa e ele for utilizado em uma alta taxa de transações, considere a configuração do valor com um tempo inferior. No entanto, se o mapa for acessado com pouca freqüência, será possível configurar o tempo com um valor superior.

Exemplo

O exemplo a seguir define um mapa, cria um novo evictor LFU, configura as propriedades do evictor e configura o mapa para utilizar o evictor:

```
//Utilizar ObjectGridManager para criar/obter o ObjectGrid. Refer to
// a seção do ObjectGridManger
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");

//Configurar propriedades assumindo 50.000 entradas do mapa
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

A utilização do evictor LRU é muito semelhante à utilização de um evictor LFU. Este é um exemplo:

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");

//Configurar propriedades assumindo 50.000 entradas do mapa
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Notice that only two lines are different from the LFUEvictor example.

Tarefas relacionadas

Ativando Evictors Programaticamente
Os evictors estão associados às instâncias do BackingMap.

Ativando Evictors com a Configuração XML
Em vez de usar a interface BackingMap para programaticamente configurar os atributos de BackingMap a serem usados pelos Evictor TTL, é possível usar um arquivo XML para configurar cada instância de BackingMap. O código a seguir demonstra como configurar tais atributos para três mapas BackingMap diferentes:

Referências relacionadas

Arquivo XML descritor do ObjectGrid
Para configurar o WebSphere eXtreme Scale, utilize um arquivo XML descritor do ObjectGrid e a API do ObjectGrid.

Ajustando o Desempenho de Bloqueio

Estratégias de bloqueio e configurações de isolamento de transação afetam o desempenho dos seus aplicativos.

Recuperar uma Instância Armazenada em Cache

Para obter mais informações, consulte “Gerenciador de Bloqueio” na página 235:

Estratégia de Bloqueio Pessimista

Utilize a estratégia de bloqueio pessimista para operações de leitura e gravação de mapas em que, normalmente, ocorrem conflitos de chaves. A estratégia de bloqueio pessimista tem o maior impacto no desempenho.

Isolamento de Transação de Leitura Committed e Uncommitted

Ao usar a estratégia de bloqueio pessimista, consulte o nível de isolamento de transação usando o método Session.setTransactionIsolation. Para o isolamento de leitura confirmada e de leitura não-confirmada, use os argumentos Session.TRANSACTION_READ_COMMITTED ou Session.TRANSACTION_READ_UNCOMMITTED dependendo do isolamento. Para reconfigurar o nível de isolamento de transação para o comportamento de bloqueio pessimista padrão, use o método Session.setTransactionIsolation com o argumento Session.REPEATABLE_READ.

O isolamento de leitura committed reduz a duração dos bloqueios compartilhados, o que pode melhorar a simultaneidade e reduzir a chance de conflitos. Este nível de isolamento deve ser utilizado quando uma transação não precisa de garantias de que os valores de leitura permanecerão inalterados ao longo da duração da transação.

Utilize uma leitura não-confirmada quando a transação não precisa visualizar os dados confirmados.

Estratégia de Bloqueio Otimista

O bloqueio otimista é a configuração padrão. Tal estratégia melhora o desempenho e a escabilidade quando comparada com a estratégia pessimista. Utilize-a quando seus aplicativos tolerarem algumas falhas de atualização otimista e o desempenho ainda mostrar-se melhor do que com a estratégia pessimista. Essa estratégia é excelente para operações de leitura e aplicativos cuja atualização não ocorre com freqüência.

Plug-in OptimisticCallback

A estratégia de bloqueio optimistic faz uma cópia das entradas de cache e as compara, conforme necessário. Esta operação pode ser custosa porque a cópia da entrada pode envolver clonagem ou serialização. Para implementar o desempenho mais rápido possível, implemente o plug-in customizado para os mapas de não entidade.

Consulte para obter informações adicionais. Consulte as informações sobre o plug-in OptimisticCallback no *Visão Geral do Produto* para obter mais informações.

Utilize Campos de Versão para Entidades

Quando você está utilizando o bloqueio optimistic com entidades, utilize a anotação @Version ou o atributo equivalente no arquivo descritor dos metadados da Entidade. A anotação da versão fornece ao ObjectGrid uma maneira muito eficiente de controlar a versão de um objeto. Se a entidade não possui um campo de versão e bloqueio optimistic é utilizado para a entidade, então, a entidade inteira deve ser copiada e comparada.

Estratégia de Bloqueio None

Não utilize nenhuma estratégia de bloqueio para aplicativos de somente leitura. A estratégia de bloqueio none não obtém nenhum bloqueio nem utilizar um gerenciador de bloqueios. Portanto, essa estratégia oferece maior simultaneidade, desempenho e escalabilidade.

Ajustando o Desempenho de Serialização

WebSphere eXtreme Scale utiliza vários processos Java para conter dados. Esses processos serializam os dados: ou seja, convertem os dados (que estão no formato de instâncias de objeto Java) em bytes e volta em objetos novamente, conforme necessário, para mover os dados entre processos do cliente e do servidor. Serializar os dados é a operação mais dispendiosa e deve ser endereçada pelo desenvolvedor de aplicativos ao projetar o esquema, configurar a grade de dados e interagir com as APIs de acesso a dados.

As rotinas de cópia e serialização Java são relativamente lentas e podem consumir de 60% a 70% do processador em uma configuração típica. As seções a seguir são escolhas para melhorar o desempenho da serialização.

 A interface ObjectTransformer foi substituída pelos plug-ins DataSerializer, que podem ser usados para armazenar dados arbitrários eficientemente no WebSphere eXtreme Scale para que as APIs do produto existentes possam ser interagir de modo eficiente com seus dados.

Gravação em um ObjectTransformer para cada BackingMap

Um ObjectTransformer pode ser associado a um BackingMap. O aplicativo pode ter uma classe que implementa a interface ObjectTransformer e fornece implementações para as seguintes operações:

- Copiando valores
- Serializando e aumentando chaves para e de fluxos
- Serializando e aumentando valores para e de fluxos

O aplicativo não precisa copiar chaves, porque elas são consideradas imutáveis.

Nota: O ObjectTransformer é chamado apenas quando o ObjectGrid conhece os dados que estão sendo transformados. Por exemplo, quando agentes de API do DataGrid são utilizados, os próprios agentes bem como os dados da instância do agente ou dados retornados do agente devem ser otimizados utilizando técnicas de serialização customizadas. O ObjectTransformer não é chamado para os agentes de API do DataGrid.

Usando Entidades

Ao utilizar a API do EntityManager com entidades, o ObjectGrid não armazena os objetos de entidade diretamente nos BackingMaps. A API do EntityManager converte o objeto de entidade em objetos de Tupla. Os mapas de entidade são automaticamente associados com um ObjectTransformer altamente otimizado. Sempre que a API do ObjectMap ou a API do EntityManager for utilizada para interagir com mapas de entidade, o ObjectTransformer da entidade será chamado.

Serialização Customizada

Há alguns casos em que os objetos devem ser modificados para usar a serialização customizada, tais como implementar a interface java.io.Externalizable ou implementar os métodos writeObject e readObject para classes implementando a interface java.io.Serializable. As técnicas de serialização customizadas devem ser empregadas quando os objetos são serializados utilizando mecanismos que não os métodos da API do ObjectGrid ou da API do EntityManager.

Por exemplo, quando objetos ou entidades são armazenados como dados da instância em um agente da API do DataGrid ou o agente retorna objetos ou entidades, tais objetos não são transformados utilizando um ObjectTransformer. O agente, entretanto, utilizará automaticamente o ObjectTransformer ao utilizar a interface `EntityMixin`. Consulte Agentes do DataGrid e Mapas Baseados em Entidade para obter mais detalhes.

Matrizes de Byte

Ao usar as APIs ObjectMap ou DataGrid, os objetos de valor e chave são serializados sempre que os clientes interagem com a grade de dados e quando os objetos são replicados. Para evitar o gasto adicional de serialização, use matrizes de byte em vez de objetos Java. As matrizes de byte são muito mais baratas para armazenar em memória porque o JDK tem menos objetos para buscar durante a coleta de lixo e elas podem ser aumentadas somente quando necessário. As matrizes de byte somente devem ser usadas se você não precisar acessar os objetos usando consultas ou índices. Como os dados são armazenados como bytes, os dados somente podem ser acessados através de sua chave.

O WebSphere eXtreme Scale pode armazenar dados automaticamente como matrizes de bytes usando a opção de configuração de mapa CopyMode.COPY_TO_BYTES, ou ele pode ser manipulado manualmente pelo cliente. Esta opção armazenará os dados de maneira eficiente na memória e também pode aumentar automaticamente os objetos dentro da matriz de bytes para uso por consulta e índices sob demanda.

Um plug-in MapSerializerPlugin pode ser associado a um plug-in BackingMap quando usar os modos de cópia COPY_TO_BYTES ou COPY_TO_BYTES_RAW. Esta associação permite que os dados sejam armazenados em formato serializado na memória, em vez de em formato de objeto Java nativo. Armazenando dados serializados preserva a memória e aprimora a replicação e o desempenho no cliente e no servidor. É possível usar um plug-in DataSerializer para desenvolver fluxos de serialização de alto desempenho que podem ser compactados, criptografados e consultados.

Ajustando a Serialização

Os plug-ins DataSerializer expõem os metadados que informa ao WebSphere eXtreme Scale quais atributos ele pode e não pode usar diretamente durante a serialização, o caminho para os dados que serão serializados e o tipo de dados que é armazenado na memória. É possível otimizar a serialização do objeto e o desempenho de desserialização a fim de interagir com eficiência com a matriz de bytes.

Visão Geral

 A interface ObjectTransformer foi substituída pelos plug-ins DataSerializer, que podem ser usados para armazenar dados arbitrários eficientemente no WebSphere eXtreme Scale para que as APIs do produto existentes possam ser interagir de modo eficiente com seus dados.

As cópias de valores são sempre feitas, exceto quando o modo NO_COPY é usado. O mecanismo de cópia padrão que é empregado no eXtreme Scale é a serialização, que é conhecida como uma operação cara. A interface ObjectTransformer é usada nesta situação. A interface ObjectTransformer usa retornos de chamadas para o aplicativo fornecer uma implementação customizada de operações comuns e caras, como serialização de objetos e cópias detalhadas em objetos. No entanto, para obter melhor desempenho na maioria dos casos, é possível usar os plug-ins DataSerializer para serializar os objetos. Você deve usar os modos de cópia COPY_TO_BYTES ou COPY_TO_BYTES_RAW para explorar os plug-ins DataSerializer. Para obter informações adicionais, consulte Serialização Usando os Plug-ins DataSerializer.

Um aplicativo pode oferecer uma implementação da interface ObjectTransformer para um mapa, e o eXtreme Scale então delega os métodos neste objeto e confia no aplicativo para oferecer uma versão otimizada de cada método na interface. A interface ObjectTransformer é a seguinte:

```
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);
}
```

É possível associar uma interface ObjectTransformer com um BackingMap usando o seguinte código de exemplo:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ajustar a Serialização e Aumento de Objetos

A serialização de objeto é normalmente a consideração mais importante de desempenho com o eXtreme Scale, que usa o mecanismo serializável padrão se um plug-in ObjectTransformer não for fornecido pelo aplicativo. Um aplicativo pode fornecer implementações de readObject e writeObject Serializáveis ou pode fazer os objetos implementarem a interface Externalizable, que é aproximadamente dez vezes mais rápida. Se os objetos no mapa não puderem ser modificados, um aplicativo poderá associar uma interface ObjectTransformer ao ObjectMap. Os métodos serialize e inflate são fornecidos para permitir que o aplicativo forneça código customizado para otimizar estas operações devido ao seu grande impacto no desempenho do sistema. O método serializa o objeto para o fluxo fornecido. O método inflate fornece um fluxo de entrada e espera que o aplicativo crie o objeto, aumente-o utilizando dados do fluxo e retorne o objeto. As implementações dos métodos serialize e inflate devem se espelhar entre si.

7.1.1+ Os plug-ins DataSerializer substituem os plug-ins do ObjectTransformer, que são descontinuados. Para serializar os dados da forma mais eficiente, use os plug-ins DataSerializer para melhorar o desempenho na maioria dos casos. Por

exemplo, se pretender usar funções, como consulta e a indexação, será possível obter vantagem imediatamente do aprimoramento de desempenho que os plug-ins do DataSerializer geram sem fazer mudanças de configuração ou programáticas para seu código do aplicativo.

Ajustando o Desempenho de Consulta

Para ajustar o desempenho de suas consultas, utilize as técnicas e dicas a seguir.

Utilizando Parâmetros

Quando uma consulta é executada, a cadeia de consultas deve ser analisada e um plano desenvolvido para executar a consulta, o que podem ter um alto custo. O WebSphere eXtreme Scale armazena em cache os planos de consulta pela cadeia de consulta. Visto que o cache tem um tamanho limitado, é importante reutilizar as cadeias de consultas sempre que possível. Utilizar os parâmetros nomeados ou posicionais também ajuda no desempenho, estimulando a reutilização do plano de consulta.

```
Positional Parameter Example  Query q = em.createQuery("select c from
Customer c where c.surname=?1");   q.setParameter(1, "Claus");
```

Utilizando Índices

A indexação adequada em um mapa pode ter um impacto significativo no desempenho da consulta, mesmo que a indexação tenha alguma sobrecarga no desempenho total do mapa. Se a indexação em atributos de objetos envolvidos em consultas, o mecanismo de consulta desempenha uma varredura de tabela para cada atributo. A varredura de tabela é a operação mais cara durante a execução de uma consulta. A indexação sobre atributos do objeto que são envolvidos em consultas permite que o mecanismo de consulta evite uma varredura de tabela desnecessária, melhorando o desempenho total da consulta. Se o aplicativo é designado para utilizar a consulta de maneira intensiva em um mapa que é em sua maior parte de leitura, configure índices para os atributos de objeto que estão envolvidos na consulta. Se o mapa for atualizado em sua maior parte, será necessário equilibrar entre o aprimoramento de desempenho de consulta e a sobrecarga de indexação no mapa.

Quando os POJO (Plain Old Java Objects) são armazenados em um mapa, a indexação adequada pode evitar uma reflexão Java. No exemplo a seguir, a consulta substitui a cláusula WHERE pela pesquisa de índice de intervalo, se o campo budget tiver um índice construído sobre ele. Caso contrário, a consulta varre o mapa inteiro e avalia a cláusula WHERE obtendo primeiro o orçamento utilizando o reflexo Java e, então, comparando o orçamento com o valor 50000:

```
SELECT d FROM DeptBean d WHERE d.budget=50000
```

Consulte “Plano de Consulta” na página 453 para obter detalhes sobre como ajustar melhor consultas individuais e como sintaxe, modelos de objetos e índices diferentes podem afetar o desempenho da consulta.

Utilizando a Paginação

Em ambientes de cliente-servidor, o mecanismo de consulta transporta o mapa de resultado inteiro para o cliente. Os dados retornados deveriam ser divididos em partes razoáveis. As interfaces EntityManager Query e ObjectMap ObjectQuery

suportam os métodos setFirstResult e setMaxResults que permitem que a consulta retorne um subconjunto dos resultados.

Valores de Primitiva de Retorno ao invés de Entidades

Com a API EntityManager Query, as entidades são retornadas como parâmetros de consulta. O mecanismo de consulta retorna atualmente as chaves para essas entidades no cliente. Quando o cliente se itera sobre essas entidades utilizando o Iterator do método getResultIterator, cada entidade é automaticamente aumentada e gerenciada, como se fosse criada com o método find na interface EntityManager. Todo o gráfico da entidade é construído a partir da entidade ObjectMap no cliente. Os atributos de valor da entidade e quaisquer entidades relacionadas são ansiosamente resolvidos.

Para evitar a construção do gráfico de alto custo, modifique a consulta para retornar os atributos individuais com navegação de caminho.

Por exemplo:

```
// Returns an entity
SELECT p FROM Person p
// Returns attributes SELECT p.name, p.address.street, p.address.city, p.gender FROM Person p
```

Plano de Consulta

Todas as consultas do eXtreme Scale possuem um plano de consulta. O plano descreve como o mecanismo de consulta interage com os ObjectMaps e índices. Exiba o plano de consulta para determinar se a cadeia de consulta ou índices estão sendo utilizados apropriadamente. O plano de consulta também pode ser usado para explorar as diferenças que as mudanças repentinas em uma sequência de consultas fazem na maneira que o eXtreme Scale executa uma consulta.

O plano de consulta pode ser visualizado de uma de duas maneiras:

- Métodos da API getPlan EntityManager Query ou ObjectQuery
- Rastreio de diagnóstico do ObjectGrid

Método getPlan

O método getPlan nas interfaces ObjectQuery e Query retorna uma String que descreve o plano de consulta. Esta cadeia pode ser exibida na saída padrão ou no log para exibir um plano de consulta.

Nota: Em um ambiente distribuído, o método getPlan não é executado junto ao servidor e não refletirá nenhum índice definido. Para visualizar o plano, utilize um agente para visualizar o plano no servidor.

Rastreio de Plano de Consulta

O plano de consulta pode ser exibido utilizando o rastreio do ObjectGrid. Para ativar o rastreio de plano de consulta, utilize a seguinte especificação de rastreio:

```
QueryEnginePlan=debug=enabled
```

Consulte “Coletando Rastreio” na página 504 para obter detalhes sobre como ativar o rastreio e localizar os arquivos de log de rastreio.

Exemplos de Plano de Consulta

O plano de consulta usa a palavra para indicar que a consulta está iterando por meio de uma coleção do ObjectMap ou por meio de uma coleção derivada, tal como: q2.getEmps(), q2.dept, ou uma coleção temporária retornada por um loop interno. Se a coleção for a partir de um ObjectMap, o plano de consulta mostrará se uma varredura sequencial (denotada por `por INDEX SCAN`), índice exclusivo ou não exclusivo, é usado. Planos de consulta utilizam uma cadeia de filtros para lista as expressões de consulta aplicadas a uma coleta.

Geralmente, um produto cartesiano não é utilizado na consulta do objeto. A consulta a seguir varre o mapa EmpBean inteiro no loop externo e varre o mapa DeptBean inteiro no loop interno:

```
SELECT e, d FROM EmpBean e, DeptBean d

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in DeptBean ObjectMap using INDEX SCAN
   returning new Tuple( q2,  q3  )
```

A consulta a seguir recupera todos os nomes de funcionários a partir de um departamento específico ao varrer sequencialmente o mapa EmpBean para obter um objeto employee. A partir do objeto employee, a consulta navega até seu objeto department e aplica o filtro d.no=1. Neste exemplo, cada funcionário possui apenas uma referência de objeto departamento, assim o loop interno é executado uma vez:

```
SELECT e.name FROM EmpBean e JOIN e.dept d WHERE d.no=1

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in  q2.dept
      filter ( q3.getNo() = 1 )
   returning new Tuple( q2.name  )
```

O exemplo a seguir é equivalente à consulta anterior. Entretanto, a consulta a seguir executa melhor porque ela primeiro limita o resultado para um objeto departamento ao usar o índice exclusivo que é definido sobre o número do campo de chave primária DeptBean. A partir do objeto department, a consulta navega para os seus objetos employee para obter seus nomes:

```
SELECT e.name FROM DeptBean d JOIN d.emps e WHERE d.no=1

Plan trace:

for q2 in DeptBean ObjectMap using UNIQUE INDEX key=(1)
     for q3 in  q2.getEmps()
   returning new Tuple( q3.name  )
```

A consulta a seguir localiza todos os funcionários que trabalham em desenvolvimento ou vendas. A consulta varre o mapa EmpBean inteiro e executa filtragem adicional avaliando as expressões: d.name = 'Sales' ou d.name='Dev'

```
SELECT e FROM EmpBean e, in (e.dept) d WHERE d.name = 'Sales' or d.name='Dev'

Plan trace:

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  )
```

A consulta a seguir é equivalente à consulta anterior, mas esta consulta executa um plano de consulta diferente e utiliza o índice de intervalo baseado no nome do campo. Em geral, esta consulta desempenha melhor porque o índice sobre o nome do campo é utilizado para restringir em objetos de departamentos, o que é executado mais rapidamente se apenas alguns departamentos forem de desenvolvimento ou vendas.

```
SELECT e FROM DeptBean d, in(d.emps) e WHERE d.name='Dev' or d.name='Sales'

Plan trace:

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()
```

A consulta a seguir localiza departamentos que não possuem nenhum funcionário:

```
SELECT d FROM DeptBean d WHERE NOT EXISTS(select e from d.emps e)

Plan trace:

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  )
```

A consulta a seguir é equivalente à consulta anterior, mas utiliza a função escalar SIZE. Esta consulta possui desempenho semelhante, mas é mais fácil de gravar.

```
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  )
```

O exemplo a seguir é uma outra forma de escrever a mesma consulta como a consulta anterior com desempenho semelhante, mas esta consulta também é mais fácil de escrever:

```
SELECT d FROM DeptBean d WHERE d.emps is EMPTY

Plan trace:

for q2 in DeptBean ObjectMap using INDEX SCAN
    filter ( q2.getEmps() IS EMPTY  )
   returning new Tuple( q2  )
```

A consulta a seguir localiza quaisquer funcionários com um endereço inicial correspondendo a pelo menos um dos endereços do funcionário cujo nome seja igual ao valor do parâmetro. O loop interno não possui nenhuma dependência do loop externo. A consulta é executada no loop interno uma 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  )
```

A consulta a seguir é equivalente à consulta anterior, mas possui uma subconsulta correlacionada; além disso, o loop interno é executado repetidamente.

```
SELECT e FROM EmpBean e WHERE EXISTS(SELECT e1 FROM EmpBean e1 WHERE e.home=e1.home and e1.name=?1)

Plan trace:

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  )
```

Otimização de Consulta Utilizando Índices

Definir e utilizar índices adequadamente pode aprimorar significativamente o desempenho da consulta.

As consultas do WebSphere eXtreme Scale podem usar plug-ins HashIndex integrados para aumentar o desempenho de consultas. Os índices podem ser definidos em atributos entity ou object. O mecanismo de consulta usará automaticamente os índices definidos se a sua cláusula WHERE usar uma das seguintes cadeias:

- Uma expressão de comparação com os seguintes operadores: =, <, >, <= ou >= (quaisquer expressões de comparação, exceto não iguais <>)
- Uma expressão BETWEEN
- Operandos das expressões são constantes ou termos simples

Requisitos

Os índices têm os seguintes requisitos quando usados pela Consulta:

- Todos os índices devem usar o plug-in HashIndex integrado.
- Todos os índices devem ser estaticamente definidos. Os índices dinâmicos não são suportados.
- A anotação @Index pode ser usada para criar automaticamente plug-ins HashIndex estáticos.
- Todos os índices de um único atributo devem ter o conjunto de propriedades RangeIndex configurado como true.
- Todos os índices compostos devem ter o conjunto de propriedades RangeIndex configurado como false.
- Todos os índices de associação (relacionamento) devem ter o conjunto de propriedades RangeIndex configurado como false.

Para obter informações sobre a configuração do HashIndex, consulte o “Plug-ins para Indexar Dados” na página 327.

Para obter informações sobre a indexação, consulte o “Indexação” na página 97.

Para obter uma maneira mais eficiente de procurar objetos armazenados em cache, consulte “Usando um Índice Composto” na página 336

Uso de Dicas para Escolher um Índice

Um índice pode ser manualmente selecionado usando o método setHint nas interfaces Query e ObjectQuery com a constante HINT_USEINDEX. Isto pode ser útil quando a otimização de uma consulta usar o melhor índice de desempenho.

Exemplos de consulta que usam índices de atributo

Os exemplos a seguir utilizam termos simples: e.empid, e.name, e.salary, d.name, d.budget e e.isManager. Os exemplos assumem que os índices são definidos sobre os campos name, salary e budget de um objeto entity ou value. O campo empid é uma chave primária e isManager não possui índice definido.

A consulta a seguir utiliza ambos os índices sobre o campos name e salary. Ela retorna todos os funcionários com nomes iguais ao valor do primeiro parâmetro ou um salário igual ao valor do segundo parâmetro:

```
SELECT e FROM EmpBean e where e.name=?1 or e.salary=?2
```

A consulta a seguir usa ambos índices sobre os campos de nome e orçamento. Ela retorna todos os departamentos nomeados 'DEV' com um orçamento que é maior que 2000.

```
SELECT d FROM DeptBean dwhere d.name='DEV' and d.budget>2000
```

A consulta a seguir retorna todos os funcionários com um salário maior do que 3000 e com um valor sinalizador isManager igual ao valor do parâmetro. A consulta utiliza o índice que é definido sobre o campo salary e executa filtragem adicional ao avaliar a expressão de comparação: `e.isManager=?1.`

```
SELECT e FROM EmpBean e where e.salary>3000 and e.isManager=?1
```

A consulta a seguir localiza todos os funcionários que ganham mais que o primeiro parâmetro ou que qualquer funcionário que é um gerente. Embora o campo salary tenha um índice definido, a consulta varre o índice integrado que é baseado em chaves primárias do campo EmpBean e avalia a expressão: `e.salary>?1` ou `e.isManager=TRUE`.

```
SELECT e FROM EmpBean e WHERE e.salary>?1 or e.isManager=TRUE
```

A consulta a seguir retorna funcionários com um nome que contém a letra a. Embora o campo name tenha um índice definido, a consulta não utiliza o índice porque o campo name é utilizado na expressão LIKE.

```
SELECT e FROM EmpBean e WHERE e.name LIKE '%a%'
```

A consulta a seguir localiza todos os funcionários com um nome que não seja "Smith". Embora o campo name tenha um índice definido, a consulta não utiliza o índice porque a consulta utiliza o operador de comparação não iguais (<>).

```
SELECT e FROM EmpBean e where e.name<>'Smith'
```

A seguinte consulta localiza todos os departamentos com um orçamento menor do que o valor do parâmetro e com um salário superior a 3000. A consulta utiliza um índice para o salário, mas não utiliza um índice para o orçamento porque

dept.budget não é um termo simples. Os objetos dept são derivados da coleta e. Não é necessário utilizar o índice de orçamento para consultar objetos dept.

```
SELECT dept from EmpBean e, in (e.dept) dept where e.salary>3000 and
dept.budget<?
```

A consulta a seguir localiza todos os funcionários com um salário maior do que o salário dos funcionários que possuem o empid e 1, 2 e 3. O salário do índice não é utilizado porque a comparação envolve uma subconsulta. O empid é uma chave primária, entretanto, ele é utilizado para uma procura de índice exclusiva porque todas as chaves primárias possuem um índice integrado 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 verificar se o índice está sendo utilizado pela consulta, é possível visualizar o "Plano de Consulta" na página 453. A seguir, está um plano de consulta de exemplo para a 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 Indexação

Os índices podem ser definidos sobre qualquer tipo de atributo único com os limitadores anteriormente definidos.

definição de índices de entidade usando @Index

Para definir um índice em uma entidade, simplesmente defina uma anotação:

Entidades usando anotações

```
  @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;
}
```

Com XML

Os índices também podem ser definidos usando XML:

Entidades sem anotações

```
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 do ObjectGrid com índices de atributos

```
<?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="Ranges are must be set to true for attributes." />
  </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="Ranges are must be set to true for attributes." />
  </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="Ranges are must be set to true for attributes." />
  </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="Ranges are must be set to true for attributes." />
  </bean>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
  </objectGridConfig>
```

Entidade XML

```
<?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>
```

Definição de índices para não-entidades usando XML

Os índices para tipos de não-entidade são definidos em XML. Não há diferença quando a criação do MapIndexPlugin para mapas de entidade e mapas de não-entidade.

Java bean

```
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 do ObjectGrid com índices de atributos

```
<?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="Ranges are must be set to true for attributes." />
  </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="Ranges are must be set to true for attributes." />
  </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="Ranges are must be set to true for attributes." />
</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="Ranges are must be set to true for attributes." />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Indexando Relacionamentos

O WebSphere eXtreme Scale armazena as chaves estrangeiras para entidades relacionadas dentro do objeto-pai. Para entidades, as chaves são armazenadas na tupla subjacente. Para objetos não-entidade, as chaves são explicitamente armazenadas no objeto-pai.

Incluir um índice em um atributo de relacionamento pode acelerar consultas que utilizam referências cíclicas ou utilizam os filtros de consulta IS NULL, IS EMPTY, SIZE e MEMBER OF. Ambas as associações únicas e com diversos valores podem ter a anotação @Index ou uma configuração de plug-in HashIndex em um arquivo XML descritor do ObjectGrid.

Definição de índices de relacionamento de entidade usando @Index

O exemplo a seguir define entidades com anotações @Index:

Entidade com anotação

```
@Entity
public class Node {
    @ManyToOne @Index
    Node parentNode;

    @OneToMany @Index
    List<Node> childrenNodes = new ArrayList();

    @OneToMany @Index
    List<BusinessUnitType> businessUnitTypes = new ArrayList();
}
```

Definição dos índices de relacionamento da entidade usando XML

O exemplo a seguir define as mesmas entidades e índices usando XML com plug-ins HashIndex:

Entidade sem anotações

```
public class Node {
int nodeId;
Node parentNode;
List<Node> childrenNodes = new ArrayList();
List<BusinessUnitType> businessUnitTypes = new ArrayList();
}
```

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="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="Ranges are not supported for association indexes." />  </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="Ranges are not supported for association indexes." />
 </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="Ranges are not supported for association indexes." />
  </bean>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
  </objectGridConfig>
```

Entidade XML

```
  <?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>
```

Usando os índices anteriormente definidos, os exemplos de consulta de entidades a seguir são otimizados:

```
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'
```

Definição dos índices de relacionamento de não-entidade

O exemplo a seguir define um plug-in HashIndex para mapas de não-entidade em um arquivo XML descritor do 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="Ranges are not supported for association indexes." />
      </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="Ranges are not supported for association indexes." />
   </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="Ranges are not supported for association indexes." />
</bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

Dadas as configurações de índice acima, o exemplos de consulta do objeto são otimizados:

```
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='TELE
```

Ajustando o Desempenho da Interface EntityManager

A interface EntityManager separa aplicativos do estado de suspensão no armazenamento de dados da grade do servidor.

O custo do uso da interface EntityManager não é alto e depende do tipo de trabalho sendo feito. Sempre use a interface EntityManager e otimize a lógica de negócios crucial após o aplicativo ser concluído. É possível retrabalhar qualquer código que utilize as interfaces EntityManager para usar mapas e tuplas. Geralmente, esse retrabalho de código pode ser necessário para 10% do código.

Se você usar relacionamentos entre objetos, o impacto no desempenho será menor, porque um aplicativo que usa mapas precisa gerenciar esses relacionamentos da mesma forma que a interface EntityManager.

Os aplicativos que usam a interface EntityManager não precisam fornecer uma implementação ObjectTransformer. Os aplicativos são otimizados automaticamente.

Retrabalhando o Código EntityManager para Mapas

Segue uma entidade de amostra:

```
@Entity
public class Person {
 @Id
```

```
 String ssn;
 String firstName;
 @Index
 String middleName;
 String surname;
}
```

Este é um trecho de código para localizar e atualizar a entidade:

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

A seguir, está o mesmo código utilizando Mapas e Tuplas:

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

// The Copy Mode is always NO_COPY for entity maps if not using COPY_TO_BYTES.
// Either we need to copy the tuple or we can ask the ObjectGrid to do it for us:
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();
```

Esses fragmentos de código têm o mesmo resultado, e um aplicativo pode utilizar um ou ambos os fragmentos.

O segundo fragmento de código mostra como utilizar mapas diretamente e como trabalhar com as tuplas (os pares de chave e valor). A tupla de valor tem três atributos: **firstName**, **middlename** e **surname**, indexado em 0, 1 e 2. A tupla de chave tem um único atributo e o número de ID é indexado em zero. É possível ver como as Tuplas são criadas utilizando os métodos EntityMetadata#getKeyMetaData ou EntityMetadata#getValueMetaData. Utilize esses métodos para criar as Tuplas para uma Entidade. Não é possível implementar a interface Tuple e passar uma instância em sua implementação de Tupla.

Tarefas relacionadas

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Referências relacionadas

"Agente de Instrumentação de Desempenho da Entidade"
É possível melhorar o desempenho de entidades de acesso a campo ativando o agente de instrumentação do WebSphere eXtreme Scale ao utilizar o Java Development Kit (JDK) Versão 1.5 ou posterior.

"Definindo um Esquema de Entidade" na página 169
Um ObjectGrid pode ter inúmeros esquemas de entidade lógicos. As entidades são definidas usando as classes Java anotadas, o XML ou uma combinação de classes XML e Java. Entidades definidas são registradas com um servidor eXtreme Scale e ligadas a BackingMaps, índices e outros plug-ins.

"Listeners de Entidade e Métodos de Retorno de Chamada" na página 185
Os aplicativos podem ser notificados quando o estado de uma entidade é alterado de estado para estado. Dois mecanismos de retorno de chamada existem para os eventos de alteração de estado: os métodos de retorno de chamada do ciclo de vida, que são definidos em uma classe de entidade e são chamados sempre que o estado da entidade for alterado, e os listeners de entidade que são mais genéricos porque o listener da entidade pode ser registrado em várias entidades.

"Exemplos do Listener de Entidade" na página 189
É possível gravar EntityListeners com base em seus requisitos. Veja a seguir vários scripts de exemplo.

"Interface EntityTransaction" na página 201
É possível utilizar a interface EntityTransaction para demarcar transações.

Agente de Instrumentação de Desempenho da Entidade

É possível melhorar o desempenho de entidades de acesso a campo ativando o agente de instrumentação do WebSphere eXtreme Scale ao utilizar o Java Development Kit (JDK) Versão 1.5 ou posterior.

Ativando o Agente do eXtreme Scale no JDK Versão 1.5 ou Acima

O agente ObjectGrid pode ser ativado com uma opção de linha de comandos Java com a seguinte sintaxe:

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

O valor *jarpath* é um caminho para um arquivo Java archive (JAR) do tempo de execução do eXtreme Scale que contém a classe do agente do eXtreme Scale e as classes de suporte como os arquivos `objectgrid.jar`, `wsobjectgrid.jar`, `ogclient.jar`, `wsogclient.jar` e `ogagent.jar`. Normalmente, em um programa Java ou em um ambiente Java Platform, Enterprise Edition independente que não executa o WebSphere Application Server, use o arquivo `objectgrid.jar` ou `ogclient.jar`. Em um ambiente do WebSphere Application Server ou de multicarregadores, é necessário usar o arquivo `ogagent.jar` na opção do agente da linha de comandos Java. Forneça o arquivo `ogagent.config` no caminho de classe ou use opções do agente para especificar informações adicionais.

Opções do Agente do eXtreme Scale

config
Substitui o nome do arquivo de configuração.

inclusão
Especifica ou substitui a definição de domínio de transformação que é a primeira parte do arquivo de configuração.

exclude
Especifica ou substitui a definição @Exclude.

fieldAccessEntity
Especifica ou substitui a definição @FieldAccessEntity.

trace Especifica um nível de rastreio. Os níveis podem ser ALL, CONFIG, FINE, FINER, FINEST, SEVERE, WARNING, INFO e OFF.

trace.file
Especifica o local do arquivo de rastreio.

O ponto e vírgula (**;**) é utilizado como um delimitador para separar cada opção. A vírgula (**,**) é utilizada como um delimitador para separar cada elemento em uma opção. O seguinte exemplo mostra a opção do agente eXtreme Scale para um programa Java:

```
-javaagent:objectgridRoot/lib/objectgrid.jar=config=myConfigFile;include=includedPackage;exclude=exc
```

Arquivo `ogagent.config`

O arquivo `ogagent.config` é o nome do arquivo de configuração do agente do eXtreme Scale designado. Se o nome do arquivo estiver no caminho de classe, o agente do eXtreme Scale localiza e analisa o arquivo. É possível substituir o nome do arquivo designado por meio da opção config do agente do eXtreme Scale. O exemplo a seguir mostra como especificar o arquivo de configuração:

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

Um arquivo de configuração do agente do eXtreme Scale possui as seguintes partes:

- **Domínio de transformação:** A parte do domínio de transformação é a primeira no arquivo de configuração. O domínio de transformação é uma lista de pacotes e classes que estão incluídos no processo de transformação da classe. Este domínio de transformação deve incluir todas as classes que são classes de entidade field-access e outras classes que fazem referência a estas classes de entidade field-access. As classes de entidade field-access e tais classes que fazem referência a estas classes de entidade field-access constroem o domínio de transformação. Se você planejar especificar classes de entidade field-access na parte @FieldAccessEntity, então não é necessário incluir classes de entidade field-access aqui. O domínio de transformação deve estar completo. Caso contrário, pode ser possível ver uma exceção FieldAccessEntityNotInstrumentedException.
- **@Exclude:** O token @Exclude indica que os pacotes e as classes listadas após este token são excluídos do domínio de transformação.
- **@FieldAccessEntity:** O token @FieldAccessEntity indica que os pacotes e as classes listados após este token são pacotes e classes Entity field-access. Se não existir nenhuma linha após o token @FieldAccessEntity, então, seu equivalente é "Nenhum @FieldAccessEntity especificado". O agente do eXtreme Scale determina que não há pacotes e classes Entity field-access definidos. Se houver linhas após o token @FieldAccessEntity, então elas representam os pacotes e as

classes de entidade de acesso a campos especificados pelo usuário. Por exemplo, "domínio de entidade de acesso a campos". O domínio de entidade de acesso a campos é um subdomínio do domínio de transformação. Os pacotes e as classes que estão listados no domínio de entidade field-access são uma parte do domínio de transformação, mesmo quando não estão listados no domínio de transformação. O token @Exclude, que lista pacotes e classes que são excluídos da transformação, não tem impacto no domínio Entity field-access. Quando o token @FieldAccessEntity é especificado, todas as entidades field-access devem estar este domínio Entity field-access. Caso contrário, uma exceção FieldAccessEntityNotInstrumentedException pode ocorrer.

Arquivo de Configuração do Agente de Exemplo (ogagent.config)

```
################################
# The # indicates comment line
################################
# This is an ObjectGrid agent config file (the designated file name is ogagent.config) that can be found and parsed by the ObjectGrid agent
# if it is in classpath.
# If the file name is "ogagent.config" and in classpath, Java program runs with -javaagent:objectgridRoot/ogagent.jar will have
# ObjectGrid agent enabled.
# If the file name is not "ogagent.config" but in classpath, you can specify the file name in config option of ObjectGrid agent
#     -javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
# See comments below for more info regarding instrumentation setting override.

# The first part of the configuration is the list of packages and classes that should be included in transformation domain.
# The includes (packages/classes, construct the instrumentation doamin) should be in the beginning of the file.
com.testpackage
com.testClass

# Transformation domain: The above lines are packages/classes that construct the transformation domain.
# The system will process classes with name starting with above packages/classes for transformation.
#
# @Exclude token : Exclude from transformation domain.
# The @Exclude token indicates packages/classes after that line should be excluded from transformation domain.
# It is used when user want to exclude some packages/classes from above specified included packages
#
# @FieldAccessEntity token: Field-access Entity domain.
# The @FieldAccessEntity token indicates packages/classes after that line are field-access Entity packages/classes.
# If there is no lilne after the @FieldAccessEntity token, it is equivalent to "No @FieldAccessEntity specified".
# The runtime will consider the user does not specify any field-access Entity packages/classes.
# The "field-acces Entity domain" is a sub-domain of transformation domain.
#
# Packages/classes listed in the "field-access Entity domain" will always be part of transformation domain,
# even they are not listed in transformation domain.
# The @Exclude, which lists packages/classes excluded from transformation, has no impact on the "field-acces Entity domain".
# Note: When @FieldAccessEntity is specified, all field-access entities must be in this field-acces Entity domain,
#       otherwise, FieldAccessEntityNotInstrumentedException may occur.
#
# The default ObjectGrid agent config file name is ogagent.config
# The runtime will look for this file as a resource in classpath and process it.
# Users can override this designated ObjectGrid agent config file name via config option of agent.
#
# e.g.
# javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
#
# The instrumentation definition, including transformation domain, @Exclude, and  @FieldAccessEntity can be overriden individually
# by corresponding designated agent options.
# Designated agent options include:
#    include              -> used to override instrumentation domain definition that is the first part of the config file
#    exclude              -> used to override @Exclude definition
#    fieldAccessEntity    -> used to override @FieldAccessEntity definition
#
# Each agent option should be separated by ";"
# Within the agent option, the package or class should be seperated by ","
#
# The following is an example that does not override the config file name:
#    -javaagent:objectgridRoot/lib/objectgrid.jar=include=includedPackage;exclude=excludedPackage;fieldAccessEntity=package1,package2
#
################################

@Exclude
com.excludedPackage
com.excludedClass

@FieldAccessEntity
```

Considerações sobre Desempenho

Para obter melhor desempenho, especifique o domínio de transformação e o domínio de entidade field-access.

Conceitos relacionados

"Ajustando o Desempenho da Interface EntityManager" na página 463
A interface EntityManager separa aplicativos do estado de suspensão no armazenamento de dados da grade do servidor.

"Objetos de Armazenamento em Cache e seus Relacionamentos (API EntityManager)" na página 166
A maioria dos produtos de cache utiliza APIs baseadas em mapa para armazenar dados como pares de chave-valor. A API ObjectMap e o cache dinâmico no WebSphere Application Server, entre outros, usam essa abordagem. Entretanto, APIs baseadas em mapas têm limitações. A API EntityManager simplifica a interação com a grade de dados ao fornecer uma maneira fácil de declarar e interagir com um gráfico complexo de objetos relacionados.

"Entity Manager em um Ambiente Distribuído" na página 177
É possível usar a API EntityManager com um ObjectGrid local ou em um ambiente distribuído do eXtreme Scale . A principal diferença é como você se conecta a esse ambiente remoto. Após você estabelecer uma conexão, não existe diferença entre o uso de um objeto Session ou uma API do EntityManager.

"Interagindo com EntityManager" na página 182
Geralmente os aplicativos primeiro obtêm uma referência do ObjectGrid e, depois, uma Sessão dessa referência para cada encadeamento. As sessões não podem ser compartilhadas entre encadeamentos. Um método extra em Session, o método getEntityManager, está disponível. Este método retorna uma referência para um gerenciador de entidades para uso para este encadeamento. A interface de EntityManager pode substituir as interfaces de Session e ObjectMap para todos os aplicativos. É possível utilizar essas APIs de EntityManager se o cliente tiver acesso às classes de entidade definidas.

"Suporte ao Plano de Carregamento do EntityManager" na página 191
Um FetchPlan é a estratégia que o gerenciador de entidade usa para recuperar objetos associados se o aplicativo precisar acessar relacionamentos.

"Filas de Consulta da Entidade" na página 196
Filas de consulte permitem que aplicativos criem uma fila qualificada por uma consulta no lado do servidor ou eXtreme Scale local sobre uma entidade. As entidades do resultado da consulta são armazenadas nesta fila. Atualmente, a fila de consulta é suportada apenas em um mapa que está utilizando a estratégia de bloqueio pessimista.

Tarefas relacionadas

"Tutorial: Armazenando Informações de Pedido nas Entidades" na página 7
Este tutorial do gerenciador de entidade mostra como utilizar o WebSphere eXtreme Scale para armazenar informações de pedido em um Web site. É possível criar um aplicativo Java Platform, Standard Edition 5 simples que utiliza um eXtreme Scale local e de memória. As entidades utilizam anotações e genéricos do Java SE 5.

Capítulo 7. Segurança

WebSphere eXtreme Scale pode proteger o acesso a dados, incluindo permissão para integração com provedores de segurança externos. Os aspectos de segurança incluem autenticação, autorização, segurança de transporte, segurança da grade de dados, segurança local e segurança JMX (MBean).

Configurando Perfis de Segurança para o Utilitário `xscmd`

Criando um perfil de segurança, é possível usar parâmetros de segurança salvos para usar o utilitário **xscmd** com ambientes seguros.

Antes de Iniciar

Para obter informações adicionais sobre como configurar o utilitário **xscmd**, consulte Administrando com o Utilitário **xscmd**.

Sobre Esta Tarefa

É possível usar o parâmetro **-ssp** *profile_name* ou **--saveSecProfile** *profile_name* com o restante de seu comando **xscmd**. Para salvar um perfil de segurança. O perfil pode conter configurações para nomes de usuário e senhas, geradores de credencial, keystores, armazenamentos confiáveis e tipos de transporte.

O do grupo de comandos **ProfileManagement** no utilitário **xscmd** contém comandos para gerenciar seus perfis de segurança.

Procedimento

- Salve um perfil de segurança.

  Para salvar um perfil de segurança, use o parâmetro **-ssp** *profile_name* ou **--saveSecProfile** *profile_name* com o restante de seu comando. A inclusão deste parâmetro em seu comando salva os parâmetros a seguir:

  ```
  -al,--alias <alias>
  -arc,--authRetryCount <integer>
  -ca,--credAuth <support>
  -cgc,--credGenClass <className>
  -cgp,--credGenProps <property>
  -cxpv,--contextProvider <provider>
  -ks,--keyStore <filePath>
  -ksp,--keyStorePassword <password>
  -kst,--keyStoreType <type>
  -prot,--protocol <protocol>
  -pwd,--password <password>
  -ts,--trustStore <filePath>
  -tsp,--trustStorePassword <password>
  -tst,--trustStoreType <type>
  -tt,--transportType <type>
  -user,--username <username>
  ```

  Os perfis de segurança são salvos no diretório *user_home*`\.xscmd\profiles\security\<`*profile_name*`>.properties`.

- Use um perfil de segurança salvo.

Para usar um perfil de segurança salvo, inclua o parâmetro **-sp** *profile_name* ou **--securityProfile** *profile_name* no comando que você está executando. Exemplo de comando: `xscmd -c listHosts -cep myhost.mycompany.com -sp myprofile`

- Liste os comandos no grupo de comandos do **ProfileManagement**.

  Execute o comando a seguir: **xscmd -lc ProfileManagement**.
- Liste os perfis de segurança existentes.

  Execute o comando a seguir: **xscmd -c listProfiles -v**.
- Exiba as configurações que são salvas em um perfil de segurança.

  Execute o comando a seguir: **xscmd -c showProfile -pn** *profile_name*.
- Remova um perfil de segurança existente.

  Execute o comando a seguir: **xscmd -c RemoveProfile -pn** *profile_name*.

Referências relacionadas

Ferramenta **xsadmin** para a Migração de Ferramenta **xscmd**
Em liberações anteriores, a ferramenta **xsadmin** é um utilitário de linha de comandos de amostra para monitorar o estado do ambiente. A ferramenta **xscmd** foi apresentada como uma ferramenta de linha de comandos administrativa e de monitoramento oficialmente suportada. Se a ferramenta **xsadmin** era usada anteriormente, considere migrar seus comandos para a nova ferramenta **xscmd**.

Programação para Segurança

Use as interfaces de programação para tratar vários aspectos da segurança em um ambiente do WebSphere eXtreme Scale.

API de Segurança

O WebSphere eXtreme Scale adota uma arquitetura de segurança aberta. Ela fornece uma estrutura de segurança básica para autenticação, autorização e segurança de transporte e requer que os usuários implementem plug-ins para completar a infraestrutura de segurança.

A seguinte imagem mostra o fluxo básico de autenticação e autorização do cliente para um servidor eXtreme Scale.

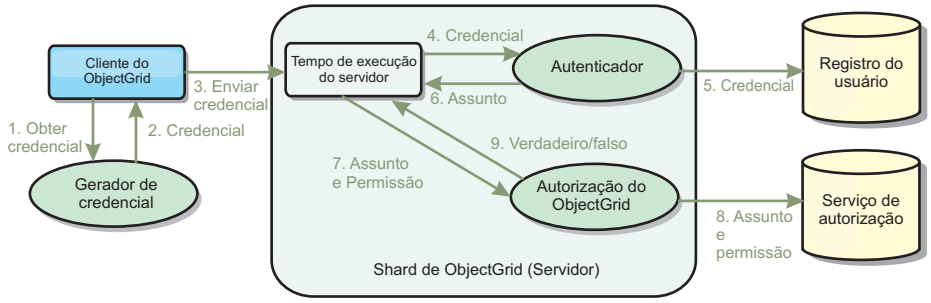


Figura 31. Fluxo de Autenticação e Autorização do Cliente

O fluxo de autenticação e o fluxo de autorização são os seguintes.

Fluxo de Autenticação

1. O fluxo de autenticação inicia com um cliente eXtreme Scale obtendo uma credencial. Isso é feito pelo plug-in com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator.

2. Um objeto CredentialGenerator sabe como gerar uma credencial de cliente válida, por exemplo, um par de ID de usuário e senha, ticket Kerberos, e assim por diante. Essa credencial gerada é enviada de volta para o cliente.
3. Depois que o cliente recuperar o objeto Credential usando o objeto CredentialGenerator, esse objeto Credential será enviado junto com o pedido eXtreme Scale para o servidor eXtreme Scale.
4. O servidor eXtreme Scale autentica o objeto Credential antes de processar o pedido do eXtreme Scale. Em seguida, o servidor utiliza o plug-in do Autenticador para autenticar o objeto Credential.
5. O plug-in do Autenticador representa uma interface com o registro do usuário, por exemplo, um servidor Lightweight Directory Access Protocol (LDAP) ou um registro do usuário do sistema operacional. O Autenticador consulta o registro do usuário e toma as decisões de autenticação.
6. Se a autenticação for bem sucedida, um objeto Subject será retornado para representar este cliente.

   Fluxo de Autorização

   O WebSphere eXtreme Scale adota um mecanismo de autorização baseado em permissão e possui categorias de permissão diferentes representadas por diferentes classes de permissão. Por exemplo, um objeto com.ibm.websphere.objectgrid.security.MapPermission representa permissões para ler, gravar, inserir, invalidar e remover as entradas de dados em um ObjectMap. Como o WebSphere eXtreme Scale suporta a autorização Java Authentication and Authorization Service (JAAS) disponível para uso imediato, é possível usar o JAAS para manipular autorização ao fornecer políticas de autorização.

   Além disso, o eXtreme Scale suporta autorizações customizadas. As autorizações customizadas são conectadas pelo plug-in com.ibm.websphere.objectgrid.security.plugins.ObjectGridAuthorization. O fluxo de autorização do cliente é o seguinte.
7. O tempo de execução do servidor envia o objeto Subject e a permissão necessária para o plug-in de autorização.
8. O plug-in de autorização consulta o serviço de Autorização e toma uma decisão de autorização. Se a permissão for concedida para esse objeto Subject, um valor `true` ou `false` será retornado.
9. Essa decisão de autorização, true ou false, é retornada para o tempo de execução do servidor.

Implementação de Segurança

Os tópicos nessa sessão discutem como programar uma implementação WebSphere eXtreme Scale segura e como programar as implementações de plug-in. A seção é organizada com base nos vários recursos de segurança. Em cada subtópico, você aprenderá sobre os plug-ins relevantes e como implementar os plug-ins. Na seção de autenticação, você saberá como se conectar a um ambiente de implementação seguro do WebSphere eXtreme Scale.

Autenticação do Cliente: O tópico de autenticação do cliente descreve como um cliente WebSphere eXtreme Scale obtém uma credencial e como um servidor autentica o cliente. Ele também discutirá como um cliente do WebSphere eXtreme Scale se conecta a um servidor WebSphere eXtreme Scale seguro.

Autorização: O tópico de autorização explica como usar o ObjectGridAuthorization para efetuar autorização do cliente além da autorização JAAS.

Autenticação de Grade: O tópico de autenticação de grade de dados descreve como o SecureTokenManager pode ser usado para transportar segredos do servidor com segurança.

Programação do Java Management Extensions (JMX): Quando o servidor WebSphere eXtreme Scale estiver protegido, o cliente JMX poderá precisar enviar uma credencial JMX com o servidor.

Programação de Autenticação de Cliente

Para autenticação, o WebSphere eXtreme Scale fornece um tempo de execução para enviar a credencial do cliente para o lado do servidor e, em seguida, chama o plug-in do autenticador para autenticar os usuários.

O WebSphere eXtreme Scale exige que você implemente os plug-ins a seguir para completar a autenticação.

- Credential: Uma Credential representa uma credencial de cliente, como um par de ID de usuário e senha.
- CredentialGenerator: Uma CredentialGenerator representa um factory de credenciais para gerar a credencial.
- Authenticator: Um Authenticator autentica a credencial do cliente e recupera as informações do cliente.

Plug-ins Credential e CredentialGenerator

Quando um cliente do eXtreme Scale se conecta a um servidor que exige autenticação, o cliente é obrigado a fornecer uma credencial do cliente. Uma credencial do cliente é representado por uma interface com.ibm.websphere.objectgrid.security.plugins.Credential. Uma credencial de cliente pode ser um par de nome de usuário e senha, um registro do Kerberos, um certificado cliente ou dados em qualquer formato concordado entre o cliente e o servidor. Esta interface explicitamente define os métodos equals(Object) e hashCode. Estes métodos são importantes porque os objetos Subject autenticados são armazenados em cache utilizando o objeto Credential como a chave no lado do servidor. O WebSphere eXtreme Scale também fornece um plug-in para gerar uma credencial. Este plug-in é representado pela interface com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator e é útil quando a credencial pode expirar. Neste caso, o método getCredential é chamado para renovar uma credencial.

A interface Credential explicitamente define os métodos equals(Object) e hashCode. Estes métodos são importantes porque os objetos Subject autenticados são armazenados em cache utilizando o objeto Credential como a chave no lado do servidor.

Você também pode usar o plug-in fornecido para gerar uma credencial. Este plug-in é representado pela interface com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator e é útil quando a credencial pode expirar. Neste caso, o método getCredential é chamado para renovar uma credencial. Consulte a documentação da API para obter mais detalhes.

Há três implementações padrão fornecidas para as interfaces da Credencial:

- A implementação da com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential, que contém um par de ID de usuário e senha.
- A implementação da com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential, que contém tokens de autenticação e autorização específicos do WebSphere Application Server. Estes tokens podem ser utilizados para propagar os atributos de segurança nos servidores de aplicativos no mesmo domínio de segurança.

O WebSphere eXtreme Scale também fornece um plug-in para gerar uma credencial. Este plug-in é representado pela interface com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator. O WebSphere eXtreme Scale fornece duas implementações integradas padrão:

- O construtor com.ibm.websphere.objectgrid.security.plugins.builtins. UserPasswordCredentialGenerator pega um ID de usuário e uma senha. Quando o método getCredential é chamado, retorna um objeto UserPasswordCredential, que contém o ID do usuário e a senha.
- O com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator representa um gerenciador de credenciais (token de segurança) ao executar no WebSphere Application Server. Quando o método getCredential é chamado, o Subject associado ao encadeamento atual é recuperado. Em seguida, as informações de segurança neste objeto Subject são convertidas em um objeto WSTokenCredential. É possível especificar se deseja recuperar um subject runAs ou um subject responsável pela chamada do encadeamento, utilizando a constante WSTokenCredentialGenerator.RUN_AS_SUBJECT ou WSTokenCredentialGenerator.CALLER_SUBJECT.

UserPasswordCredential e UserPasswordCredentialGenerator

Para propósitos de teste, o WebSphere eXtreme Scale fornece as seguintes implementações de plug-in:

1. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential`
2. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator`

A credencial de senha do usuário armazena um ID de usuário e uma senha. O gerador de credenciais de senha de usuário contém este ID de usuário e senha.

O código de exemplo a seguir mostra como implementar estes dois plug-ins.

UserPasswordCredential.java

```
// This sample program is provided AS IS and may be used, executed, copied and modified
// without royalty payment by customer
// (a) for its own instruction and study,
// (b) in order to develop applications designed to run with an IBM WebSphere product,
// either for customer's own internal use or for redistribution by customer, as part of such an
// application, in customer's own products.
// Licensed Materials - Property of 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;

/**
 * This class represents a credential containing a user ID and password.
 *
 * @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;

/**
 * Creates a UserPasswordCredential with the specified user name and
 * password.
 *
 * @param userName the user name for this credential
 * @param password the password for this credential
 *
 * @throws IllegalArgumentException if userName or password is <code>null</code>
 */
public UserPasswordCredential(String userName, String password) {
    super();
    if (userName == null || password == null) {
        throw new IllegalArgumentException("User name and password cannot be null.");
    }
    this.ivUserName = userName;
    this.ivPassword = password;
}

/**
 * Gets the user name for this credential.
 *
 * @return   the user name argument that was passed to the constructor
 *           or the <code>setUserName(String)</code>
 *           method of this class
 *
 * @see #setUserName(String)
 */
public String getUserName() {
    return ivUserName;
}

/**
 * Sets the user name for this credential.
 *
 * @param userName the user name to set.
 *
 * @throws IllegalArgumentException if userName is <code>null</code>
 */
public void setUserName(String userName) {
    if (userName == null) {
        throw new IllegalArgumentException("User name cannot be null.");
    }
    this.ivUserName = userName;
}

/**
 * Gets the password for this credential.
 *
 * @return   the password argument that was passed to the constructor
 *           or the <code>setPassword(String)</code>
 *           method of this class
 *
 * @see #setPassword(String)
 */
public String getPassword() {
    return ivPassword;
}

/**
 * Sets the password for this credential.
 *
 * @param password the password to set.
 *
 * @throws IllegalArgumentException if password is <code>null</code>
 */
public void setPassword(String password) {
    if (password == null) {
        throw new IllegalArgumentException("Password cannot be null.");
    }
    this.ivPassword = password;
}

/**
 * Checks two UserPasswordCredential objects for equality.
 * <p>
 * Two UserPasswordCredential objects are equal if and only if their user names
 * and passwords are equal.
 *
 * @param o the object we are testing for equality with this object.
 *
 * @return <code>true</code> if both UserPasswordCredential objects are equivalent.
 *
 * @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;
    }

    /**
     * Returns the hashcode of the UserPasswordCredential object.
     *
     * @return the hash code of this object
     *
     * @see Credential#hashCode()
     */
    public int hashCode ()  {
        return ivUserName.hashCode() + ivPassword.hashCode();
    }
}
```

UserPasswordCredentialGenerator.java

```
// This sample program is provided AS IS and may be used, executed, copied and modified
// without royalty payment by customer
// (a) for its own instruction and study,
// (b) in order to develop applications designed to run with an IBM WebSphere product,
// either for customer's own internal use or for redistribution by customer, as part of such an
// application, in customer's own products.
// Licensed Materials - Property of 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;

/**
 * This credential generator creates <code>UserPasswordCredential</code> objects.
 * <p>
 * UserPasswordCredentialGenerator has a one to one relationship with
 * UserPasswordCredential because it can only create a UserPasswordCredential
 * representing one identity.
 *
 * @since WAS XD 6.0.1
 * @ibm-api
 *
 * @see CredentialGenerator
 * @see UserPasswordCredential
 */
public class UserPasswordCredentialGenerator implements CredentialGenerator {

    private String ivUser;

    private String ivPwd;

    /**
     * Creates a UserPasswordCredentialGenerator with no user name or password.
     *
     * @see #setProperties(String)
     */
    public UserPasswordCredentialGenerator() {
        super();
    }

    /**
     * Creates a UserPasswordCredentialGenerator with a specified user name and
     * password
     *
     * @param user the user name
     * @param pwd the password
     */
    public UserPasswordCredentialGenerator(String user, String pwd) {
        ivUser = user;
        ivPwd = pwd;
    }

    /**
     * Creates a new <code>UserPasswordCredential</code> object using this
     * object's user name and password.
     *
     * @return a new <code>UserPasswordCredential</code> instance
     *
     * @see CredentialGenerator#getCredential()
     * @see UserPasswordCredential
     */
    public Credential getCredential() {
        return new UserPasswordCredential(ivUser, ivPwd);
    }
```

```
/**
 * Gets the password for this credential generator.
 *
 * @return   the password argument that was passed to the constructor
 */
public String getPassword() {
    return ivPwd;
}

/**
 * Gets the user name for this credential.
 *
 * @return   the user argument that was passed to the constructor
 *           of this class
 */
public String getUserName() {
    return ivUser;
}
/**
 * Sets additional properties namely a user name and password.
 *
 * @param properties a properties string  with a user name and
 *                   a password separated by a blank.
 *
 * @throws IllegalArgumentException if the format is not valid
 */
public void setProperties(String properties) {
    StringTokenizer token = new StringTokenizer(properties, " ");
    if (token.countTokens() != 2) {
        throw new IllegalArgumentException(
            "The properties should have a user name and password and separated by a blank.");
    }

    ivUser = token.nextToken();
    ivPwd = token.nextToken();
}
/**
 * Checks two UserPasswordCredentialGenerator objects for equality.
 * <p>
 * Two UserPasswordCredentialGenerator objects are equal if and only if
 * their user names and passwords are equal.
 *
 * @param obj the object we are testing for equality with this object.
 *
 * @return <code>true</code> if both UserPasswordCredentialGenerator objects
 *         are equivalent.
 */
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;
}

/**
 * Returns the hashcode of the UserPasswordCredentialGenerator object.
 *
 * @return the hash code of this object
 */
public int hashCode ()  {
```

```
        return ivUser.hashCode() + ivPwd.hashCode();
    }

}
```

A classe UserPasswordCredential contém dois atributos: nome de usuário e senha. O UserPasswordCredentialGenerator serve como uma factory que contém os objetos UserPasswordCredential.

WSTokenCredential e WSTokenCredentialGenerator

Quando os clientes e servidores do WebSphere eXtreme Scale são todos implementados no WebSphere Application Server, o aplicativo cliente pode usar estas duas implementações integradas quando as seguintes condições forem satisfeitas:

1. A segurança global do WebSphere Application Server estiver ativada.
2. Todos os clientes e servidores do WebSphere eXtreme Scale estão em execução no WebSphere Application Server Java Virtual Machines.
3. Os servidores de aplicativos estiverem no mesmo domínio de segurança.
4. O cliente já estiver autenticado no WebSphere Application Server.

Nesta situação, o cliente pode usar a classe com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator para gerenciar uma credencial. O servidor usa a classe de implementação WSAuthenticator para autenticar a credencial.

Este cenário aproveita as vantagens do fato de que o cliente do eXtreme Scale já ter sido autenticado. Como os servidores de aplicativos que possuem servidores estão no mesmo domínio de segurança que os servidores de aplicativos que hospedam os clientes, os tokens de segurança podem ser propagados do cliente para o servidor para que o mesmo registro de usuário não precise ser autenticado novamente.

Nota: Não assuma que um CredentialGenerator sempre gera a mesma credencial. Para uma credencial expirável e atualizável, o CredentialGenerator deve poder gerar a credencial válida mais recente para certificar-se de que a autenticação foi bem-sucedida. Um exemplo é usar o bilhete Kerberos como um objeto Credential. Quando o bilhete Kerberos é atualizado, o CredentialGenerator deve recuperar o bilhete atualizado quando o CredentialGenerator.getCredential é chamado.

Plug-in Authenticator

Após o cliente do eXtreme Scale recuperar o objeto Credential utilizando o objeto CredentialGenerator, este objeto Credential do cliente é enviado junto o pedido do cliente para o servidor eXtreme Scale. O servidor autentica o objeto de Credencial antes de processar a solicitação. Se o objeto Credential for autenticado com êxito, um objeto Subject será retornado para representar este cliente.

Este objeto Subject é então armazenado em cache e expira após seu tempo de vida alcançar o valor de tempo limite da sessão. O valor de tempo limite da sessão de login pode ser configurado utilizando a propriedade loginSessionExpirationTime no arquivo XML do cluster. Por exemplo, configurar `loginSessionExpirationTime="300"` fará com que o objeto Subject expire em 300 segundos.

Esse objeto Subject é, então, utilizado para autorizar o pedido, que é mostrado posteriormente. Um servidor eXtreme Scale utiliza o plug-in do Autenticador para autenticar o objeto Credential. Consulte as informações sobre o Autenticador na documentação da API para obter mais detalhes.

O plug-in Autenticador é onde o tempo de execução do eXtreme Scale autentica o objeto Credential a partir do registro do usuário do cliente como, por exemplo, um servidor protocolo LDAP (Lightweight Directory Access Protocol).

O WebSphere eXtreme Scale não fornece uma configuração de registro do usuário disponível imediatamente. A configuração e o gerenciamento do registro do usuário são deixados fora do WebSphere eXtreme Scale para simplificação e flexibilidade. Este plug-in implementa a conexão e a autenticação com o registro do usuário. Por exemplo, uma implementação do Autenticador extrai o ID do usuário e a senha da credencial, utiliza-os para conectar-se e validar em um servidor LDAP e cria um objeto Subject como resultado da autenticação. A implementação pode usar módulos de login JAAS. Um objeto Subject é retornado como resultado de autenticação.

Observe que este método cria duas exceções: InvalidCredentialException e ExpiredCredentialException. A exceção InvalidCredentialException indica que a credencial não é válida. A exceção ExpiredCredentialException indica que a credencial expirou. Se uma destas duas exceções resultar do método authenticate, as exceções serão enviadas de volta para o cliente. Porém, o tempo de execução do cliente manipula estas duas exceções diferentemente:

- Se o erro for uma exceção InvalidCredentialException, o tempo de execução do cliente exibe esta exceção. Seu aplicativo deve tratar a exceção. É possível corrigir o CredentialGenerator, por exemplo e, em seguida, tentar a operação novamente.
- Se o erro for uma exceção ExpiredCredentialException, e a quantidade de novas tentativas não for 0, o tempo de execução do cliente chama o método CredentialGenerator.getCredential novamente, e envia o novo objeto Credential para o servidor. Se a nova autenticação de credencial for bem-sucedida, o servidor processará o pedido. Se a nova autenticação da credencial falhar, a exceção será enviada de volta para o cliente. Se o número de novas tentativas atingir o valor suportado e o cliente permanece obtendo uma exceção ExpiredCredentialException, ocorre a exceção ExpiredCredentialException. O aplicativo deve tratar o erro.

A interface Authenticator oferece grande flexibilidade. É possível implementar a interface Authenticator de sua própria maneira específica. Por exemplo, é possível implementar essa interface para suportar dois registros do usuário diferentes.

O WebSphere eXtreme Scale oferece amostra de implementações de plug-in do autenticador. Exceto para o plug-in do autenticador do WebSphere Application Server, as outras implementações são apenas amostras para fins de teste.

KeyStoreLoginAuthenticator

Este exemplo usa uma implementação integrada do eXtreme Scale: KeyStoreLoginAuthenticator, que é para fins de teste e amostra (uma armazenamento de chave é um registro de usuário simples e não deve ser usado para um ambiente de produção). Novamente, a classe é exibida para demonstrar melhor como implementar um autenticador.

KeyStoreLoginAuthenticator.java

```
// This sample program is provided AS IS and may be used, executed, copied and modified
// without royalty payment by customer
// (a) for its own instruction and study,
// (b) in order to develop applications designed to run with an IBM WebSphere product,
// either for customer's own internal use or for redistribution by customer, as part of such an
// application, in customer's own products.
// Licensed Materials - Property of 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;

/**
 * This class is an implementation of the <code>Authenticator</code> interface
 * when a user name and password are used as a credential.
 * <p>
 * When user ID and password authentication is used, the credential passed to the
 * <code>authenticate(Credential)</code> method is a UserPasswordCredential object.
 * <p>
 * This implementation will use a <code>KeyStoreLoginModule</code> to authenticate
 * the user into the key store using the JAAS login module "KeyStoreLogin". The key
 * store can be configured as an option to the <code>KeyStoreLoginModule</code>
 * class. Please see the <code>KeyStoreLoginModule</code> class for more details
 * about how to set up the JAAS login configuration file.
 * <p>
 * This class is only for sample and quick testing purpose. Users should
 * write your own Authenticator implementation which can fit better into
 * the environment.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Authenticator
 * @see KeyStoreLoginModule
 * @see UserPasswordCredential
 */
public class KeyStoreLoginAuthenticator implements Authenticator {

    /**
     * Creates a new KeyStoreLoginAuthenticator.
     */
    public KeyStoreLoginAuthenticator() {
        super();
    }

    /**
     * Authenticates a <code>UserPasswordCredential</code>.
     * <p>
     * Uses the user name and password from the specified UserPasswordCredential
     * to login to the KeyStoreLoginModule named "KeyStoreLogin".
     *
     * @throws InvalidCredentialException if credential isn't a
     *         UserPasswordCredential or some error occurs during processing
     *         of the supplied UserPasswordCredential
     *
     * @throws ExpiredCredentialException if credential is expired.  This exception
     *         is not used by this implementation
     *
     * @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

```
// This sample program is provided AS IS and may be used, executed, copied and modified
// without royalty payment by customer
// (a) for its own instruction and study,
// (b) in order to develop applications designed to run with an IBM WebSphere product,
// either for customer's own internal use or for redistribution by customer, as part of such an
// application, in customer's own products.
// Licensed Materials - Property of 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;

/**
 * A KeyStoreLoginModule is keystore authentication login module based on
 * JAAS authentication.
 * <p>
 * A login configuration should provide an option "<code>keyStoreFile</code>" to
 * indicate where the keystore file is located. If the <code>keyStoreFile</code>
 * value contains a system property in the form, <code>${system.property}</code>,
 * it will be expanded to the value of the system property.
 * <p>
 * If an option "<code>keyStoreFile</code>" is not provided, the default keystore
 * file name is <code>"${java.home}${/}.keystore"</code>.
 * <p>
 * Here is a Login module configuration example:
 * <pre><code>
 *    KeyStoreLogin {
 *        com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule required
 *            keyStoreFile="${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();

    /**
     * Key store file property name
     */
    public static final String KEY_STORE_FILE_PROPERTY_NAME = "keyStoreFile";

    /**
     * Key store type. Only JKS is supported
     */
```

```
public static final String KEYSTORE_TYPE = "JKS";

/**
 * The default key store file name
 */
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;

/**
 * Creates a new KeyStoreLoginModule.
 */
public KeyStoreLoginModule() {
}

/**
 * Initializes the login module.
 *
 * @see LoginModule#initialize(Subject, CallbackHandler, Map, Map)
 */
public void initialize(Subject sub, CallbackHandler callbackHandler,
        Map mapSharedState, Map mapOptions) {

    // initialize any configured options
    debug = "true".equalsIgnoreCase((String) mapOptions.get("debug"));

    if (sub == null)
        throw new IllegalArgumentException("Subject is not specified");

    if (callbackHandler == null)
        throw new IllegalArgumentException(
        "CallbackHander is not specified");

    // Get the key store path
    String sKeyStorePath = (String) mapOptions
        .get(KEY_STORE_FILE_PROPERTY_NAME);

    // If there is no key store path, the default one is the .keystore
    // file in the java home directory
    if (sKeyStorePath == null) {
        sKeyStorePath = DEFAULT_KEY_STORE_FILE;
    }

    // Replace the system enviroment variable
    sKeyStorePath = ObjectGridUtil.replaceVar(sKeyStorePath);

    File fileKeyStore = new File(sKeyStorePath);

    try {
        KeyStore store = KeyStore.getInstance("JKS");
        store.load(new FileInputStream(fileKeyStore), null);

        // Save the key store
        keyStore = store;

        if (debug) {
            System.out.println("[KeyStoreLoginModule] initialize: Successfully loaded key store");
        }
    }
    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;
}

/**
 * Authenticates a user based on the keystore file.
 *
 * @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) {
        // treat a NULL password as an empty password
        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);
    }

    // Validate the user name and password
    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;
}

/**
 * Indicates the user is accepted.
 * <p>
 * This method is called only if the user is authenticated by all modules in
 * the login configuration file. The principal objects will be added to the
 * stored subject.
 *
 * @return false if for some reason the principals cannot be added; true
 *         otherwise
 *
 * @exception LoginException
 *                LoginException is thrown if the subject is readonly or if
 *                any unrecoverable exceptions is encountered.
 *
 * @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;
}

/**
 * Indicates the user is not accepted
 *
 * @see LoginModule#abort()
 */
public boolean abort() throws LoginException {
    boolean b = logout();
    return b;
}

/**
 * Logs the user out. Clear all the maps.
 *
 * @see LoginModule#logout()
 */
public boolean logout() throws LoginException {


    // Clear the instance variables
    principals.clear();
    publicCreds.clear();
    privateCreds.clear();

    // clear maps in the subject
    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;
}

/**
 * Validates the user name and password based on the keystore.
 *
 * @param userName user name
 * @param password password
 * @throws SecurityException if any exceptions encountered
 */
private void validate(String userName, char password[])
    throws SecurityException {

    PrivateKey privateKey = null;

    // Get the private key from the keystore
    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);
    }

    // Check the certificats
    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) {

            // If the first certificate is an X509Certificate
            if (certs[0] instanceof X509Certificate) {
                try {
                    // Get the first certificate which represents the user
                    X509Certificate certX509 = (X509Certificate) certs[0];

                    // Create a principal
                    X500Principal principal = new X500Principal(certX509
                            .getIssuerDN()
                            .getName());
                    principals.add(principal);

                    if (debug) {
                        System.out.println("  Principal added: " + principal);
                    }
                    // Create the certification path object and add it to the
                    // public credential set
                    CertificateFactory factory = CertificateFactory
                        .getInstance("X.509");
                    java.security.cert.CertPath certPath = factory
                        .generateCertPath(Arrays.asList(certs));
                    publicCreds.add(certPath);

                    // Add the private credential to the private credential set
                    privateCreds.add(new X500PrivateCredential(certX509,
                            privateKey, userName));

                }
                catch (CertificateException ce) {
                    SecurityException se = new SecurityException();
                    se.initCause(ce);
                    throw se;
                }
            }
            else {
                // The first certificate is not an X509Certificate
                // We just add the certificate to the public credential set
                // and the private key to the private credential set.
                publicCreds.add(certs[0]);
                privateCreds.add(privateKey);
            }
        }
    }
}
```

Uso do Plug-in do Autenticador LDAP

Você recebe a implementação padrão do com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator para manipular a autenticação de nome do usuário e senha para um servidor LDAP. Esta implementação usa o módulo de login LDAPLogin para efetuar login do usuário em um servidor LDAP (Lightweight Directory Access Protocol). O fragmento a seguir demonstra como o método authenticate é implementado:

```
/**
* @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);
        }
    }
```

Além disso, o eXtreme Scale envia um módulo de login com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule para esta finalidade. Você deve fornecer as duas opções a seguir no arquivo de configuração de login JAAS.

- providerURL: A URL do provedor do servidor LDAP
- factoryClass: A classe de implementação do depósito de informações do contexto do LDAP

O módulo LDAPLoginModule chama o método com.ibm.websphere.objectgrid.security.plugins.builtins. LDAPAuthenticationHelper.authenticate. O trecho de código a seguir mostra como é possível implementar o método authenticate do LDAPAuthenticationHelper.

```
/**
* Authenticate the user to the LDAP directory.
* @param user the user ID, e.g., uid=xxxxxx,c=us,ou=bluepages,o=ibm.com
* @param pwd the password
*
* @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);

    // Look up for the user
    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("");

    // Check the 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;
    }
}
```

Se a autenticação for bem-sucedida, o ID e a senha serão considerados válidos. Então o módulo de login obtém as informações de ID e informações do departamento a partir deste método authenticate. O módulo de login cria dois proprietários: SimpleUserPrincipal e SimpleDeptPrincipal. É possível utilizar o subject autenticado para autorização de grupo (neste caso, o departamento é um grupo) e para autorização individual.

O exemplo a seguir mostra uma configuração de módulo de login utilizado para efetuar login no servidor LDAP:

```
LDAPLogin { com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule required
    providerURL="ldap://directory.acme.com:389/"
    factoryClass="com.sun.jndi.ldap.LdapCtxFactory";
};
```

Na configuração anterior, o servidor LDAP aponta para `ldap://directory.acme.com:389/server`. Altere esta configuração para seu servidor LDAP. Este módulo de login usa o ID e a senha fornecidos para se conectar ao servidor LDAP. Esta implementação serve apenas para fins de teste.

Utilização do Plug-in Autenticador do WebSphere Application Server

Além disso, o eXtreme Scale fornece a implementação integrada do com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator para usar a infraestrutura de segurança do WebSphere Application Server. Esta implementação integrada pode ser usada quando as seguintes condições forem verdadeiras.

1. A segurança global do WebSphere Application Server estiver ativada.
2. Todos os clientes e servidores eXtreme Scale tiverem sido ativados nos JVMs do WebSphere Application Server.
3. Estes servidores de aplicativos estão no mesmo domínio de segurança.
4. O cliente do eXtreme Scale já estiver autenticado no WebSphere Application Server.

O cliente pode usar a classe com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator para gerar uma credencial. O servidor usa esta classe de implementação Authenticator para autenticar a credencial. Se o token for autenticado com êxito, será retornado um objeto Subject.

Este cenário tira vantagens do fato de o cliente já ter sido autenticado. Como os servidores de aplicativos que possuem servidores estão no mesmo domínio de segurança que os servidores de aplicativos que hospedam os clientes, os tokens de segurança podem ser propagados do cliente para o servidor para que o mesmo registro de usuário não precise ser autenticado novamente.

Utilização do Plug-in Autenticador do Tivoli Access Manager

O Tivoli Access Manager é amplamente utilizado como um servidor de segurança. Também é possível implementar o Authenticator usando os módulos de login fornecidos pelo Tivoli Access Manager.

Para autenticar um usuário para o Tivoli Access Manager, aplique o módulo de login com.tivoli.mts.PDLoginModule, o que exige que o aplicativo que efetua a chamada forneça as seguintes informações:

1. Um nome de proprietário, especificado como um nome abreviado ou um nome X.500 (DN)
2. Uma senha

O módulo de login autentica o shard primário e retorna a credencial do Tivoli Access Manager. O módulo espera que o aplicativo de chamada forneça as seguintes informações:

1. O nome de usuário, por meio de um objeto javax.security.auth.callback.NameCallback.
2. A senha, por meio de um objeto javax.security.auth.callback.PasswordCallback.

Quando a credencial do Tivoli Access Manager é recuperada com êxito, o JAAS LoginModule cria um Subject e um PDPrincipal. Nenhum módulo integrado para autenticação do Tivoli Access Manager é fornecido pois ele está apenas com o módulo PDLoginModule. Consulte a Autorização do IBM Tivoli Access Manager Java Classes Developer Reference para obter mais detalhes.

Conexão Segura com o WebSphere eXtreme Scale

Para se conectar um cliente do eXtreme Scale seguramente a um servidor, é possível usar qualquer método de conexão na interface ObjectGridManager que usa um objeto do ClientSecurityConfiguration. A seguir a um breve exemplo.

```
public ClientClusterContext connect(String catalogServerAddresses,
   ClientSecurityConfiguration securityProps,
     URL overRideObjectGridXml) throws ConnectException;
```

Este método usa um parâmetro do tipo ClientSecurityConfiguration, que é uma interface representando uma configuração de segurança do cliente. É possível usar a API pública do com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory para criar uma instância com valores padrão, ou é possível criar uma instância por meio da transmissão do arquivo de propriedades do cliente do WebSphere eXtreme Scale. Este arquivo contém as seguintes propriedades que estão relacionadas à autenticação. O valor marcado com um sinal de mais (+) é o padrão.

- `securityEnabled (true, false+):` Esta propriedade indica se a segurança está ativada. Quando um cliente se conecta a um servidor, os valores de securityEnabled no lado do cliente e do servidor devem ser ambos `true` ou ambos `false`. Por exemplo, se a segurança do servidor conectado estiver ativada, o cliente terá que configurar esta propriedade como true para conectar-se ao servidor.
- authenticationRetryCount (um valor de número inteiro, 0+): Esta propriedade determina quantas novas tentativas devem ser feitas para efetuar login quando uma credencial tiver expirado. Se o valor for 0, nenhuma nova tentativa será feita. A nova tentativa de autenticação se aplicará apenas ao caso em que a credencial tiver expirado. Se a credencial não for válida, não haverá nova tentativa. Seu aplicativo é responsável por tentar novamente a operação.

Após criar um objeto com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration, configure o objeto credentialGenerator no cliente utilizando o seguinte método:

```
/**
* Configure o objeto {@link CredentialGenerator} para este cliente.
* @param generator o objeto CredentialGenerator associado a este cliente
*/
void setCredentialGenerator(CredentialGenerator generator);
```

É possível configurar o objeto CredentialGenerator no arquivo de propriedades do cliente do WebSphere eXtreme Scale também, da seguinte forma.

- credentialGeneratorClass: O nome da implementação da classe para o objeto CredentialGenerator. Ele deve ter um construtor padrão.
- credentialGeneratorProps: As propriedades para a classe CredentialGenerator. Se o valor não for nulo, ele será configurado como o objeto CredentialGenerator construído utilizando o método setProperties(String).

A seguir está uma amostra para instanciar um a ClientSecurityConfiguration e, em seguida, utilizá-lo para conectar-se ao servidor.

```
/**
* Obter um ClientClusterContext seguro
* @return um 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);
}
```

Quando connect é chamado, o cliente do WebSphere eXtreme Scale chama o método CredentialGenerator.getCredential para obter a credencial do cliente. Esta credencial é enviada junto com o pedido de conexão com o servidor para autenticação.

Utilização de uma instância do CredentialGenerator diferente por sessão

Em alguns casos, um cliente do WebSphere eXtreme Scale representa apenas uma identidade do cliente mas, em outros, ele pode representar múltiplas identidades. Aqui há um cenário para o caso mais recente: Um cliente do WebSphere eXtreme Scale é criado e compartilhado em um servidor da web. Todos os servlets neste servidor da web usam este cliente do WebSphere eXtreme Scale. Como cada servlet representa um web client diferente, use credenciais diferentes ao enviar solicitações aos servidores do WebSphere eXtreme Scale.

O WebSphere eXtreme Scale permite a mudança da credencial no nível de sessão. Cada sessão pode usar um objeto CredentialGenerator diferente. Assim, os cenários anteriores podem ser implementados deixando o servlet obter uma sessão com um objeto CredentialGenerator diferente. O exemplo a seguir ilustra o ObjectGrid.getSession(CredentialGenerator)method na interface ObjectGridManager.

```
/**
     * Get a session using a <code>CredentialGenerator</code>.
     * <p>
     * This method can only be called by the ObjectGrid client in an ObjectGrid
     * client server environment. If ObjectGrid is used in a local model, that is,
     * within the same JVM with no client or server existing, <code>getSession(Subject)</code>
     * or the <code>SubjectSource</code> plugin should be used to secure the ObjectGrid.
```

```
     *
     * <p>If the <code>initialize()</code> method has not been invoked prior to
     * the first <code>getSession</code> invocation, an implicit initialization
     * will occur.  This ensures that all of the configuration is complete
     * before any runtime usage is required.</p>
     *
     * @param credGen A <code>CredentialGenerator</code> for generating a credential
     *                for the session returned.
     *
     * @return An instance of <code>Session</code>
     *
     * @throws ObjectGridException if an error occurs during processing
     * @throws TransactionCallbackException if the <code>TransactionCallback</code>
     *         throws an exception
     * @throws IllegalStateException if this method is called after the
     *         <code>destroy()</code> method is called.
     *
     * @see #destroy()
     * @see #initialize()
     * @see CredentialGenerator
     * @see Session
     * @since WAS XD 6.0.1
 */
Session getSession(CredentialGenerator credGen) throws
ObjectGridException, TransactionCallbackException;
```

A seguir está um exemplo:

```
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();

CredentialGenerator credGenManager = new UserPasswordCredentialGenerator("manager", "xxxxxx");
CredentialGenerator credGenEmployee = new UserPasswordCredentialGenerator("employee", "xxxxxx");

ObjectGrid og = ogManager.getObjectGrid(ctx, "accounting");

// Get a session with CredentialGenerator;
Session session = og.getSession(credGenManager );

// Get the employee map
ObjectMap om = session.getMap("employee");

// start a transaction.
session.begin();

Object rec1 = map.get("xxxxxx");

session.commit();

// Get another session with a different CredentialGenerator;
session = og.getSession(credGenEmployee );

// Get the employee map
om = session.getMap("employee");

// start a transaction.
session.begin();

Object rec2 = map.get("xxxxx");

session.commit();
```

Se você usar o método ObjectGird.getSession para obter um objeto de Sessão, a sessão usa o conjunto de objetos CredentialGenerator no objeto ClientConfigurationSecurity. O método ObjectGrid.getSession(CredentialGenerator) substitui a configuração CredentialGenerator no objeto ClientSecurityConfiguration.

Se for possível reutilizar o objeto de Sessão, isto resulta em ganho de desempenho. Porém, a chamada do método ObjectGrid.getSession(CredentialGenerator) não é muito cara. A principal sobrecarga é o tempo maior de coleta de lixo do objeto. Certifique-se de liberar as referências quando tiver concluído os objetos Session. Geralmente, se o seu objeto de Sessão puder compartilhar a identidade, tente reutilizar o objeto de Sessão. Se não, utilize o método ObjectGrid.getSession(CredentialGenerator).

Informações relacionadas

API de Credencial

Programação de Autorização de Cliente

O WebSphere eXtreme Scale suporta a autorização Java Authentication and Authorization Service (JAAS) que está pronta para uso e também suporta a autorização customizada usando a interface ObjectGridAuthorization.

O plug-in ObjectGridAuthorization é usado para autorizar acessos do ObjectGrid, do ObjectMap e do JavaMap aos Principals representados por um objeto Subject de uma maneira customizada. Uma implementação típica deste plug-in é recuperar os Principals do objeto Subject e, em seguida, verificar se as permissões especificadas foram concedidas aos Principals.

Uma permissão passada para o método checkPermission(Subject, Permission) pode ser uma das seguintes permissões:

- MapPermission
- ObjectGridPermission
- ServerMapPermission
- AgentPermission

Consulte a documentação da API do ObjectGridAuthorization para obter mais detalhes.

MapPermission

A classe pública com.ibm.websphere.objectgrid.security.MapPermission representa permissões para os recursos ObjectGrid, especificamente os métodos de interfaces theObjectMap ou JavaMap. O WebSphere eXtreme Scale define as seguintes cadeias de permissões para acesso aos métodos de ObjectMap e JavaMap:

- **read**: Permissão para ler os dados do mapa. A constante de número inteiro é definida como `MapPermission.READ`.
- **write**: Permissão para atualizar os dados no mapa. A constante de número inteiro é definida como `MapPermission.WRITE`.
- **insert**: Permissão para inserir os dados no mapa. A constante de número inteiro é definida como `MapPermission.INSERT`.
- **remove**: Permissão para remover os dados do mapa. A constante de número inteiro é definida como `MapPermission.REMOVE`.
- **invalidate**: Permissão para invalidar os dados a partir do mapa. A constante de número inteiro é definida como `MapPermission.INVALIDATE`.
- **all**: Todas as permissões acima: read, write, insert, remote e invalidate. A constante de número inteiro é definida como `MapPermission.ALL`.

Consulte a documentação da API da ServerMapPermission para obter mais detalhes.

É possível construir um objeto MapPermission transmitindo o nome completo do mapa do ObjectGrid (no formato [ObjectGrid_name].[ObjectMap_name]) e a cadeia de permissão ou valor inteiro. Uma cadeia de permissão pode ser uma cadeia delimitada por vírgulas das cadeias de permissão anteriores, tais como read, insert, ou podem ser todas. Um valor de número inteiro de permissão pode ser qualquer constante de número inteiro mencionada anteriormente ou um valor matemático

de diversas constantes de permissão de número inteiro, tal como MapPermission.READ | MapPermission.WRITE.

A autorização ocorre quando um método ObjectMap ou JavaMap é chamado. O tempo de execução eXtreme Scale verifica diferentes permissões para métodos diferentes. Se as permissões requeridas não forem concedidas ao cliente, isso resultará em um AccessControlException.

Tabela 11. Lista de Métodos e a MapPermission Necessária

Permissão	**ObjectMap/JavaMap**
read	Boolean containsKey(Object)
	Boolean equals(Object)
	Object get(Object)
	Object get(Object, Serializable)
	List getAll(List)
	List getAll(List keyList, Serializable)
	List getAllForUpdate(List)
	List getAllForUpdate(List, Serializable)
	Object getForUpdate(Object)
	Object getForUpdate(Object, Serializable)
	public Object getNextKey(long)
write	Object put(Object key, Object value)
	void put(Object, Object, Serializable)
	void putAll(Map)
	void putAll(Map, Serializable)
	void update(Object, Object)
	void update(Object, Object, Serializable)
insert	public void insert (Object, Object)
	void insert(Object, Object, Serializable)
remove	Object remove (Object)
	void removeAll(Collection)
	void clear()
invalidate	public void invalidate (Object, Boolean)
	void invalidateAll(Collection, Boolean)
	void invalidateUsingKeyword(Serializable)
	int setTimeToLive(int)

A autorização é baseada exclusivamente em qual método é utilizado, ao invés do que o método realmente faz. Por exemplo, um método put pode inserir ou atualizar um registro, dependendo de existir ou não o registro. Entretanto, os casos de inserir ou atualizar não estão discriminados.

Um tipo de operação pode ser obtido por combinações de outros tipos. Por exemplo, uma atualização pode ser obtida por uma remoção e, em seguida, por uma inserção. Considere essas combinações quando projetar suas políticas de autorização.

ObjectGridPermission

Uma com.ibm.websphere.objectgrid.security.ObjectGridPermission representa permissões para o ObjectGrid:

- Query: permissão para criar uma consulta de objeto ou consulta de entidade. A constante de número inteiro é definida como ObjectGridPermission.QUERY.
- Dynamic map: permissão para criar um mapa dinâmico baseado no modelo de mapa. A constante de número inteiro é definida como ObjectGridPermission.DYNAMIC_MAP.

Consulte a documentação da API do ObjectGridAuthorization para obter mais detalhes.

A tabela a seguir resume os métodos e ObjectGridPermission necessários:

Tabela 12. Lista de Métodos e a ObjectGridPermission Necessária

Ação da permissão	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

Uma ServerMapPermission representa permissões para um ObjectMap hospedado em um servidor. O nome da permissão é o nome completo do nome do mapa do ObjectGrid. Executar as seguintes ações:

- 1. replicate: permissão para replicar um mapa de servidor no cache local.
- 2. dynamicIndex: permissão para um cliente criar ou remover um índice dinâmico em um servidor

Consulte a documentação da API da ServerMapPermission para obter mais detalhes. Os métodos detalhados, que requerem ServerMapPermission diferente, estão relacionados na seguinte tabela:

Tabela 13. Permissões para um ObjectMap Hospedado por Servidor

Ação da permissão	Métodos
replicate	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

Uma AgentPermission representa as permissões para os agentes datagrid. O nome da permissão é o nome completo do mapa ObjectGrid, e a ação é uma cadeia limitada por vírgulas de nomes de classe de implementação do agente ou de nomes do pacote.

Consulte a documentação da API AgentPermission para obter informações adicionais.

Os métodos a seguir na classe com.ibm.websphere.objectgrid.datagrid.AgentManager exigem 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 Autorização

O WebSphere eXtreme Scale suporta dois tipos de mecanismos de autorização: Autorização JAAS (Java Authentication and Authorization Service) e autorização customizada. Esses mecanismos aplicam-se a todas as autorizações. A autorização JAAS muda as políticas de segurança Java com controles de acesso centrados no usuário. As permissões podem ser concedidas com base não apenas em qual código está a execução, mas também em quem a está executando. Autorização JAAS é parte do SDK Versão 5 e posterior.

Além disso, o WebSphere eXtreme Scale também suporta a autorização customizada com o seguinte plug-in:

- ObjectGridAuthorization: maneira customizada para autorizar o acesso a todos os artefatos.

É possível implementar seu próprio mecanismo de autorização, se não desejar utilizar a autorização JAAS. Usando um mecanismo de autorização customizado é possível usar o banco de dados de políticas, servidor de políticas ou Tivoli Access Manager para gerenciar as autorizações.

É possível configurar o mecanismo de autorização de duas maneiras:

- Configuração XML

  1. *Configuração XML:* É possível utilizar o arquivo XML do ObjectGrid para definir um ObjectGrid e configurar o mecanismo de autorização como AUTHORIZATION_MECHANISM_JAAS ou AUTHORIZATION_MECHANISM_CUSTOM. A seguir está o arquivo secure-objectgrid-definition.xml que é utilizado no ObjectGridSample do aplicativo corporativo:

  ```
  <objectGrids>
   <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
    <bean id="TransactionCallback"
  classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- Configuração Programática

  2. *Configuração Programática:* Se desejar criar um ObjectGrid utilizando o método ObjectGrid.setAuthorizationMechanism(int), é possível chamar o seguinte método para configurar o mecanismo de autorização. A chamada deste método aplica-se somente ao modelo de programação local do WebSphere eXtreme Scale quando você instancia diretamente a instância do ObjectGrid:

  ```
  /**
   * Set the authorization Mechanism. The default is
   * com.ibm.websphere.objectgrid.security.SecurityConstants.
   * AUTHORIZATION_MECHANISM_JAAS.
   * @param authMechanism the map authorization mechanism
  */
  void setAuthorizationMechanism(int authMechanism);
  ```

Autorização JAAS

Um objeto javax.security.auth.Subject representa um usuário autenticado. Um Subject consiste em um conjunto de principals e cada Principal representa uma

identidade para esse usuário. Por exemplo, um Subject pode ter um nome principal, por exemplo, Joe Smith, e um grupo principal, por exemplo, gerente.

Utilizando a política de autorização JAAS, as permissões podem ser concedidas a Principals específicos. O WebSphere eXtreme Scale associa o Subject ao contexto de controle de acesso atual. Para cada chamada para o método ObjectMap ou Javamap, o tempo de execução do Java automaticamente determina se a política concede a permissão necessária somente a um Principal específico e se for, a operação será permitida somente se o Subject associado ao contexto do controle de acesso contiver o Principal designado.

É necessário estar familiarizado com a sintaxe de política do arquivo de políticas. Para obter uma descrição detalhada da autorização do JAAS, consulte o Guia de Referência do JAAS.

O WebSphere eXtreme Scale possui uma base de código especial que é usada para verificação da autorização JAAS para as chamadas de método ObjectMap e JavaMap. Esta base de código especial é `http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction`. Utilize esta base de código ao conceder as permissões ObjectMap ou JavaMap a proprietários. Este código especial foi criado porque o arquivo JAR (Java Archive) para eXtreme Scale é concedido com todas as permissões.

O modelo da política para conceder a permissão MapPermission é:

```
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";
  };
```

Um campo de Principal é semelhante ao seguinte exemplo:

```
principal
Principal_class "principal_name"
```

Nesta política, somente as permissões de inserção e leitura são concedidas a esses quatro mapas para um determinado principal. O outro arquivo de políticas, `fullAccessAuth.policy`, concede todas as permissões para estes mapas a um principal. Antes de executar o aplicativo, altere o principal_name e a classe do proprietário para os valores apropriados. O valor de principal_name depende do registro do usuário. Por exemplo, se o S.O. local for usado como registro do usuário, o nome da máquina será MACH1, o ID do usuário será user1 e o principal_name será `MACH1/user1`.

A política de autorização JAAS pode ser colocada diretamente no arquivo de políticas Java, ou pode ser colocada em um arquivo de autorização JAAS separado e, em seguida, configurado usando

- Use o seguinte argumento JVM:
  `-Djava.security.auth.policy=file:[JAAS_AUTH_POLICY_FILE]`
- Use a seguinte propriedade no arquivo java.security:
  `-Dauth.policy.url.x=file:[JAAS_AUTH_POLICY_FILE]`

Autorização de ObjectGrid Customizada

O plug-in ObjectGridAuthorization é utilizado para autorizar acessos ObjectGrid, ObjectMap e JavaMap para Principals representados por um objeto Subject de modo customizado. Uma implementação típica desse plug-in é recuperar os Principals do objeto Subject, e então verificar se as permissões especificadas estão concedidas ou não aos Principals.

Uma permissão passada para o método checkPermission(Subject, Permission) poderia ser uma das seguintes:

- MapPermission
- ObjectGridPermission
- AgentPermission
- ServerMapPermission

Consulte a documentação da API do ObjectGridAuthorization para obter mais detalhes.

O plug-in ObjectGridAuthorization pode ser configurado da seguinte forma:

- Configuração XML

  É possível usar o arquivo XML do ObjectGrid para definir um plug-in de ObjectAuthorization. Veja um exemplo a seguir:

```
<objectGrids>
 <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
...
  <bean id="ObjectGridAuthorization"
className="com.acme.ObjectGridAuthorizationImpl" />
</objectGrids>
```

- Configuração Programática

  Se você quiser criar um ObjectGrid usando o método de API ObjectGrid.setObjectGridAuthorization(ObjectGridAuthorization), pode chamar o seguinte método para configurar o plug-in de autorização. Este método aplica-se somente ao modelo de programação local do eXtreme Scale quando você instancia diretamente a instância do ObjectGrid.

```
/**
     * Sets the <code>ObjectGridAuthorization</code> for this ObjectGrid instance.
     * <p>
     * Passing <code>null</code> to this method removes a previously set
     * <code>ObjectGridAuthorization</code> object from an earlier invocation of this method
     * and indicates that this <code>ObjectGrid</code> is not associated with a
     * <code>ObjectGridAuthorization</code> object.
     * <p>
     * This method should only be used when ObjectGrid security is enabled. Se
     * the ObjectGrid security is disabled, the provided <code>ObjectGridAuthorization</code> object
     * will not be used.
     * <p>
     * A <code>ObjectGridAuthorization</code> plugin can be used to authorize
     * access to the ObjectGrid and maps. Please refer to <code>ObjectGridAuthorization</code> for more details.
     *
     * <p>
     * As of XD 6.1, the <code>setMapAuthorization</code> is deprecated and
     * <code>setObjectGridAuthorization</code> is recommended for use. However,
     * if both <code>MapAuthorization</code> plugin and <code>ObjectGridAuthorization</code> plugin
     * are used, ObjectGrid will use the provided <code>MapAuthorization</code> to authorize map accesses,
     * even though it is deprecated.
     * <p>
     * Note, to avoid an <code>IllegalStateException</code>, this method must be
     * called prior to the <code>initialize()</code> method.  Also, keep in mind
     * that the <code>getSession</code> methods implicitly call the
     * <code>initialize()</code> method if it has yet to be called by the
     * application.
     *
     * @param ogAuthorization the <code>ObjectGridAuthorization</code> plugin
     *
     * @throws IllegalStateException if this method is called after the
     *         <code>initialize()</code> method is called.
     *
     * @see #initialize()
     * @see ObjectGridAuthorization
```

```
     * @since WAS XD 6.1
     */
    void setObjectGridAuthorization(ObjectGridAuthorization ogAuthorization);
```

Implementação de ObjectGridAuthorization

O método Boolean checkPermission(Subject subject, Permission permission) da interface ObjectGridAuthorization é chamado pelo tempo de execução do WebSphere eXtreme Scale para verificar se o objeto subject transmitido possui a permissão de passagem. A implementação da interface ObjectGridAuthorization retorna true se o objeto possui a permissão e false se não possui.

Uma implementação típica deste plug-in é recuperar os proprietários do objeto Subject e verificar se as permissões especificadas serão concedidas aos proprietários consultando políticas específicas. Estas políticas são definidas por usuários. Por exemplo, as políticas podem ser definidas em um banco de dados, um arquivo simples ou em um servidor de políticas Tivoli Access Manager.

Por exemplo, podemos usar o servidor de políticas Tivoli Access Manager para gerenciar a política de autorização e usar sua API para autorizar o acesso. Para saber como usar as APIs do Tivoli Access Manager Authorization, consulte o IBM Tivoli Access Manager Authorization Java Classes Developer Reference para obter detalhes adicionais.

Esta implementação da amostra faz as seguintes suposições:

- Somente autorizações de verificação para MapPermission. Para outras permissões, retorne sempre `true`.
- O objeto Subject contém um proprietário com.tivoli.mts.PDPrincipal.
- O servidor de políticas do Tivoli Access Manager definiu as permissões a seguir para o objeto de nome ObjectMap ou JavaMap. O objeto definido no servidor de política deve ter o mesmo nome que o nome ObjectMap ou JavaMap no formato de [ObjectGrid_name].[ObjectMap_name]. A permissão é o primeiro caractere das cadeias de permissão definidas na permissão MapPermission. Por exemplo, a permissão "r" definida no servidor de política representa a permissão de leitura para o mapa ObjectMap.

O fragmento de código a seguir demonstra como implementar o 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) {

  // For non-MapPermission, we always authorize.
  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;
}
```

Informações relacionadas

Módulo 4: Usar a Autorização do Java Authentication and Authorization Service (JAAS) no WebSphere Application Server
Agora que você configurou a autenticação de clientes, é possível configurar ainda mais a autenticação para conceder aos usuários diferentes permissões. Por exemplo, um usuário operador pode apenas visualizar dados, enquanto que um usuário administrador pode executar todas as operações.

Autenticação da Grade de Dados

É possível utilizar o plug-in do gerenciador de token seguro para ativar a autenticação servidor-para-servidor, que requer que você implemente a interface SecureTokenManager.

O método generateToken(Object) obtém uma proteção de objeto, e depois gera um token que não pode ser compreendido pelos outros. O método verifyTokens(byte[]) faz o processo inverso: converte o token de volta ao objeto original.

Uma implementação SecureTokenManager simples usa um algoritmo de codificação simples, como um algoritmo XOR, para codificar o objeto na forma serializada e depois usa o algoritmo de codificação correspondente para decodificar o token. Esta implementação não é segura e é fácil de ser interrompida.

Implementação padrão do **WebSphere eXtreme Scale**

O WebSphere eXtreme Scale fornece uma implementação imediatamente disponível para esta interface. Esta implementação padrão utiliza um par de chaves para assinar e verificar a assinatura e utiliza uma chave secreta para criptografar o conteúdo. Cada servidor tem um armazenamento de chaves de tipo JCKES para armazenar o par de chaves, uma chave privada e uma chave pública e uma chave secreta. O armazenamento de chaves tem que ser do tipo JCKES para armazenar as chaves secretas. Estas chaves são utilizadas para criptografar e assinar ou verificar a cadeia de segredo na extremidade de envio. Além disso, o token está associado ao tempo de expiração. Na extremidade de recebimento, os dados são verificados, decriptografados e comparados com a cadeia de segredo do receptor. Os protocolos de comunicação Secure Sockets Layer (SSL) não são necessários entre um par de servidores para autenticação, porque as chaves privadas e as chaves públicas servem para a mesma finalidade. No entanto, se a comunicação do servidor não for criptografada, os dados poderão ser roubados por violação na comunicação.

Como o token expira em breve, a ameaça de ataque à reprodução é minimizada. Esta possibilidade é significativamente reduzida se todos os servidores forem implementados atrás de um firewall.

A desvantagem desta abordagem é que os administradores do WebSphere eXtreme Scale precisam gerar chaves e transportá-las para todos os servidores, o que pode causar violação de segurança durante o transporte.

Programação de Segurança Local

WebSphere eXtreme Scale fornece vários terminais de segurança para permitir que você integre mecanismos customizados. No modelo de programação local, a principal função de segurança é a autorização e não possui suporte à autenticação. É necessário autenticar fora do WebSphere Application Server. Entretanto, são fornecidos plug-ins para obter e validar objetos Subject.

Autenticação

No modelo de programação local, o eXtreme Scale não fornece nenhum mecanismo de autenticação, mas conta com o ambiente, servidores de aplicativos ou aplicativos, para autenticação. Quando o eXtreme Scale é usado no WebSphere Application Server ou WebSphere Extended Deployment, os aplicativos podem usar o mecanismo de autenticação de segurança do WebSphere Application Server. Quando o eXtreme Scale está sendo executado em um ambiente J2SE (Java 2 Platform, Standard Edition, o aplicativo tem que gerenciar as autenticações com autenticação JAAS (Java Authentication and Authorization Service) ou outros mecanismos de autenticação. Para obter informações adicionais sobre como utilizar a autenticação JAAS, consulte o Guia de Referência do JAAS. O contrato entre um aplicativo e uma instância do ObjectGrid é o objeto javax.security.auth.Subject. Quando o cliente é autenticado pelo servidor de aplicativos ou pelo aplicativo, o aplicativo pode recuperar o objeto javax.security.auth.Subject autenticado e utilizar este objeto Subject para obter uma sessão da instância do ObjectGrid, chamando o método ObjectGrid.getSession(Subject). Este objeto Subject é utilizado para autorizar o acesso aos dados do mapa. Este contrato é chamado de mecanismo de transmissão de subject. O exemplo a seguir ilustra a API ObjectGrid.getSession(Subject).

```
/**
 * This API allows the cache to use a specific subject rather than the one
 * configured on the ObjectGrid to get a session.
 * @param subject
 * @return An instance of Session
 * @throws ObjectGridException
 * @throws TransactionCallbackException
 * @throws InvalidSubjectException the subject passed in is not valid based
 * on the SubjectValidation mechanism.
 */
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;
```

O método ObjectGrid.getSession() na interface ObjectGrid também pode ser utilizando para obter um objeto Session:

```
/**
 * This method returns a Session object that can be used by a single thread at a time.
 * You cannot share this Session object between threads without placing a
 * critical section around it. While the core framework allows the object to move
 * between threads, the TransactionCallback and Loader might prevent this usage,
 * especially in J2EE environments. When security is enabled, this method uses the
 * SubjectSource to get a Subject object.
 *
 * If the initialize method has not been invoked prior to the first
 * getSession invocation, then an implicit initialization occurs.  This
```

```
 * initialization ensures that all of the configuration is complete before
 * any runtime usage is required.
 *
 * @see #initialize()
 * @return An instance of Session
 * @throws ObjectGridException
 * @throws TransactionCallbackException
 * @throws IllegalStateException if this method is called after the
 *         destroy() method is called.
 */
public Session getSession()
throws ObjectGridException, TransactionCallbackException;
```

Conforme especifica a documentação da API, quando a segurança é ativada, este método utiliza o plug-in SubjectSource para obter um objeto Subject. O plug-in SubjectSource é um dos plug-ins de segurança no eXtreme Scale para suportar a propagação de objetos Subject. Consulte Plug-ins Relacionados à Segurança para obter informações adicionais. O método getSession(Subject) pode ser chamado na instância do ObjectGrid local apenas. Se você chamar o método getSession(Subject) em um lado do cliente em uma configuração distribuída do eXtreme Scale, o resultado será um IllegalStateException.

Plug-ins de Segurança

O WebSphere eXtreme Scale oferece dois plug-ins de segurança que estão relacionados ao mecanismo de transmissão de assunto: os plug-ins SubjectSource e SubjectValidation.

Plug-in SubjectSource

O plug-in SubjectSource, representado pela interface com.ibm.websphere.objectgrid.security.plugins.SubjectSource, é um plug-in usado para obter um objeto Subject de um ambiente de execução do eXtreme Scale. Este ambiente pode ser um aplicativo usando o ObjectGrid ou um servidor de aplicativos que hospeda o aplicativo. Considere o plug-in SubjectSource uma alternativa para o mecanismo de transmissão de subject. Utilizando o mecanismo de transmissão de subject, o aplicativo recupera o objeto Subject e utiliza-o para obter o objeto de sessão do ObjectGrid. Com o plug-in SubjectSource, o tempo de execução do eXtreme Scale recupera o objeto Subject e o utiliza para obter o objeto de sessão. O mecanismo de transmissão de subject fornece o controle de objetos Subject para aplicativos, enquanto o mecanismo do plug-in SubjectSource libera aplicativos de recuperar o objeto Subject. É possível utilizar o plug-in SubjectSource para obter um objeto Subject que representa um cliente do eXtreme Scale que é utilizado para autorização. Quando o método ObjectGrid.getSession é chamado, o Subject getSubject emite uma ObjectGridSecurityException se a segurança estiver ativada. O WebSphere eXtreme Scale oferece uma implementação padrão deste plug-in: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl. Esta implementação pode ser usada para recuperar um assunto do responsável pela chamada ou um assunto RunAs do encadeamento quando um aplicativo está em execução no WebSphere Application Server. É possível configurar esta classe no arquivo XML descritor do ObjectGrid como a classe de implementação SubjectSource ao usar o eXtreme Scale no WebSphere Application Server. O trecho de código a seguir mostra o fluxo principal do método WSSubjectSourceImpl.getSubject.

```
Subject s = null;
try {
  if (finalType == RUN_AS_SUBJECT) {
    // obter o subject RunAs
    s = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
```

```
  }
  else if (finalType == CALLER_SUBJECT) {
    // obter o callersubject
    s = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
  }
}
catch (WSSecurityException wse) {
  throw new ObjectGridSecurityException(wse);
}

return s;
```

Para obter outros detalhes, consulte a documentação da API para o plug-in SubjectSource e a implementação do WSSubjectSourceImpl.

Plug-in SubjectValidation

O plug-in SubjectValidation, que é representado pela interface com.ibm.websphere.objectgrid.security.plugins.SubjectValidation, é outro plug-in de segurança. O plug-in SubjectValidation pode ser utilizado para validar que um javax.security.auth.Subject, quer transmitido para o ObjectGrid quer recuperado pelo plug-in SubjectSource, é um Subject válido desde que não violado.

O método SubjectValidation.validateSubject(Subject) na interface SubjectValidation obtém um objeto Subject e retorna um objeto Subject. Tudo está definido nas suas implementações: se o objeto Subject é ou não válido e qual objeto Subject será retornado. Se o objeto Subject não for válido, o resultado será um InvalidSubjectException.

É possível utilizar esse plug-in se não confiar no objeto Subject transmitido para esse método. Isso dificilmente ocorrerá, desde que você confie nos desenvolvedores que redigiram o código do aplicativo para recuperar o objeto Subject.

Uma implementação deste plug-in precisa de suporte do criador do objeto Subject, porque apenas o criador sabe se o objeto Subject foi violado. No entanto, alguns criadores de subjects podem não saber se o Subject foi violado. Neste caso, este plug-in não é útil.

O WebSphere eXtreme Scale oferece uma implementação padrão de SubjectValidation: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl. É possível usar esta implementação para validar o assunto autenticado pelo WebSphere Application Server. Você pode configurar esta classe com a classe de implementação SubjectValidation ao usar o eXtreme Scale no WebSphere Application Server. A implementação do WSSubjectValidationImpl levará em consideração um objeto Subject válido apenas se o token da credencial associado a tal Subject não estiver violado. Você pode alterar outras partes do objeto Subject. A implementação WSSubjectValidationImpl solicita ao WebSphere Application Server o Subject original correspondendo ao token de credencial e retorna o Subject original como o objeto Subject validado. Portanto, as alterações feitas no conteúdo do Subject diferentes do token de credencial não têm nenhum efeito. O trecho de código a seguir mostra o fluxo básico do WSSubjectValidationImpl.validateSubject(Subject).

```
// Create a LoginContext with scheme WSLogin and
// pass a Callback handler.
LoginContext lc = new LoginContext("WSLogin",
new WSCredTokenCallbackHandlerImpl(subject));
```

```
// When this method is called, the callback handler methods
// will be called to log the user in.
lc.login();

// Get the subject from the LoginContext
return lc.getSubject();
```

O trecho de código anterior cria um objeto de manipulador de retorno de chamada do token de credencial, WSCredTokenCallbackHandlerImpl, com o objeto Subject a ser validado. Em seguida, um objeto LoginContext é criado com o esquema de login WSLogin. Quando o método lc.login é chamado, a segurança WebSphere Application Server recupera o token de credencial do objeto Subject e, em seguida, retorna o Subject correspondente como o objeto Subject validado.

Para obter outros detalhes, consulte as APIs Java da implementação SubjectValidation e WSSubjectValidationImpl.

Configuração do Plug-in

Você pode configurar os plug-ins SubjectValidation e SubjectSource de dois modos:

- **Configuração XML**É possível utilizar o arquivo XML do ObjectGrid para definir um ObjectGrid e configurar estes dois plug-ins. A seguir está um exemplo, no qual a classe WSSubjectSourceImpl está configurada como o plug-in SubjectSource e a classe WSSubjectValidation está configurada como o 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>
  ```

- **Programação** Se você desejar criar um ObjectGrid através de APIs, é possível chamar os seguintes métodos para configurar os plug-ins SubjectSource ou SubjectValidation.

  ```
  **
  * Set the SubjectValidation plug-in for this ObjectGrid instance. A
  * SubjectValidation plug-in can be used to validate the Subject object
  * passed in as a valid Subject. Refer to {@link SubjectValidation}
  * for more details.
  * @param subjectValidation the SubjectValidation plug-in
  */
  void setSubjectValidation(SubjectValidation subjectValidation);


  /**
  * Set the SubjectSource plug-in. A SubjectSource plug-in can be used
  * to get a Subject object from the environment to represent the
  * ObjectGrid client.
  *
  * @param source the SubjectSource plug-in
  */
  void setSubjectSource(SubjectSource source);
  ```

Gravar Seu Código de Autenticação JAAS

Você pode escrever seu próprio código de autenticação JAAS (Java Authentication and Authorization Service) para manipular a autenticação. É necessário gravar seus próprios módulos de login e, em seguida, configurá-los para seu módulo de autenticação.

O módulo de login recebe informações sobre um usuário e autentica o usuário. Estas informações podem ser tudo o que pode identificar o usuário. Por exemplo, as informações podem ser um ID de usuário e senha, certificado do cliente, e assim por diante. Depois de receber tais informações, o módulo de login verifica se elas representam um Subject válido e cria o objeto Subject. No momento, várias implementações de módulos de login estão disponíveis para o público.

Depois que um módulo de login for gravado, configure-o para o tempo de execução a ser utilizado. Configure um módulo de login do JAAS. Este módulo de login contém o módulo de login e seu esquema de autenticação. Por exemplo:

```
FileLogin
{
    com.acme.auth.FileLoginModule required
};
```

O esquema de autenticação é FileLogin e o módulo de login é com.acme.auth.FileLoginModule. O token requerido indica que o módulo FileLoginModule deve validar tal login ou todo o esquema falhará.

A configuração do arquivo de configuração do módulo de login JAAS pode ser feita de um dos seguintes modos:

- Configure o arquivo de configuração do módulo de login do JAAS na propriedade login.config.url no arquivo java.security, por exemplo:
  ```
  login.config.url.1=file:${java.home}/lib/security/file.login
  ```
- Configure o arquivo de configuração de módulo de login JAAS a partir da linha de comandos usando os argumentos JVM (Java Virtual Machine) **-Djava.security.auth.login.config** como, por exemplo,
  ```
  -Djava.security.auth.login.config ==$JAVA_HOME/lib/security/file.login
  ```

Se o seu código estiver em execução no WebSphere Application Server, configure o login do JAAS no console administrativo e armazene esta configuração de login na configuração do servidor de aplicativos. Consulte a configuração de login para Java Authentication and Authorization Service para obter detalhes.

Capítulo 8. Resolução de Problemas

Além dos logs e do rastreio, de mensagens e notas sobre a liberação discutidos nesta seção, é possível usar as ferramentas de monitoramento para descobrir problemas, como o local de dados no ambiente, a disponibilidade de servidores na grade de dados, e assim por diante. Se você estiver executando um ambiente do WebSphere Application Server, poderá usar a Performance Monitoring Infrastructure (PMI). Se você estiver executando em um ambiente independente, poderá usar uma ferramenta de monitoramento do fornecedor, como CA Wily Introscope ou Hyperic HQ. Também é possível usar e customizar o utilitário **xscmd** para exibir informações de texto sobre o ambiente.

Ativando a Criação de Log

É possível usar logs para monitorar e solucionar problemas em seu ambiente.

Sobre Esta Tarefa

Os logs são salvos em diferentes locais e formatos dependendo da sua configuração.

Procedimento

- **Ative os logs em um ambiente independente.**

  Com servidores de catálogos independentes, os logs estão no local onde o comando **startOgServer** é executado. Para os servidores de contêiner, é possível usar o local padrão ou definir um local de log customizado:

  – **Local de log padrão:** O logs estão no diretório onde o comando do servidor foi executado. Se você iniciar os servidores no diretório *wxs_home*/bin, os arquivos de log e de rasteio estão nos diretórios logs/<*server_name*> no diretório bin.

  – **Local de log customizado:** Para especificar um local alternativo para os logs do servidor de contêiner, crie um arquivo de propriedades, como server.properties, com o seguinte conteúdo:

  ```
  workingDirectory=<directory>
  traceSpec=
  systemStreamToFileEnabled=true
  ```

  A propriedade **workingDirectory** é o diretório-raiz para os logs e para o arquivo de rastreio opcional. O WebSphere eXtreme Scale cria um diretório com o nome do servidor de contêiner com um arquivo SystemOut.log, um arquivo SystemErr.log e um arquivo de rastreio. Para usar um arquivo de propriedades durante a inicialização de contêiner, use a opção **-serverProps** e forneça o local do arquivo de propriedades do servidor.

- **Ative os logs no WebSphere Application Server.**

  ConsulteWebSphere Application Server: Ativando e Desativando Criação de Log para obter mais informações.

- **Recupere os arquivos FFDC.**

  Os arquivos FFDC servem para que o suporte IBM auxilie na depuração. Estes arquivos poderão ser solicitados pelo suporte IBM quando ocorrer um problema. Esses arquivos aparecem no diretório ffdc e contêm arquivos semelhantes ao seguinte:

  ```
  server2_exception.log
  server2_20802080_07.03.05_10.52.18_0.txt
  ```

O que Fazer Depois

Visualize os arquivos de log e seus locais especificados. As mensagens comuns para procurar no arquivo `SystemOut.log` são as mensagens de confirmação de início, como o seguinte exemplo:

```
CWOBJ1001I: ObjectGrid Server catalogServer01 is ready to process requests.
```

Para obter mais informações sobre uma mensagem específica nos arquivos de log, consulte Mensagens.

Referências relacionadas

"Opções de Rastreio" na página 506
É possível ativar o rastreio para fornecer informações sobre o seu ambiente para o suporte IBM.

Mensagens
Quando encontrar uma mensagem em um log ou em outras partes de uma interface de produto, será possível procurar pela mensagem pelo prefixo do componente para obter mais informações.

Coletando Rastreio

É possível usar o rastreio para monitorar e solucionar problemas em seu ambiente. O rastreio deve ser fornecido para um servidor quando trabalhar com o suporte IBM.

Sobre Esta Tarefa

Coletar rastreio pode ajudá-lo a monitorar e corrigir problemas em sua implementação do WebSphere eXtreme Scale. A maneira como o rastreio é coletado depende da sua configuração. Consulte "Opções de Rastreio" na página 506 para obter a lista das especificações de rastreio diferentes que podem ser coletadas.

Procedimento

- **Colete o rastreio dentro de um ambiente WebSphere Application Server.**

  Se o seus servidores de catálogos e de contêiner estiverem em um ambiente WebSphere Application Server, consulte WebSphere Application Server: Trabalhando com Rastreio para obter mais informações.

- **Colete o rastreio com o comando start do servidor de catálogos ou de contêiner independente.**

  É possível configurar o rastreio em um servidor de serviço de catálogo ou de contêiner usando os parâmetros **-traceSpec** e **-traceFile** com o comando **startOgServer**. Por exemplo:

  ```
  startOgServer.sh catalogServer -traceSpec ObjectGridPlacement=all=enabled -traceFile  /home/user1/logs/trace.log
  ```

  O parâmetro **-traceFile** é opcional. Se não configurar um local **-traceFile**, o arquivo de rastreio irá para o mesmo local dos arquivos de log do sistema.Para obter mais informações sobre esses parâmetros, consulte as informações sobre o script startOgServer no *Guia de Administração*.

- **Colete o rastreio no servidor de catálogos ou de contêiner independente com um arquivo de propriedades.**

  Para coletar rastreio a partir de um arquivo de propriedades, crie um arquivo, como `server.properties`, com o seguinte conteúdo:

  ```
  workingDirectory=<directory>
  traceSpec=<trace_specification>
  systemStreamToFileEnabled=true
  ```

A propriedade **workingDirectory** é o diretório-raiz para os logs e para o arquivo de rastreio opcional. Se o valor **workingDirectory** não estiver configurado, o diretório de trabalho padrão será o local usado para iniciar os servidores, como *wxs_home*/bin. Para usar um arquivo de propriedades durante a inicialização do servidor, utilize o parâmetro **-serverProps** com o comando **startOgServer** e forneça o local do arquivo de propriedades do servidor.Para obter mais informações sobre o arquivo de propriedades do servidor e como usar esse arquivo, consulte as informações sobre o arquivo de propriedades do servidor no *Guia de Administração*.

- **Colete o rastreio em um cliente independente.**

  É possível iniciar a coleção de rastreio em um cliente independente ao incluir propriedades do sistema no script de inicialização para o aplicativo cliente. No exemplo a seguir, as configurações de rastreio são especificadas para o aplicativo 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: Ativando Rastreio em Aplicativos Clientes e Independentes para obter mais informações.

- **Colete o rastreio com a interface ObjectGridManager.**

  Também é possível configurar o rastreio durante o tempo de execução em uma interface ObjectGridManager. A configuração de um rastreio em uma interface ObjectGridManager pode ser usada para obter rastreio em um cliente eXtreme Scale enquanto ele se conecta com um eXtreme Scale e confirma as transações. Para configurar o rastreio em uma interface ObjectGridManager, forneça uma especificação de rastreio e um log de rastreio.

  ```
  ObjectGridManager manager= ObjectGridManagerFactory.getObjectGridManager();
  ...
  manager.setTraceEnabled(true);
  manager.setTraceFileName("logs/myClient.log");
  manager.setTraceSpecification("ObjectGridReplication=all=enabled");
  ```

  Para obter mais informações sobre a interface ObjectGridManager, consulte as informações sobre como interagir com o ObjectGrid usando a interface ObjectGridManager no *Guia de Programação*.

- **Colete o rastreio em servidores de contêiner com o utilitário xscmd.**

  Para coletar o rastreio com o utilitário **xscmd**, use o comando **-c setTraceSpec**. Use o utilitário **xscmd** para coletar o rastreio em um ambiente independente durante o tempo de execução em vez de durante a inicialização. É possível coletar o rastreio em todos os servidores e serviços de catálogo ou filtrar os servidores com base no nome do ObjectGrid, e em outras propriedades. Por exemplo, para coletar o rastreio ObjectGridReplication com acesso ao servidor de serviço de catálogo, execute:

  ```
  xscmd -c setTraceSpec "ObjectGridReplication=all=enabled"
  ```

  Também é possível desativar o rastreio ao configurar a especificação de rastreio para *=all=disabled..

Resultados

Os arquivos de rastreio são gravados no local especificado.

Referências relacionadas

"Opções de Rastreio"
É possível ativar o rastreio para fornecer informações sobre o seu ambiente para o suporte IBM.

Mensagens
Quando encontrar uma mensagem em um log ou em outras partes de uma interface de produto, será possível procurar pela mensagem pelo prefixo do componente para obter mais informações.

Opções de Rastreio

É possível ativar o rastreio para fornecer informações sobre o seu ambiente para o suporte IBM.

Sobre o Rastreio

O rastreio do WebSphere eXtreme Scale é dividido em vários componentes diferentes. É possível especificar o nível de rastreio a ser usado. Os níveis comuns de rastreio incluem: all, debug, entryExit e event.

Um exemplo de cadeia de rastreio é o seguinte:

```
ObjectGridComponent=level=enabled
```

É possível concatenar as cadeias de rastreio. Use o sinal de asterisco (*) para especificar um valor de curinga, como `ObjectGrid*=all=enabled`. Se for necessário fornecer um rastreio para o suporte IBM, uma cadeia de rastreio específica será solicitada. Por exemplo, se ocorrer um problema com a replicação, a cadeia de rastreio `ObjectGridReplication=debug=enabled` pode ser solicitada.

Especificação de Rastreio

ObjectGrid
Mecanismo de cache principal geral.

ObjectGridCatalogServer
Serviço de catálogo geral.

ObjectGridChannel
Comunicações de topologia de implementação estática.

ObjectGridClientInfo
Informações do cliente do DB2.

ObjectGridClientInfoUser
Informações sobre o usuário do DB2.

ObjectgridCORBA
Comunicações de topologia de implementação dinâmica.

ObjectGridDataGrid
A API do AgentManager.

ObjectGridDynaCache
O provedor de cache dinâmico do WebSphere eXtreme Scale.

ObjectGridEntityManager
A API do EntityManager. Utilize com a opção Projector.

ObjectGridEvictors
Evictors integrados do ObjectGrid.

ObjectGridJPA
Carregadores do Java Persistence API (JPA).

ObjectGridJPACache
Plug-ins do Cache JPA

ObjectGridLocking
Gerenciador de bloqueios de entrada de cache do ObjectGrid.

ObjectGridMBean
Beans de gerenciamento.

ObjectGridMonitor
Infraestrutura de monitoramento histórico.

7.1.1+ **ObjectGridNative**
Rastreio do código nativo do WebSphere eXtreme Scale, incluindo o código nativo eXtremeMemory.

7.1.1+ **ObjectGridOSGi**
Os componente de integração de OSGi do WebSphere eXtreme Scale.

ObjectGridPlacement
Serviço de disposição de shards do servidor de catálogos.

ObjectGridQuery
Consulte ObjectGrid.

ObjectGridReplication
Serviço de replicação.

ObjectGridRouting
Detalhes de roteamento do cliente/servidor.

ObjectGridSecurity
Rastreio de segurança.

7.1.1+ **ObjectGridSerializer**
A infraestrutura do plug-in DataSerializer.

ObjectGridStats
Estatísticas do ObjectGrid.

ObjectGridStreamQuery
API de Consulta do Fluxo.

7.1.1+ **ObjectGridTransactionManager**
O gerenciador de transações do WebSphere eXtreme Scale.

ObjectGridWriteBehind
Atributo write-behind do ObjectGrid.

7.1.1+ **ObjectGridXM**
Rastreio geral do IBM eXtremeMemory.

7.1.1+ **ObjectGridXMEviction**
Rastreio de despejo do eXtremeMemory.

7.1.1+ **ObjectGridXMTransport**
Rastreio de transporte geral do eXtremeMemory.

7.1.1+ **ObjectGridXMTransportInbound**
Rastreio de transporte específico da entrada do eXtremeMemory.

7.1.1+ **ObjectGridXMTransportOutbound**
Rastreio de transporte específico da entrada do eXtremeMemory.

Projector
O mecanismo dentro da API do EntityManager.

QueryEngine
O mecanismo de consulta para a API de Consulta do Objeto e a API de Consulta do EntityManager.

QueryEnginePlan
Rastreio do plano de consulta.

7.1.1+ **TCPChannel**
O canal TCP/IP do IBM eXtremeIO.

7.1.1+ **XsByteBuffer**
Rastreio do buffer de bytes do WebSphere eXtreme Scale.

Tarefas relacionadas

"Ativando a Criação de Log" na página 503
É possível usar logs para monitorar e solucionar problemas em seu ambiente.

"Coletando Rastreio" na página 504
É possível usar o rastreio para monitorar e solucionar problemas em seu ambiente. O rastreio deve ser fornecido para um servidor quando trabalhar com o suporte IBM.

Iniciando Servidores Independentes
Quando estiver executando uma configuração independente, o ambiente é composto por servidores de catálogos, servidores de contêineres e processos do cliente. Os servidores do WebSphere eXtreme Scale também podem ser integrados em aplicativos Java existentes usando a API do servidor integrado. É necessário configurar e iniciar estes processos manualmente.

Administrando com o Utilitário **xscmd**
Com o **xscmd**, é possível concluir tarefas administrativas no ambiente como: estabelecer links de replicação multimestre, substituir o quorum e parar grupos de servidores com o comando teardown.

Analisando Dados de Log e de Rastreio

É possível usar as ferramentas de análise de log para analisar como seu tempo de execução está executando e resolver problemas que ocorrem no ambiente.

Sobre Esta Tarefa

É possível gerar relatórios a partir dos arquivos existentes de log e de rastreio no ambiente. Esses relatórios visuais podem ser usados para as seguintes finalidades:

- **Analisar o status e o desempenho do ambiente de tempo de execução:**
  - Implementar consistência do ambiente
  - Criar log da frequência
  - Topologia de execução versus topologia configurada
  - Mudanças de topologia não planejadas
  - Status de quorum
  - Status de replicação de partição
  - Estatísticas de memória, rendimento, uso do processador, e assim por diante
- **Para resolver problemas no ambiente:**
  - Visualizações de topologia em pontos no tempo específicos
  - Estatísticas de memória, rendimento, uso do processador durante falhas do cliente

– Níveis de fix pack atuais, ajuste de configurações
– Status de quorum

Visão Geral de Análise de Log

É possível usar a ferramenta **xsLogAnalyzer** para ajudar a resolver problemas no ambiente.

Todas as Mensagens de Failover

Exibe o número total de mensagens de failover como um gráfico ao longo do tempo. Também exibe uma lista das mensagens de failover, incluindo os servidores que foram afetados.

Todas as Mensagens Críticas do eXtreme Scale

Exibe os IDs de mensagem junto com as explicações e ações do usuário associadas, o que pode economizar o tempo de procura por mensagens.

Todas as Exceções

Exibe as cinco principais exceções, incluindo as mensagens e quantas vezes elas ocorreram e quais servidores foram afetados pela exceção.

Resumo de Topologia

Exibe um diagrama de como sua topologia é configurada de acordo com os arquivos de log. Este resumo pode ser usado para comparar a configuração real, possivelmente a identificação de erros de configuração.

Consistência de Topologia: Tabela de Comparação do Object Request Broker (ORB)

Exibe configurações ORB no ambiente. É possível usar essa tabela para ajudar a determinar se as configurações estão consistentes em seu ambiente.

Visualização de Linha de Tempo de Eventos

Exibe um diagrama de linha de tempo de diferentes ações que ocorreram na grade de dados, incluindo eventos de ciclo de vida, exceções, mensagens críticas e eventos de captura de dados de erros (FFDC).

Executando Análise de Log

É possível executar a ferramenta **xsLogAnalyzer** em um conjunto de arquivos de log e de rastreio a partir de qualquer computador.

Antes de Iniciar

- Ative os logs e rastreio. Consulte “Ativando a Criação de Log” na página 503 e “Coletando Rastreio” na página 504 para obter mais informações.
- Colete os arquivos de log. Os arquivos de log podem estar em vários locais dependendo de como eles foram configurados. Se você estiver usando as configurações de log padrão, os arquivos de log poderão ser obtidos a partir dos seguintes locais:
  – Em uma instalação independente:*wxs_install_root*/bin/logs/<*server_name*>

  - Em uma instalação integrada ao WebSphere Application Server: *was_root*/logs/*<server_name>*
- Colete os arquivos de rastreio. Os arquivos de rasteio podem estar em vários locais dependendo de como eles foram configurados. Se você estiver usando as configurações de rastreio padrão, os arquivos de trace poderão ser obtidos a partir dos seguintes locais:
  - Em uma instalação independente: Se nenhum valor de rastreio específico for configurado, os arquivos de rastreio serão gravados para o mesmo local que o sistema de arquivos de log.
  - Em uma instalação que é integrada com o WebSphere Application Server: *was_root*/profiles/*server_name*/logs.

  Copie os arquivos de log e rastreio no computador a partir do qual você está planejando usar a ferramenta Log Analyzer.
- Se desejar criar scanners customizados no relatório gerado, crie um arquivo de propriedades de especificações do scanner e o arquivo de configuração antes de executar a ferramenta. Para obter informações adicionais, consulte "Criando Scanners Customizados para Análise do Log" na página 511.

Procedimento

1. Execute a ferramenta **xsLogAnalyzer**.

   O script está nos seguintes locais:
   - Em uma instalação independente : *wxs_install_root*/ObjectGrid/bin
   - Em uma instalação integrada ao WebSphere Application Server: *was_root*/bin

   Dica: Se seus arquivos de log forem grandes, considere usar os parâmetros **-startTime**, **-endTime** e **-maxRecords** quando executar o relatório para restringir o número de entradas de log que são varridos. Usar esses parâmetros quando você executa o relatório torna os relatórios mais fáceis de ler e de executar com mais eficiência. É possível executar diversos relatórios no mesmo conjunto de arquivos de log.

   ```
   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 o caminho absoluto para o diretório de log que você deseja avaliar (necessário).

   -outDir
   Especifica um diretório existente para gravar a saída do relatório. Se você não especificar um valor, o relatório será gravado no local raiz da ferramenta **xsLogAnalyzer**.

   -startTime
   Especifica o horário de início para avaliar nos logs. A data está no formato a seguir: *year.month.day.hour.minute.second.millisecond*

   -endTime
   Especifica o horário de encerramento para avaliar nos logs. A data está no formato a seguir: *year.month.day.hour.minute.second.millisecond*

   -trace Especifica uma sequência de rastreio, tal como ObjectGrid*=all=enabled.

   -maxRecords
   Especifica o número máximo de registros a serem gerados no relatório.

O padrão é `100`. Se você especificar o valor como 50, os primeiros 50 registros serão gerados para o período de tempo especificado.

2. Abra os arquivos gerados. Se um diretório de saída não foi definido, os relatórios serão gerados em uma pasta chamada `report_`*`date_time`*. Para abrir a página principal dos relatórios, abra o arquivo `index.html`.
3. Use os relatórios para analisar os dados do log. Use as dicas a seguir para maximizar o desempenho das exibições de relatório:
   - Para maximizar o desempenho de consultas nos dados de log, use informações o mais específicas possível. Por exemplo, uma consulta para `server` leva muito mais tempo para ser executada e retorna mais resultados do que `server_host_name`.
   - Algumas visualizações têm um número limitado de pontos de dados que são exibidos de uma vez. É possível ajustar o segmento de tempo que está sendo visualizado alterando os dados atuais, tais como a hora de início e de encerramento, na visualização.

O que Fazer Depois

Para obter mais informações sobre resolução de problemas da ferramenta **`xsLogAnalyzer`** e os relatórios gerados, consulte "Resolução de Problemas da Análise do Log" na página 512.

Criando Scanners Customizados para Análise do Log

É possível criar scanners customizados para análise do log. Depois de configurar o scanner, os resultados são gerados nos relatórios quando você executa a ferramenta **`xsLogAnalyzer`**. O scanner customizado varre os logs para os registros de eventos com base nas expressões regulares que você especificou.

Procedimento

1. Crie um arquivo de propriedades de especificações do scanner que especifica a expressão geral para executar o scanner customizado.
   a. Crie e salve um arquivo de propriedades. O arquivo deve estar no diretório *`loganalyzer_root`*`/config/custom`. É possível nomear o arquivo como: you like. O arquivo é usado pelo novo scanner, portanto, nomear o scanner no arquivo de propriedades é útil, por exemplo: *`my_new_server_scanner_spec`*`.properties`.
   b. Inclua as propriedades a seguir no arquivo *`my_new_server_scanner_spec`*`.properties`:

      ```
      include.regular_expression = REGULAR_EXPRESSION_TO_SCAN
      ```

      A variável *REGULAR_EXPRESSION_TO_SCAN* é uma expressão regular na qual filtrar os arquivos de log.

      Exemplo: Para varrer em busca de instâncias de linhas que contêm as sequências `"xception"` e `"rror"` independentemente da ordem, configure a propriedade **`include.regular_expression`** com o valor a seguir:

      ```
      include.regular_expression = (xception.+rror)|(rror.+xception)
      ```

      Esta expressão regular faz com que os eventos sejam registrados se a sequência `"rror"` vier antes ou após a sequência `"xception"`.

      Exemplo:Para varrer através de cada linha nos logs em busca de instâncias de linhas que contêm as sequências de frase `"xception"` ou a frase `"rror"` independetemente da ordem, configure a propriedade **`include.regular_expression`** com o valor a seguir:

```
include.regular_expression = (xception)|(rror)
```

Esta expressão regular faz com que os eventos sejam registrados se a sequência "rror" vier antes ou após a sequência "xception".

2. Crie um arquivo de configuração que a ferramenta **xsLogAnalyer** usa para criar o scanner.
   a. Crie e salve um arquivo de propriedades. O arquivo deve estar no diretório *loganalyzer_root*/config/custom. É possível nomear o arquivo como *scanner_name*Scanner.config, em que *scanner_name* é um nome exclusivo para o novo scanner. Por exemplo, você pode nomear o arquivo serverScanner.config
   b. Inclua as propriedades a seguir no arquivo *scanner_name*Scanner.config:
   ```
   scannerSpecificationFiles = LOCATION_OF_SCANNER_SPECIFICATION_FILE
   ```

   A variável *LOCATION_OF_SCANNER_SPECIFICATION_FILE* é o caminho e o local do arquivo de especificação que você criou na etapa anterior. Por exemplo: *loganalyzer_root*/config/custom/my_new_scanner_spec.properties. Também é possível especificar diversos arquivos de especificação de scanner usando uma lista separada por ponto e vírgula:

```
scannerSpecificationFiles = LOCATION_OF_SCANNER_SPECIFICATION_FILE1;LOCATION_OF_SCANNER_SPECIFICATION_FILE2
```

3. Execute a ferramenta **xsLogAnalyzer**. Para obter informações adicionais, consulte "Executando Análise de Log" na página 509.

Resultados

Depois de executar a ferramenta **xsLogAnalyzer**, o relatório contém novas guias no relatório para os scanners customizados que você configurou. Cada guia contém as visualizações a seguir:

Gráficos
Um gráfico plotado que ilustra os eventos registrados. Os eventos são exibidos na ordem na qual os eventos foram localizados.

Tabelas
Uma representação tabular dos eventos registrados.

Relatórios Resumo

Resolução de Problemas da Análise do Log

Use as informações de resolução de problemas a seguir para diagnosticar e corrigir problemas com a ferramenta **xsLogAnalyzer** e seus relatórios gerados.

Procedimento

- **Problema:** Ocorrem condições de falta de memória quando você está usando a ferramenta **xsLogAnalyzer** para gerar relatórios. Um exemplo de um erro que pode ocorrer é o seguinte: `java.lang.OutOfMemoryError: Limite de sobrecarga de GC excedido`.

  Solução: A ferramenta **xsLogAnalyzer** é executada dentro de uma Java virtual machine (JVM). É possível configurar a JVM para aumentar o tamanho de heap antes de executar a ferramenta **xsLogAnalyzer** especificando algumas configurações quando você executa a ferramenta. O aumento do tamanho de heap permite que mais registros de eventos sejam armazenados na memória da JVM. Comece com uma configuração de 2048 M, assumindo que o sistema operacional possui memória principal suficiente. Na mesma instância de linha

de comandos na qual você está planejando executar a ferramenta **xsLogAnalyzer**, configure o tamanho máximo de heap da JVM:

```
java -Xmx<HEAP_SIZE>m
```

O valor de *HEAP_SIZE* pode ser qualquer número inteiro e representa o número de megabytes que são alocados para o heap da JVM. Por exemplo, você pode executar `java -Xmx2048m`. Se as mensagens de falta de memória continuarem, ou você não tem os recursos para alocar 2048m ou mais de memória, limite o número de eventos que estão sendo mantidos no heap. É possível limitar o número máximo de eventos no heap transmitindo o parâmetro **-maxRecords** para o comando **xsLogAnalyzer**

- **Problema:** Quando você abre um relatório gerado a partir da ferramenta **xsLogAnalyzer**, o navegador é interrompido ou não carrega a página.

  Causa: Os arquivos HTML gerados são muito grandes e não podem ser carregados pelo navegador. Esses arquivos são grandes porque o escopo dos arquivos de log que você está analisando é muito amplo.

  Solução: Considere usar os parâmetros **-startTime**, **-endTime** e **-maxRecords** quando você executar a ferramenta **xsLogAnalyzer** para restringir o número de entradas de log que são varridas. Usar esses parâmetros quando você executa o relatório torna os relatórios mais fáceis de ler e de executar com mais eficiência. É possível executar diversos relatórios no mesmo conjunto de arquivos de log.

Resolução de Problemas de Conectividade do Cliente

Há vários problemas comuns específicos para clientes e de conectividade do cliente que podem ser resolvidos conforme descrito nas seções a seguir.

Procedimento

- **Problema:** Se você estiver usando a API EntityManager ou mapas de matriz de bytes com o modo de cópia COPY_TO_BYTES, os métodos de acesso a dados do cliente resultarão em várias exceções relacionadas à serialização ou uma exceção NullPointerException.
  - O seguinte erro ocorre quando você está usando o modo de cópia COPY_TO_BYTES:

```
java.lang.NullPointerException
        at com.ibm.ws.objectgrid.map.BaseMap$BaseMapObjectTransformer2.inflateObject(BaseMap.java:5278)
        at com.ibm.ws.objectgrid.map.BaseMap$BaseMapObjectTransformer.inflateValue(BaseMap.java:5155)
```

  - O seguinte erro ocorre quando você está usando a API EntityManager :

```
java.lang.NullPointerException
  at com.ibm.ws.objectgrid.em.GraphTraversalHelper.fluffFetchMD(GraphTraversalHelper.java:323)
  at com.ibm.ws.objectgrid.em.GraphTraversalHelper.fluffFetchMD(GraphTraversalHelper.java:343)
  at com.ibm.ws.objectgrid.em.GraphTraversalHelper.getObjectGraph(GraphTraversalHelper.java:102)
  at com.ibm.ws.objectgrid.ServerCoreEventProcessor.getFromMap(ServerCoreEventProcessor.java:709)
  at com.ibm.ws.objectgrid.ServerCoreEventProcessor.processGetRequest(ServerCoreEventProcessor.java:323)
```

  Causa: A API EntityManager e o modo de cópia COPY_TO_BYTES usam um repositório de metadados que é integrado na grade de dados. Quando os clientes se conectam, a grade de dados armazena os identificadores de repositório no cliente e armazena em cache os identificadores enquanto o cliente estiver conectado. Se você reiniciar a grade de dados, todos os metadados serão perdidos e os identificadores regenerados não corresponderão aos identificadores em cache no cliente.

  Solução: Se você estiver usando a API EntityManager ou o modo de cópia COPY_TO_BYTES, desconecte e reconecte todos os clientes se o ObjectGrid for interrompido e reiniciado. Desconectar e reconectar os clientes atualiza o cache

do identificador de metadados. Os clientes podem ser desconectados usando o método ObjectGridManager.disconnect ou o método ObjectGrid.destroy.

- **Problema:** O cliente é interrompido durante uma chamada de método de getObjectGrid.

  Um cliente pode ser interrompido ao chamar o método getObjectGrid no ObjectGridManager ou lançar uma exceção: com.ibm.websphere.projector.MetadataException. O repositório EntityMetadata não está disponível e o limite de tempo limite é alcançado.

  Causa: O motivo é que o cliente está aguardando os metadados da entidade no servidor ObjectGrid estarem disponíveis.

  Solução: Este erro pode ocorrer quando um servidor de contêiner tiver sido iniciado, mas o posicionamento ainda não foi iniciado. Execute as ações a seguir:

  – Examine a política de implementação para o ObjectGrid e verifique se o número de contêineres ativos é maior ou igual aos atributos numInitialContainers e minSyncReplicas no arquivo descritor da política de implementação.

  – Examine a configuração para a propriedade **placementDeferralInterval** no arquivo de propriedade de servidor do servidor de contêiner para ver quanto tempo precisa passar antes de as operações de posicionamento ocorrerem.

  – Se você usou o comando **xscmd -c suspendBalancing** para parar o balanceamento de shards para um conjunto de mapas e grade de dados específicos, use o comando **xscmd -c resumeBalancing** para iniciar novamente o balanceamento.

Conceitos relacionados

"Criando Instâncias do ObjectGrid com a Interface ObjectGridManager" na página 136
Cada um desses métodos cria uma instância local de um ObjectGrid.

Resolvendo Problemas da Integração de Cache

Use estas informações para resolver problemas com a configuração de integração de seu cache, incluindo sessões HTTP e configurações de cache dinâmico.

Procedimento

- 7.1.1+ **Problema:** Os IDs da sessão de HTTP não estão sendo reutilizados.

  Causa: É possível reutilizar os IDs de sessão. Se você criar uma grade de dados para persistência de sessão na Versão 7.1.1 ou posterior, a reutilização do ID de sessão será automaticamente ativada. No entanto, se você criou configurações anteriores, esta configuração já poderá ser configurada com o valor incorreto.

  Solução: Consulte as seguintes configurações para verificar se a reutilização do ID de sessão de HTTP estáativada:

  – A propriedade `reuseSessionId` no arquivo `splicer.properties` deve ser configurada para `true`.

  – O valor da propriedade customizada `HttpSessionIdReuse` deve ser configurado para `true`. Essa propriedade customizada pode ser configurada em um dos seguintes caminhos no console administrativo do WebSphere Application Server:

    - **Servidores** > *server_name* > **Gerenciamento de Sessões** > **Propriedades customizadas**

    - **Clusters Dinâmicos** > *dynamic_cluster_name* > **Modelo doServidor** > **Gerenciamento de Sessão** > **Propriedades Customizadas**

- **Servidores** > **Tipos de Servidor** > **Servidores de Aplicativos WebSphere** > *server_name* e, em seguida, em Infraestrutura do Servidor, clique em **Gerenciamento Java e processos** > **Definição de Processo** > **Java virtual machine** > **Propriedades Customizadas**
- **Servidores** > **Tipos de Servidor** > **Servidores de Aplicativos do WebSphere** > *server_name* > **Configurações do Contêiner da Web** > **Contêiner da Web**.

Se você atualizar quaisquer valores de propriedade customizada, reconfigure o gerenciamento de sessão eXtreme Scale para que o arquivo `splicer.properties` reconheça a mudança.

- **Problema:** Quando você estiver usando uma grade de dados para armazenar as sessões HTTP e o carregamento de transações for alta, uma mensagem CWOBJ0006W será exibida no arquivo `SystemOut.log`.

  ```
  CWOBJ0006W: Ocorreu uma exceção:
  com.ibm.websphere.objectgrid.ObjectGridRuntimeException:
  java.util.ConcurrentModificationException
  ```

  Esta mensagem ocorre apenas quando o parâmetro **`replicationInterval`** no arquivo `splicer.properties` é configurado com um valor maior que zero e o aplicativo da Web modifica um objeto List que foi configurado como um atributo na HTTPSession.

  Solução: Clone o atributo que contém o objeto List modificado e coloque o atributo clonado no objeto de sessão.

Referências relacionadas

Arquivos XML para Configuração do Gerenciador de Sessões HTTP
Quando você inicia um servidor de contêiner que armazena dados da sessão HTTP, é possível usar os arquivos XML padrão ou você pode especificar arquivos XML customizados. Estes arquivos criam nomes de ObjectGrid específicos, número de réplicas, etc.

Parâmetros de inicialização do contexto do servlet
A lista de parâmetros de inicialização de contexto de servlet a seguir pode ser especificada no arquivo splicer properties conforme requerido no método de conexão escolhido.

Arquivo `splicer.properties`
O arquivo `splicer.properties` contém todas as opções de configuração para configurar um gerenciador de sessões baseado em filtro de servlet.

Resolução de Problemas do Plug-in do Cache JPA

Use estas informações para resolver problemas com sua configuração de plug-in do cache JPA. Estes problemas podem ocorrer em ambas as configurações Hibernate e OpenJPA.

Procedimento

- **Problema:** A exceção a seguir é exibida: `CacheException: Falha ao obter o servidor ObjectGrid`.

  Com um valor de atributo **`ObjectGridType`** de `EMBEDDED` ou `EMBEDDED_PARTITION`, o cache eXtreme Scale tenta obter uma instância do servidor a partir do tempo de execução. Em um ambiente Java Platform, Standard Edition, um servidor eXtreme Scale com serviço de catálogo integrado é iniciado. O serviço de catálogo integrado tentará atender na porta 2809. Se essa porta estiver sendo usada por outro processo, ocorrerá um erro.

Solução: Se terminais de serviço de catálogo externo forem especificados, por exemplo, com o arquivo `objectGridServer.properties`, este erro ocorrerá se o nome do host ou a porta for especificada incorretamente. Corrija o conflito de porta.

- **Problema:** A exceção a seguir é exibida: `CacheException: Falha ao obter o REMOTE ObjectGrid para o REMOTE ObjectGrid configurado. objectGridName = [ObjectGridName], Nome da PU = [persistenceUnitName]`

  Esse erro ocorre porque o cache não pode obter a instância do ObjectGrid a partir dos terminais de serviço de catálogo fornecidos.

  Solução: Esse problema geralmente ocorre devido a um nome de host ou porta incorreto.

- **Problema:** A exceção a seguir é exibida: `CacheException: Não é possível ter duas PUs [persistenceUnitName_1, persistenceUnitName_2] configuradas com o mesmo ObjectGridName [ObjectGridName] do EMBEDDED ObjectGridType`

  Essa exceção ocorrerá se você tiver muitas unidades de persistência configuradas e os caches do eXtreme Scale destas unidades forem configurados com o mesmo nome de ObjectGrid e valor de atributo `EMBEDDED` do **`ObjectGridType`**. Estas configurações de unidade de persistência podem estar no mesmo arquivo `persistence.xml` ou diferente.

  Solução: É necessário verificar se o nome do ObjectGrid é exclusivo para cada unidade de persistência quando o valor do atributo **`ObjectGridType`** for `EMBEDDED`.

- **Problema:** A exceção a seguir é exibida: `CacheException: O REMOTE ObjectGrid [ObjectGridName] não inclui os BackingMaps [mapName_1, mapName_2,...] necessários`

  Com um tipo de ObjectGrid REMOTE, se o ObjectGrid do lado do cliente obtido não tiver os mapas de apoio de entidade completos para suportar o cache da unidade de persistência, esta exceção ocorrerá. Por exemplo, cinco classes de entidade são listadas na configuração da unidade de persistência, mas o ObjectGrid obtido possui apenas dois BackingMaps. Embora o ObjectGrid obtido possa ter 10 BackingMaps, se algum destes cinco BackingMaps de entidade necessários não for localizado nos dez BackingMaps, esta exceção ainda ocorrerá.

  Solução: Certifique-se de que a configuração do mapa de apoio suporte o cache da unidade de persistência.

Resolução de Problemas de Administração

Use as informações a seguir para resolver problemas de administração, incluindo iniciar e iniciar os servidores, usando o utilitário **`xscmd`**, e assim por diante.

Procedimento

- **Problema:** Scripts de administração estão ausentes do diretório *`profile_root`*`/bin` de uma instalação do WebSphere Application Server.

  Causa: Quando você atualiza a instalação, novos arquivos de script não são instalados automaticamente nos perfis.

  Solução: Se você deseja executar um script a partir de seu diretório *`profile_root`*`/bin`, cancele o aumento e aumente novamente o perfil com a liberação mais recente. Para obter mais informações, consulte Cancelando o Aumento de um Perfil Usando o Prompt de Comandos e Criando e Alterando Perfis para o WebSphere eXtreme Scale.

- **Problema:** Quando você está executando um comando **`xscmd`**, a seguinte mensagem é impressa na tela:

```
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: Um problema de conexão ocorreu com o servidor de catálogo.

Solução: Verifique se seus servidores de catálogos estão em execução e estão disponíveis pela rede. Esta mensagem também pode ocorrer quando você possui um domínio de serviço de catálogo definido, mas menos de dois servidores de catálogos estão em execução. O ambiente não está disponível até dois servidores de catálogo serem iniciados.

Conceitos relacionados

Melhor Prática: Armazenando em Cluster o Serviço de Catálogo com Domínios de Serviço de Catálogo
Quando estiver usando o serviço de catálogo, no mínimo dois servidores de catálogos são necessários para evitar um ponto de falha único. Dependendo do número de nós em seu ambiente, é possível criar configurações diferentes para assegurar que pelo menos dois servidores de catálogos estejam sempre em execução.

Administrando

Resolução de Problemas de Várias Configurações do Datacenter

Use estas informações para resolver problemas de diversas configurações do datacenter,incluindo vinculação entre domínios do serviço de catálogo.

Procedimento

Problema: Os dados estão ausentes em um ou mais domínios do serviço de catálogo. Por exemplo, você pode executar o comando `xscmd -c establishLink`. Quando você examina os dados para cada domínio do serviço de catálogo vinculado, os dados têm uma outra aparência, por exemplo, a partir do comando `xscmd -c showMapSizes`.
Solução: É possível solucionar esse problema com o comando `xscmd -c showLinkedPrimaries`. Esse comando imprime cada shard primário, incluindo quais primários estrangeiros são vinculados.
No cenário descrito, é possível descobrir, a partir da execução do comando `xscmd -c showLinkedPrimaries`, que os shards primários do primeiro domínio do serviço de catálogo são vinculados aos shards primários do segundo domínio de serviço de catálogo, porém o segundo domínio de serviço de catálogo não possui links para o primeiro domínio de serviço de catálogo. Você pode considerar executar novamente o comando `xscmd -c establishLink` a partir do segundo domínio de serviço de catálogo para o primeiro domínio de serviço de catálogo.

Resolução de Problemas de Carregadores

Use estas informações para resolver problemas com os carregadores de banco de dados.

Procedimento

- **Problema:** Quando estiver usando um carregador OpenJPA com DB2 no WebSphere Application Server, uma exceção de cursor fechado ocorre.

  A exceção a seguir é do DB2 no arquivo de log org.apache.openjpa.persistence.PersistenceException:

  ```
  [jcc][t4][10120][10898][3.57.82] Invalid operation: result set is closed.
  ```

  Solução: Por padrão, o servidor de aplicativos configura a propriedade customizada resultSetHoldability com um valor de 2 (CLOSE_CURSORS_AT_COMMIT). Esta propriedade faz com o DB2 feche seu resultSet/cursor nos limites da transação. Para remover a exceção, altere o valor da propriedade customizada para 1 (HOLD_CURSORS_OVER_COMMIT). Configure a propriedade customizada resultSetHoldability no seguinte caminho na célula do WebSphere Application Server: **Recursos** > **Provedor JDBC** > **Provedor Driver Universal JDBC** > **DataSources** > *data_source_name* > **Propriedades Customizadas** > **Novo**.

- **Problema** O DB2 exibe uma exceção: `A transação atual foi retrocedida devido a um conflito ou tempo limite. Código de razão "2".. SQLCODE=-911, SQLSTATE=40001, DRIVER=3.50.152`

  Essa exceção ocorre devido a um problema de contenção de bloqueio quando estiver executando com o OpenJPA com DB2 no WebSphere Application Server. O nível de isolamento padrão para a Leitura Repetida (RR) do WebSphere Application Server, que obtém bloqueios de longa duração com o DB2.**Solução:**

  Configure o nível de isolamento para Leitura Confirmada para reduzir a contenção de bloqueio. Configure a propriedade customizada da origem de dados webSphereDefaultIsolationLevel para configurar o nível de isolamento para 2(TRANSACTION_READ_COMMITTED) no seguinte caminho na célula WebSphere Application Server: **Recursos** > **Provedor JDBC** > *JDBC_provider* > **Origens de Dados** > *data_source_name* > **Propriedades Customizadas** > **Novo**. Para obter informações adicionais sobre a propriedade customizada webSphereDefaultIsolationLevel e níveis de isolamento de transação, consulte Requisitos para Configurar os Níveis de Isolamento de Acesso a Dados.

- **Problema:** Quando estiver usando a função de pré-carregamento do JPALoader ou JPAEntityLoader, a seguinte mensagem CWOBJ1511 não é exibida para a partição em um servidor de contêiner: `CWOBJ1511I: GRID_NAME:MAPSET_NAME:PARTITION_ID (primário) está aberto para negócios.`

  Em vez disso, ocorrerá uma exceção TargetNotAvailableException no servidor de contêiner, que ativa a partição que é especificada pela propriedade preloadPartition.

  Solução: Configure o atributo preloadMode para `true` se você utilizar um JPALoader ou JPAEntityLoader para pré-carregar dados no mapa. Se a propriedade preloadPartition da JPALoader ou JPAEntityLoader estiver configurada para um valor entre `0` e `total_number_of_partitions-1`, então o JPALoader e o JPAEntityLoader tentarão pré-carregar os dados a partir do banco de dados backend no mapa. O trecho de código a seguir ilustra como o atributo preloadMode é configurado para ativar o pré-carregamento assíncrono:

  ```
  BackingMap bm = og.defineMap( "map1" );
  bm.setPreloadMode( true );
  ```

  Também é possível configurar o atributo preloadMode usando um arquivo XML conforme ilustrado no seguinte exemplo:

  ```
   <backingMap name="map1" preloadMode="true" pluginCollectionRef="map1"
  lockStrategy="OPTIMISTIC" />
  ```

Conceitos relacionados

"Programação para Integração de JPA" na página 396
O Java Persistence API (JPA) é uma especificação que permite o mapeamento de objetos Java para bancos de dados relacionais. O JPA contém uma especificação completa de object-relational mapping (ORM) usando anotações de metadados da linguagem Java, descritores XML, ou ambos para definir o mapeamento entre objetos Java e um banco de dados relacional. Inúmeras implementações comerciais e de software livre estão disponíveis.

Configurando a Integração de Cache
O WebSphere eXtreme Scale pode integrar-se com outros produtos relacionados ao armazenamento em cache. Também é possível usar o provedor de cache dinâmico do WebSphere eXtreme Scale para plugar o WebSphere eXtreme Scale no componente de cache dinâmico no WebSphere Application Server. Outra extensão para o WebSphere Application Server é o gerenciador de sessões HTTP do WebSphere eXtreme Scale, que pode ajudar a armazenar em cache as sessões HTTP.

Resolvendo Problemas de Conflitos

As seções a seguir descrevem alguns dos cenários de conflitos mais comuns e sugestões sobre como evitá-los.

Antes de Iniciar

Implemente a manipulação de exceção no seu aplicativo. Consulte o "Implementando Manipulação de Exceção em Cenários de Bloqueio" na página 250 para obter informações adicionais.

A seguinte exceção é exibida como resultado:

`com.ibm.websphere.objectgrid.plugins.LockDeadlockException: Message`

Essa mensagem representa a cadeia que é transmitida como um parâmetro quando a exceção é criada e emitida.

Procedimento

- **Problema:** Exceção LockTimeoutException.

  Descrição: Quando uma transação ou cliente solicita que um bloqueio seja concedido para uma entrada de mapa específica, a solicitação sempre aguarda o cliente atual liberar o bloqueio antes que a solicitação seja enviada. Se a solicitação de bloqueio permanecer inativa por um longo período de tempo, e o bloqueio nunca for concedido, a exceção LockTimeoutException será criada para evitar um conflito, que é descrito com mais detalhes na seção a seguir. É mais provável que esta exceção seja exibida quando usar uma estratégia de bloqueio pessimista porque o bloqueio nunca é liberado antes que transação seja confirmada.

  Recuperar mais detalhes:

  A exceção LockTimeoutException contém o método getLockRequestQueueDetails, que retorna uma sequência. Este método pode ser usado para ver uma descrição detalhada da situação que aciona a exceção. A seguir há um exemplo de código que captura a exceção e exibe uma mensagem de erro.

```
try {
    ...
}
catch (LockTimeoutException lte) {
    System.out.println(lte.getLockRequestQueueDetails());
}
```

O resultado da saída é:

```
lock request queue
->[TX:163C269E-0105-4000-E0D7-5B3B090A571D, state =
    Granted 5348 milli-seconds ago, mode = U]
->[TX:163C2734-0105-4000-E024-5B3B090A571D, state =
    Waiting for 5348 milli-seconds, mode = U]
->[TX:163C328C-0105-4000-E114-5B3B090A571D, state =
    Waiting for 1402 milli-seconds, mode = U]
```

Se você receber a exceção em um bloco de captura uma exceção ObjectGridException, o código a seguir determinará a exceção e exibe os detalhes da fila. O método do utilitário findRootCause também é usado.

```
try {
...
}
catch (ObjectGridException oe) {
    Throwable Root = findRootCause( oe );
    if (Root instanceof LockTimeoutException) {
        LockTimeoutException lte = (LockTimeoutException)Root;
        System.out.println(lte.getLockRequestQueueDetails());
    }
}
```

Solução: Uma uma exceção LockTimeoutException evita possíveis conflitos em seu aplicativo. Uma exceção desse tipo resulta quando a exceção aguarda uma quantidade de tempo configurada. É possível configurar a quantidade de tempo que a exceção aguarda usando o método setLockTimeout(int), que está disponível para o BackingMap. Se um conflito não existir realmente no aplicativo, ajuste o tempo limite de bloqueio para evitar a LockTimeoutException.

O código a seguir mostra como criar um objeto ObjectGrid, definir um mapa e configurar seu valor LockTimeout para 30 segundos:

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("MapName");
bMap.setLockTimeout(30);
```

Use o exemplo codificado permanentemente para configurar as propriedades do ObjectGrid e do mapa. Se você criar o ObjectGrid a partir de um arquivo XML, configure o atributo **LockTimeout** dentro do elemento backingMap. A seguir há um exemplo de um elemento backingMap que configura um valor LockTimeout do mapa para 30 segundos.

```
<backingMap name="MapName" lockStrategy="PESSIMISTIC" lockTimeout="30">
```

- **Problema:** Conflitos de chave única.

  Descrição: Os seguintes cenários descrevem como podem ocorrer conflitos quando uma única chave é acessada utilizando um bloqueio S e posteriormente atualizada. Quando isto acontece em duas transações simultaneamente, o resultado é um conflito.

Tabela 14. Cenário de conflitos de uma única chave

	Encadeamento 1	**Encadeamento 2**	
1	session.begin()	session.begin()	Cada encadeamento estabelece uma transação independente.
2	map.get(key1)	map.get(key1)	O bloqueio S é concedido a ambas as transações para key1.

Tabela 14. Cenário de conflitos de uma única chave (continuação)

	Encadeamento 1	**Encadeamento 2**	
3	map.update(Key1,v)		Nenhum bloqueio U. A atualização é executada no cache transacional.
4		map.update(key1,v)	Nenhum bloqueio U. A atualização é executada no cache transacional
5	session.commit()		Bloqueado: O bloqueio S para key1 não pode ser atualizado para um bloqueio X porque o Encadeamento 2 possui um bloqueio S.
6		session.commit()	Conflito: O bloqueio S para key1 não pode ser atualizado para um bloqueio X porque T1 possui um bloqueio S.

Tabela 15. Conflitos de uma única chave, continuação

	Encadeamento 1	**Encadeamento 2**	
1	session.begin()	session.begin()	Cada encadeamento estabelece uma transação independente.
2	map.get(key1)		Bloqueio S concedido para key1
3	map.getForUpdate(key1,v)		O bloqueio S é atualizado para um bloqueio U para key1.
4		map.get(key1)	Bloqueio S concedido para key1.
5		map.getForUpdate(key1,v)	Bloqueado: T1 já possui um bloqueio U.
6	session.commit()		Conflito: O bloqueio U para key1 não pode ser atualizado.
7		session.commit()	Conflito: O bloqueio S para key1 não pode ser atualizado.

Tabela 16. Conflitos de uma única chave, continuação

	Encadeamento 1	**Encadeamento 2**	
1	session.begin()	session.begin()	Cada encadeamento estabelece uma transação independente
2	map.get(key1)		Bloqueio S concedido para key1.
3	map.getForUpdate(key1,v)		O bloqueio S é atualizado para um bloqueio U para key1
4		map.get(key1)	O bloqueio S é concedido para key1.
5		map.getForUpdate(key1,v)	Bloqueado: O Encadeamento 1 já possui um bloqueio U.
6	session.commit()		Conflito: O bloqueio U para key1 não pode ser atualizado para um bloqueio X porque o Encadeamento 2 possui um bloqueio S.

Se ObjectMap.getForUpdate for utilizado para evitar o bloqueio S, o conflito será evitado:

Tabela 17. Conflitos de uma única chave, continuação

	Encadeamento 1	Encadeamento 2	
1	session.begin()	session.begin()	Cada encadeamento estabelece uma transação independente.
2	map.getForUpdate(key1)		Bloqueio U concedido para o encadeamento 1 para key1.
3		map.getForUpdate(key1)	O pedido do bloqueio U é bloqueado.
4	map.update(key1,v)	<blocked>	
5	session.commit()	<blocked>	O bloqueio U para key1 pode ser atualizado com êxito para um bloqueio X.
6		<liberado>	O bloqueio U é finalmente concedido para key1 para o encadeamento 2.
7		map.update(key2,v)	Bloqueio U concedido ao encadeamento 2 para key2.
8		session.commit()	O bloqueio U para key1 pode ser atualizado com êxito para um bloqueio X.

Soluções:

1. Use o método getForUpdate ao invés do método get para adquirir um bloqueio U ao invés de um bloqueio S.
2. Utilize um nível de isolamento de transação da leitura confirmada para evitar reter bloqueios S. Reduzir o nível de isolamento de transação aumenta a possibilidade de leituras não-repetíveis. No entanto, leituras não repetitivas de um cliente são possíveis apenas se o cache de transição estiver explicitamente invalidado pelo mesmo cliente.
3. Utilize a estratégia de bloqueio otimista. A utilização da estratégia de bloqueio optimistic requer a manipulação de exceções de colisão otimistas.

- **Problema:** Diversos conflitos de chaves solicitados.

  Descrição: Este cenário descreve o que acontece se duas transações tentarem atualizar a mesma entrada diretamente e manter bloqueios S com outras entradas.

Tabela 18. Cenário de conflito de múltiplas chaves em ordem

	Encadeamento 1	Encadeamento 2	
1	session.begin()	session.begin()	Cada encadeamento estabelece uma transação independente.
2	map.get(key1)	map.get(key1)	O bloqueio S é concedido a ambas as transações para key1.
3	map.get(key2)	map.get(key2)	Bloqueio S concedido para ambas as transações para key2.
4	map.update(key1,v)		Nenhum bloqueio U. Atualização executada no cache transacional.
5		map.update(key2,v)	Nenhum bloqueio U. A atualização é executada no cache transacional.
6.	session.commit()		Bloqueado: O bloqueio S para key 1 não pode ser atualizado para um bloqueio X porque o encadeamento 2 possui um bloqueio S.

Tabela 18. Cenário de conflito de múltiplas chaves em ordem (continuação)

	Encadeamento 1	Encadeamento 2	
7		session.commit()	Conflito: O bloqueio S para key 2 não pode ser atualizado porque o encadeamento 1 possui um bloqueio S.

É possível utilizar o método ObjectMap.getForUpdate para evitar o bloqueio S, assim, é possível evitar o bloqueio:

Tabela 19. Cenário de conflito de múltiplas chaves em ordem, continuação

	Encadeamento 1	Encadeamento 2	
1	session.begin()	session.begin()	Cada encadeamento estabelece uma transação independente.
2	map.getForUpdate(key1)		Bloqueio U concedido para a transação T1 para key1.
3		map.getForUpdate(key1)	O pedido do bloqueio U é bloqueado.
4	map.get(key2)	<blocked>	Bloqueio S concedido para T1 para key2.
5	map.update(key1,v)	<blocked>	
6	session.commit()	<blocked>	O bloqueio U para key1 pode ser atualizado com êxito para um bloqueio X.
7		<liberado>	O bloqueio U finalmente é concedido para key1 para T2
8		map.get(key2)	Bloqueio S concedido para T2 para key2.
9		map.update(key2,v)	Bloqueio U concedido para T2 para key2.
10		session.commit()	O bloqueio U para key1 pode ser atualizado com êxito para um bloqueio X.

Soluções:

1. Use o método getForUpdate ao invés do método get para adquirir um bloqueio U diretamente para a primeira chave. Esta estratégia funciona apenas se o método order for determinístico.
2. Utilize um nível de isolamento de transação da leitura confirmada para evitar reter bloqueios S. Esta solução é a mais fácil para implementar se o método order não for determinístico. Reduzir o nível de isolamento de transação aumenta a possibilidade de leituras não-repetíveis. Entretanto, as leituras não-repetíveis são possíveis apenas se o cache de transição estiver explicitamente invalidado.
3. Utilize a estratégia de bloqueio otimista. A utilização da estratégia de bloqueio optimistic requer a manipulação de exceções de colisão otimistas.

- **Problema:** Fora de ordem com bloqueio U

  Descrição: Se a ordem na qual as chaves são solicitadas não puder ser garantida, um conflito ainda poderá ocorrer.

Tabela 20. Fora de ordem com cenário com bloqueio U

	Encadeamento 1	Encadeamento 2	
1	session.begin()	session.begin()	Cada encadeamento estabelece uma transação independente.
2	map.getforUpdate(key1)	map.getForUpdate(key2)	Bloqueios U concedidos com êxito para key1 e key2.

Tabela 20. Fora de ordem com cenário com bloqueio U (continuação)

	Encadeamento 1	Encadeamento 2	
3	map.get(key2)	map.get(key1)	Bloqueio S concedido para key1 e key2.
4	map.update(key1,v)	map.update(key2,v)	
5	session.commit()		O bloqueio U não pode ser atualizado para um bloqueio X porque T2 possui um bloqueio S.
6		session.commit()	O bloqueio U não pode ser atualizado para um bloqueio X porque T1 possui um bloqueio S.

Soluções:

1. Quebre todo o trabalho com um bloqueio U global único (mutex). Este método reduz a simultaneidade, mas trata todos os cenários quando o acesso e a ordem são não-determinísticos.
2. Utilize um nível de isolamento de transação da leitura confirmada para evitar reter bloqueios S. Esta solução é a mais fácil para implementar se o método order não for determinístico e fornece a maior quantidade de simultaneidade. Reduzir o nível de isolamento de transação aumenta a possibilidade de leituras não-repetíveis. Entretanto, as leituras não-repetíveis são possíveis apenas se o cache de transição estiver explicitamente invalidado.
3. Utilize a estratégia de bloqueio otimista. A utilização da estratégia de bloqueio optimistic requer a manipulação de exceções de colisão otimistas.

Conceitos relacionados

"Bloqueios" na página 246
Bloqueios têm ciclos de vida e os tipos diferentes de bloqueios são compatíveis com outros de várias maneiras. Os bloqueios devem ser manipulados na ordem correta para evitar cenários de conflito.

IBM Support Assistant for WebSphere eXtreme Scale

É possível usar o IBM Support Assistant para coletar dados, analisar sintomas e acessar informações do produto.

IBM Support Assistant Lite

O IBM Support Assistant Lite for WebSphere eXtreme Scale fornece uma coleta de dados automática e suporte à análise de sintomas para cenários de determinação de problemas.

O IBM Support Assistant Lite reduz o período de tempo que leva para reproduzir um problema com os níveis de rastreamento Reliability, Availability and Serviceability adequados configurados (os níveis de rastreamento são automaticamente configurados pela ferramenta) para simplificar a determinação de problemas. Se você precisar de assistência adicional, o IBM Support Assistant Lite também reduz o esforço necessário para enviar as informações de log apropriadas para o IBM Support.

O IBM Support Assistant Lite é incluído em cada instalação do WebSphere eXtreme Scale Versão 7.1.0

IBM Support Assistant

O IBM® Support Assistant (ISA) fornece acesso rápido a recursos do produto, de educação e de suporte que podem ajudar você a responder questões e resolver sozinho problemas com os produtos de software IBM, sem precisar entrar em contato com o IBM Support. Plug-ins diferentes específicos do produto permitem customizar o IBM Support Assistant para os produtos particulares que você instalou. O IBM Support Assistant também podem coletar dados do sistema, arquivos de log e outras informações para ajudar o IBM Support a determinar a causa de um problema particular.

O IBM Support Assistant é um utilitário a ser instalado em sua estação de trabalho, não diretamente no sistema do servidor WebSphere eXtreme Scale em si. Os requisitos de memória e de recurso para o Assistant podem afetar negativamente o desempenho do sistema do servidor WebSphere eXtreme Scale. Os componentes de diagnóstico móveis incluídos são projetados para um impacto mínimo para a operação normal de um servidor.

É possível usar o IBM Support Assistant para ajudá-lo das seguintes maneiras:

- Para pesquisar em fontes de conhecimento e de informações IBM e não IBM em vários produtos IBM para responder uma questão ou resolver um problema
- Para localizar informações adicionais por meio de recursos da web específicos do produto; incluindo páginas iniciais do produto e de suporte, grupos de notícias e fóruns de clientes, qualificações e recursos de treinamento e informações sobre como resolver problemas e as perguntas mais comuns
- Para aumentar sua capacidade para diagnosticar problemas específicos do produto com as ferramentas de diagnóstico de destino disponíveis no Support Assistant
- Para simplificar a coleta de dados de diagnóstico para ajudar você e a IBM a resolver seus problemas (coletar dados gerais ou dados específicos do produto/sintoma)
- Para ajudar no relatório de incidentes de problemas ao IBM Support por meio de uma interface online customizada, incluindo a capacidade de conectar os dados de diagnóstico mencionados acima ou qualquer outra informação a incidentes novos ou existentes

Finalmente, é possível usar o recurso Updater integrado para obter suporte para produtos de software e recursos adicionais assim que eles forem disponibilizados. Para configurar o IBM Support Assistant para ser usado com o WebSphere eXtreme Scale, primeiro instale o IBM Support Assistant usando os arquivos fornecidos na imagem transferida por download da página da web Visão Geral do IBM Support em: http://www-947.ibm.com/support/entry/portal/Overview/Software/Other_Software/IBM_Support_Assistant. A seguir, use o IBM Support Assistant para localizar e instalar qualquer atualização do produto. Também é possível optar por instalar plug-ins disponíveis para outro software IBM em seu ambiente. Mais informações e a versão mais recente do IBM Support Assistant estão disponíveis a partir da página da web do IBM Support Assistant em: http://www.ibm.com/software/support/isa/.

Avisos

Referências nesta publicação a produtos, programas ou serviços IBM não significam que a IBM pretende torná-los disponíveis em todos os países onde opera. Qualquer referência a produtos, programas ou serviços IBM não significa que apenas produtos, programas ou serviços IBM possam ser utilizados. Qualquer produto, programa ou serviço funcionalmente equivalente, que não infrinja nenhum direito de propriedade intelectual da IBM poderá ser utilizado em substituição a este produto, programa ou serviço. A avaliação e verificação da operação em conjunto com outros produtos, exceto aqueles expressamente designados pela IBM, são de inteira responsabilidade do Cliente.

A IBM pode ter patentes ou solicitações de patentes pendentes relativas a assuntos tratados nesta publicação. O fornecimento desta publicação não garante ao Cliente nenhum direito sobre tais patentes. Pedidos de licença devem ser enviados, por escrito, para:

Gerência de Relações Comerciais e Industriais da IBM Brasil
Av. Pasteur, 138-146
Botafogo
Rio de Janeiro, RJ
CEP 22290-240

Licenciados deste programa que desejam obter mais informações sobre este assunto com objetivo de permitir: (i) a troca de informações entre programas criados independentemente e outros programas (incluindo este) e (ii) a utilização mútua das informações trocadas, devem entrar em contato com:

Gerência de Relações Comerciais e Industriais da IBM Brasil
Av. Pasteur, 138-146,
Botafogo
Rio de Janeiro, RJ
CEP 22290-240
Brasil

Tais informações podem estar disponíveis, sujeitas a termos e condições apropriados, incluindo em alguns casos, o pagamento de uma taxa.

Marcas Registradas

Os termos a seguir são marcas registradas da IBM Corporation nos Estados Unidos e/ou em outros países:

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java e todas as marcas registradas baseadas em Java são marcas registradas da Sun Microsystems, Inc. nos Estados Unidos e/ou em outros países.

LINUX é uma marca registrada de Linus Torvalds nos Estados Unidos e/ou em outros países.

Microsoft, Windows, Windows NT e o logotipo do Windows são marcas registradas da Microsoft Corporation nos Estados Unidos e/ou em outros países.

UNIX é uma marca registrada do The Open Group nos Estados Unidos e em outros países.

Outros nomes de empresas, produtos e serviços podem ser marcas registradas ou marcas de serviços de terceiros.

Índice Remissivo

A

B

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

U

V

W

X

Impresso no Brasil