WebSphere® eXtreme Scale Versão 7.1
Guia de Programação

Guia de Programação do WebSphere eXtreme Scale

IBM

Esta edição aplica-se à versão 7, release 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 *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.
- 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.

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.

Quem Deve Utilizar este Manual

Este manual é destinado principalmente a desenvolvedores de aplicativos.

Como este Manual Está Estruturado

O manual contém informações sobre os seguintes tópicos principais:

- **Capítulo 1** inclui informações sobre introdução ao WebSphere eXtreme Scale.
- **Capítulo 2** inclui informações sobre como programar o WebSphere eXtreme Scale.
- **Capítulo 3** inclui informações sobre acesso aos dados.
- **Capítulo 4** inclui informações sobre as APIs e plug-ins do Sistema.
- **Capítulo 5** inclui informações sobre a integração com a estrutura Spring.
- **Capítulo 6** inclui informações sobre a API de segurança.
- **Capítulo 7** inclui informações sobre a API de administração.
- **Capítulo 8** inclui informações sobre as considerações de desempenho.
- **Capítulo 9** inclui informações sobre a resolução de problemas.
- **Capítulo 10** inclui o glossário do produto.

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. Introdução ao WebSphere eXtreme Scale

Depois de instalar o WebSphere eXtreme Scale em um ambiente independente, use as seguintes etapas como uma simples introdução para sua capacidade como uma grade de dados em memória.

A instalação independente do WebSphere eXtreme Scale inclui uma amostra que pode ser usada para verificar a sua instalação e ver como uma grade e um cliente simples do eXtreme Scale podem ser usados. A amostra de iniciação está localizada no diretório *installRoot*/ObjectGrid/gettingstarted.

A amostra de iniciação é fornecida para uma introdução rápida à operação básica e de funcionalidade do eXtreme Scale. A amostra consiste de scritps de shell e lote projetados para iniciar uma grade simples com muito pouca customização necessária. Além disso, um programa cliente, incluindo a fonte, é fornecido para executar funções simples de criação, leitura, atualização e exclusão (CRUD) nesta grade básica.

Scripts e suas Funções

Essa amostra fornece os seguintes scripts:

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`

O script runcat.sh|bat inicia o processo de serviço de catálogo do eXtreme Scale no sistema local.

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

O script runcontainer.sh|bat inicia um processo de servidor de contêiner. É possível executar esse script várias vezes com nomes do servidor exclusivos especificados para iniciar qualquer número de contêineres. Essas instâncias podem funcionar juntas para hospedar informações particionadas e redundantes na grade.

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

O script runclient.sh|bat executa o cliente CRUD simples e inicia a operação especificada.

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

Para *command*, use uma das seguintes opções:

- Especifique `i` para inserir *value2* na grade 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*

Nota: O arquivo *installRoot*`/ObjectGrid/ gettingstarted/src/Client.java` é o programa cliente que demonstra como se conectar a um servidor de catálogo, obter uma instância de ObjectGrid, e usar a API de ObjectMap.

Etapas Básicas

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

1. Abra uma janela de sessão do terminal ou linha de comandos.
2. Utilize o seguinte comando para navegar para o diretório `gettingstarted`:

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

   Substitua *installRoot* pelo caminho para o diretório-raiz da instalação do eXtreme Scale ou o caminho do arquivo-raiz do trial do eXtreme Scale extraído *installRoot*.
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.
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. Repita as etapas 5 e 6 se desejar iniciar mais instâncias do servidor de contêiner para suportar a replicação.
5. Abra outra janela de sessão de terminal ou linha de comandos para executar comandos do cliente.
   - Incluir dados na grade:
     - UNIX Linux `./runclient.sh i key1 helloWorld`
     - Windows `runclient.bat i key1 helloWorld`
   - Procurar e exibir o valor:
     - UNIX Linux `./runclient.sh g key1`
     - Windows `runclient.bat g key1`
   - Atualizar o valor:
     - UNIX Linux `./runclient.sh u key1 goodbyeWorld`
     - Windows `runclient.bat u key1 goodbyeWorld`
   - Excluir o valor:
     - UNIX Linux `./runclient.sh d key1`
     - Windows `runclient.bat d key1`
6. Use <ctrl+c> para parar o processo de serviço de catálogo e os servidores de contêiner em suas respectivas janelas.

Definindo um ObjectGrid

A amostra usa os arquivos `objectgrid.xml` e `deployment.xml` que estão no diretório *installRoot*`/ObjectGrid/gettingstarted/xml` para iniciar um servidor de contêineres. O arquivo `objectgrid.xml` é o arquivo XML descritor do ObjectGrid e o arquivo `deployment.xml` é o arquivo XML descritor de política de implementação do ObjectGrid. Ambos os arquivos juntos definem uma topologia de ObjectGrid distribuída.

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 BackingMap. Esses BackingMaps são o armazenamento de dados atual para dados em 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. Este arquivo de exemplo define o Grid ObjectGrid com BackingMaps 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 XML descritor de política de implementação é passado para um servidor de contêineres ObjectGrid 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 XML do seu 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.

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.

Usando o ObjectGrid

O arquivo `Client.java` no diretório *`installRoot`*`/ObjectGrid/gettingstarted/src/` é o programa cliente que demonstra como se conectar a um servidor de catálogos, obter uma instância de ObjectGrid, e usar a API de ObjectMap.

Da perspectiva de um aplicativo cliente, o uso do WebSphere eXtreme Scale pode ser dividido nas seguintes etapas.

1. Conexão com o serviço de catálogo por meio da obtenção de uma instância de ClientClusterContext.
2. Obtenção de uma instância do ObjectGrid do cliente.
3. Obtenção de uma instância da Sessão.
4. Obtenção de uma instância do ObjectMap.
5. Uso de métodos ObjectMap.

1. **Conectando ao serviço de catálogo por meio da obtenção de uma instância do ClientClusterContext**

   Para se conectar a um servidor de catálogos, use o método connect da API ObjectGridManager. O método connect usado requer apenas o terminal do servidor de catálogos no formato de `hostname:port`, como `localhost:2809`. Se a conexão com o servidor de catálogos for bem-sucedida, o método connect retornará uma instância do ClientClusterContext. A instância do ClientClusterContext é necessária para a obtenção do ObjectGrid a partir da

API do ObjectGridManager. O trecho de código a seguir demonstra como se conectar a um servidor de catálogos e obter uma instância do ClientClusterContext.

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

2. **Obtendo uma Instância do ObjectGrid**

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

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

3. **Obtendo uma Instância da 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. **Obtendo uma Instância do ObjectMap**

   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 passar o nome do mapa como parâmetro para o método getMap para obter a instância do ObjectMap. O fragmento de código a seguir demonstra como obter ObjectMap chamando o método getMap da API de Sessão.

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

5. **Usando os métodos ObjectMap**

   Após a obtenção de um ObjectMap, é 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 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();
   ```

Informações adicionais

Esta amostra demonstra como iniciar o servidor de catálogos e o servidor de contêiner e usar a API do ObjectMap em um ambiente independente. Também é possível usar a API do EntityManager.

Em um ambiente WebSphere Application Server com oWebSphere eXtreme Scale instalado ou ativado, o cenário mais comum é uma topologia conectada à rede. Em uma topologia conectada à rede, o servidor de catálogos é hospedado no processo do gerenciador de implementação do WebSphere Application Server e cada

instância doWebSphere Application Server hospeda um servidor eXtreme Scale automaticamente. Os aplicativos Java™ Platform, Enterprise Edition precisam somente incluir ambos, o arquivo XML descritor do ObjectGrid e o arquivo XML descritor da política de implementação do ObjectGrid, no diretório `META-INF` de qualquer módulo e o ObjectGrid se torna disponível automaticamente. Então, um aplicativo pode se conectar a um servidor de catálogos disponível localmente e obter uma instância do ObjectGrid para uso.

Capítulo 2. Preparando para Desenvolver Aplicativos

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

Interfaces de Programação do WebSphere eXtreme Scale

O WebSphere eXtreme Scale fornece diversos recursos que são acessados programaticamente usando a linguagem de programação Java por meio 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.

7.0.0.0 FIX 2+ 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.

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, o cliente eXtreme Scale e os processos do contêiner devem incluir as classes ou JARs no caminho de classe ao iniciar o processo. Ao projetar um aplicativo para uso com o eXtreme Scale, separe qualquer lógica de negócios dos objetos de dados persistentes.

Consulte Carregamento de classe no WebSphere Application Server Centro de Informações de Implementação de Rede 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*.

Para obter as configurações relacionadas ao uso do agente de instrumentação do WebSphere eXtreme Scale, consulte o tópico do agente de instrumentação no *Guia de Programação*.

Configurando um Ambiente de Desenvolvimento

Configure um ambiente de desenvolvimento integrado baseado no Eclipse para construir e executar um aplicativo SE Java com o eXtreme Scale.

Procedimento

- Configure o Eclipse para construir e executar um aplicativo SE Java com o eXtreme Scale.
  1. Defina uma biblioteca de usuário para permitir que seu aplicativo faça referência a interfaces de programação de aplicativo do 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) Navegue e selecione os arquivos `objectgrid.jar` e `cglib.jar` do diretório `wxs_root/lib`. Clique em **OK**.
2) Para incluir Javadoc para APIs ObjectGrid, selecione o local do Javadoc para o arquivo `objectgrid.jar` que você incluiu 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 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

Tenha você 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 Versão 7.1.
- Faça download e extraia a versão de teste do WebSphere eXtreme Scale.
- Instale o Apache Tomcat 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 seu 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 que execute 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...**, navegue para wxs_home/lib/objectgrid.jar e selecione-o. 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 **Navegar...** e navegue para tomcat_root e selecione-o. Clique em **OK**.

e. Clique em **Avançar**.

f. Selecione seu aplicativo Java EE na área de janela Disponível à esquerda e 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. Todos os erros para o Projeto devem ser colocados na área de janela Problemas. Se não forem, execute as etapas a seguir.

a. Clique em **Projeto** → **Limpar** → *project_name*. Clique em **OK**. Construa o projeto.

b. Clique com o botão direito do mouse no projeto Java EE desejado 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 Apache Tomcat, eXtremeScaleClient e Java 1.5 JRE estejam no caminho.
- **Para aplicativos do servidor:** Certifique-se de que Apache Tomcat, eXtremeScale e 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 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

Se você tiver configurado e executado um aplicativo da Web para usar um cliente do WebSphere eXtreme Scale no Rational Application Developer, desenvolva, agora, um servlet que use as APIs do WebSphere eXtreme Scale para armazenar e recuperar dados de uma grade remota do WebSphere eXtreme Scale.

Se tiver ativado um aplicativo Java EE na interface do Rational Application Developer com uma instalação independente do WebSphere eXtreme Scale, desenvolva, agora, um servlet que use as APIs de sistema do WebSphere eXtreme Scale para iniciar e parar serviços de catálogo.

Executando um Aplicativo Cliente ou do Servidor 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 do WebSphere Application Server integrado no Rational Application Developer. Se você estiver configurando um servidor, iniciar o WebSphere Application Server automaticamente inicia 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 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.

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 em **WebSphere Application Server v7.0** e selecione **Iniciar**.

Capítulo 3. 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.

interface ObjectGrid

Os seguintes métodos permitem interagir com uma instância ObjectGrid.

Criar e Inicializar

Consulte o tópico da interface ObjectGridManager para obter as etapas necessárias para criação de uma instância do ObjectGrid. Existem dois métodos distintos para criar uma instância do ObjectGrid: programaticamente ou com arquivos de configuração XML. Consulte a documentação da API para obter informações adicionais.

Métodos Get ou Set e Factory

Quaisquer métodos configurados devem ser chamados antes de inicializar a instância do ObjectGrid. Se você chamar o método set depois de o método initialize ser chamado, o resultado será uma exceção java.lang.IllegalStateException. Cada um dos métodos getSession da interface ObjectGrid também chama implicitamente o método initialize. Portanto, é necessário chamar os métodos set antes de chamar qualquer um dos métodos getSession. A única exceção desta regra é com a configuração, a inclusão e a remoção de objetos ObjectGridEventListener. Esses objetos podem ser processados após a conclusão do processo de inicialização.

A interface ObjectGrid contém os seguintes métodos principais.

Tabela 1. interface ObjectGrid. Métodos principais de ObjectGrid.

Método	Descrição
BackingMap defineMap(String name);	defineMap: é um método factory para definir um BackingMap exclusivamente denominado. Para obter informações adicionais sobre os mapas de apoio, consulte a Interface BackingMap.
BackingMap getMap(String name);	getMap: Retorna um BackingMap anteriormente definido chamando defineMap. Utilizando este método, é possível configurar o BackingMap, se ainda não estiver configurado por meio da configuração XML.

Tabela 1. interface ObjectGrid (continuação). Métodos principais de ObjectGrid.

Método	Descrição
BackingMap createMap(String name);	createMap: Cria um BackingMap, mas não o armazena em cache para uso por este ObjectGrid. Use este método com o método setMaps(List) da interface ObjectGrid, que captura BackingMaps para uso com este ObjectGrid. Utilize estes métodos quando estiver configurando um ObjectGrid com Spring Framework.
void setMaps(List mapList);	setMaps: Limpa quaisquer BackingMaps que foram anteriormente definidos neste ObjectGrid e os substitui pela lista de BackingMaps que é fornecida.
public Session getSession() throws ObjectGridException, TransactionCallbackException;	getSession: Retorna um Session, que fornece a funcionalidade begin, commit e rollback para uma Unidade de Trabalho. Para obter informações adicionais sobre os objetos Session, consulte a interface Session.
Session getSession(CredentialGenerator cg);	getSession(CredentialGenerator cg): Obtém uma sessão com um objeto CredentialGenerator. Este método pode ser chamado apenas pelo cliente do ObjectGrid em um ambiente do servidor do cliente.
Session getSession(Subject subject);	getSession(Subject subject): Permite o uso de um objeto Subject específico ao invés de um configurado no ObjectGrid para obter um Session.
void initialize() throws ObjectGridException;	initialize: O ObjectGrid é inicializado e está disponível para uso geral. Este método é chamado implicitamente quando o método getSession é chamado, se o ObjectGrid não estiver em um estado inicializado.
void destroy();	destroy: A estrutura é desmontada e não pode ser utilizada após este método ser chamado.
void setTxTimeout(int timeout);	setTxTimeout: Utilize este método para configurar a quantidade de tempo, em segundos, que uma transação que é iniciada por uma sessão que esta instância do ObjectGrid criou tem permissão para ser concluída. Se uma transação não for concluída dentro de uma quantidade de tempo especificada, a Sessão que iniciou a transação será marcada como "expirada". Uma Sessão marcada com tempo limite expirado faz com que o próximo método ObjectMap que for chamado pela Sessão marcada resulte em uma exceção . O objeto Session é marcado apenas como rollback, o que faz com que ocorra um retrocesso na transação mesmo se o aplicativo chamar o método commit ao invés do método rollback após a exceção TransactionTimeoutException ser capturada pelo aplicativo. Um valor de tempo limite de 0 indica que a transação tem permissão para uma quantidade de tempo ilimitada para ser concluída. A transação não expira se um valor do tempo limite de 0 for utilizado. Se este método não for chamado, então, qualquer objeto Session que é retornado pelo método getSession desta interface possui um valor de tempo limite configurado com 0, por padrão. Um aplicativo pode substituir a configuração de tempo limite da transação em uma base por Sessão utilizando o método setTransactionTimeout da interface com.ibm.websphere.objectgrid.Session. Você também pode configurar o tempo limite da transação com o arquivo `objectGrid.xml` no caso distribuído.
int getTxTimeout();	getTxTimeout: Retorna o valor de tempo limite da transação em segundos. Este método retorna o mesmo valor que foi transmitido como o parâmetro de tempo limite no método setTxTimeout. Se o método setTxTimeout não foi chamado, então, o método retorna 0 para indicar que a transação tem permissão para uma quantidade de tempo ilimitada para ser concluída.
public int getObjectGridType();	Retorna o tipo de ObjectGrid. O valor de retorno é equivalente a uma das constantes declaradas nesta interface: LOCAL, SERVER ou CLIENT.
public void registerEntities(Class[] entities);	Registre uma ou mais entidades com base nos metadados da classe. O registro da entidade é necessário antes da inicialização do ObjectGrid para ligar uma Entidade a um BackingMap e qualquer índice definido. Esse método pode ser chamado várias vezes.
public void registerEntities(URL entityXML);	Registra uma ou mais entidades de um arquivo XML da entidade. O registro da entidade é necessário antes da inicialização do ObjectGrid para ligar uma Entidade a um BackingMap e qualquer índice definido. Esse método pode ser chamado várias vezes.
void setQueryConfig(QueryConfig queryConfig);	Configure o objeto QueryConfig para este ObjectGrid. Um objeto QueryConfig fornece configurações de consulta para executar consultas de objeto nos mapas nesse ObjectGrid.
//Keywords	
void associateKeyword(Serializable parent, Serializable child);	Reprovado: Utilize o Índice ou a função de consulta para obter Objects com atributos específicos. O método associateKeyword fornece um mecanismo de invalidação flexível baseado em palavras-chave. Este método vincula as duas palavras-chave em um relacionamento direcional. Se o pai for invalidado, o filho também será invalidado. A invalidação do filho não tem nenhum impacto sobre o pai.
//Security	
void setSecurityEnabled()	setSecurityEnabled: Ativa a segurança. A segurança é desativada por padrão.

Tabela 1. interface ObjectGrid (continuação). Métodos principais de ObjectGrid.

Método	Descrição
void setPermissionCheckPeriod(long period);	setPermissionCheckPeriod: Este método obtém um parâmetro único que indica com que frequência verificar a permissão que é utilizada para permitir o acesso de um cliente. Se o parâmetro for 0, todos os métodos solicitam ao mecanismo de autorização, autorização JAAS ou autorização customizada, para verificar se o objeto atual tem permissão. Esta estratégia pode causar problemas de desempenho, dependendo da implementação de autorização. No entanto, este tipo de autorização está disponível se for requerido. Alternativamente, se o parâmetro for menor do que 0, ele indica o número de milissegundos a armazenar um conjunto de permissões em cache antes de retornar para o mecanismo de autorização para atualizá-las. Este parâmetro fornece um desempenho muito melhor mas, se as permissões de backend forem alteradas durante este período, o ObjectGrid poderá permitir ou impedir o acesso mesmo que o provedor de segurança de backend tenha sido modificado.
void setAuthorizationMechanism(int authMechanism);	setAuthorizationMechanism: Configurar o mecanismo de autorização. O padrão é SecurityConstants.JAAS_AUTHORIZATION.
setMapAuthorization(MapAuthorization ma);	Reprovado: Use o método setObjectGridAuthorization (ObjectGridAuthorization) em vez do plug-in de autorizações customizadas. setMapAuthorization: Configura o plug-in MapAuthorization para esta instância do ObjectGrid. Este plug-in pode ser utilizado para autorizar acessos de ObjectMap ou JavaMap aos proprietários que estão contidos no objeto Subject. Uma implementação típica deste plug-in é recuperar os proprietários do objeto Subject e, em seguida, verificar se as permissões especificadas são concedidas aos proprietários.
setSubjectSource(SubjectSource ss);	setSubjectSource: Configura o plug-in SubjectSource. Este plug-in pode ser utilizado para obter um objeto Subject que representa o cliente do ObjectGrid. Este subject é utilizado para autorização do ObjectGrid. O método SubjectSource.getSubject é chamado pelo tempo de execução do ObjectGrid quando o método ObjectGrid.getSession é utilizado para obter uma sessão e a segurança está ativada. Este plug-in é útil para um cliente já autenticado: ele pode recuperar o objeto Subject autenticado e, em seguida, transmitir para a instância do ObjectGrid. Outra autenticação não é necessária.
setSubjectValidation(SubjectValidation sv);	setSubjectValidation: Configura o plug-in SubjectValidation para esta instância do ObjectGrid. Este plug-in pode ser utilizado para validar se um subject javax.security.auth.Subject transmitido para o ObjectGrid é um subject válido que não foi violado. 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, um criador do subject pode não saber se o Subject foi violado. Neste caso, este plug-in não deve ser utilizado.
void setObjectGridAuthorization(ObjectGridAuthorization ogAuthorization);	Configura o ObjectGridAuthorization para esta instância do ObjectGrid. Transmitir nulo para este método remove um objeto ObjectGridAuthorization configurado anteriormente de uma chamada anterior deste método e indica que <code>ObjectGrid</code> não está associado a um objeto ObjectGridAuthorization. Esse método deve ser utilizado somente quando a segurança do ObjectGrid estiver ativada. Se a segurança do ObjectGrid estiver desativada, o objeto ObjectGridAuthorization não será utilizado. Um plug-in do ObjectGridAuthorization pode ser utilizado para autorizar o acesso ao ObjectGrid e aos mapas. A partir do XD 6.1, o setMapAuthorization é reprovado e o setObjectGridAuthorization é recomendado para uso. Entretanto, se o plug-in de MapAuthorization e o plug-in de ObjectGridAuthorization forem utilizados, o ObjectGrid utilizará o MapAuthorization fornecido para autorizar os acessos ao mapa, mesmo que seja reprovado.

Interface do ObjectGrid: Plug-ins

A interface ObjectGrid possui diversos pontos de plug-in opcionais para interações mais extensíveis.

```
void addEventListener(ObjectGridEventListener cb);
void setEventListeners(List cbList);
void removeEventListener(ObjectGridEventListener cb);
void setTransactionCallback(TransactionCallback callback);
int reserveSlot(String);
// Security related plug-ins
void setSubjectValidation(SubjectValidation subjectValidation);
void setSubjectSource(SubjectSource source);
void setMapAuthorization(MapAuthorization mapAuthorization);
```

- ObjectGridEventListener: Uma interface ObjectGridEventListener é utilizada para receber notificações quando ocorrem eventos significativos no ObjectGrid. Estes eventos incluem inicialização do ObjectGrid, início de uma transação,

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 ObjectGrid. Estes listeners estão associados a cada Sessão. Consulte Interface de Listeners e Sessão para obter informações adicionais.

- TransactionCallback: Uma interface de listener TransactionCallback permite que eventos transacionais, tais como sinais begin, commit e rollback, para envia para esta interface. Geralmente, uma interface do listener TransactionCallback é utilizada com um Utilitário de Carga. Para obter informações adicionais, consulte o plug-in deTransactionCallback e os Utilitários de Carga. Estes eventos podem então ser utilizados para coordenar transações com um recurso externo ou em vários utilitários de carga.
- reserveSlot: Permite que plug-ins neste ObjectGrid reservem slots para uso em instâncias do objeto que possuem slots como TxID.
- SubjectValidation. Se a segurança estiver ativada, este plug-in pode ser utilizado para validar uma classe javax.security.auth.Subject que é passada para o ObjectGrid.
- MapAuthorization. Se a segurança estiver ativada, este plug-in poderá ser utilizado para autorizar acessos do ObjectMap aos proprietários representados pelo objeto Subject.
- SubjectSource: Se a segurança estiver ativada, este plug-in pode ser utilizado para obter um objeto Subject que representa o cliente do ObjectGrid. Este subject é então utilizado para autorização do ObjectGrid.

Para obter informações adicionais sobre plug-ins, consulte a introdução aos plug-ins no *Guia de Programação*.

Interface BackingMap

Cada instância do ObjectGrid contém uma coleta de objetos de BackingMap. Use o método defineMap ou o método createMap da interface ObjectGrid para nomear e incluir cada BackingMap em uma instância do ObjectGrid. Estes métodos retornam uma instância do BackingMap que é então utilizada para definir o comportamento de um Mapa individual. Um BackingMap pode ser considerado como um cache de memória de dados com commit para um mapa individual.

Interface Session

A interface Session é usada para iniciar uma transação e obter o ObjectMap ou JavaMap que é necessário para execução de interação transacional entre um aplicativo e um objeto BackingMap. No entanto, as alterações na transação não serão aplicadas ao objeto de BackingMap até que a transação seja confirmada. Um BackingMap pode ser considerado como um cache de memória de dados com commit para um mapa individual. Para obter informações adicionais, consulte as informações sobre o uso de Sessões para acesso a dados no *Guia de Programação*.

A interface BackingMap oferece métodos para configuração de atributos do BackingMap. Alguns dos métodos set permitem a extensibilidade de um BackingMap por meio de vários plug-ins projetados customizados. Consulte a lista dos métodos de configuração a seguir para configurar atributos e fornece suporte a plug-ins desenvolvidos de forma customizada:

```
// For setting BackingMap attributes.
public void setReadOnly(boolean readOnlyEnabled);
public void setNullValuesSupported(boolean nullValuesSupported);
public void setLockStrategy( LockStrategy lockStrategy );
```

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

// For setting an optional custom plug-in provided by application.
public abstract void setObjectTransformer(ObjectTransformer t);
public abstract void setOptimisticCallback(OptimisticCallback checker);
public abstract void setLoader(Loader loader);
public abstract void setPreloadMode(boolean async);
public abstract void setEvictor(Evictor e);
public void setMapEventListeners( List /*MapEventListener*/ eventListenerList );
public void addMapEventListener(MapEventListener eventListener );
public void removeMapEventListener(MapEventListener eventListener );
public void addMapIndexPlugin(MapIndexPlugin index);
public void setMapIndexPlugins(List /\* MapIndexPlugin \*/ indexList );
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb);
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback cb);
public void removeDynamicIndex(String name);
```

Existe um método get correspondente para cada um dos métodos set listados.

Atributos de BackingMap

Cada BackingMap possui os seguintes atributos que podem ser configurados para modificar ou controlar o comportamento de BackingMap:

- ReadOnly: Este atributo indica se o Mapa é um Mapa somente leitura ou um Mapa de leitura e gravação. Se este atributo nunca for configurado para o Mapa, o Mapa será padronizado para ser um Mapa de leitura e gravação. Quando um BackingMap é configurado para ser de leitura, o ObjectGrid otimiza o desempenho para de leitura quando possível.
- NullValuesSupported: Este atributo indica se um valor nulo pode ser colocado no Mapa. Se este atributo nunca for configurado, o Mapa não suportará valores nulos. Se o Mapa suportar valores nulos, uma operação get que retorna nulo poderá significar que o valor é nulo ou o mapa não contém a chave especificada pela operação get.
- LockStrategy: Este atributo determina se um gerenciador de bloqueios é utilizado por este BackingMap. Se um gerenciador de bloqueios for utilizado, o atributo LockStrategy será utilizado para indicar se será utilizada uma abordagem de bloqueio otimista ou pessimista para bloquear as entradas do mapa. Se este atributo não for configurado, será utilizado o LockStrategy otimista. Consulte o tópico Bloqueio para obter detalhes sobre as estratégias de bloqueio suportadas.
- CopyMode: Este atributo determina se uma cópia de um objeto de valor é feita no BackingMap quando um valor é lido a partir do mapa ou é colocado no BackingMap durante o ciclo de commit de uma transação. Vários modos de cópia são suportados para permitir que o aplicativo faça o equilíbrio entre desempenho e integridade de dados. Se este atributo não estiver configurado, então, o modo de cópia COPY_ON_READ_AND_COMMIT é utilizado. Este modo de cópia não tem o melhor desempenho, mas tem a maior proteção contra problemas de integridade de dados. Se o BackingMap estiver associado a uma entidade da API EntityManager, então a configuração CopyMode não tem efeito a menos que o valor esteja configurado como COPY_TO_BYTES. Se qualquer outro CopyMode estiver configurado, ele sempre será configurado como

NO_COPY. Para substituir o CopyMode para uma mapa de entidades, use o método ObjectMap.setCopyMode. Para obter informações mais detalhadas sobre os modos de cópia, consulte as informações sobre as boas práticas do método CopyMode no *Guia de Programação*.

- CopyKey: Este atributo determina se o BackingMap faz uma cópia de um objeto de chave quando uma entrada é criada pela primeira vez no mapa. A ação padrão é não fazer uma cópia de objetos de chave, porque as chaves normalmente são objetos inalteráveis.
- NumberOfBuckets: Este atributo indica o número de depósitos de hash a serem utilizados pelo BackingMap. A implementação de BackingMap utiliza um mapa hash para sua implementação. Se existirem muitas entradas no BackingMap, mais depósitos significam melhor desempenho. O número de chaves que possuem o mesmo depósito se torna mais baixo conforme aumenta o número de depósitos. Mais depósitos também significam mais simultaneidade. Este atributo é útil para ajuste de desempenho. Um valor padrão indefinido será utilizado se o aplicativo não configurar o atributo NumberOfBuckets.
- NumberOfLockBuckets: Este atributo indica o número de depósitos de bloqueio que são utilizados pelo gerenciador de bloqueios para este BackingMap. Quando LockStrategy estiver configurado como OPTIMISTIC ou PESSIMISTIC, será criado um gerenciador de bloqueios para o BackingMap. O gerenciador de bloqueios utiliza um mapa hash para rastrear as entradas que estão bloqueadas por uma ou mais transações. Se existirem muitas entradas no mapa de hash, mais depósitos de bloqueios resultarão em melhor desempenho, porque o número de chaves que colidem no mesmo depósito é mais baixo conforme aumenta o número de depósitos. Mais depósitos de bloqueios também significam mais simultaneidade. Quando o atributo LockStrategy for configurado como NONE, nenhum gerenciador de bloqueios será utilizado por este BackingMap. Neste caso, a configuração de numberOfLockBuckets não tem nenhum efeito. Se este atributo não for configurado, será utilizado um valor indefinido.
- LockTimeout: Este atributo é utilizando quando o BackingMap está utilizando um gerenciador de bloqueios. O BackingMap utilizará um gerenciador de bloqueios quando o atributo LockStrategy for configurado como OPTIMISTIC ou PESSIMISTIC. O valor de atributo está em segundos e determina por quanto tempo o gerenciador de bloqueios espera que um bloqueio seja concedido. Se este atributo não estiver configurado, então são usados 15 segundos como o valor de LockTimeout. Consulte Bloqueio Pessimista para obter detalhes sobre as exceções de tempo limite de espera de bloqueio que podem ocorrer.
- TtlEvictorType: Cada BackingMap possui seu próprio evictor time to live integrado que utilize um algoritmo baseado em tempo para determinar quais entradas de mapa despejar. Por padrão, o evictor time to live integrado não está ativo. É possível ativar o tempo para ativar evictor chamando o método setTtlEvictorType com um desses valores: CREATION_TIME, LAST_ACCESS_TIME, LAST_UPDATE_TIME ou NONE. Um valor de CREATION_TIME indica que o evictor inclui o atributo TimeToLive no horário em que a entrada de mapa foi criada no BackingMap para determinar quando o evictor deve despejar a entrada de mapa do BackingMap. Um valor de LAST_ACCESS_TIME (ou LAST_UPDATE_TIME) indica que o evictor inclui o atributo TimeToLive na hora que a entrada de mapa foi acessada pela última vez (ou acessada *e* atualizada) por alguma transação que o aplicativo está executando para determinar quando o evictor deve liberar a entrada de mapa. A entrada do mapa será liberada apenas se uma entrada de mapa nunca for acessada por nenhuma transação durante um período de tempo especificado pelo atributo TimeToLive. Um valor de NONE indica que o evictor deve permanecer inativo e nunca liberar nenhuma das entradas de mapa. Se este

atributo nunca for configurado, NONE será utilizado como o padrão e o evictor time to live não será ativado. Consulte Evictors para obter detalhes sobre o evictor time to live interno.

- TimeToLive: Este atributo é utilizado para especificar o número de segundos que o evictor time to live integrado precisa incluir no horário da criação ou no horário do último acesso para cada entrada, conforme descrito para o atributo TtlEvictorType. Se este atributo nunca for configurado, será utilizado o valor especial de zero para indicar que o time to live é infinito. Se este atributo for configurado como infinito, as entradas do mapa nunca serão liberadas pelo evictor.

O exemplo a seguir demonstra como definir o someMap BackingMap na instância someGrid ObjectGrid e configura diversos atributos do BackingMap utilizando os métodos set da interface BackingMap:

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

...

ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("someGrid");
BackingMap bm = objectGrid.getMap("someMap");
bm.setReadOnly( true ); // override default of read/write
bm.setNullValuesSupported(false); // override default of allowing Null values
bm.setLockStrategy( LockStrategy.PESSIMISTIC ); // override default of OPTIMISTIC
bm.setLockTimeout( 60 ); // override default of 15 seconds.
bm.setNumberOfBuckets(251); // override default (prime numbers work best)
bm.setNumberOfLockBuckets(251); // override default (prime numbers work best)
```

Plug-ins do BackingMap

A interface BackingMap possui vários pontos de conexão opcionais para interações mais extensíveis com o ObjectGrid:

- **Plug-in ObjectTransformer**: Para algumas operações do mapa, um BackingMap talvez precise serializar, desserializar ou copiar uma chave ou valor de uma entrada no BackingMap. O BackingMap pode desempenhar estas ações fornecendo uma implementação padrão da interface ObjectTransformer. Um aplicativo pode melhorar o desempenho fornecendo um plug-in ObjectTransformer projetado customizado que é utilizado pelo BackingMap para serializar, desserializar ou copiar uma chave ou valor de uma entrada no BackingMap. Consulte as informações sobre o plug-in ObjectTransformer no *Guia de Programação* para obter informações adicionais.
- **Plug-in do Evictor**: O evictor time to live interno utiliza um algoritmo baseado em tempo para decidir quando uma entrada no BackingMap deve ser liberada. Alguns aplicativos talvez precisem utilizar um algoritmo diferente para decidir quando uma entrada em um BackingMap precisa ser liberada. O plug-in do Evictor disponibiliza um Evictor projetado customizado para ser utilizado pelo BackingMap. O plug-in do Evictor é uma inclusão no evictor time to live interno. Ele não substitui o evictor time to live. O ObjectGrid fornece um plug-in do Evictor customizado que implementa algoritmos bem conhecidos, como "menos utilizado recentemente" ou "menos utilizado freqüentemente". Os aplicativos podem conectar um dos plug-ins do Evictor fornecidos ou podem fornecer seu próprio plug-in do Evictor. Consulte as informações sobre o despejo no *Guia de Programação*.

- **Plug-in MapEventListener**: Um aplicativo talvez queira saber sobre eventos de BackingMap, como uma evicção de entrada do mapa ou um pré-carregamento de uma conclusão de BackingMap. Um BackingMap chama métodos no plug-in MapEventListener para notificar um aplicativo sobre eventos do BackingMap. Um aplicativo pode receber notificação de vários eventos de BackingMap, utilizando o método setMapEventListener para fornecer um ou mais plug-ins MapEventListener projetados customizados para o BackingMap. O aplicativo pode modificar os objetos MapEventListener listados usando o método addMapEventListener ou o método removeMapEventListener. Consulte as informações sobre o plug-in MapEventListener no *Guia de Programação* para obter informações adicionais.
- **Plug-in do Loader**: Um BackingMap é um cache de memória de um Mapa. Um plug-in Carregador é uma opção que é usada pelo BackingMap para mover dados entre a memória e um armazenamento persistente. Por exemplo, um utilitário de carga JDBC (Java Database Connectivity) pode ser usado para mover dados para dentro e para fora de um BackingMap e uma ou mais tabelas relacionais de um banco de dados relacional. Um banco de dados relacional não precisa ser utilizado como o armazenamento persistente para um BackingMap. O utilitário de carga também pode ser usado para dados movidos entre um BackingMap e um arquivo, entre um BackingMap e um mapa Hibernate, entre um BackingMap e um bean de entidade Java 2 Platform, Enterprise Edition (J2EE), entre um BackingMap e outro servidor de aplicativos, e assim por diante. O aplicativo deve fornecer um plug-in do Loader projetado customizado para mover dados entre o BackingMap e o armazenamento persistente para cada tecnologia utilizada. Se um Loader não for fornecido, o BackingMap se tornará um cache de memória simples. Consulte as informações sobre o uso de um utilitário de carga no *Guia de Programação* para obter informações adicionais.
- **Plug-in OptimisticCallback**: Quando o atributo LockStrategy para um BackingMap for configurado como OPTIMISTIC, o BackingMap ou um plug-in do Loader deverá desempenhar operações de comparação para os valores do mapa. O plug-in OptimisticCallback é utilizado pelo BackingMap e pelo Loader para desempenhar as operações de comparação de controle de versões otimistas. Consulte as informações sobre o plug-in OptimisticCallback no *Guia de Programação* para obter informações adicionais.
- **Plug-in MapIndexPlugin**: Um plug-in MapIndexPlugin ou, abreviando, um Index, é uma opção utilizada pelo BackingMap para construir um índice baseado no atributo especificado do objeto armazenado. O índice permite que o aplicativo localize objetos por um valor específico ou intervalo de valores. Existem dois tipos de índice: estático e dinâmico. Consulte Indexação para obter informações detalhadas.

Para obter informações adicionais sobre plug-ins, consulte a introdução aos plug-ins no *Guia de Programação*.

Conectando-se ao ObjectGrid Distribuído

É possível se conectar a um ObjectGrid distribuído com um ponto de extremidade de conexão do serviço de catálogo. É necessário ter o nome do host e a porta de terminal do servidor de catálogo para o qual deseja se conectar.

Para conectar-se com uma grade distribuída, é necessário configurar o ambiente do lado do servidor com um serviço de catálogo e servidores de contêiner.

O método getObjectGrid(ClientClusterContext ccc, String objectGridName) conecta-se com o serviço de catálogo especificado e retorna uma instância ObjectGrid de cliente correspondente a uma instância ObjectGrid do lado do servidor.

O fragmento de código a seguir é um exemplo de como conectar-se com uma grade distribuída.

```
// 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.

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

// Obtain a distributed ObjectGrid using ObjectGridManager and providing
// the ClientClusterContext.

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

Interação com um ObjectGrid Usando o ObjectGridManager

A classe ObjectGridManagerFactory e a interface ObjectGridManager fornecem um mecanismo para criar, acessar e armazenar em cache instâncias do ObjectGrid. A classe ObjectGridManagerFactory é uma classe auxiliar estática para acessar a interface ObjectGridManager, 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.

Modelo de programação

Antes de usar a funcionalidade do eXtreme Scale como uma grade de dados de memória, é necessário criar e interagir com as instâncias do ObjectGrid com os seguintes métodos.

- Métodos createObjectGrid
- Métodos getObjectGrid
- Métodos removeObjectGrid
- Controlando o Ciclo de Vida de um ObjectGrid

Métodos createObjectGrid

Este tópico descreve os sete métodos createObjectGrid na 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 do ObjectGridManager permite a criação de uma instância do ObjectGrid baseada na configuração XML. A interface do 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 mudança 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) 1.4.2 possui suporte para validação 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; // turn XML validation on
boolean cacheInstance = true; // Cache the instance
String objectGridName="Employees"; // Name of Object Grid URL
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. Para obter a sintaxe XML, consulte a configuração ObjectGrid. Existem sete métodos createObjectGrid e estão documentados no bloco de códigos a seguir.

```
/**
 * 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;
```

Interrupções do Cliente durante uma Chamada de Método 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. O motivo é que o cliente está aguardando que os metadados da entidade no servidor ObjectGrid fiquem disponíveis. Este erro pode ocorrer quando um contêiner tiver sido iniciado mas o número inicial de contêineres ou o número mínimo de réplicas síncronas não tiver sido alcançado. 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.

Métodos getObjectGrid

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 “Conectando-se ao ObjectGrid Distribuído” na página 22 para obter mais informações.

Métodos removeObjectGrid

É possível 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. O 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

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(){
        // Starting the startup bean
        // This method is called when the application starts
        objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
        try {
            // create 2 ObjectGrids and cache these instances
            ObjectGrid bookstoreGrid =
      objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
            ObjectGrid videostoreGrid =
      objectGridManager.createObjectGrid("videostore", true);
            // within the JVM,
            // these ObjectGrids can now be retrieved from the
            //ObjectGridManager using the getObjectGrid(String) method
        } catch (ObjectGridException e) {
            e.printStackTrace();
            return false;
        }

        return true;
    }

    public void stop()  {
        // Stopping the startup bean
        // This method is called when the application is stopped
        try {
            // remove the cached ObjectGrids and destroy them
```

```
            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 for armazenada em cache, ela será recuperada e manipulada no código do 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");
        // perform operations on the cached ObjectGrid
        // ...
    }

    public void destroy() {
        super.destroy();
        try {
            // remove and destroy the cached bookstore ObjectGrid
            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 no WebSphere eXtreme Scale

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 mudança 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.

7.0.0.0 FIX 2+ 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

.

Boas Práticas de CopyMode

WebSphere eXtreme Scale faz uma cópia do valor com base nas seis configurações de CopyMode disponíveis. Determine qual configuração funciona melhor para seus requisitos de implementação.

É possível usar o método da API BackingMap setCopyMode(CopyMode, valueInterfaceClass) para configurar o modo de cópia para um dos seguintes

campos estáticos finais que foram definidos na classe com.ibm.websphere.objectgrid.CopyMode.

Quando um aplicativo usar a interface ObjectMap para obter uma referência para uma entrada de mapa, use tal referência somente dentro da transação WebSphere eXtreme Scale 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 você esteja usando o modo de cópia COPY_TO_BYTES. 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 tem uma referência para ser 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. Usando a configuração COPY_TO_BYTES, é possível reduzir a quantidade de memória que um gráfico grande de Objetos pode consumir. Consulte “Mapas de Matriz de Byte” na página 40 para obter informações adicionais.

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.

Mapas de Matriz de Byte

É possível armazenar os pares de chave-valor 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 “Desempenho de Serialização” na página 231.

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" />
```

Consulte o tópico sobre o arquivo XML do descritor ObjectGrid no *Guia de Administração* para obter mais informações.

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.

Uso de Sessões para Acessar Dados na Grade

Os aplicativos podem iniciar e terminar transações por meio 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

Os métodos a seguir estão disponíveis com a interface Session. Consulte a documentação da API para obter informações adicionais sobre os seguintes métodos:

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

    void begin() throws TransactionAlreadyActiveException, TransactionException;

    void beginNoWriteThrough() throws TransactionAlreadyActiveException, TransactionException;

    public void commit() throws NoActiveTransactionException, TransactionException;

    public void rollback() throws NoActiveTransactionException, TransactionException;

    public void flush() throws TransactionException;

    TxID getTxID() throws NoActiveTransactionException;

    boolean isWriteThroughEnabled();

    void setTransactionType(String tranType);

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

    ObjectGrid getObjectGrid();
```

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

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

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

Método Get

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 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();
// make some changes
...
session.flush(); // push these changes to the Loader, but don't commit yet
// make some more changes
...
session.commit();
```

Método NoWriteThrough

Alguns mapas do eXtreme Scale são suportados por um utilitário de carga, 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 no mapa de memória do eXtreme Scale e não são confirmados no armazenamento persistente fornecido pelo Utilitário de Carga. 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 que está ativa no momento.

```
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
// make some changes ...
session.commit(); // these changes will not get pushed to the 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.

```
// Use the Session object that was passed in during
//ObjectGridEventListener.initialization...
session.beginNoWriteThrough();
// process the received LogSequence
try {
 session.processLogSequence(receivedLogSequence);
} catch(Exception e) {
 session.rollback(); throw e;
}
// commit the changes
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

Ao usar uma política de posicionamento de partição por contêiner, é possível usar um SessionHandle. Uma instância de SessionHandle contém informações de partição para a Sessão atual e pode ser reutilizada para uma nova Sessão.

Um SessionHandle inclui informações para a partição a 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 uma instância do SessionHandle, é possível aplicar esse identificador em uma Sessão com o método setSessionHandle(SessionHandle target), passando o identificador como o destino. É possível recuperar o SessionHandle com o método Session.getSessionHandle.

Como ele é aplicável em um cenário de colocação por contêiner, usar o SessionHandle gera uma IllegalStateException se um ObjectGrid específico tiver vários conjuntos de mapas por contêiner ou não tiver nenhum. Se você não chamar o método setSessionHandle antes de chamar o método getSessionHandle, o SessionHandle adequado será selecionado com base em sua configuração de ClientProperties.

Também é possível usar a classe auxiliar SessionHandleTransformer para converter o identificador em formatos diferentes. Os métodos desta classe pode alterar uma representação do identificador da matriz de byte para a instância, cadeia para instância e vice-versa para ambos os casos, e também pode gravar o conteúdo do manipulador no fluxo de saída.

Para obter um exemplo de como é possível usar um SessionHandle, consulte o tópico de roteamento preferencial por zona na *Visão Geral do Produto*.

Integração de SessionHandle

Um objeto SessionHandle inclui informações de partição para o Session ao qual está ligado e facilita o roteamento de pedido. Objetos SessionHandle aplicam-se apenas ao cenário de posicionamento de partição por contêiner.

Objeto SessionHandle para Roteamento de Pedido

É possível ligar um objeto SessionHandle a um Session das seguintes formas:

Dica: Em cada uma das seguintes chamadas de método, após um SessionHandle ser ligado a um Session, o SessionHandle pode ser obtido do método Session.getSessionHandle que é utilizado com o método Session.setSessionHandle.

- Invoque o método Session.getSessionHandle: No momento em que esse método for invocado, se não houver um SessionHandle ligado ao Session, um SessionHandle será selecionado aleatoriamente e ligado ao Session.
- Invoque operações create, read, update, delete (CRUD) transacionais: No momento em que esses métodos forem invocados ou no momento de uma confirmação, se não houver um SessionHandle ligado ao Session, um SessionHandle será selecionado aleatoriamente e ligado ao Session.
- Invoque o método ObjectMap.getNextKey: No momento em que esse método for invocado, se não houver um SessionHandle ligado ao Session, o pedido de operação será roteado aleatoriamente para partições individuais até uma chave ser obtida. Se uma chave for retornada de uma partição, um SessionHandle correspondente a essa partição será ligado ao Session. Se nenhuma chave for localizada, nenhum SessionHandle será ligado ao Session.
- Invoque o método QueryQueue.getNextEntity ou QueryQueue.getNextEntities: No momento em que esse método for invocado, se não houver um SessionHandle ligado ao Session, o pedido de operação será roteado aleatoriamente para partições individuais até um objeto ser obtido. Se um objeto for retornado de uma partição, um SessionHandle correspondente a essa partição será ligado ao Session. Se nenhum objeto for localizado, o SessionHandle será ligado ao Session.

- Configure um SessionHandle com o método Session.setSessionHandle(SessionHandle sh): Se um SessionHandle for obtido do método Session.getSessionHandle, o SessionHandle poderá ser ligado a um Session. A configuração de um SessionHandle influencia no roteamento de pedidos dentro do escopo do Session ao qual ele está ligado.

O método Session.getSessionHandle sempre retornará um SessionHandle e ligará automaticamente um SessionHandle ao Session se não houver um SessionHandle ligado ao Session. Se você quiser verificar apenas se um Session tem um SessionHandle, chame o método Session.isSessionHandleSet. Se esse método retornar um valor de false, nenhum SessionHandle está ligado atualmente ao Session.

novo método incluído na API de Session

```
   /**
     * Determina se um SessionHandle está atualmente configurado
neste Session.
     *
     * @retorne true se SessionHandle estiver atualmente configurado
nesta sessão.
     *
     * @since 7.1
     */

    public boolean isSessionHandleSet();
```

Principais Tipos de Operações no Cenário de Posicionamento por Contêiner

A seguir está 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 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 CRUD transacionais (API de ObjectMap e API de EntityManager): SessionHandle
- **Objeto Session sem objeto SessionHandle ligado**
  - Índice - API de MapIndex e MapRangeIndex: Todas as partições ativas atuais
  - Query e ObjectQuery: Partição especificada via 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 um SessionHandle.
- QueryQueue: Especifica uma partição com o método QueryQueue.setPartition. Se nenhuma partição for configurada, o método irá selecionar aleatoriamente uma partição para retornar. Se um objeto for retornado, o atual Session será ligado ao SessionHandle que está ligado à partição que retorna o objeto. Se nenhum objeto for retornado, o Session não será ligado a um SessionHandle.
- Operações CRUD transacionais (API de ObjectMap e API de EntityManager): Selecionar uma partição aleatoriamente.

Na maioria dos casos, você deve utilizar 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. Após o armazenamento em cache do SessionHandle, é possível configurá-lo em outro Session para poder rotear pedidos para a partição especificada pelo SessionHandle armazenado em cache. Para executar operações como ObjectMap.clear sem o SessionHandle, é possível configurar temporariamente o SessionHandle como nulo chamando Session.setSessionHandle(null). Sem um SessionHandle especificado, as operações serão executadas em todas as partições ativas atuais.

- **Comportamento de roteamento de QueryQueue**

  No cenário de posicionamento de partição por contêiner, é possível utilizar SessionHandle para controlar o roteamento dos métodos getNextEntity e getNextEntities da API de QueryQueue. Se o Session estiver ligado a um SessionHandle, pedidos serão roteados para a partição à qual o SessionHandle está ligado. Se o Session não estiver ligado a um SessionHandle, pedidos serão roteados para a partição configurada com o método QueryQueue.setPartition se uma partição tiver sido configurada dessa forma. Se o Session não tiver um SessionHandle ou uma partição ligados, uma partição selecionada aleatoriamente será retornada. Se nenhuma partição for localizada, o processo vai parar e nenhum SessionHandle será ligado ao atual Session.

O fragmento de código a seguir mostra como utilizar o SessionHandle.

Exemplo de programação para o objeto 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, você deve utilizar o SessionHandle para a maioria das operações. O SessionHandle controla o roteamento para as partições. Para recuperar dados, o SessionHandle que você liga ao Session deve ser o mesmo SessionHandle de qualquer transação de inserção de dados.

Quando quiser executar uma operação sem um SessionHandle configurado no Session, será possível desvincular um SessionHandle de um Session fazendo uma chamada de método Session.setSessionHandle(null).

Quando um Session for ligado a um SessionHandle, todos os pedidos de operação serão roteados para a partição especificada pelo SessionHandle. Sem o SessionHandle configurado, as operações são 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 61.

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.

Introdução ao ObjectMap

A interface ObjectMap é utilizada para interação transacional entre aplicativos e BackingMaps.

Finalidade

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 o recurso de mapa dinâmico, é possível criar mapas quando a grade já tiver sido inicializada.

Nas versões anteriores, o 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 restrição de ter de definir todos os mapas antes da inicialização. Pelo uso de mapas modelos, agora os mapas podem 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 Cadeia que corresponde à expressão regular de um mapa modelo resultará 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 definir 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.

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 que corresponde a esta expressão regular, o aplicativo cria um novo mapa.

Nota: Se você definiu mais de um mapa modelo, certifique-se de que o nome de qualquer argumento para o método Session.getMap(String) não correspondente a mais de um mapa modelo.

Exemplo

A configuração de um mapa modelo é necessária para criar um mapa dinâmico. Inclua um booleano modelo para 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. Isso identifica em qual mapSet os mapas dinâmicos estarão.

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 informações adicionais, consulte emas informações sobre autorização de 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.partition.info">
    <backingMap name="partition.info" readOnly="false" lockStrategy="PESSIMISTIC"
       ttlEvictorType="NONE" copyMode="NO_COPY" numberOfBuckets="107"/>
    <backingMap name="clone.info" readOnly="false" lockStrategy="PESSIMISTIC"
      ttlEvictorType="NONE" copyMode="NO_COPY" numberOfBuckets="107"
      lockTimeout="300"/>
   </objectGrid>

   <objectGrid name="session">
    <bean id="ObjectGridEventListener"
     className="com.ibm.ws.xs.sessionmanager.SessionHandleManager"/>
    <backingMap name="objectgrid.session.metadata"
     pluginCollectionRef="objectgrid.session.metadata.dynamicmap.*
     template= true " readOnly="false" lockStrategy="PESSIMISTIC"
     ttlEvictorType="LAST_ACCESS_TIME" copyMode="NO_COPY"/>
    <backingMap name="objectgrid.session.attribute"
     pluginCollectionRef="objectgrid.session.attribute.dynamicmap.*"
     template=true readOnly="false" lockStrategy="OPTIMISTIC"
     ttlEvictorType="NONE" copyMode="NO_COPY"/>
    <backingMap name="datagrid.session.global.ids" readOnly="false"
     lockStrategy="PESSIMISTIC" ttlEvictorType="NONE" copyMode="NO_COPY"/>
   </objectGrid>

  </objectGrids>
</objectGridConfig>
```

objectGridDeployment.xml

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

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

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

```
    <map ref="datagrid.session.global.ids"/>
   </mapSet>
  </objectgridDeployment>

</deploymentPolicy>
```

Limitações e considerações:

Limitações:

- Não é possível usar mapas dinâmicos com Consulta.
- O 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 por meio 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.

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 do ObjectMap e o cache dinâmico no WebSphere Application Server, entre outros, utilizam essa abordagem. Entretanto, APIs baseadas em mapas têm limitações. A API EntityManager simplifica a interação com o cache do eXtreme Scale fornecendo uma maneira fácil de declarar e interagir com um gráfico complexo de objetos relacionados.

Limitações de API Baseada em Mapa

Se estiver utilizando uma API baseada em mapa, como o cache dinâmico no WebSphere Application Server ou a API do ObjectMap, você terá que considerar as seguintes limitações:

- O cache precisa usar reflexo para extrair os dados dos objetos que contém; isso afeta o desempenho.
- Dois aplicativos não podem compartilhar um cache se utilizarem objetos diferentes para os mesmos dados.
- Não é possível utilizar evolução de dados porque não pode incluir facilmente um atributo em um objeto Java em cache.
- É difícil trabalhar com gráficos de objetos. O aplicativo precisa armazenar referências artificiais entre objetos e uni-los manualmente.

Utilizando EntityManager

A API EntityManager utiliza a infra-estrutura baseada em Mapa existente, contudo os objetos da entidade são convertidos para/de tuplas antes de serem armazenados ou lidos no 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 facilita significativamente o uso do eXtreme Scale seguindo o estilo Plain Old Java Object (POJO) de programação que é adotado pela maioria das estruturas.

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 “EntityManager em um Ambiente Distribuído” na página 72.

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.

7.0.0.0 FIX 2+ 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.

Arquivo `emd.xsd`

Use a definição de esquema XML de metadados de entidade para criar um arquivo descritor XML e definir um esquema de entidade para o WebSphere eXtreme Scale.

Consulte as informações sobre o arquivo descritor de metadados de entidade no *Guia de Administração* para obter descrições de cada elemento e atributo do arquivo `emd.xsd`.

Arquivo `emd.xsd`

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

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

    <!-- ************************************************** -->
    <xsd:complexType name="entity">
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

<!-- ************************************************** -->

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

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

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

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

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

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

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

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

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

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

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

</xsd:schema>
```

EntityManager em um Ambiente Distribuído

É possível usar EntityManager com um ObjectGrid local ou em um ambiente do eXtreme Scale distribuído. 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 da grade ou implementação

Esses arquivos especificam quais entidades e BackingMaps um servidor hospedará.

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`. Observe que o

fragmento do 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 refere-se a entity.xml com o atributo **entityMetadataXMLFile**. O local deste arquivo é relativo ao local do arquivo objectgrid.xml. 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>
```

Esse exemplo supõe que a classe Order teria os campos orderNumber e desc anotados de forma semelhante.

Um arquivo entity.xml sem classe equivalente seria como a seguir

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 um servidor eXtreme Scale, consulte *Iniciando WebSphere eXtreme Scale processos do servidor* no *Guia de Administração*, que usa ambos os arquivosdeployment.xml e objectgrid.xml para iniciar o servidor de catálogo.

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");
   ```

7.0.0.0 FIX 2+ 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 utilizará 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 memória em um cliente porque o cliente tem somente as informações que ele precisa em vez de todas as informações que estão 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 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 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” na página 69 para obter informações adicionais sobre o arquivo descritor de metadados.

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 do gerenciador de entidades do no*Visão Geral do Produto* para obter informações adicionais.

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 89 para obter mais informações.

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 mudança 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 por meio 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.

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.

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.

Impacto do Desempenho da Interface EntityManager

Um ambiente que exigisse que cada aplicativo utilizasse os mesmos objetos de acesso a dados para um determinado armazenamento de dados não seria nada prático. Em contraste, a interface EntityManager fornecida com o WebSphere eXtreme Scale 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 utilize a interface EntityManager e otimize a lógica de negócios crucial após o aplicativo se concluído. É possível fazer um retrabalho em qualquer código que utilize interfaces EntityManager para utilizar mapas e tuplas. Geralmente, esse retrabalho de código pode ser necessário para dez por cento do código.

Se você utilizar relacionamentos entre objetos, o impacto no desempenho será menor, pois um aplicativo que está utilizando mapas precisa gerenciar esses relacionamentos de forma semelhante à interface EntityManager.

Os aplicativos que utilizam a interface EntityManager não precisam fornecer um ObjectTransformer porque ele é otimizado automaticamente.

Retrabalho do Código de 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, indexados em 0, 1 e 2 respectivamente. A tupla de chave tem um único atributo, e o número do 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.

Agente de Instrumentação

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=excludedPackage;fieldAccessEntity=package1,package2
```

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.

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 105.

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.

Interface EntityTransaction

É possível utilizar a interface EntityTransaction para demarcar transações.

Finalidade

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 .

Tutorial do Entity Manager: Visão Geral

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.

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.

Tabela 2. Outros Métodos (continuação)

Método	Resultado
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 117
- "Referência para Consultas do eXtreme Scale" na página 109

Os tópicos a seguir descrevem os meios para usar a API de Consulta:

- "API de Consulta EntityManager" na página 105
- "Uso da API ObjectQuery" na página 100

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*

Inserindo Dados para Fusos Horários Diferentes

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. O uso dos mesmos objetos de data e hora baseados 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 utilizará o fuso horário local e poderá 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.

Para obter mais informações, consulte o tópico sobre como consultar dados em vários fusos horários no *Guia de Programação.*

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

É possível 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.

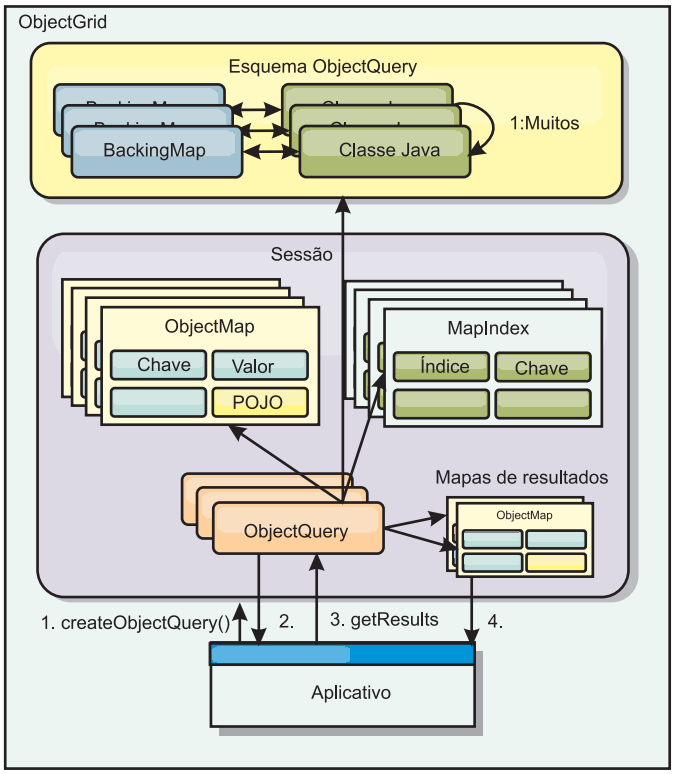


Figura 1. A interação da consulta com os mapas de objetos e como um esquema é definido para classes e associado a um mapa ObjectGrid

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 uso do arquivo XML descritor do ObjectGrid, consulte o 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 o 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. É possível consultar os tipos de atributos a seguir:

- Os tipos de primitivas Java que incluem 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 Tutorial de consulta de objetos o 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.

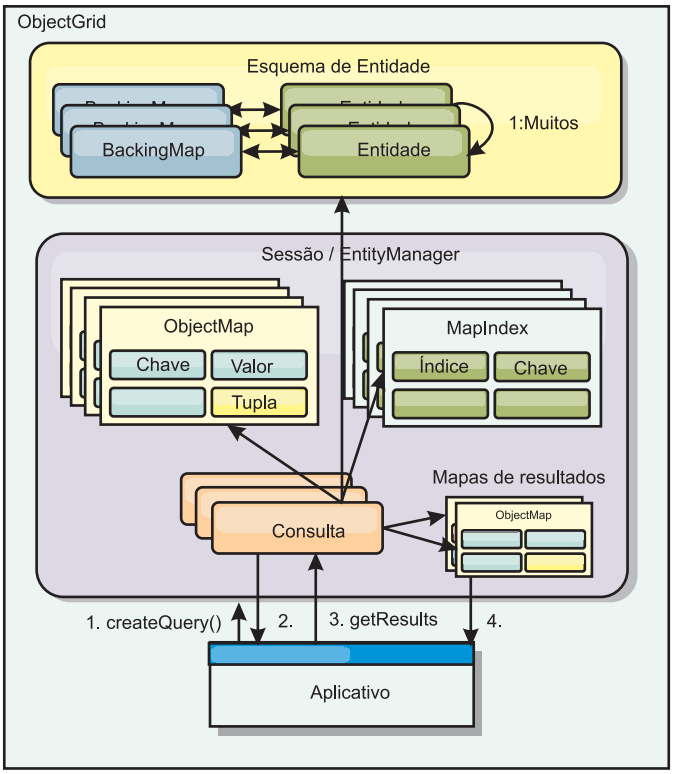


Figura 2. A interação da consulta com os mapas de objetos ObjectGrid e como o esquema da entidade é definido e associado com um mapa ObjectGrid.

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 é possível 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.

É possível 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)
```

Ajuste de 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. Consulte Indexação para obter mais informações.

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" 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 irá interagir com 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 utilizado para explorar as diferenças que as alterações sutis em uma cadeia de consultas fazem na maneira que o eXtreme Scale executa uma consulta.

O plano de consulta pode ser visualizado de uma de duas maneiras:

- Consulta EntityManager ou métodos de API ObjectQuery getPlan
- Rastreio de diagnóstico do ObjectGrid

Método getPlan

O método getPlan nas interfaces ObjectQuery e Query retorna uma Cadeia 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 “Logs e Rastreio” na página 383 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 utiliza a palavra para indicar que a consulta está iterando por meio de uma coleta do ObjectMap ou por meio de uma coleta derivada, tal como: q2.getEmps(), q2.dept ou uma coleta temporária retornada por um loop interno. Se a coleta for a partir de um ObjectMap, o plano de consulta mostra se uma varredura sequencial (denotada por INDEX SCAN), índice exclusivo ou não-exclusivo é utilizado. 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 department, assim o loop interno é executado apenas 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 abaixo executa melhor porque ela primeiro limita o resultado para um objeto departamento utilizando o índice exclusivo que é definido sobre o número do

campo de chave primária. 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 executa o 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 Configurando o HashIndex.

Para obter informações sobre indexação, consulte Indexação.

Para obter uma maneira mais eficiente de procurar objetos armazenados em cache, consulte “HashIndex Composto” na página 245

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 121. 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='TELECOM'
```

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

Processamento de Sessões e 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.

Transações

As transações possuem muitas vantagens para o armazenamento e a manipulação de dados. É possível utilizar transações para proteger a grade 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. Consulte o *Guia de Programação* para obter informações adicionais sobre o eXtreme Scale e a interação com transações externas. Se você estiver usando o WebSphere Application Server, é possível usar o plug-in WebSphereTranscationCallback. Consulte o *Guia de Programação* para obter informações adicionais sobre os plug-ins que estão disponíveis com o WebSphere eXtreme Scale.

Atributo CopyMode

É possível ajustar o número de cópias definindo o atributo CopyMode do BackingMap ou objetos ObjectMap.

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

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"/>
```

Consulte as informações sobre as boas práticas do método copyMode no *Guia de Programação* para obter mais informações.

Bloqueio de Entrada de Mapa

Um BackingMap do ObjectGrid suporta várias estratégias de bloqueio para mapas para manter a consistência de entradas de cache.

Cada BackingMap pode ser configurado para utilizar uma das seguintes estratégias de bloqueio:

1. Modo de Bloqueio Otimista
2. Modo de Bloqueio Pessimista
3. Nenhum(a)

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, é possível 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 estratégias de bloqueio, consulte as informações sobre as estratégias de bloqueio no *Visão Geral do Produto*.

Especificando uma Estratégia de Bloqueio

O exemplo a seguir demonstra como a estratégia de bloqueio pode ser configurada nos BackingMaps map1, map2 e map3, em que cada mapa está utilizando uma estratégia de bloqueio diferente. O primeiro fragmento mostra como usar o XML para a configuração de estratégia de bloqueio e o segundo fragmento mostra uma abordagem programática.

Abordagem XML

Configuração de BackingMap - Exemplo 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="map1"
                lockStrategy="PESSIMISTIC" numberOfLockBuckets="31"/>
            <backingMap name="map2"
                lockStrategy="OPTIMISTIC" numberOfLockBuckets="409"/>
            <backingMap name="map3"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Abordagem Programática

Configuração BackingMap - exemplo programático
```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
 ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap( "map1" );
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setNumberOfLockBuckets(31);
bm = og.defineMap( "map2" );
```

```
bm.setNumberOfLockBuckets(409);
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

Para evitar uma exceção java.lang.IllegalStateException, o método setLockStrategy deve ser chamado antes de usar os métodos initialize ou getSession na instância do ObjectGrid local.

Para obter mais informações, consulte o tópico sobre estratégias de bloqueio em *Visão Geral do Produto*.

Configuração do Gerenciador de Bloqueios

Quando a estratégia de bloqueio pessimistic ou optimistic 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.

LockDeadlockException

Este exemplo de código que mostra como capturar a exceção e a mensagem resultante que será exibida.

```
try {
...
} catch (ObjectGridException oe) {
System.out.println(oe);
}
```

O resultado é:

```
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.

Causa da Exceção

O tipo mais comum de exceção de conflito ocorre ao utilizar uma estratégia de bloqueio pessimista e dois clientes separados, cada qual possuindo um bloqueio compartilhado sobre um determinado objeto. Em seguida, ambos tentam evoluir

para um bloqueio exclusivo sobre tal objeto. O diagrama a seguir ilustra tal situação, incluindo os blocos de transação que fazem com que a exceção seja emitida.

O trecho de código Java a seguir demonstra como transmitir um arquivo de configuração XML para criar um ObjectGrid.

Esta é uma visualização abstrata do que está ocorrendo em seu programa quando ocorre a exceção. Em um aplicativo com muitos encadeamentos atualizando o mesmo ObjectMap, é possível encontrar esta situação. Este é um exemplo de dois clientes que executam blocos de código de transação, conforme ilustrado na figura anterior.

Soluções Possíveis

É possível encontrar a situação ilustrada na Figura 1 quando muitos encadeamentos iniciam transações em um determinado mapa. Nesse caso, uma exceção será emitida para evitar a interrupção do seu programa. É possível notificar a si mesmo e acrescentar código no bloco de captura para obter mais detalhes sobre a causa. Como essa exceção só ocorre numa estratégia de bloqueio pessimista, uma solução simples é apenas utilizar a estratégia de bloqueio otimista. Se precisar de uma estratégia de bloqueio pessimista, no entanto, poderá utilizar o método getForUpdate em vez de o método get. Isso elimina o recebimento de exceções para a situação descrita anteriormente.

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 a implementação de uma abordagem de bloqueio optimistic, consulte a seção OptimisticCallback em as informações sobre as considerações do utilitário de carga 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 139.

JMS para Distribuição de Mudanças de Transação

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 as informações sobre a configuração da 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 o uso de JMS para mudanças de transação distribuída, consulte as informações sobre mudanças de distribuição entre Java Virtual Machines peer no *Guia de Administração*.

Transações de partição única e de partição de grade cruzada

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 importantes de transações a serem considerados são transações de partição única e de cada partição (grade cruzada).

Em geral, as interações com o cache podem ser categorizadas como transações de partição única ou transações de grade cruzada, conforme discutido a seguir.

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 ser substituído e retornado para a grade, o cache voltará para 100% da capacidade de rendimento.

Transações de Grade Cruzada

Em termos de desempenho, disponibilidade e escalabilidade, as transações de grade cruzada são o oposto das transações de partição única As transações de grade cruzada acessam cada partição e, portanto, cada computador na configuração. Cada computador na grade é instruído a procurar alguns dados e retornar o resultado. A transação não pode ser concluída até cada computador ter respondido e, dessa forma, o rendimento da grade inteira será limitado pelo computador mais lento. Incluir computadores não agiliza o computador mais lento e, assim, não melhora o rendimento do cache.

As transações de grade cruzada têm um efeito semelhante em 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 não poderá 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 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 é limitado ao rendimento do computador mais lento em uma transação de grade cruzada, em média, o rendimento será reduzido em 10%.

As transações de partição única são preferidas para as transações de grade cruzada para serem escaladas 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, mas é possível transformar as transações de grade 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 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. 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 distribuída para garantir a consistência de 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 mudança 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 não será escalável se um registro único estiver sendo usado por um número de vezes desproporcional, comparado ao restante dos registros. O desempenho da grade 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 cruzada, mas para a maior parte, isso já ficou ineficiente como uma operação de grade cruzada porque o desempenho dependeu 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 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 prática porque todas as operações baseadas no ID do cliente se transformam em transações de grade 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 manter 10.000 produtos, apenas 1/10.000 da grade precisará ser bloqueada para executar a atualização. Com a organização B, a grade contém apenas 50 lojas, então, 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 A, essa transação se torna uma transação de grade 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 cruzada é escalar o processamento de dados. Se uma grade tiver 5 computadores e uma transação de grade cruzada for despachada, que classifica cerca 100.000 registros em cada computador, essa transação classificará 500.000 registros. Se o computador mais lento na grade puder executar 10 dessas transações por segundo, a grade 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 para 5 transações por segundo. Além disso, a grade classifica cerca de 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 permanece o mesmo nos 10 pedidos por segundo, mas agora cada transação processa 1.000.000 de registros dobrando, assim, a capacidade da grade em termos de processamento de registros para 10.000.000 por segundo.

Com os aplicativos, como um mecanismo de procura que precisa ser escalado em termos de processamento de dados para acomodar o tamanho crescente da Internet e de rendimento para acomodar o crescimento de usuários, é necessário criar várias grades, com um round robin dos

pedidos entre as grades. Se for preciso escalar o rendimento, inclua computadores e outra grade para atender aos pedidos. Se o processamento de dados precisar ser escalado, inclua mais computadores e mantenha o número de grades constante.

Manipulando 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.

Tempos Limite do Bloqueio

Cada BackingMap tem um valor de tempo limite de espera de bloqueio padrão. O valor de tempo limite é utilizado para garantir que um aplicativo não aguarde infinitamente por um modo de bloqueio a ser concedido devido a uma condição de conflito que ocorre devido a um erro de aplicativo. O aplicativo pode utilizar a interface BackingMap para substituir o valor de tempo limite de espera de bloqueio padrão. 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.

Quando ocorre uma LockTimeoutException, o aplicativo deve determinar se o tempo limite está ocorrendo porque o aplicativo está em execução de forma mais lenta 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.

Tempo limite de espera para ObjectMaps

O tempo limite de espera do bloqueio pode ser substituído por uma instância única do ObjectMap utilizando o método ObjectMap.setLockTimeout. 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 compartilhados, atualizáveis 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 é possível 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.

Cenários de Conflito Comuns

As seções a seguir descrevem alguns dos cenários de conflitos mais comuns e sugestões sobre como evitá-los.

Cenário: Conflitos de chave únicos

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 5. 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.
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 6. 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 7. 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.

Tabela 7. Conflitos de uma única chave, continuação (continuação)

	Encadeamento 1	**Encadeamento 2**	
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 8. 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. Utilize 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. 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.

Cenário: Bloqueio de chaves múltiplas solicitado

Este cenário descreve o que acontece se duas transações tentam atualizar a mesma entrada diretamente e contêm bloqueios S com outras entradas.

Tabela 9. 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.
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 10. 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. Utilize 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.

Cenário: Fora de ordem com bloqueio U

Se a ordem na qual as chaves são solicitadas não puder ser garantida, então, ainda pode ocorrer um conflito:

Tabela 11. 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.
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.

Manipulação de Exceção nos Cenários de Bloqueio

Os exemplos anteriores não possuem nenhuma manipulação de exceção. 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. Altere o trecho de código anterior conforme demonstrado no exemplo a seguir:

```
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 finally no trecho de código assegura que seja efetuado rollback de uma transação, 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.

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 a implementação de uma abordagem de bloqueio optimistic, consulte a seção OptimisticCallback em as informações sobre as considerações do utilitário de carga 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 139.

Boas Práticas de Desempenho de Bloqueio

Estratégias de bloqueio e configurações de isolamento de transação afetam o desempenho dos seus aplicativos.

Recuperar uma Instância Armazenada em Cache

Consulte as informações sobre o bloqueio de entrada de mapa no *Guia de Administração* para obter mais informações.

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.

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 )
{
     // Start with Throwable that occurred as the root.
     Throwable root = t;

     // Follow cause chain until last Throwable in chain is found.
     Throwable cause = root.getCause();
     while ( cause != null )
     {
        root = cause;
        cause = root.getCause();
     }

     // Return last Throwable in the chain as the root cause.
     return root;
}
```

Configurando Clientes com o WebSphere eXtreme Scale

É possível configurar um cliente eXtreme Scale com base em seus requisitos, como a necessidade de substituir configurações.

Configure o Cliente com XML

Um arquivo XML ObjectGrid pode ser usado para alterar configurações no lado do cliente. Para alterar as configurações em um cliente eXtreme Scale, você deve criar um arquivo XML ObjectGrid semelhante na estrutura ao arquivo que foi usado para o servidor eXtreme Scale.

Suponha que o seguinte arquivo XML tenha sido emparelhado com um arquivo XML de política de implementação e que esses arquivos tenham sido usados para iniciar um servidor eXtreme Scale.

companyGridServerSide.xml

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

    <objectGrids>
        <objectGrid name="CompanyGrid">
            <bean id="TransactionCallback"
                className="com.company.MyTxCallback" />
            <bean id="ObjectGridEventListener"
                className="com.company.MyOgEventListener" />
            <backingMap name="Customer"
                pluginCollectionRef="customerPlugins"/>
            <backingMap name="Item" />
            <backingMap name="OrderLine" numberOfBuckets="1049"
                timeToLive="1600" ttlEvictorType="LAST_ACCESS_TIME" />
```

```
            <backingMap name="Order" lockStrategy="PESSIMISTIC"
                pluginCollectionRef="orderPlugins" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="customerPlugins">
            <bean id="Evictor"
                className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
            <bean id="MapEventListener"
                className="com.company.MyMapEventListener" />
        </backingMapPluginCollection>
        <backingMapPluginCollection id="orderPlugins">
            <bean id="MapIndexPlugin"
                className="com.company.MyMapIndexPlugin" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Em um servidor eXtreme Scale, a instância do ObjectGrid denominada CompanyGrid se comporta como definido pelo arquivo companyGridServerSide.xml. Por padrão, o cliente CompanyGrid tem as mesmas configurações que a instância CompanyGrid em execução no servidor. Entretanto, algumas das configurações podem ser substituídas no cliente da seguinte forma:

1. Crie uma instância ObjectGrid específica do cliente.
2. Copie o arquivo XML ObjectGrid que foi utilizado para abrir o servidor.
3. Edite o novo arquivo para customizar para o lado do cliente.
   - Para configurar ou atualizar qualquer um dos atributos no cliente, especifique um novo valor ou altere o valor existente.
   - Para remover um plug-in do cliente, use a cadeia vazia como o valor para o atributo className.
   - Para alterar um plug-in existente, especifique um novo valor para o atributo className.
   - Também é possível incluir qualquer plug-in suportado para uma substituição de cliente: TRANSACTION_CALLBACK, OBJECTGRID_EVENT_LISTENER, EVICTOR, MAP_EVENT_LISTENER.
4. Crie um cliente com o arquivo XML de substituição de cliente recém criado.

O seguinte arquivo XML ObjectGrid pode ser usado para especificar alguns dos atributos e plug-ins no cliente CompanyGrid.

companyGridClientSide.xml

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

    <objectGrids>
        <objectGrid name="CompanyGrid">
            <bean id="TransactionCallback"
                className="com.company.MyClientTxCallback" />
            <bean id="ObjectGridEventListener" className="" />
            <backingMap name="Customer" numberOfBuckets="1429"
                pluginCollectionRef="customerPlugins"/>
            <backingMap name="Item" />
            <backingMap name="OrderLine" numberOfBuckets="701"
                timeToLive="800" ttlEvictorType="LAST_ACCESS_TIME" />
            <backingMap name="Order" lockStrategy="PESSIMISTIC"
                pluginCollectionRef="orderPlugins" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="customerPlugins">
            <bean id="Evictor"
                className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
            <bean id="MapEventListener" className="" />
```

```
        </backingMapPluginCollection>
        <backingMapPluginCollection id="orderPlugins">
            <bean id="MapIndexPlugin"
                className="com.company.MyMapIndexPlugin" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

- TransactionCallback no cliente é com.company.MyClientTxCallback em vez da configuração do lado do servidor de com.company.MyTxCallback.
- O cliente não tem um plug-in ObjectGridEventListener porque o valor className é a cadeia vazia.
- O cliente configura numberOfBuckets como 1429 para Customer backingMap, retém seu plug-in Evictor e remove o plug-in MapEventListener.
- Os atributos numberOfBuckets e timeToLive de OrderLine backingMap mudaram
- Embora um atributo lockStrategy diferente seja especificado, não há nenhum efeito, pois o atributo lockStrategy não é suportado para uma substituição de cliente.

Para criar o cliente CompanyGrid usando o arquivo companyGridClientSide.xml, passe o arquivo XML ObjectGrid como uma URL para um dos métodos de conexão no ObjectGridManager.

Criando o cliente para XML

```
ObjectGridManager ogManager =
 ObjectGridManagerFactory.ObjectGridManager();
ClientClusterContext clientClusterContext = ogManager.connect
("MyServer1.company.com:2809", null, new URL(
                "file:xml/companyGridClientSide.xml"));
```

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.

Configurar o Cliente na Estrutura Spring

As configurações de ObjectGrid do lado do cliente também podem ser substituídas utilizando a Estrutura Spring. O arquivo XML de exemplo a seguir mostra como construir um elemento ObjectGridConfiguration e utilizá-lo para substituir algumas configurações do lado do cliente. Esse exemplo chama as mesmas APIs que são demonstradas na configuração programática. O exemplo também é uma funcionalidade equivalente ao exemplo na configuração XML ObjectGrid.

configuração do cliente com 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>
```

Depois de criar o arquivo XML, carregue o arquivo e crie o ObjectGrid com o seguinte fragmento de código.

```
BeanFactory beanFactory = new XmlBeanFactory(new

UrlResource("file:test/companyGridSpring.xml"));

ObjectGrid companyGrid = (ObjectGrid) beanFactory.getBean("companyGrid");
```

Leia sobre a Visão geral de integração de estrutura spring para obter mais informações sobre como criar um arquivo descritor XML.

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.

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 {
    // Get a SQL connection to use.
    Connection conn = getConnection(tx);
    try
    {
    // Process the list of changes and build a set of prepared
    // statements for executing a batch update, insert, or delete
    // SQL operations. The statements are cached in 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;
        }
    }
    // Run the batch statements that were built by above loop.
```

```
    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, é possível 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.

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.

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.

Capítulo 4. Acessando Dados com o Serviço de Dados REST

Desenvolva aplicativos que executam operações usando protocolos do serviço de dados REST.

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 capígulo 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 Modelo de dados escalável no eXtreme Scale e 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.

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.

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 linha dentro da resposta do serviço de dados como um Atom Feed Document (relação para-muitos) ou Atom Entry Document (relação para-um).

Dica: Para obter mais detalhes sobre o protocolo RetrieveEntity definido nos Serviços de Dados WCF, consulte MSDN: Pedido 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 nos Serviços de Dados WCF Microsoft, 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

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.

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 a 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

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 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 do pedido, o serviço de dados REST irá configurar o valor da propriedade da entidade para 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. Um código de resposta 500 (Erro do Servidor Interno) será retornado para indicar um Erro do Servidor Interno. Se uma propriedade estiver ausente na carga útil do pedido HTTP MERGE, o serviço de dados REST não mudará o valor da propriedade existente.

Se houver propriedades duplicadas especificadas na carga útil, a última propriedade será utilizada. 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. Consulte a seção 6.3.1.4 para obter mais detalhes.

Para obter mais detalhes sobre o pedido UpdateValue, consulte: Biblioteca MSDN: Pedido UpdateValue

Veja aqui um exemplo do pedido UpdateValue. Neste exemplo, atualizamos 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).

Atualizar um link entre duas instâncias de entidade do eXtreme Scale pode não apenas estabelecer associações, mas também remover associações. Por exemplo, se o cliente estabelecer uma associação para-um entre uma entidade Order(orderId=5000,customer_customerId='IBM'), uma entidade e uma instância de Customer('ALFKI'), ela terá que desassociar a entidade Order(orderId=5000,customer_customerId='IBM') e a entidade da instância de Customer atualmente associada.

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 do pedido UpdateLink contiver vários links, o serviço de dados REST só analisará 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') com 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 do Pedido: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- Cabeçalho do Pedido: Tipo de Conteúdo: application/xml
- Carga Útil do Pedido: {"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')"}
- Carga Útil de Resposta: Nenhuma
- Código de Resposta: 204 Nenhum Conteúdo

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

Simultaneidade Otimista

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 61.

Capítulo 5. Programação com 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.

Introdução aos 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 ObjectGrid:

- **TransactionCallback**: Um plug-in TransactionCallback fornece eventos do ciclo de vida de transação.
- **ObjectGridEventListener**: Um plug-in ObjectGridEventListener fornece eventos de ciclo de vida ObjectGrid para o ObjectGrid, shards e transações.
- **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)
- **JPATxCallback** (Apenas do lado do servidor)
- **Subclasses de JPATxCallback**

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.
- **Indexação** - Use o recurso de indexação, que é representado pelo plug-in MapIndexplug-in, para construir um ou mais índice 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)

Ciclos de Vida de Plug-in

A maioria dos plug-ins possui ambos os métodos inicializar e destruir ou métodos equivalentes, além dos métodos para os quais eles foram designados para operar. Esses métodos especializados de cada plug-in estão disponíveis para serem chamados em pontos funcionais designados. Esses dois métodos inicializar e destruir definem o ciclo de vida de plug-ins, que são controlados pelos 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.

Quando os objetos do proprietário estão inicializando, eles chamarão o método inicializar dos plug-ins possuídos. Durante o ciclo de vida dos objetos proprietários, o método destruir dos plug-ins também será chamado. Para obter detalhes sobre as características dos métodos inicializar e destruir, junto com outros métodos aptos com 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, as instâncias do plug-in, são independentes de todos os ObjectGrids e instâncias de plug-in do lado do servidor.

Nessa topologia distribuída, suponha que você possua um ObjectGrid chamado "myGrid" definido no arquivo objectGrid.xml e configurado com um ObjectGridEventListener customizado chamado myObjectGridEventListener. O arquivo objectGridDeployment.xml define a política de implementação para myGrid ObjectGrid. Ambos objectGrid.xml e objectGridDeployment.xml são usados para iniciar servidores de contêiner. Durante a inicialização do servidor de contêiner, a instância myGrid ObjectGrid do lado do servidor será inicializada e o método inicializar da instância myObjectGridEventListener de propriedade da instância myObjectGrid será chamada. Depois que o servidor de contêiner for iniciado, o aplicativo poderá se conectar à instância myGrid ObjectGrid do lado do servidor e obter uma instância do lado do cliente.

Ao obter a instância myGrid ObjectGrid do lado do cliente, a instância myGrid do lado do cliente passará pelo próprio ciclo de inicialização e chamará o método inicializar da 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 ObjectGrid do lado do cliente.

Se o aplicativo desconectar ou destruir a instância myGrid ObjectGrid do lado do cliente, o método destroy da instância myObjectGridEventListener do lado do cliente possuída será chamado automaticamente. Entretanto, isso não afeta a instância myObjectGridEventListener no lado do servidor. O método destroy da instância myObjectGridEventListener do lado do servidor será chamado apenas durante o ciclo de vida da instância myGrid ObjectGrid do lado do servidor ao parar um servidor de contêiner. Ou seja, ao parar um servidor de contêiner, as instâncias ObjectGrid contidas serão destruídas e o método destroy de todos os plug-ins possuídos será 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 um BackingMap e é necessário ter cuidado ao determinar suas configurações para os plug-ins que podem ser gravados com base nessas considerações do ciclo de vida.

Plug-ins para Despejar Objetos de Cache

O WebSphere eXtreme Scale fornece um mecanismo padrão para despejar entradas do cache e um plug-in para criar evictors customizados. Um evictor controla a associação de entradas em cada BackingMap. O evictor padrão utiliza uma política de despejo de TTL (Time-to-Live) para cada BackingMap. Se você fornecer um mecanismo de evictor conectável, geralmente ele utilizará uma política de evicção baseada no número de entradas em vez de baseada no tempo.

Propriedade TimeToLive

Um evictor TTL padrão é criado com cada mapa de apoio. O evictor padrão remove entradas com base em um conceito de time-to-live. Este comportamento é definido pelo atributo ttlType, que tem os seguintes tipos.

Nenhum

Especifica que as entradas nunca expiram e, portanto, nunca são removidas do mapa.

Tempo de criação

Especifica que as entradas são despejadas dependendo de quando foram criadas.

Se você estiver usando o ttlType CREATION_TIME, o evictor despejará uma entrada quando seu tempo de criação for igual ao valor do seu atributo TimeToLive, que é configurado em milissegundos na configuração do aplicativo. Se você configurar o valor do atributo TimeToLive como 10 segundos, a entrada será despejada automaticamente dez segundos após ser inserida.

É importante ter atenção ao configurar este valor para o CREATION_TIME ttlType. Este evictor é melhor utilizado quando existem quantidades de inclusões no cache razoavelmente altas que são utilizadas apenas durante

uma quantidade de tempo configurada. Com esta estratégia, tudo o que é criado é removido após a quantidade de tempo configurada.

O ttlType CREATION_TIME é útil em cenários como a atualização de cotas de estoque a cada 20 minutos ou menos. Suponha que um aplicativo da Web obtenha cotas de estoque e que obter as cotas mais recentes não seja crítico. Neste caso, os preços de ações são armazenados em cache em um ObjectGrid por 20 minutos. Após 20 minutos, as entradas do mapa do ObjectGrid expiram e são liberadas. A cada vinte minutos ou quase, o mapa do ObjectGrid usa o plug-in do Carregador para atualizar os dados do mapa com dados atualizados do banco de dados. O banco de dados é atualizado a cada 20 minutos com os preços de ações mais recentes.

Horário do último acesso

Especifica que as entradas são despejadas dependendo de quando foram acessadas pela última vez, se foram lidas ou atualizadas.

Horário da última atualização

Especifica que as entradas são despejadas dependendo de quando foram atualizadas pela última vez.

Se você estiver usando o atributo ttlType LAST_ACCESS_TIME ou LAST_UPDATE_TIME, configure TimeToLive para um número menor do que se você estivesse usando o ttlType CREATION_TIME, porque o atributo TimeToLive de entradas é reconfigurado toda vez que é acessado. Em outras palavras, se o atributo TimeToLive for igual a 15 e existir uma entrada por 14 segundos, mas for acessada, ela não expirará novamente durante outros 15 segundos. Se você configurar o TimeToLive como um número relativamente alto, muitas entradas poderão nunca ser liberadas. No entanto, se você configurar o valor como algo semelhante a 15 segundos, as entradas poderão ser removidas quando não forem acessadas com freqüência.

O ttlType LAST_ACCESS_TIME ou LAST_UPDATE_TIME é útil em cenários como a manutenção de dados da sessão de um cliente, usando um mapa do ObjectGrid. Os dados de sessão devem ser destruídos se o cliente não utilizá-los por algum período de tempo. Por exemplo, é excedido o tempo limite dos dados de sessão após 30 minutos sem atividade do cliente. Neste caso, usar um tipo TTL de LAST_ACCESS_TIME ou LAST_UPDATE_TIME com o atributo TimeToLive configurado para 30 minutos é adequado para este aplicativo.

O exemplo a seguir cria um mapa de suporte, configura seu atributo ttlType do evictor padrão e configura sua propriedade TimeToLive.

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

A maior parte das configurações de evictor deve ser feita antes de inicializar o ObjectGrid.

Também é possível gravar seus próprios evictors: Para obter mais informações, consulte as informações sobre como gravar um evictor customizado no *Guia de Programação*.

Evictors Opcionais

O evictor TTL padrão utiliza uma política de evicção baseada no tempo e o número de entradas no BackingMap não tem efeito sobre o tempo de expiração de uma entrada. É possível utilizar um evictor conectável opcional para despejar entradas com base no número de entradas existentes em vez de no tempo.

Os seguintes evictors plugáveis opcionais fornecem alguns algoritmos comumente usados para decidir quais entradas despejar quando um BackingMap crescer além de algum limite de tamanho:

- O evictor LRUEvictor utiliza um algoritmo LRU (Least Recently Used) para decidir quais entradas serão despejadas quando o BackingMap exceder um número máximo de entradas.
- O evictor LFUEvictor utiliza um algoritmo LFU (Least Frequently Used) para decidir quais entradas serão despejadas quando o BackingMap exceder um número máximo de entradas.

O BackingMap informa um evictor conforme as entradas são criadas, modificadas ou removidas de uma transação. O BackingMap acompanha estas entradas e escolhe quando liberar uma ou mais entradas do BackingMap.

Um BackingMap não possui informações de configuração para um tamanho máximo. Em vez disso, as propriedades do evictor são configuradas para controlar o comportamento do evictor. O LRUEvictor e o LFUEvictor possuem uma propriedade de tamanho máximo utilizada para fazer o evictor começar a liberar entradas quando o tamanho máximo for excedido. Assim como o evictor TTL, os evictores LRU e LFU podem não liberar imediatamente uma entrada quando o número máximo de entradas for atingido para minimizar o impacto no desempenho.

Se o algoritmo LRU ou LFU não for adequado para um determinado aplicativo, será possível redigir seus próprios evictors para criar a sua estratégia de despejo.

Despejo Baseado em Memória

Importante: O despejo baseado em memória é suportado somente no Java Platform, Enterprise Edition Versão 5 ou posterior.

Todos os evictores integrados suportam despejo baseado em memória, que pode ser ativado na interface BackingMap por meio da configuração do atributo evictionTriggers de BackingMap como `MEMORY_USAGE_THRESHOLD`. Para obter mais informações sobre como configurar o atributo evictionTriggers na BackingMap, consulte as informações sobre a interface BackingMap e o arquivo XML do descritor do ObjectGrid no *Guia de Programação*.

O despejo baseado em memória é baseado no limite de uso do heap. Quando o despejo baseado em memória é ativado no BackingMap e o BackingMap possui algum evictor integrado, o limite de uso é configurado com uma porcentagem padrão de memória total se o limite ainda não tiver sido configurado anteriormente.

Ao usar o despejo baseado em memória, você deveria configurar o limite de coleta de lixo para o mesmo valor que a utilização de heap de destino. Por exemplo, se o limite de despejo baseado em memória for configurado em 50 por cento e o limite de coleta de lixo estiver configurado no nível padrão de 70 por cento, então a

utilização de heap pode chegar no máximo a 70 por cento. Esta aumento da utilização de heap ocorre proque o despejo baseado em memória é acionado somente depois de um ciclo de coleta de lixo.

O algoritmo de despejo baseado em memória usado pelo WebSphere eXtreme Scale é sensível ao comportamento do algoritmo de coleta de lixo em uso. O melhor algoritmo para despejo baseado em memória é o padrão da IBM por meio do coletor. A geração de algoritmos de coleta de lixo podem provocar comportamento indesejado e, dessa forma, você não deve usar estes algoritmos com o despejo baseado em memória.

Para alterar a porcentagem de limite de uso, configure a propriedade memoryThresholdPercentage no contêiner e os arquivos de propriedades do servidor para os processos do servidor do eXtreme Scale.

Durante o tempo de execução, se o uso da memória exceder o limite de uso destinado, os evictors baseados em memória iniciam o despejo de entradas e tentam manter o uso da memória abaixo do limite de uso destinado. Porém, não há garantia de que a velocidade do despejo seja rápida o suficiente para evitar um possível erro de falta de memória se o tempo de execução do sistema continuar consumindo memória rapidamente.

Evictor TimeToLive (TTL)

O WebSphere eXtreme Scale fornece um mecanismo padrão para despejar entradas do cache e um plug-in para criar evictors customizados. Um Evictor controla a associação de entradas em cada instância do BackingMap.

Ativar o Evictor TTL Programaticamente

Os evictors TTL estão associados a instâncias do BackingMap. O evictor padrão utiliza uma política de despejo time-to-live (TTL) para cada instância de BackingMap. Se você fornecer um mecanismo de evictor conectável, geralmente ele utilizará uma política de evicção baseada no número de entradas em vez de baseada no tempo.

O fragmento de código a seguir utiliza a interface BackingMap para configurar o prazo de expiração de cada entrada para 10 minutos após a entrada ser criada.

```
evictor time-to-live
programáticoimport
com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.TTLType;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid("grid");
BackingMap bm = og.defineMap( "myMap" );
bm.setTtlEvictorType( TTLType.CREATION_TIME );
bm.setTimeToLive( 600 );
```

O argumento do método setTimeToLive é 600 porque isso indica que o valor de time-to-live está em segundos. O código anterior deve ser executado antes de o método initialize ser chamado na instância do ObjectGrid. Estes atributos de BackingMap não podem ser alterados após a inicialização da instância do ObjectGrid. Após a execução do código, qualquer entrada inserida no BackingMap myMap tem um tempo de expiração. Ao final do tempo de expiração, o evictor de TTL remove a entrada.

Para configurar o prazo de expiração para o horário do último acesso mais 10 minutos, altere o argumento que é transmitido para o método setTtlEvictorType de TTLType.CREATION_TIME para TTLType.LAST_ACCESS_TIME. Com este valor, o tempo de expiração é calculado como a hora do último acesso mais 10 minutos. Quando uma entrada é criada pela primeira vez, a hora do último acesso é a hora de criação. Para basear o prazo de expiração na última *atualização*, em vez de simplesmente no último *acesso* (se ele envolveu ou não uma atualização), substitua a configuração TTLType.LAST_UPDATE_TIME pela configuração TTLType.LAST_ACCESS_TIME.

Ao usar a configuração TTLType.LAST_ACCESS_TIME ou TTLType.LAST_UPDATE_TIME, é possível usar as interfaces ObjectMap e JavaMap para substituir o valor time-to-live de BackingMap. Esse mecanismo permite que um aplicativo utilize um valor de time-to-live diferente para cada entrada criada. Suponha que o fragmento de conjunto de códigos precedente configure o atributo ttlType como LAST_ACCESS_TIME e configure o valor de time-to-live como 10 minutos. Qualquer aplicativo pode substituir o valor de time-to-live para cada entrada executando o seguinte código antes de criar ou modificar uma entrada:

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.ObjectMap;
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
int oldTimeToLive1 = om.setTimeToLive( 1800 );
om.insert("key1", "value1" );
int oldTimeToLive2 = om.setTimeToLive( 1200 );
om.insert("key2", "value2" );
```

No fragmento de código anterior, a entrada com a chave key1 tem um prazo de expiração composto pelo tempo de inserção mais 30 minutos como resultado da solicitação de método setTimeToLive(1800) na instância de ObjectMap. A variável oldTimeToLive1 é configurada como 600 porque o valor de time-to-live de BackingMap é utilizado como um valor padrão se o método setTimeToLive não foi chamado anteriormente na instância de ObjectMap.

A entrada com a chave key2 tem um prazo de expiração composto pelo tempo de inserção mais 20 minutos como resultado da chamada de método setTimeToLive(1200) na instância de ObjectMap. A variável oldTimeToLive2 é configurada como 1800 porque o valor de time-to-live da solicitação de método ObjectMap.setTimeToLive anterior configura o valor de time-to-live como 1800.

O exemplo anterior mostra duas entradas de mapa sendo inseridas no mapa myMap para as chaves key1 e key2. Posteriormente, o aplicativo ainda pode atualizar essas entradas de mapa enquanto retém os valores de time-to-live que são utilizados no tempo de inserção para cada entrada de mapa. O exemplo a seguir ilustra como reter os valores de TTL (Time-to-Live), utilizando uma constante definida na interface ObjectMap:

```
Session session = og.getSession();
ObjectMap om = session.getMap( "myMap" );
om.setTimeToLive( ObjectMap.USE_DEFAULT );
session.begin();
om.update("key1", "updated value1" );
om.update("key2", "updated value2" );
om.insert("key3", "value3" );
session.commit();
```

Como o valor especial de ObjectMap.USE_DEFAULT é utilizado na chamada de método setTimeToLive, a chave key1 retém seu valor de time-to-live de 1800

segundos e a chave key2 retém seu valor de time-to-live de 1200 segundos, pois esses valores foram utilizados quando essas entradas de mapa foram inseridas pela transação anterior.

O exemplo anterior também mostra uma nova entrada de mapa para a inserção da chave key3. Neste caso, o valor especial USE_DEFAULT indica a utilização da configuração padrão do valor time-to-live para este mapa. O valor padrão é definido pelo atributo time-to-live de BackingMap. Consulte atributos da interface BackingMap para obter informações sobre como o atributo time-to-live é definido na instância do BackingMap.

Consulte a documentação da API para o método setTimeToLive nas interfaces ObjectMap e JavaMap. A documentação explica que o resultado será uma exceção IllegalStateException se o método BackingMap.getTtlEvictorType retornar qualquer coisa diferente do valor TTLType.LAST_ACCESS_TIME ou TTLType.LAST_UPDATE_TIME. As interfaces ObjectMap e JavaMap podem substituir o valor time-to-live apenas quando você está usando a configuração LAST_ACCESS_TIME ou TTLType.LAST_UPDATE_TIME para o tipo de evictor TTL. O método setTimeToLive não pode ser utilizado para substituir o valor de time-to-live quando você estiver utilizando a configuração de tipo de evictor CREATION_TIME ou NONE.

Ativar o Evictor TTL Utilizando 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:

Ativando evictor de tempo de vida 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="grid1">
        <backingMap name="map1" ttlEvictorType="NONE" />
        <backingMap name="map2" ttlEvictorType="LAST_ACCESS_TIME|LAST_UPDATE_TIME"
     timeToLive="1800" />
        <backingMap name="map3" ttlEvictorType="CREATION_TIME" timeToLive="1200" />
    </objectgrid>
</objectGrids>
```

O exemplo anterior mostra que a instância map1 de BackingMap utiliza um tipo de evictor NONE TTL. A instância BackingMap map2 usa o tipo de evictor TTL LAST_ACCESS_TIME ou LAST_UPDATE_TIME – especifique uma ou outra dessas configurações – e tem um valor time-to-live de 1800 segundos ou 30 minutos. A instância map3 de BackingMap é definida para utilizar um tipo de evictor CREATION_TIME TTL e tem um valor de time-to-live de 1200 segundos ou 20 minutos.

Conexão de um Evictor programável

Como os evictors são associados com BackingMaps, utilize a interface BackingMap para especificar o evictor conectável.

Evictores Conectáveis Opcionais

O evictor TTL padrão utiliza uma política de evicção baseada no tempo e o número de entradas no BackingMap não tem efeito sobre o tempo de expiração de uma entrada. É possível utilizar um evictor conectável opcional para despejar entradas com base no número de entradas existentes em vez de no tempo.

Os seguintes evictores conectáveis opcionais fornecem alguns algoritmos comumente utilizados para decidir quais entradas liberar quando um BackingMap crescer além de algum limite de tamanho.

- O evictor LRUEvictor utiliza um algoritmo LRU (Least Recently Used) para decidir quais entradas serão despejadas quando o BackingMap exceder um número máximo de entradas.
- O evictor LFUEvictor utiliza um algoritmo LFU (Least Frequently Used) para decidir quais entradas serão despejadas quando o BackingMap exceder um número máximo de entradas.

O BackingMap informa um evictor conforme as entradas são criadas, modificadas ou removidas de uma transação. O BackingMap acompanha estas entradas e escolhe quando liberar uma ou mais entradas da instância do BackingMap.

Uma instância do BackingMap não possui informações de configuração para um tamanho máximo. Em vez disso, as propriedades do evictor são configuradas para controlar o comportamento do evictor. O LRUEvictor e o LFUEvictor possuem uma propriedade de tamanho máximo utilizada para fazer o evictor começar a liberar entradas quando o tamanho máximo for excedido. Assim como o evictor TTL, os evictores LRU e LFU podem não liberar imediatamente uma entrada quando o número máximo de entradas for atingido para minimizar o impacto no desempenho.

Se o algoritmo LRU ou LFU não for adequado para um determinado aplicativo, será possível redigir seus próprios evictors para criar a sua estratégia de despejo.

Utilizando Evictors Conectáveis Opcionais

Para incluir evictors conectáveis opcionais na configuração do BackingMap, é possível utilizar tanto a configuração programática quanto a configuração do XML.

Conectando um Evictor Conectável Programaticamente

Como os evictors estão associados aos BackingMaps, use a interface BackingMap para especificar o evictor conectável. O trecho de código a seguir é um exemplo de especificação de um evictor LRUEvictor para o BackingMap map1 e um evictor LFUEvictor para a instância do BackingMap map2:

Conectando um evictor programaticamente

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor;
import com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid("grid");
BackingMap bm = og.defineMap( "map1" );
LRUEvictor evictor = new LRUEvictor();
evictor.setMaxSize(1000);
evictor.setSleepTime( 15 );
```

```
evictor.setNumberOfLRUQueues( 53 );
bm.setEvictor(evictor);
bm = og.defineMap( "map2" );
LFUEvictor evictor2 = new LFUEvictor();
evictor2.setMaxSize(2000);
evictor2.setSleepTime( 15 );
evictor2.setNumberOfHeaps( 211 );
bm.setEvictor(evictor2);
```

O snippet anterior mostra um evictor LRUEvictor sendo utilizado para o map1 BackingMap com um número aproximado de entradas de 53.000 (53 * 1000). O evictor LFUEvictor é utilizado para o map2 BackingMap com um número máximo aproximado de entradas de 422.000 (211 * 2000). Os evictores LRU e LFU têm uma propriedade de tempo de suspensão que indica por quanto tempo o evictor fica suspenso antes de ser ativado e verificar se as entradas precisam ser liberadas. O tempo de suspensão é especificado em segundos. Um valor de 15 segundos é uma boa garantia entre o impacto no desempenho e a prevenção para que o BackingMap não se torne muito grande. A meta é utilizar o maior tempo de suspensão possível sem fazer o BackingMap crescer a um tamanho excessivo.

O método setNumberOfLRUQueues configura a propriedade LRUEvictor que indica quantas filas de LRU o evictor utiliza para gerenciar informações de LRU. Uma coleta de filas é utilizada para que cada entrada não mantenha informações de LRU na mesma fila. Esta abordagem pode melhorar o desempenho minimizando o número de entradas do mapa que precisam ser sincronizadas no mesmo objeto de fila. Aumentar o número de filas é uma boa maneira de minimizar o impacto que o evictor LRU pode causar no desempenho. Um bom ponto de partida é utilizar dez por cento do número máximo de entradas como o número de filas. A utilização de um número primo geralmente é melhor do que utilizar um número que não seja primo. O método setMaxSize indica quantas entradas são permitidas em cada fila. Quando uma fila alcança seu número máximo entradas, a entrada ou as entradas utilizadas menos recentemente nesta fila são despejadas na próxima vez que o evictor verifica se há alguma entrada que precisa ser despejada.

O método setNumberOfHeaps configura a propriedade LFUEvictor para determinar quantos objetos de heap binários o LFUEvictor utiliza para gerenciar informações de LFU. Mais uma vez é utilizada uma coleta para melhorar o desempenho. A utilização de dez por cento do número máximo de entrada é um bom ponto de partida e utilizar um número primo geralmente é melhor do que utilizar um número que não seja primo. O método setMaxSize indica quantas entradas são permitidas em cada heap. Quando um heap alcança seu número máximo entradas, a entrada ou as entradas utilizadas menos recentemente neste heap são despejadas na próxima vez que o evictor verifica se há alguma entrada que precisa ser despejada.

Abordagem de Configuração XML para Conectar um Evictor Conectável

Em vez de utilizar várias APIs para conectar programaticamente um evictor e configurar suas propriedades, um arquivo XML pode ser utilizado para configurar cada BackingMap conforme ilustrado na amostra a seguir:

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

```
        <backingMap name="map1" ttlEvictorType="NONE" pluginCollectionRef="LRU" />
        <backingMap name="map2" ttlEvictorType="NONE" pluginCollectionRef="LFU" />
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPlugincollection id="LRU">
        <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
            <property name="maxSize" type="int" value="1000" description="set max size for each LRU queue" />
            <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
            <property name="numberOfLRUQueues" type="int" value="53" description="set number of LRU queues" />
        </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="LFU">
        <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
            <property name="maxSize" type="int" value="2000" description="set max size for each LFU heap" />
            <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
            <property name="numberOfHeaps" type="int" value="211" description="set number of LFU heaps" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Despejo Baseado em Memória

Todos os evictors integrados suportam despejo baseado em memória que pode ser ativado na interface BackingMap configurando o atributo evictionTriggers de BackingMap como "MEMORY_USAGE_THRESHOLD". Para obter mais informações sobre como configurar o atributo evictionTriggers no BackingMap, consulte a interface BackingMap e a referência de configuração do eXtreme Scale.

O despejo baseado em memória é baseado no limite de uso do heap. Quando o despejo baseado em memória é ativado no BackingMap e o BackingMap possui algum evictor integrado, o limite de uso é configurado com uma porcentagem padrão de memória total se o limite ainda não tiver sido configurado anteriormente.

Para alterar a porcentagem de limite de uso, configure a propriedade memoryThresholdPercentage no contêiner e o arquivo de propriedades do servidor para o processo do servidor do eXtreme Scale. Para configurar o limite de uso de destino em um processo do cliente do eXtreme Scale, é possível utilizar o MemoryPoolMXBean. Consulte também: Arquivo containerServer.props e Iniciando Processos do Servidor eXtreme.

Durante o tempo de execução, se o uso da memória exceder o limite de uso destinado, os evictors baseados em memória iniciam o despejo de entradas e tentam manter o uso da memória abaixo do limite de uso destinado. Entretanto, não há garantia de que a velocidade do despejo seja rápida o suficiente para evitar um potencial erro de falta de memória se o tempo de execução do sistema continuar a consumir memória rapidamente.

Gravando um Evictor Customizado

WebSphere eXtreme Scale permite que você grave uma implementação de despejo customizada.

É necessário criar um evictor customizado que implemente a interface do evictor e siga as convenções de plug-in do eXtreme Scale comuns. A interface é a seguinte:

```
public interface Evictor
{
    void initialize(BackingMap map, EvictionEventCallback callback);
    void activate();
    void apply(LogSequence sequence);
    void deactivate();
    void destroy();
}
```

- O método initialize é chamado durante a inicialização do objeto BackingMap. Este métodos inicializa um plug-in Evictor com uma referência ao BackingMap e uma referência a um objeto que implementa a interface com.ibm.websphere.objectgrid.plugins.EvictionEventCallback.
- O método activate é chamado para ativar o Evictor. Após este método ser chamado, o Evictor pode usar a interface EvictionEventCallback para despejar entradas de mapa. Se o Evictor tentar utilizar EvictionEventCallbackinterface para despejar entradas de mapas antes do método activate ser chamado, resultará em uma exceção IllegalStateException.
- O método apply é chamado quando as transações que acessam uma ou mais entradas do BackingMap são consolidadas. O método apply recebe uma referência a um objeto que implementa a interface com.ibm.websphere.objectgrid.plugins.LogSequence. A interface LogSequence permite que um plug-in Evictor determine quais entradas de BackingMap foram criadas, modificadas ou removidas pela transação. Um Evictor utiliza estas informações ao decidir quando e quais entradas serão liberadas.
- O método deactivate é chamado para desativar o Evictor. Após este método ser chamado, o Evictor deve parar a utilização da interface EvictionEventCallback para despejar entradas de mapas. Se o Evictor utilizar a interface EvictionEventcallback após este método ser chamado, resultará em uma exceção IllegalStateException.
- O método destroy é chamado quando o BackingMap está sendo destruído. Este método permite que um Evictor termine quaisquer encadeamentos que ele possa ter criado.

A interface EvictionEventCallback tem os seguintes métodos:

```
public interface EvictionEventCallback
{
    void evictMapEntries(List evictorDataList) throws ObjectGridException;
    void evictEntries(List keysToEvictList) throws ObjectGridException;
    void setEvictorData(Object key, Object data);
    Object getEvictorData(Object key);
}
```

Os métodos EvictionEventCallback são usados por um plug-in Evictor para retornar à estrutura eXtreme Scale como a seguir:

- O método setEvictorData é usado por um evictor para solicitar à estrutura que é usada para armazenar e associar algum objeto evictor que ele cria com a entrada indicada pelo argumento-chave. Os dados são específicos do evictor e são determinados pelas informações requeridas pelo evictor para implementar o algoritmo que está sendo utilizado. Por exemplo, em um algoritmo least frequently used, o evictor mantém uma contagem no objeto de dados do evictor para rastrear quantas vezes o método apply é chamado com um LogElement que se refere a uma entrada para uma chave especificada.
- O método getEvictorData é usado por um evictor para recuperar os dados que ele passa ao método setEvictorData durante uma solicitação de método apply anterior. Se os dados do Evictor para o argumento de chave especificado não for localizado, um objeto especial KEY_NOT_FOUND que é definido na interface EvictorCallback será retornado.
- O método evictMapEntries é usado por um evictor para recuperar o despejo de uma ou mais entradas de mapa. Cada objeto no parâmetro evictorDataList deve implementar a interface com.ibm.websphere.objectgrid.plugins.EvictorData. Além disso, a mesma instância EvictorData transmitida para o método setEvictorData deve estar no parâmetro da lista de dados do evictor deste método. O método getKey da interface EvictorData é utilizado para determinar

qual entrada do mapa será liberada. A entrada do mapa será liberada se a entrada de cache contiver exatamente a mesma instância EvictorData que está na lista de dados do evictor para esta entrada de cache.

- O método evictEntries é usado por um evictor para solicitar despejo de uma ou mais entradas de mapa. Este método será utilizado apenas se o objeto transmitido para o método setEvictorData não implementar a interface com.ibm.websphere.objectgrid.plugins.EvictorData.

Após uma transação ser concluída, o eXtreme Scale chama o método apply da interface do Evictor. Todos os bloqueios de transação que foram adquiridos pela transação concluída não estão mais suspensos. Provavelmente, vários encadeamentos podem chamar o método apply ao mesmo tempo e cada encadeamento pode concluir sua própria transação. Como os bloqueios de transação já foram liberados pela transação concluída, o método apply deve fornecer sua própria sincronização para assegurar que o método apply seja seguro em encadeamento.

A razão para implementar a interface EvictorData e utilizar o método evictMapEntries em vez do método evictEntries é fechar uma possível janela de cronometragem. Considere a seguinte seqüência de eventos:

1. A transação 1 é concluída e chama o método apply com uma LogSequence que exclui a entrada do mapa para a chave 1.
2. A transação 2 é concluída e chama o método apply com uma LogSequence que insere uma nova entrada do mapa para a chave 1. Em outras palavras, a transação 2 recria a entrada do mapa que foi excluída pela transação 1.

Como o evictor é executado assincronicamente a partir de encadeamentos que executam transações, é possível que quando o evictor decida liberar a chave 1, ele possa liberar a entrada do mapa existente antes da conclusão da transação 1 ou possa liberar a entrada do mapa que foi recriada pela transação 2. Para eliminar janelas de cronometragens e eliminar a incerteza quanto à qual versão da entrada do mapa da chave 1 o evictor pretendia liberar, implemente a interface EvictorData pelo objeto transmitido ao método setEvictorData. Utilize a mesma instância EvictorData durante a existência de uma entrada do mapa. Quando essa entrada do mapa for excluída e, em seguida, recriada por outra transação, o evictor deverá utilizar uma nova instância da implementação de EvictorData. Utilizando a implementação de EvictorData e utilizando o método evictMapEntries, o evictor pode assegurar que a entrada do mapa seja liberada apenas se a entrada de cache associada à entrada do mapa contiver a instância EvictorData correta.

As interfaces Evictor e EvictonEventCallback permitem que um aplicativo conecte um evictor que implementa um algoritmo definido pelo usuário para evicção. O fragmento de código a seguir ilustra como é possível implementar o método initialize da interface Evictor:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
import java.util.LinkedList;
// Instance variables
private BackingMap bm;
private EvictionEventCallback evictorCallback;
private LinkedList queue;
private Thread evictorThread;
public void initialize(BackingMap map, EvictionEventCallback callback)
{
    bm = map;
    evictorCallback = callback;
    queue = new LinkedList();
    // spawn do encadeamento do evictor
    evictorThread = new Thread( this );
```

```
    String threadName = "MyEvictorForMap-" + bm.getName();
    evictorThread.setName( threadName );
    evictorThread.start();
}
```

O código anterior salva as referências ao mapa e objetos de retorno de chamada em variáveis da instância para que fiquem disponíveis para os métodos apply e destroy. Neste exemplo, uma lista vinculada é criada para utilização como uma fila de primeiro a entrar e primeiro a sair para implementação de um algoritmo LRU (Least Recently Used). É efetuado spawn de um encadeamento e uma referência ao encadeamento é mantida como uma variável de instância. Mantendo esta referência, o método destroy pode interromper e terminar o encadeamento no qual foi efetuado spawn.

Ignorando os requisitos de sincronização para tornar um código seguro em encadeamento, o trecho de código a seguir ilustra como o método apply da interface Evictor pode ser implementado:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.EvictorData;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;

public void apply(LogSequence sequence)
{
    Iterator iter = sequence.getAllChanges();
    while(iter.hasNext())
    {
        LogElement elem = (LogElement)iter.next();
        Object key = elem.getKey();
        LogElement.Type type = elem.getType();
        if ( type == LogElement.INSERT )
        {
            // do insert processing here by adding to front of LRU queue.
            EvictorData data = new EvictorData(key);
            evictorCallback.setEvictorData(key, data);
            queue.addFirst( data );
        }
        else if ( type == LogElement.UPDATE || type == LogElement.FETCH || type == LogElement.TOUCH )
        {
            // do update processing here by moving EvictorData object to
            // front of queue.
            EvictorData data = evictorCallback.getEvictorData(key);
            queue.remove(data);
            queue.addFirst(data);
        }
        else if ( type == LogElement.DELETE || type == LogElement.EVICT )
        {
            // do remove processing here by removing EvictorData object
            // from queue.
            EvictorData data = evictorCallback.getEvictorData(key);
            if ( data == EvictionEventCallback.KEY_NOT_FOUND )
            {
                // Assumption here is your asynchronous evictor thread
                // evicted the map entry before this thread had a chance
                // to process the LogElement request. So you probably
                // need to do nothing when this occurs.
            }
            else
            {
                // Key was found. So process the evictor data.
                if ( data != null )
                {
                    // Ignore null returned by remove method since spawned
                    // evictor thread may have already removed it from queue.
                    // But we need this code in case it was not the evictor
                    // thread that caused this LogElement to occur.
                    queue.remove( data );
                }
                else
                {
                    // Depending on how you write you Evictor, this possibility
                    // may not exist or it may indicate a defect in your evictor
                    // due to improper thread synchronization logic.
                }
            }
        }
    }
}
```

O processamento de inserção no método apply geralmente manipula a criação de um objeto de dados do evictor transmitido para o método setEvictorData da interface EvictionEventCallback. Como este evictor ilustra a implementação de um LRU, o EvictorData também é incluído na frente da fila que foi criada pelo método initialize. O processamento de atualização no método apply geralmente atualiza o objeto de dados do evictor criado por alguma chamada anterior do método apply (por exemplo, pelo processamento de inserção do método apply). Como este evictor é a implementação de um LRU, ele precisa mover o objeto EvictorData de sua posição de fila atual para a frente da fila. O encadeamento do evictor no qual foi efetuado spawn remove o último objeto EvictorData na fila, porque o último elemento da fila representa a entrada least recently used. A suposição é que o objeto EvictorData contenha um método getKey para que o encadeamento do evictor saiba quais são as chaves das entradas que precisam ser liberadas. Observe que este exemplo está ignorando os requisitos de sincronização para tornar o código seguro em encadeamento. Um evictor customizado real é mais complicado, porque lida com sincronização e gargalos de desempenho que ocorrem como resultado dos pontos de sincronização.

Os seguintes trechos de código ilustram o método destroy e o método run do encadeamento executável no qual foi efetuado spawn pelo método initialize:

```
// Destroy method simply interrupts the thread spawned by the initialize method.
public void destroy()
{
    evictorThread.interrupt();
}

// Here is the run method of the thread that was spawned by the initialize method.
public void run()
{
    // Loop until destroy method interrupts this thread.
    boolean continueToRun = true;
    while ( continueToRun )
    {
        try
        {
            // Sleep for a while before sweeping over queue.
            // The sleepTime is a good candidate for a evictor
            // property to be set.
            Thread.sleep( sleepTime );
            int queueSize = queue.size();
            // Evict entries if queue size has grown beyond the
            // maximum size. Obviously, maximum size would
            // be another evictor property.
            int numToEvict = queueSize - maxSize;
            if ( numToEvict > 0 )
            {
                // Remove from tail of queue since the tail is the
                // least recently used entry.
                List evictList = new ArrayList( numToEvict );
                while( queueSize > ivMaxSize )
                {
                    EvictorData data = null;
                    try
                    {
                        EvictorData data = (EvictorData) queue.removeLast();
                        evictList.add( data );
                        queueSize = queue.size();
                    }
                    catch ( NoSuchElementException nse )
                    {
                        // The queue is empty.
                        queueSize = 0;
                    }
                }
                // Request eviction if key list is not empty.
                if ( ! evictList.isEmpty() )
                {
                    evictorCallback.evictMapEntries( evictList );
                }
            }
        }
        catch (InterruptedException e)
        {
            continueToRun = false;
        }
    } // end while loop
} // end run method.
```

Interface RollBackEvictor Opcional

A interface com.ibm.websphere.objectgrid.plugins.RollbackEvictor pode ser opcionalmente implementada por um plug-in Evictor. Implementando esta interface, um evictor pode ser chamado não apenas quando as transações forem confirmadas, mas também quando for efetuado rollback das transações.

```
public interface RollbackEvictor
{
    void rollingBack( LogSequence ls );
}
```

O método apply será chamado apenas de uma transação for confirmada. Se for efetuado rollback de uma transação e a interface RollbackEvictor for implementada pelo evictor, o método rollingBack será chamado. Se a interface RollbackEvictor não for implementada e for efetuado rollback da transação, o método apply e o método rollingBack não serão chamados.

Boas Práticas de Desempenho do Evictor de Plug-in

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 evictores least frequently used (LFU) e least recently 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:

```
//Use ObjectGridManager to create/get the ObjectGrid. Refer to
// the ObjectGridManger section
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");

//Set properties assuming 50,000 map entries
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");

//Set properties assuming 50,000 map entries
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.

Plug-ins para Transformar Objetos Armazenados em Cache

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 detectam colisões automaticamente. Quando um domínio detecta uma colisão, ele chama um árbitro. Geralmente, as colisões são resolvidas usando o árbitro de colisão padrão. No entanto, um aplicativo pode fornecer um árbitro de colisão customizado.

Sobre Esta Tarefa

Se um domínio recebe uma entrada replicada que colide com um registro local, o árbitro padrão usa as mudanças do domínio nomeado mais baixo lexicalmente. 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 para 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 fornece as seguintes informações para um árbitro de colisão customizado:

- A versão externa do registro
- A versão local 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 indicando sua decisão. O método chamado pelo domínio para chamar o plug-in deve retornar verdadeiro ou falso, onde falso significa ignorar a colisão – a versão local permanece inalterada e eXtreme Scale esquecerá que já viu a versão externa. 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 usar o plug-in de 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>
```

Plug-in ObjectTransformer

Com o plug-in ObjectTransformer, é possível serializar, desserializar e copiar objetos no cache para aumentar o desempenho.

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.

Finalidade

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.

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>
```

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);
```

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);
```

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. Delegar os dados é a operação mais dispendiosa e deve ser endereçada pelo desenvolvedor de aplicativos ao projetar o esquema, configurar a grade 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.

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.

Para obter mais informações, consulte Plug-ins para Serializar e Copiar Objetos em Cache e Boas Práticas da Interface de ObjectTransformer.

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. Consulte Para obter mais informações, consulte o tópico sobre o uso de um utilitário de carga com os mapas e tuplas de entidade no *Guia de Programação*. 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 nos quais os objetos devem ser modificados para utilizar a serialização customizada, tais como a implementação da interface java.io.Externalizable ou pela implementação dos 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 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 por meio 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.

Boas Práticas da 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.

Visão Geral

Para obter detalhes sobre a interface ObjectTransformer, consulte "Plug-in ObjectTransformer" na página 227. A partir de um ponto de vista de desempenho, e das informações do método CopyMode que estão no tópico de boas práticas do método CopyMode, o eXtreme Scale claramente copia os valores para todos os casos, 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 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.

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.

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.

Finalidade

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 chama o método 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.

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>
```

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 );
```

Plug-ins para Fornecer Listeners de Eventos

É possível utilizar os plug-ins ObjectGridEventListener e MapEventListener para configurar notificações para vários eventos no cache do 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 240. 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 utilizando os métodos addEventListener, setEventListeners 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 240.

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”. 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, setMapEventListeners 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”.

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 “Introdução aos Plug-ins” na página 209. Para os outros tipos de plug-ins de listener, consulte “Plug-ins para Fornecer Listeners de Eventos” na página 238.

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.

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);
```

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);
```

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 por meio da interface ObjectGridEventGroup.TransactionEvents.

Vantagens desta Abordagem

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 por meio 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.

Configure um plug-in 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);
```

Configure um plug-in 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-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.

Para obter informações gerais sobre indexação, consulte Indexação.

Para obter informações sobre o uso da indexação, consulte "Utilizando a Indexação para Acessar Dados Sem Chave" na página 248.

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) {
            // this is a tuple map
            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);
        }
    }
```

HashIndex 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 (somente para mapas de entidade) com anotações de entidade.

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>
```

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.

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.

Utilizando a Indexação para Acessar Dados Sem Chave

O uso de indexação como uma alternativa para acessar dados com chave é muito mais eficiente.

Etapas Necessárias

1. Inclua os plug-ins de índice estático ou dinâmico no BackingMap.
2. Obtenha o objeto de proxy do índice do aplicativo, emitindo o método getIndex do ObjectMap.
3. Direcione o objeto de proxy de índice a uma interface de índice de aplicativo apropriado, como MapIndex, MapRangeIndex, ou uma interface de índice customizada.
4. Utilize os métodos definidos na interface do índice 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.

Nota: Em um ambiente distribuído, se o objeto do índice for obtido de um ObjectGrid do cliente, ele terá objeto de índice do tipo cliente e todas as operações de índice serão executadas em um ObjectGrid do servidor remoto. Se o mapa for particionado, a operação de índice executará em cada partição remotamente e os resultados de cada partição serão mesclados antes de serem retornados ao aplicativo. O desempenho será determinado pelo número de partições e pelo tamanho do resultado retornado por cada partição. Desempenho insuficiente pode ocorrer se ambos os fatores forem altos.

Para obter informações sobre a configuração do HashIndex, consulte Configurando o HashIndex.

Se desejar gravar seu próprio plug-in de índice, consulte “Plug-ins para Indexação Customizada de Objetos de Cache” na página 243.

Para obter informações sobre indexação, consulte Indexação e “HashIndex Composto” na página 245.

Incluindo Plug-ins de Índice Estático

É possível utilizar duas abordagens para incluir plug-ins de índice estático na configuração de BackingMap: configuração XML e configuração programática. O exemplo a seguir ilustra a abordagem da configuração XML.

Incluindo plug-ins de índice estático: abordagem de 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. O HashIndex suporta propriedades que os usuários podem configurar, como Name, RangeIndex e AttributeName no exemplo anterior.

- A propriedade Name é configurada como "CODE", uma cadeia 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 converter o objeto do índice recuperado para a interface MapRangeIndex. Se a propriedade RangeIndex for configurada como "false", o aplicativo pode somente converter o 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 de atributo único for usado pela consulta, a propriedade RangeIndex deverá ser definida para "true" em índices de atributo único. Para um índice de relacionamento e índice composto, a propriedade RangeIndex deverá ser definida 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 múltiplos atributos, a propriedade AttributeName pode ser configurada para uma lista de atributos delimitados por vírgula, rendendo um índice composto.

Consulte as informações sobre a configuração do HashIndex no *Guia de Administração* para obter mais informações.

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 de código a seguir ilustra a abordagem da configuração programática:

Incluindo plug-ins de índices estáticos: abordagem da configuração programática

```
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");

    // use the builtin HashIndex class as the index plugin class.
    HashIndex mapIndexplugin = new HashIndex();
    mapIndexplugin.setName("CODE");
    mapIndexplugin.setAttributeName("EmployeeCode");
    mapIndexplugin.setRangeIndex(true);
    personBackingMap.addMapIndexplugin(mapIndexplugin);
```

Utilizando 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.

O código de exemplo a seguir ilustra como recuperar e usar índices estáticos.

Exemplo de utilização de índices estáticos

```
Session session = ivObjectGrid.getSession();
    ObjectMap map = session.getMap("person");
    MapRangeIndex codeIndex = (MapRangeIndex) m.getIndex("CODE");
    Iterator iter = codeIndex.findLessEqual(new Integer(15));
    while (iter.hasNext()) {
        Object key = iter.next();
        Object value = map.get(key);
    }
```

Incluindo, Removendo e Utilizando 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 API de BackingMap para obter informações adicionais sobre os métodos createDynamicIndex e removeDynamicIndex.

Utilizando o exemplo de índices dinâmicos

```
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 {
            // If not using DynamicIndexCallback, need to wait for the Index to be ready.
            // The waiting time depends on the current size of the map
            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");
```

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. 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 documentação da APIAPI DynamicIndexCallback na documentação da API para obter informações adicionais.

O exemplo de código a seguir ilustra a utilização da interface DynamicIndexCallback:

Utilizando a interface DynamicIndexCallback

```
BackingMap personBackingMap = ivObjectGrid.getMap("person");
    DynamicIndexCallback callback = new DynamicIndexCallbackImpl();
    personBackingMap.createDynamicIndex("CODE", true, "employeeCode", callback);


    class DynamicIndexCallbackImpl implements DynamicIndexCallback {
        public DynamicIndexCallbackImpl() {
        }

        public void ready(String indexName) {
            System.out.println("DynamicIndexCallbackImpl.ready() -> indexName = " + indexName);

            // 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);
        }
    }
```

Plug-ins para a Comunicação com Armazenamentos Persistentes

Com um plug-in de utilitário de carga eXtreme Scale, um mapa ObjectGrid pode se comportar como um cache de memória para dados que são tipicamente 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 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).

Consulte as informações sobre os conceitos de armazenamento em cache no *Visão Geral do Produto* para obter informações adicionais.

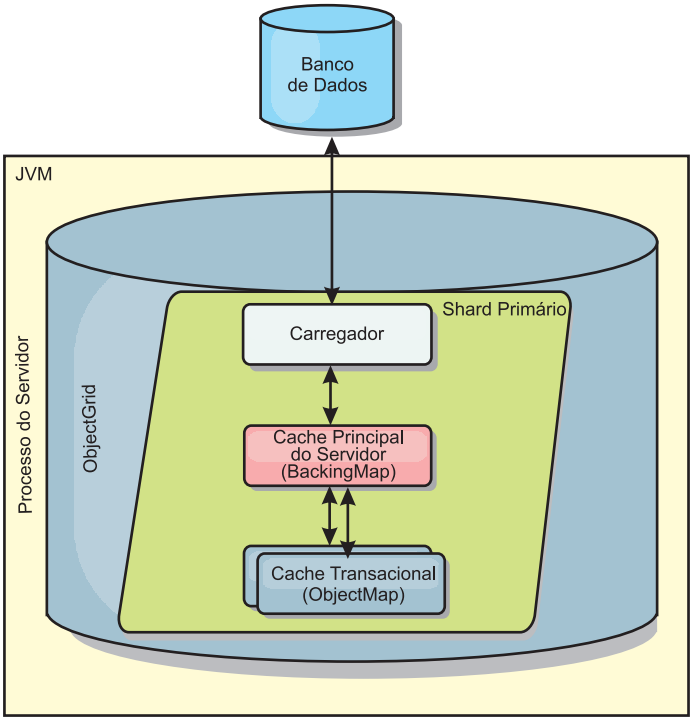


Figura 3. Utilitário de Carga

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.

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 utilitário de carga MyLoader ao mapa de apoio map1 com o mesmo nome de banco de dados e propriedades do Utilitário de Carga do nível de isolamento:

```
<?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>
```

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 as informações sobre os Carregadores no *Visão Geral do Produto* para obter mais informações.

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 272.

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 {
        // Get a SQL connection to use.
        Connection conn = getConnection(tx);
        try {
            // Process the list of changes and build a set of prepared
            // statements for executing a batch update, insert, or delete
            // SQL operation.
            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;
                }
            }
            // Execute the batch statements that were built by above loop.
            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 "Programação do utilitário de pré-carregamento JPA baseado em cliente" na página 309 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();
            // Get a auto-commit connection to use that is set to
            // a read committed isolation level.
            conn = getAutoCommitConnection(tx);
            // Preload the Employee Map with EmployeeRecord
            // objects. Read all Employees from table, but
            // limit preload to first 100 rows.
            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;
            }
            // Commit the transaction.
            session.commit();
            tranActive = false;
        } catch (Throwable t) {
            throw new LoaderException("preload failure: " + t, t);
        } finally {
            if (tranActive) {
                try {
                    session.rollback();
                } catch (Throwable t2) {
                    // Tolerate any rollback failures and
                    // allow original Throwable to be thrown.
                }
            }
```

```
            // Be sure to clean up other databases resources here
            // as well such a closing statements, result sets, 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;

    // Loader instance variables.
    MyTransactionCallback ivTcb; // MyTransactionCallback

    // extends TransactionCallback
    MyOptimisticCallback ivOcb; // MyOptimisticCallback

    // implements OptimisticCallback
    // ...
    public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException [Replication programming]
        // Cache TransactionCallback and OptimisticCallback objects
        // in instance variables of this Loader.
```

```
        ivTcb = (MyTransactionCallback) session.getObjectGrid().getTransactionCallback();
        ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
        // The remainder of preloadMap code (such as shown in prior example).
    }
```

Para obter informações sobre o pré-carregamento e o pré-carregamento recuperável já que pertence ao failover de replicação, consulte as informações 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 262.

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 as 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.

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.

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.

Visão Geral

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.

O 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 254.

O 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 272 para obter detalhes.

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.

Visão Geral

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 Utilitários de Carga, consulte as informações sobre os utilitários de carga 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 254.

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);

                    // get the preloadStatusMap to record preload status.
                    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 = " + whileCounter);

                        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.ivBackingMapName + ",
             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.

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.

Finalidade

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 as informações 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

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 272

página 272, 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)
{
    // remember that this localid means this session is saved for later.
    localIdToSession.put(new Integer(jta.getLocalId()), session);
    return true;
}
```

Recuperar uma Transação Externa

Algumas vezes é possível 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 279.

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.

Ativação do Plug-in WebSphereTransactionCallback

É possível ativar o WebSphereTransactionCallback na configuração ObjectGrid com a configuração programática ou XML.

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>
```

Conectar Programaticamente no Objeto WebSphereTransactionCallback

O seguinte fragmento de código cria 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);
```

Capítulo 6. Programação para Tarefas Administrativas

Além das interfaces de programação de aplicativos (APIs) do sistema, o WebSphere eXtreme Scale também inclui APIs de administração que permitem que os aplicativos monitorem e administrem servidores e clientes.

API do Servidor Integrado

O WebSphere eXtreme Scale inclui interfaces de programação de aplicativos (APIs) e interfaces de programação do sistema para integrar clientes e servidores eXtreme Scale dentro de seus aplicativos Java existentes. O tópico a seguir descreve as APIs de servidor integradas disponíveis.

Instanciando o Servidor eXtreme Scale

É possível utilizar várias propriedades para configurar a instância do servidor eXtreme Scale, que pode ser recuperada do método ServerFactory.getServerProperties. O objeto ServerProperties é um singleton, portanto, cada chamada para o método getServerProperties recupera a mesma instância.

É possível criar um novo servidor com o seguinte código.

```
Server server = ServerFactory.getInstance();
```

Todas as propriedades configuradas antes da primeira chamada de getInstance são utilizadas para inicializar o servidor.

Configurando Propriedade de Servidor

É possível configurar as propriedades do servidor até que ServerFactory.getInstance seja chamado pela primeira vez. A primeira chamada do método getInstance instancia o servidor eXtreme Scale e lê todas as propriedades configuradas. Configurar as propriedades após a criação não tem efeito. O exemplo a seguir mostra como configurar propriedades antes de criar uma instância do Servidor.

```
// Get the server properties associated with this process.
ServerProperties serverProperties = ServerFactory.getServerProperties();

// Set the server name for this process.
serverProperties.setServerName("EmbeddedServerA");

// Set the name of the zone this process is contained in.
serverProperties.setZoneName("EmbeddedZone1");

// Set the end point information required to bootstrap to the catalog service.
serverProperties.setCatalogServiceBootstrap("localhost:2809");

// Set the ORB listener host name to use to bind to.
serverProperties.setListenerHost("host.local.domain");

// Set the ORB listener port to use to bind to.
serverProperties.setListenerPort(9010);

// Turn off all MBeans for this process.
```

```
serverProperties.setMBeansEnabled(false);

Server server = ServerFactory.getInstance();
```

Integrando o Serviço de Catálogo

Qualquer configuração JVM que é sinalizada pelo método CatalogServerProperties.setCatalogServer pode hospedar o serviço de catálogo para o eXtreme Scale. Este método indica ao tempo de execução do servidor do eXtreme Scale para instanciar o serviço de catálogo quando o servidor for iniciado. O código a seguir mostra como instanciar o servidor de catálogos do eXtreme Scale:

```
CatalogServerProperties catalogServerProperties =
 ServerFactory.getCatalogProperties();
catalogServerProperties.setCatalogServer(true);

Server server = ServerFactory.getInstance();
```

Integrando o Contêiner do eXtreme Scale

Emita o método Server.createContainer para qualquer JVM para hospedar vários contêineres do eXtreme Scale. O código a seguir mostra como instanciar um contêiner do eXtreme Scale:

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new
File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL());
Container container = server.createContainer(policy);
```

Processo do Servidor Autocontido

É possível iniciar todos os serviços juntos, o que é útil para o desenvolvimento e também prático na produção. Ao iniciar os serviços juntos, um único processo faz todo o seguinte: inicia o serviço de catálogo, inicia um conjunto de contêineres e executa a lógica de conexão do cliente. A inicialização dos serviços desta forma classifica os problemas de programação antes da implementação em um ambiente distribuído. O código a seguir mostra como instanciar um servidor eXtreme Scale autocontido:

```
CatalogServerProperties catalogServerProperties =
 ServerFactory.getCatalogProperties();
catalogServerProperties.setCatalogServer(true);

Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new
File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL());
Container container = server.createContainer(policy);
```

Incorporando o eXtreme Scale no WebSphere Application Server

A configuração para o eXtreme Scale é definida automaticamente quando você instalar o WebSphere Extended Deployment DataGrid em um ambiente WebSphere Application Server. Você não precisa configurar quaisquer propriedades antes de acessar o servidor para criar um contêiner. O código a seguir mostra como instanciar um servidor do eXtreme Scale no WebSphere Application Server:

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new
```

```
File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL);
Container container = server.createContainer(policy);
```

Para obter um exemplo passo a passo de como iniciar um serviço de catálogo integrado e contêiner programaticamente, consulte “Utilizando a API do Servidor Integrado”.

Utilizando a API do Servidor Integrado

Com o WebSphere eXtreme Scale, é possível usar uma API programática para gerenciar o ciclo de vida de servidores e contêineres integrados. É possível configurar programaticamente o servidor com qualquer uma das opções que também podem ser configuradas com as opções de linha de comandos ou com as propriedades de servidor baseadas em arquivo. É possível configurar o servidor integrado para que seja um servidor de contêiner, um serviço de catálogo ou ambos.

Antes de Iniciar

É necessário ter um método para executar o código a partir de um Java Virtual Machine já existente. As classes doeXtreme Scale deverão estar disponíveis na árvore do carregador de classes.

Sobre Esta Tarefa

É possível executar muitas tarefas de administração com a API de Administração. Um uso comum da API é como um servidor interno para armazenar o estado do aplicativo da Web. O servidor da Web pode iniciar um servidorWebSphere eXtreme Scale integrado, relatar o servidor de contêiner para o serviço de catálogo e o servidor será, em seguida, incluído como um membro de uma grade distribuída maior. Esse uso pode fornecer escalabilidade e alta disponibilidade para um armazém de dados voláteis do contrário.

É possível controlar programaticamente o ciclo de vida completo de um servidor eXtreme Scale integrado. Os exemplos são genéricos o máximo possível e mostram apenas exemplos de código diretos para etapas realçadas.

Procedimento

1. Obtenha o objeto ServerProperties da classe ServerFactory e configure quaisquer opções necessárias.

   Cada servidor eXtreme Scale possui um conjunto de propriedades configuráveis.Quando um servidor é iniciado na linha de comandos, essas propriedades são configuradas para os padrões, mas é possível substituir várias propriedades ao fornecer uma origem ou arquivo externo. No escopo integrado, é possível configurar diretamente as propriedades com um objeto ServerProperties. Essas propriedades devem ser configuradas antes de obter uma instância do servidor a partir da classe ServerFactory. O seguinte exemplo de fragmento de código obtém um objeto ServerProperties, configura o campo CatalogServiceBootStrap e inicializa várias configurações do servidor opcionais. Consulte a documentação da API para obter uma lista de definições configuráveis.

   ```
   ServerProperties props = ServerFactory.getServerProperties();
   props.setCatalogServiceBootstrap("host:port"); // required to connect to specific catalog service
   props.setServerName("ServerOne"); // name server
   props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // Sets trace spec
   ```

2. Se desejar que o servidor seja um serviço de catálogo, obtenha o objeto CatalogServerProperties.

   Cada servidor integrado pode ser um serviço de catálogo, um servidor de contêiner ou ambos. O seguinte exemplo obtém o objeto CatalogServerProperties, ativa a opção de serviço de catálogo e define várias configurações de serviço de catálogo.

   ```
   CatalogServerProperties catalogProps = ServerFactory.getCatalogProperties();
   catalogProps.setCatalogServer(true); // false by default, it is required to set as a catalog service
   catalogProps.setQuorum(true); // enables / disables quorum
   ```

3. Obtenha a instância Servidor a partir da classe ServerFactory. A instância doServidor é um singleton de escopo definido de processo responsável por gerenciar a associação na grade. Depois que essa instância for inicializada, esse processo será conectado e altamente disponibilizado com outros servidores na grade. A seguir há um exemplo de como criar a instância Servidor:

   ```
   Server server = ServerFactory.getInstance();
   ```

   Revisando o exemplo anterior, a classeServerFactory fornece um método estático que retorna uma instância Servidor. A classe ServerFactory deve ser a única interface para obter uma instância Servidor. Portanto, a classe garante que a instância seja um singleton ou uma instância para cada JVM ou carregador de classes isolado. O método getInstance inicializa a instância Servidor. É necessário configurar todas as propriedades do servidor antes de inicializar a instância. A classe Servidor é responsável por criar novas instâncias do Contêiner. É possível usar ambas as classes ServerFactory e Servidor para gerenciar o ciclo de vida da instância do Servidor integrada.

4. Inicie uma instância Contêiner usando a instância do Servidor.

   Antes que os shards possam ser colocados em um servidor integrado, é necessário criar um contêiner no servidor. A interface Servidor possui um método createContainer que utiliza um argumentoDeploymentPolicy. O seguinte exemplo utiliza a instância do servidor obtida para criar um contêiner usando um arquivo DeploymentPolicy criado. Note que os contêineres requerem um carregador de classes que possui os binários do aplicativo disponíveis para serialização. É possível tornar esses binários disponíveis ao chamar o métodocreateContainer com o carregador de classes de contexto Encadeamento para configurar o carregador de classes que deseja usar.

   ```
   DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy
   (new URL("file://urltodeployment.xml"),
    new URL("file://urltoobjectgrid.xml"));
   Container container = server.createContainer(policy);
   ```

5. Remover e limpar um contêiner.

   É possível remover e limpar um servidor de contêiner usando o método teardown em execução na instância do Contêiner obtida. Executar o método teardown adequadamente em um contêiner limpa o contêiner e remove o contêiner do servidor integrado.

   O processo de limpeza do contêiner inclui o movimento e a remoção de todos os shards que são colocados dentro desse contêiner. Cada servidor pode conter muitos contêineres e shards. Limpar um contêiner não afeta o ciclo de vida da instância do Servidor pai. O seguinte exemplo demonstra como executar o método teardown em um servidor. O método teardown é disponibilizado por meio da interface ContainerMBean. Ao usar a interface ContainerMBean, se você não tiver mais acesso programático nesse contêiner, ainda poderá remover e limpar o contêiner com o MBean. Um métodoterminate também existe na interface Contêiner, mas não use-o a menos que realmente seja necessário. Esse método é mais forte e não coordena o movimento e a limpeza apropriados dos shards.

```
container.teardown();
```

6. Parar o servidor integrado.

   Ao parar um servidor integrado, todos os contêineres e shards que estiverem em execução também são parados no servidor. Ao parar um servidor integrado, é necessário limpar todas as conexões abertas e mover ou remover todos os shards. O seguinte exemplo mostra como parar um servidor e o uso do métodowaitFor na interface do Servidor para garantir que a instância do Servidor seja parada completamente. Da mesma forma que o exemplo de contêiner, o método stopServer é disponibilizado por meio da interface ServerMBean. Com essa interface, é possível parar um servidor com o bean gerenciado (MBean) correspondente.

```
ServerFactory.stopServer();  // Uses the factory to kill the Server singleton
 // or
server.stopServer(); // Uses the Server instance directly
server.waitFor(); // Returns when the server has properly completed its shutdown procedures
```

   Código de exemplo completo:

```
 import java.net.MalformedURLException;
import java.net.URL;

import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.deployment.DeploymentPolicy;
import com.ibm.websphere.objectgrid.deployment.DeploymentPolicyFactory;
import com.ibm.websphere.objectgrid.server.Container;
import com.ibm.websphere.objectgrid.server.Server;
import com.ibm.websphere.objectgrid.server.ServerFactory;
import com.ibm.websphere.objectgrid.server.ServerProperties;

public class ServerFactoryTest {

    public static void main(String[] args) {

        try {

            ServerProperties props = ServerFactory.getServerProperties();
            props.setCatalogServiceBootstrap("catalogservice-hostname:catalogservice-port");
            props.setServerName("ServerOne"); // name server
            props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // TraceSpec

            /*
             * Na maioria dos casos, o servidor atuará apenas como um servidor de contêiner
             * e será conectado a um serviço de catálogo externo. Essa é a forma mais
             * disponibilizada de fazer as coisas. O trecho de código comentado a seguir
             * ativará esse Servidor como um serviço de catálogo.
             *
             *
             * CatalogServerProperties catalogProps =
             * ServerFactory.getCatalogProperties();
             * catalogProps.setCatalogServer(true); // enable catalog service
             * catalogProps.setQuorum(true); // enable quorum
             */

            Server server = ServerFactory.getInstance();

            DeploymentPolicy policy =
DeploymentPolicyFactory.createDeploymentPolicy
     (new URL("url to deployment xml"),  new URL("url to objectgrid
xml file"));
            Container container = server.createContainer(policy);

            /*
             * O shard será agora colocado nesse contêiner se os requisitos de
implementação forem atendidos.
             * Isso inclui a criação de servidor e de contêiner integrado.
             *
             * As linhas abaixo demonstrarão simplesmente a chamada
de métodos de limpeza
             */

            container.teardown();
            server.stopServer();
            int success = server.waitFor();

        } catch (ObjectGridException e) {
            // Container failed to initialize
        } catch (MalformedURLException e2) {
```

```
            // invalid url to xml file(s)
        }

    }

}
```

Monitorando com a API de Estatísticas

A API de Estatísticas é a interface direta para a árvore de estatísticas interna. As estatísticas são desativadas por padrão, mas podem ser ativadas configurando uma interface StatsSpec. Uma interface StatsSpec define como o WebSphere eXtreme Scale deve monitorar estatísticas.

Sobre Esta Tarefa

É possível usar a API StatsAccessor local para consultar dados e acessar estatísticas em qualquer instância do ObjectGrid que esteja no mesmo Java Virtual Machine (JVM) que o código de execução. Para obter mais informações sobre as interfaces específicas, consulte a documentação da API. Conclua as seguintes etapas para ativar o monitoramento da árvore de estatísticas internas.

Procedimento

1. Recupere o objeto StatsAccessor. A interface StatsAccessor segue o padrão do singleton. Assim, além dos problemas relacionados ao carregador de classes, uma instância do StatsAccessor deverá existir para cada JVM. Esta classe atua como a interface principal para todas as operações locais de estatísticas. O seguinte código é um exemplo de como recuperar a classe do acessador. Chame essa operação antes de qualquer outra chamada ObjectGrid.

   ```
   public class LocalClient {

      public static void main(String[] args) {

         // retrieve a handle to the StatsAccessor
         StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      }

   }
   ```

2. Configure a interface StatsSpec da grade. Configure esse JVM para coletar todas as estatísticas apenas no nível do ObjectGrid. É necessário garantir que um aplicativo ative todas as estatísticas que podem ser necessárias antes de iniciar quaisquer transações. O exemplo a seguir configura a interface StatsSpec utilizando um campo constante estático e utilizando uma spec String. Usar um campo de constante estático é mais simples porque o campo já definiu a especificação. Entretanto, ao utilizar uma spec String, é possível ativar qualquer combinação de estatísticas que são necessárias.

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

         // retrieve a handle to the StatsAccessor
         StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

         // Set the spec via the static field
         StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
         accessor.setStatsSpec(spec);

         // Set the spec via the spec String
         StatsSpec spec = new StatsSpec("og.all=enabled");
         accessor.setStatsSpec(spec);

   }
   ```

3. Envie as transações para a grade para forçar a coleta dos dados para monitoramento. Para coletar dados úteis para as estatísticas, é necessário enviar as transações para a grade. O seguinte extrato de código insere um registro no MapA, que é um ObjectGridA. Como as estatísticas estão no nível do ObjectGrid, qualquer mapa dentro do ObjectGrid produz os mesmos resultados.

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

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
    ObjectGridmanagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Drive insert
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();
}
```

4. Consulte um StatsFact utilizando a API do StatsAccessor. Cada caminho de estatísticas é associado com uma interface StatsFact. A interface StatsFact é um marcador genérico que é utilizado para organizar e conter um objeto StatsModule. Antes de poder acessar o módulo de estatísticas real, o objeto StatsFact deve ser recuperado.

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

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
    ObjectGridManagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Drive insert
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();

      // Retrieve StatsFact

      StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
    StatsModule.MODULE_TYPE_OBJECT_GRID);

}
```

5. Interaja com o objeto StatsModule. O objeto StatsModule está contido na interface StatsFact. É possível obter uma referência para o módulo utilizando a interface StatsFact. Como a interface StatsFact é uma interface genérica, é necessário converter o módulo retornado para o tipo StatsModule esperado. Como esta tarefa coleta estatísticas do eXtreme Scale, o objeto StatsModule

retornado é convertido para um tipo OGStatsModule. Após o módulo ser convertido, é possível acessar todas as estatísticas disponíveis.

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

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
    ObjectGridmanagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Drive insert
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();

      // Retrieve StatsFact
      StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
    StatsModule.MODULE_TYPE_OBJECT_GRID);

      // Retrieve module and time
      OGStatsModule module = (OGStatsModule)fact.getStatsModule();
      ActiveTimeStatistic timeStat =
    module.getTransactionTime("Default", true);
      double time = timeStat.getMeanTime();

}
```

Monitorando com a PMI do WebSphere Application Server

O WebSphere eXtreme Scale suporta Performance Monitoring Infrastructure (PMI) ao executar em um WebSphere Application Server ou servidor de aplicativos WebSphere Extended Deployment. A PMI coleta dados de desempenho em aplicativos de tempo de execução e fornece interfaces que suportam aplicativos externos para monitorar dados de desempenho. É possível utilizar o console administrativo ou a ferramenta wsadmin para acessar dados de monitoramento.

Antes de Iniciar

É possível utiliza a PMI para monitorar seu ambiente quando você está utilizando oWebSphere eXtreme Scale combinado com o WebSphere Application Server.

Sobre Esta Tarefa

O WebSphere eXtreme Scale utiliza o recurso PMI customizado do WebSphere Application Server para incluir sua própria instrumentação PMI. Com esta abordagem, é possível ativar e desativar a PMI do WebSphere eXtreme Scale com o console administrativo ou com as interfaces Java Management Extensions (JMX) na ferramenta wsadmin. Além disso, é possível acessar as estatísticas do WebSphere eXtreme Scale com a PMI padrão e as interfaces JMX que são utilizadas pela ferramentas de monitoramento, incluindo o Tivoli Performance Viewer.

Procedimento

1. Ative a PMI do eXtreme Scale. É necessário ativar a PMI para visualizar as estatísticas de PMI. Consulte "Ativando a PMI" para obter mais informações.
2. Recupere as estatísticas de PMI do eXtreme Scale. Visualize o desempenho dos seus aplicativos eXtreme Scale com o Tivoli Performance Viewer. Consulte "Recuperando Estatísticas PMI" na página 295 para obter mais informações.

O que Fazer Depois

Para obter mais informações sobre a ferramenta wsadmin, consulte "Acessando MBeans Utilizando a Ferramenta wsadmin" na página 303.

Ativando a PMI

É possível utilizar a WebSphere Application Server Performance Monitoring Infrastructure (PMI) para ativar ou desativar estatísticas em qualquer nível. Por exemplo, é possível optar por ativar as estatísticas de taxa de acesso do mapa para um mapa específico, mas não o número de estatísticas de entrada ou as estatísticas de tempo de atualização de lote do utilitário de carga. É possível ativar a PMI no console administrativo ou com scripts.

Antes de Iniciar

Seu servidor de aplicativos deve ser iniciado e ter um aplicativo ativado para o eXtreme Scale instalado. Para ativar a PMI com scripts, você também deve estar apto a efetuar login e utilizar a ferramenta wsadmin. Para obter informações adicionais sobre a ferramenta wsadmin, consulte o tópico ferramenta wsadmin no centro de informações do WebSphere Application Server.

Sobre Esta Tarefa

Utilize a PMI do WebSphere Application Server para fornecer um mecanismo granular com o qual é possível ativar ou desativar estatísticas em qualquer nível. Por exemplo, é possível optar por ativar as estatísticas de taxa de acesso do mapa para um mapa específico, mas não o número de estatísticas de entrada ou as estatísticas de tempo de atualização de lote do utilitário de carga. Esta seção mostra como utilizar o console administrativo e os scripts wsadmin para ativar a PMI do ObjectGrid.

Procedimento

- **Ativar a PMI no console administrativo.**
  1. No console administrativo, clique em **Monitoramento e Ajuste** → **Performance Monitoring Infrastructure** → *server_name*.
  2. Verifique se opção Ativar PMI (Performance Monitoring Infrastructure) está selecionada. Essa definição é ativada por padrão. Se a configuração não estiver ativada, selecione a caixa de opções e reinicie o servidor.
  3. Clique em **Customizar**. Na árvore de configuração, selecione o ObjectGrid e o módulo Mapas do ObjectGrid. Ative as estatísticas para cada módulo.

  A categoria de tipo de transação para estatísticas do ObjectGrid é criada no tempo de execução. É possível visualizar apenas as subcategorias das estatísticas do ObjectGrid e do Mapa na guia **Tempo de Execução**.
- **Ativar a PMI com scripts.**

1. Abra um prompt de linha de comandos. Navegue para o diretório install_root/bin. Digite wsadmin para iniciar a ferramenta de linha de comandos wsadmin.
2. Modifique a configuração do tempo de execução do PMI do eXtreme Scale. Verifique se a PMI está ativada para o servidor, por meio dos seguintes comandos:

   ```
   wsadmin>set s1 [$AdminConfig getid
   /Cell:CELL_NAME/Node:NODE_NAME/
       Server:APPLICATION_SERVER_NAME/]
   wsadmin>set pmi [$AdminConfig list PMIService $s1]
   wsadmin>$AdminConfig show $pmi.
   ```

   Se a PMI não estiver ativada, execute os seguintes comandos para ativá-la:

   ```
   wsadmin>$AdminConfig modify $pmi {{enable true}}
   wsadmin> $AdminConfig save
   ```

   Se precisar ativar a PMI, reinicie o servidor.
3. Configure as variáveis para customizar o conjunto de estatísticas utilizando os seguintes comandos:

   ```
   wsadmin>set perfName [$AdminControl completeObjectName type=Perf,
   process=APPLICATION_SERVER_NAME,*]
   wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
   wsadmin>set params [java::new {java.lang.Object[]} 1]
   wsadmin>$params set 0 [java::new java.lang.String custom]
   wsadmin>set sigs [java::new {java.lang.String[]} 1]
   wsadmin>$sigs set 0 java.lang.String
   ```
4. Configure o conjunto de estatísticas para customizar o seguinte comando:

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
   ```
5. Utilize os comandos a seguir para configurar as variáveis de modo a ativar a estatística da PMI objectGridModule:

   ```
   wsadmin>set params [java::new {java.lang.Object[]} 2]
   wsadmin>$params set 0 [java::new java.lang.String objectGridModule=1]
   wsadmin>$params set 1 [java::new java.lang.Boolean false]
   wsadmin>set sigs [java::new {java.lang.String[]} 2]
   wsadmin>$sigs set 0 java.lang.String
   wsadmin>$sigs set 1 java.lang.Boolean
   ```
6. Configure a cadeia de estatísticas por meio deste comando:

   ```
   wsadmin>set params2 [java::new {java.lang.Object[]} 2]
   wsadmin>$params2 set 0 [java::new java.lang.String mapModule=*]
   wsadmin>$params2 set 1 [java::new java.lang.Boolean false]
   wsadmin>set sigs2 [java::new {java.lang.String[]} 2]
   wsadmin>$sigs2 set 0 java.lang.String
   wsadmin>$sigs2 set 1 java.lang.Boolean
   ```
7. Configure a cadeia de estatísticas por meio deste comando:

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params2 $sigs2
   ```

Essas etapas ativam o PMI do tempo de execução do eXtreme Scale, mas não modificam a configuração do PMI. Se você reiniciar o servidor de aplicativos, as configurações da PMI serão perdidas, exceto para a ativação da PMI principal.

Exemplo

É possível executar as seguintes etapas para ativar as estatísticas da PMI para o aplicativo de amostra:

1. Ative o aplicativo utilizando o endereço da Web `http://host:port/ObjectGridSample`, em que host e porta são o nome do host e número de porta HTTP do servidor no qual a amostra está instalada.

2. No aplicativo de amostra, clique consecutivamente em ObjectGridCreationServlet e nos botões de ação 1, 2, 3, 4 e 5 para gerar ações para o ObjectGrid e os mapas. Não feche esta página do servlet neste momento.
3. No console administrativo, clique em **Monitoramento e Ajuste** → **Performance Monitoring Infrastructure** → *server_name* Clique na guia **Tempo de Execução**.
4. Clique no botão de rádio **Customizado**.
5. Expanda o módulo Mapas do ObjectGrid na árvore de tempo de execução e clique no link clusterObjectGrid. No grupo Mapas do ObjectGrid, há uma instância do ObjectGrid denominada clusterObjectGrid; há quatro mapas debaixo do grupo clusterObjectGrid: contadores, funcionários, escritórios e sites. Na instância do ObjectGrids, existe uma instância do clusterObjectGrid e sob esta há um tipo de transação denominado DEFAULT.
6. É possível ativar as estatísticas de seu interesse. Por exemplo, é possível ativar o número de entradas para o mapa funcionários e o tipo de resposta de transação para o tipo de transação DEFAULT.

O que Fazer Depois

Quando ativar a PMI, será possível visualizar estatísticas de PMI com o console administrativo ou por meio de scripts.

Recuperando Estatísticas PMI

Ao recuperar estatísticas PMI, é possível visualizar o desempenho dos aplicativos eXtreme Scale.

Antes de Iniciar

- Ative o controle de estatísticas PMI para o seu ambiente. Consulte "Ativando a PMI" na página 293 para obter mais informações.
- Os caminhos nesta tarefa assumem que você está recuperando estatísticas para o aplicativo de amostra, mas é possível utilizar estas estatísticas para qualquer outro aplicativo com etapas semelhantes.
- Se estiver utilizando o console administrativo para recuperar estatísticas, você deve estar apto a efetuar login no console administrativo. Se estiver utilizando scripts, você deve estar apto a efetuar login no wsadmin.

Sobre Esta Tarefa

É possível recuperar estatísticas PMI para visualização no Tivoli Performance Viewer concluindo etapas no console administrativo ou com scripts.

- Etapas do console administrativo
- Etapas de scripts

Para obter mais informações sobre as estatísticas que podem ser recuperadas, consulte "Módulos PMI" na página 296.

Procedimento

- Recupere estatísticas PMI no console administrativo.
  1. No console administrativo, clique em **Monitoramento e Ajuste** → **Performance Viewer** → **Atividade Atual**
  2. Selecione o servidor que deseja monitorar utilizando o Tivoli Performance Viewer, em seguida, ative o monitoramento.
  3. Clique no servidor para visualizar a página do Performance Viewer.

4. Expanda a árvore de configuração. Clique em **Mapas do ObjectGrid** → **clusterObjectGrid** e selecione **employees**. Expanda **ObjectGrids** → **clusterObjectGrid** e selecione **DEFAULT**.
5. No aplicativo de amostra do ObjectGrid, retorne ao servlet ObjectGridCreationServlet, clique no botão 1, em seguida, ocupar mapas. É possível visualizar a estatística no visualizador.

- Recupere estatísticas PMI com scripts.
  1. Em um prompt de linha de comandos, navegue até o diretório `install_root/bin`. Digite `wsadmin` para iniciar a ferramenta wsadmin.
  2. Configure as variáveis para o ambiente utilizando os seguintes comandos:

```
wsadmin>set perfName [$AdminControl completeObjectName type=Perf,*]
wsadmin>set perfOName [$AdminControl makeObjectName $perfName]
wsadmin>set mySrvName [$AdminControl completeObjectName type=Server,
 name=APPLICATION_SERVER_NAME,*]
```

  3. Configure as variáveis para obter o ambiente mapModule utilizando os seguintes comandos:

```
wsadmin>set params [java::new {java.lang.Object[]} 3]
wsadmin>$params set 0 [$AdminControl makeObjectName $mySrvName]
wsadmin>$params set 1 [java::new java.lang.String mapModule]
wsadmin>$params set 2 [java::new java.lang.Boolean true]
wsadmin>set sigs [java::new {java.lang.String[]} 3]
wsadmin>$sigs set 0 javax.management.ObjectName
wsadmin>$sigs set 1 java.lang.String
wsadmin>$sigs set 2 java.lang.Boolean
```

  4. Obtenha a estatística mapModule utilizando os seguintes comandos:

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params $sigs
```

  5. Configure as variáveis para obter a estatística objectGridModule utilizando os seguintes comandos:

```
wsadmin>set params2 [java::new {java.lang.Object[]} 3]
wsadmin>$params2 set 0 [$AdminControl makeObjectName $mySrvName]
wsadmin>$params2 set 1 [java::new java.lang.String objectGridModule]
wsadmin>$params2 set 2 [java::new java.lang.Boolean true]
wsadmin>set sigs2 [java::new {java.lang.String[]} 3]
wsadmin>$sigs2 set 0 javax.management.ObjectName
wsadmin>$sigs2 set 1 java.lang.String
wsadmin>$sigs2 set 2 java.lang.Boolean
```

  6. Obtenha a estatística objectGridModule utilizando os seguintes comandos:

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params2 $sigs2
```

Resultados

É possível visualizar estatísticas no Tivoli Performance Viewer.

Módulos PMI

É possível monitorar o desempenho dos seus aplicativos com os módulos Performance Monitoring Infrastructure (PMI).

objectGridModule

O objectGridModule contém uma estatística de tempo: tempo de resposta de transação. Uma transação é definida como uma duração entre a chamada de método Session.begin e a chamada de método Session.commit. Esta duração é rastreada como o tempo de resposta da transação. O elemento-raiz do objectGridModule, "root", serve como ponto de entrada para as estatísticas do WebSphere eXtreme Scale. Este elemento-raiz possui ObjectGrids como seus

elementos-filhos, que possuem tipos de transações como seus elementos-filhos. A estatística de tempo de resposta está associada a cada tipo de transação.

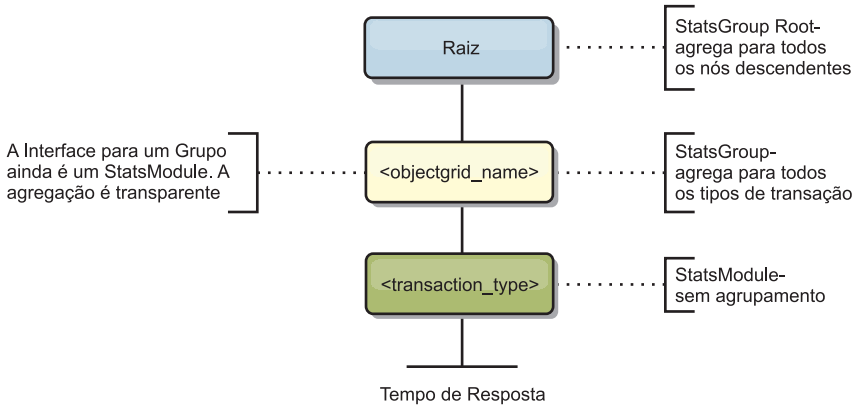


Figura 4. Estrutura do Módulo ObjectGridModule

O seguinte diagrama mostra um exemplo da estruturaObjectGridModule. Neste exemplo, duas instâncias ObjectGrid existem no sistema: ObjectGrid A e ObjectGrid B. A instância ObjectGrid A possui dois tipos de transações: A e padrão. A instância ObjectGrid B possui apenas o tipo de transação padrão.

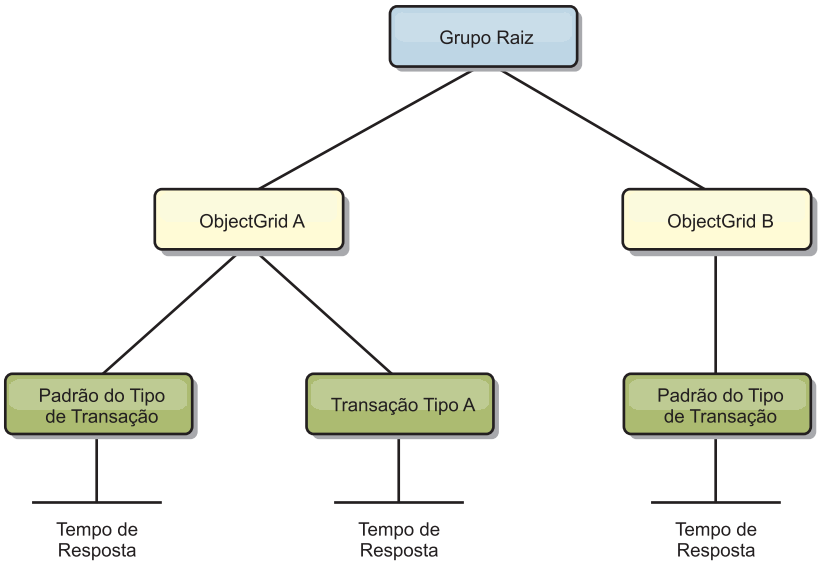


Figura 5. Exemplo da Estrutura do Módulo ObjectGridModule

Os tipos de transações são definidos por desenvolvedores de aplicativos, porque eles sabem quais tipos de transações seus aplicativos utilizam. O tipo de transação é configurando utilizando o método Session.setTransactionType(String) a seguir:

```
/**
* Configura o tipo de transação para futuras transações.
*
* Quando este método é chamado, todas as futuras transações terão o mesmo tipo
* até que outro tipo de transação seja configurado. Se nenhum tipo de transação
* estiver configurado, será utilizado o tipo de transação TRANSACTION_TYPE_DEFAULT
* padrão.
*
* Os tipos de transações são utilizados principalmente para finalidade de
rastreamento de dados estatísticos.
```

```
* Os usuários podem predefinir tipos de transações que são executadas em um
* application. A idéia é categorizar transações com as mesmas características
* para uma categoria (tipo), portanto, uma estatística de tempo de resposta de
 transação pode ser
* utilizada para rastrear cada tipo de transação.
*
* Este rastreamento é útil quando seu aplicativo tem diferentes tipos de
* transações.
* Entre eles, alguns tipos de transações, como transações de atualização,
têm um processamento
* mais longo que outras transações, como transações de leitura. Utilizando o tipo de
* transação, diferentes transações são rastreadas por diferentes estatísticas, portanto,
* as estatísticas podem ser mais úteis.
*
* @param tranType o tipo de transação para futuras transações.
*/
void setTransactionType(String tranType);
```

O exemplo a seguir configura o tipo de transação como `updatePrice`:

```
// Set the transaction type to updatePrice
// The time between session.begin() and session.commit() will be
// tracked in the time statistic for "updatePrice".
session.setTransactionType("updatePrice");
session.begin();
map.update(stockId, new Integer(100));
session.commit();
```

A primeira linha indica que o tipo de transação subseqüente é updatePrice. Uma estatística updatePrice existe na instância ObjectGrid que corresponde à sessão no exemplo. Utilizando interfaces Java Management Extensions (JMX), é possível obter o tempo de resposta da transação para transações updatePrice. Também é possível obter a estatística agregada para todos os tipos de transações na instância ObjectGrid especificada.

mapModule

O mapModule contém três estatísticas relacionadas aos mapas eXtreme Scale:

- **Taxa de ocorrências do mapa** - *BoundedRangeStatistic*: Controla a taxa de ocorrências de um mapa. A taxa de ocorrência é um valor flutuante entre 0 e 100 inclusivamente, que representa a porcentagem de ocorrências do mapa em relação a operações get do mapa.
- **Número de entradas**-*CountStatistic*: Controla o número de entradas no mapa.
- **Tempo de resposta de atualização de lote do utilitário de carga**-*TimeStatistic*: Controla o tempo de resposta que é utilizada para a operação de atualização de lote do utilitário de carga.

O elemento-raiz do mapModule, "root", serve como ponto de entrada para as estatísticas do Mapa ObjectGrid. Este elemento-raiz possui ObjectGrids como seus elementos-filhos, que possuem mapas como seus elementos-filhos. Cada instância do mapa possui três estatísticas listadas. A estrutura mapModule é mostrada no diagrama a seguir:

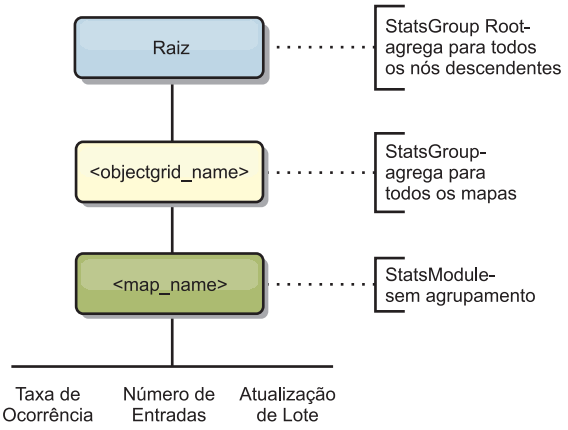
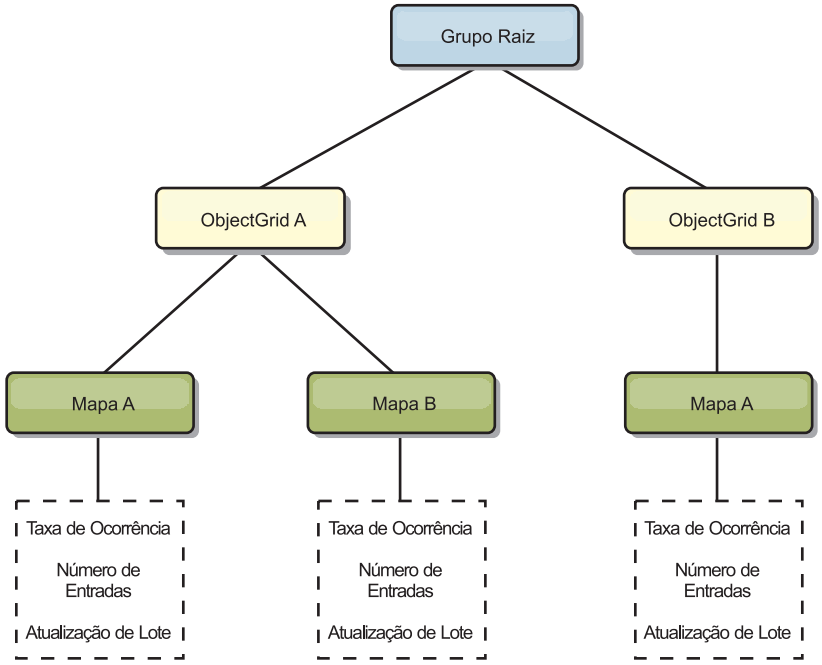


Figura 6. Estrutura do mapModule

O diagrama a seguir mostra um exemplo da estrutura mapModule:

Figura 7. Exemplo de Estrutura do Módulo mapModule



hashIndexModule

O hashIndexModule contém as seguintes estatísticas que são relacionadas aos índices de nível de Mapa:

- **Contagem de Localizações**-*CountStatistic*: O número de chamadas para a operação find do índice.
- **Contagem de Colisões**-*CountStatistic*: O número de colisões para a operação find.
- **Contagem de Falhas**-*CountStatistic*: O número de falhas para a operação find.
- **Contagem de Resultados**-*CountStatistic*: O número de chaves retornadas da operação find.

- **Contagem de BatchUpdate**-*CountStatistic*: O número de atualizações de lote junto a este índice. Quando o mapa correspondente é alterado de qualquer maneira, o método doBatchUpdate() do índice será chamado. Esta estatística informará com que frequência seu índice está sendo alterado ou atualizado.
- **Tempo de Duração da Operação Find**-*TimeStatistic*: A quantidade de tempo que a operação find leva para ser concluída

O elemento-raiz do hashIndexModule, "root", serve como ponto de entrada para as estatísticas do HashIndex. Este elemento-raiz possui ObjectGrids como seus elementos-filhos, os ObjectGrids possuem mapas como seus elementos-filhos, que, finalmente, possuem HashIndexes como seus elementos-filhos e nós-folhas da árvore. Cada instância do HashIndex possui três estatísticas listadas. A estrutura hashIndexModule é mostrada no diagrama a seguir:

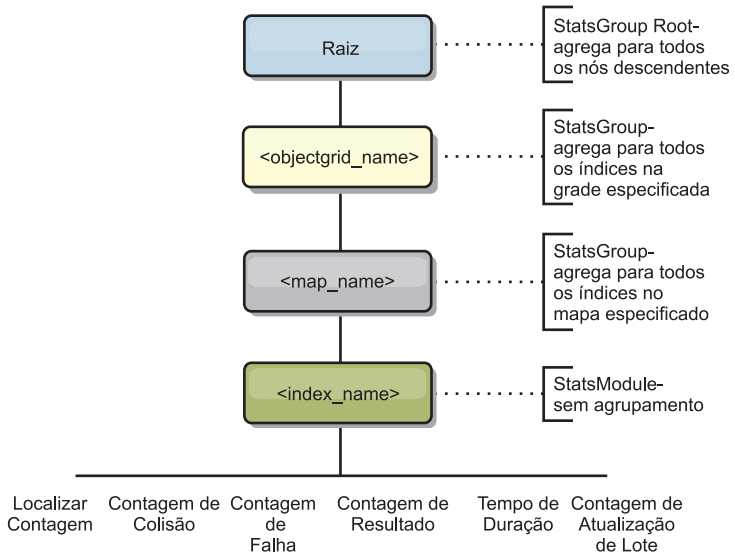


Figura 8. Estrutura do Módulo hashIndexModule

O diagrama a seguir mostra um exemplo da estrutura hashIndexModule:

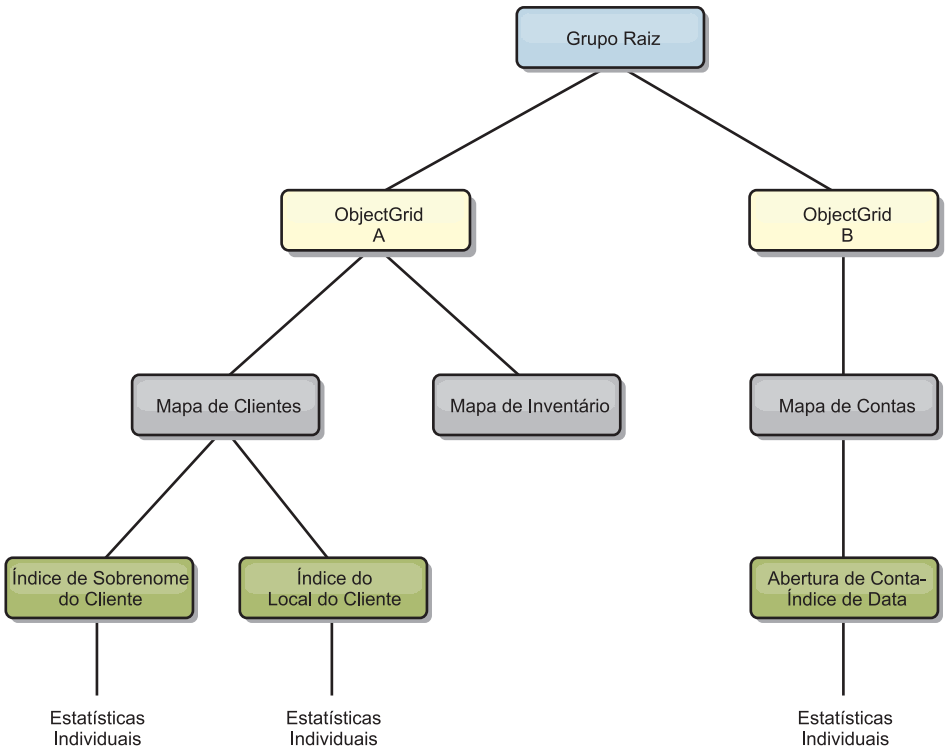


Figura 9. Exemplo da Estrutura do Módulo hashIndexModule

agentManagerModule

O agentManagerModule contém as estatísticas que são relacionadas aos agentes de nível de mapa:

- **Tempo de Redução**: *TimeStatistic* - A quantidade de tempo para o agente concluir a operação reduce.
- **Tempo de Duração Total**: *TimeStatistic* - A quantidade total de tempo para o agente concluir todas as operações.
- **Tempo de Serialização do Agente**: *TimeStatistic* - A quantidade de tempo para serialização do agente.
- **Tempo de Aumento do Agente**: *TimeStatistic* - A quantidade de tempo para aumentar o agente no servidor.
- **Tempo de Serialização do Resultado**: *TimeStatistic* - A quantidade de tempo para serializar os resultados do agente.
- **Tempo de Aumento do Resultado**: *TimeStatistic* - A quantidade de tempo para aumentar os resultados do agente.
- **Contagem de Falhas**: *CountStatistic* - O número de vezes que o agente falhou.
- **Contagem de Chamada**: *CountStatistic* - O número de vezes que o AgentManager foi chamado.
- **Contagem de Partições**: *CountStatistic* - O número de partições para as quais o agente é enviado.

O elemento-raiz do agentManagerModule, "root", serve como ponto de entrada para as estatísticas do AgentManager. O elemento-raiz possui ObjectGrids como seus elementos-filhos, os ObjectGrids possuem seus elementos-filhos, que, finalmente, possuem instâncias do AgentManager como seus elementos-filhos e nós-folhas da árvore. Cada instância do AgentManager possui três estatísticas

listadas.

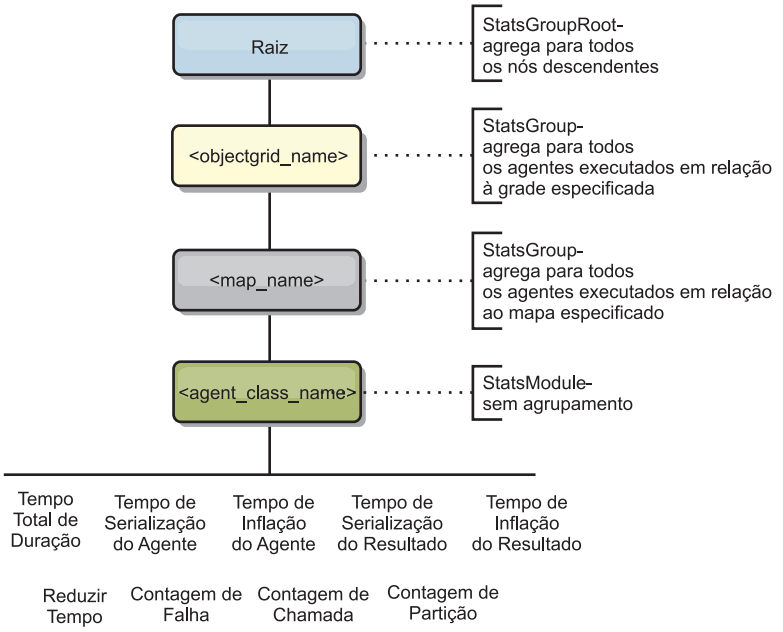


Figura 10. Estrutura agentManagerModule

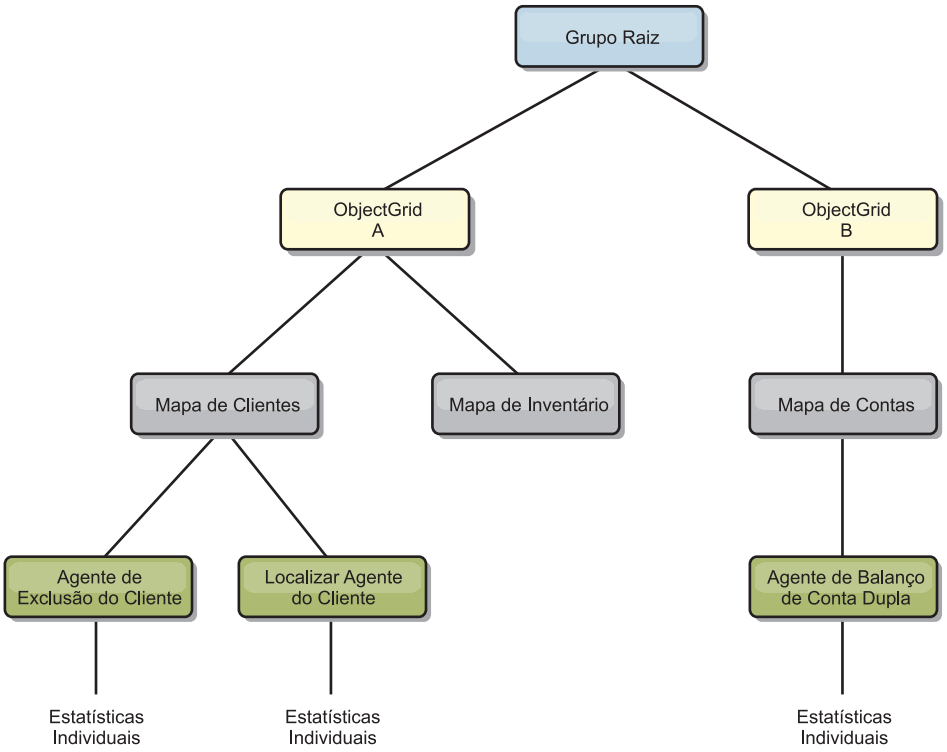


Figura 11. Exemplo da Estrutura agentManagerModule

queryModule

O queryModule contém estatísticas relacionadas às consultas do eXtreme Scale:

- **Tempo de Criação do Plano**: *TimeStatistic* - A quantidade de tempo para criar o plano de consulta.

- **Tempo de Execução**: *TimeStatistic* - A quantidade de tempo para executar a consulta.
- **Contagem de Execução**: *CountStatistic* - O número de vezes que a consulta foi executada.
- **Contagem de Resultados**: *CountStatistic* - A contagem para cada conjunto de resultados de cada execução de consulta.
- **FailureCount**: *CountStatistic* - O número de vezes que a consulta falhou.

O elemento-raiz do queryModule, "root", serve como ponto de entrada para as estatísticas do Query. Este elemento-raiz possui ObjectGrids como seus elementos-filhos, que possuem objetos Query como seus elementos-filhos e nós-folhas da árvore. Cada instância do Query possui as três estatísticas listadas.

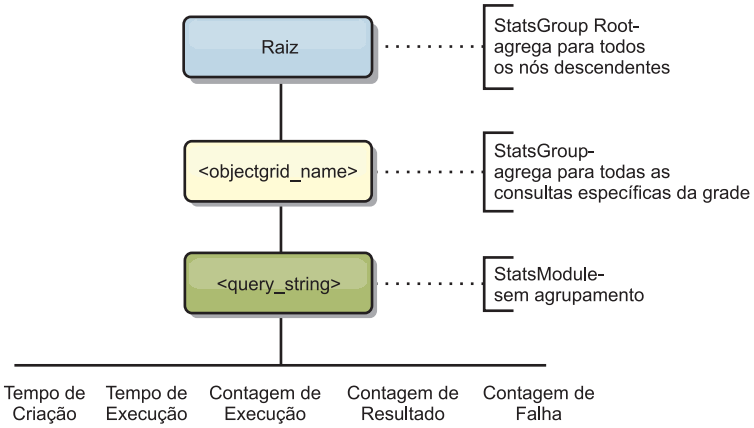


Figura 12. Estrutura queryModule

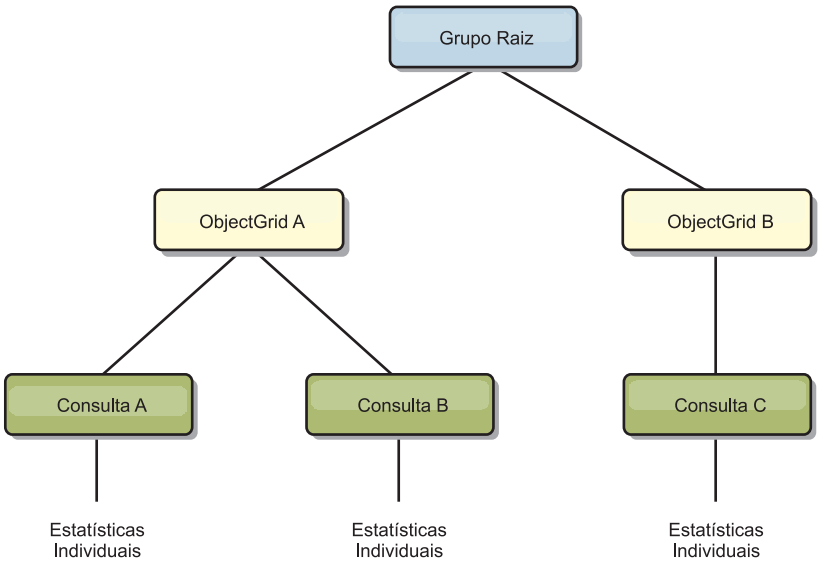


Figura 13. Exemplo da Estrutura queryModule QueryStats.jpg

Acessando MBeans Utilizando a Ferramenta wsadmin

É possível usar o utilitário wsadmin fornecido no WebSphere Application Server para acessar informações de MBean.

Execute a ferramenta wsadmin a partir do diretório bin em sua instalação do WebSphere Application Server. O exemplo a seguir recupera uma visualização da disposição do shard atual em um eXtreme Scale dinâmico. O wsadmin pode ser executado a partir de qualquer instalação na qual o eXtreme Scale está em execução. Não é necessário executar o wsadmin no serviço de catálogo.

```
$ wsadmin.sh -lang jython
wsadmin>placementService = AdminControl.queryNames
 ("com.ibm.websphere.objectgrid:*,type=PlacementService")
wsadmin>print AdminControl.invoke(placementService,
 "listObjectGridPlacement","library ms1")

<objectGrid name="library" mapSetName="ms1">
  <container name="container-0" zoneName="DefaultDomain"
  hostName="host1.company.org" serverName="server1">
     <shard type="Primary" partitionName="0"/>
     <shard type="SynchronousReplica" partitionName="1"/>
  </container>
  <container name="container-1" zoneName="DefaultDomain"
  hostName="host2.company.org" serverName="server2">
     <shard type="SynchronousReplica" partitionName="0"/>
     <shard type="Primary" partitionName="1"/>
  </container>
  <container name="UNASSIGNED" zoneName="_ibm_SYSTEM"
  hostName="UNASSIGNED" serverName="UNNAMED">
    <shard type="SynchronousReplica" partitionName="0"/>
    <shard type="AsynchronousReplica" partitionName="0"/>
  </container>
</objectGrid>
```

Capítulo 7. 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.

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).

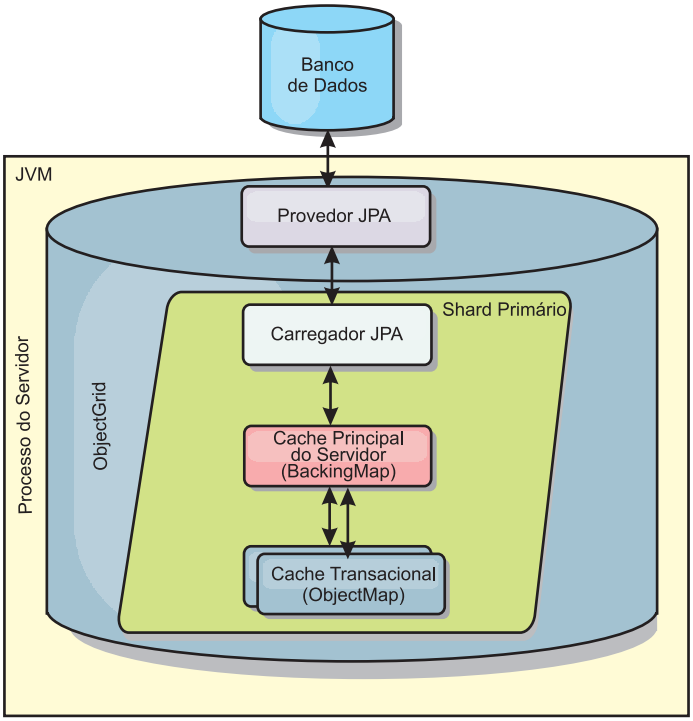


Figura 14. Arquitetura do Utilitário de Carga do JPA

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 informações, consulte “Plug-in JPAEntityLoader” na página 260.

Métodos

Utilitários de Carga Fornecem Três Métodos Principais:

1. get: Retorna uma lista de valores que correponde 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.

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 do eXtreme Scale quando as consultas ao banco de dados não podem 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.

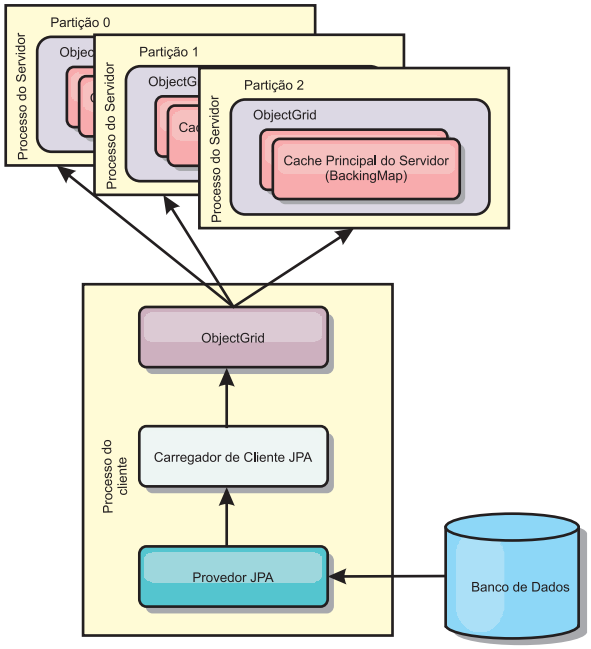


Figura 15. Utilitário de Carga do Cliente que usa Implementação JPA para Carregar o ObjectGrid

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 12. 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.
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 a configuração do utilitário de pré-carregamento JPA baseado em cliente, consulte oas informações no *Guia de Programação*

Programação 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. É possível implementar o pré-carregamento e o recarregamento de dados no seu aplicativo.

Usando a interface StateManager

Use o método setObjectGridState da interface StateManager para configurar o estado ObjectGrid para um dos seguintes valores: OFFLINE, ONLINE, QUIESCE ou PRELOAD. A interface StateManager impede que outros clientes acessem o ObjectGrid quando ainda não estiver on-line.

Por exemplo, configure o estado do ObjectGrid como PRELOAD antes de carregar os mapas com dados. Depois dos dados serem concluídos, configure o estado do ObjectGrid de volta para ONLINE. Consulte as informações sobre a configuração da disponibilidade de um ObjectGrid no *Guia de Administração* para obter mais informações.

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.

Exemplo de Pré-Carregamento Baseado no Cliente

O fluxo de pré-carregamento de dados é o seguinte:

1. Limpe o mapa a ser pré-carregado. No caso de um mapa de entidade, se qualquer relação for configurada como remoção em cascata, os mapas relacionados também serão limpos.
2. Execute a consulta ao JPA para as entidades em um lote. O tamanho do batch é 1000.

3. Para cada batch, crie uma lista de chaves e uma lista de valores para cada partição.
4. Para cada partição, chame o agente da grade de dados para inserir ou atualizar os dados no lado do servidor diretamente se ele for um cliente do eXtreme Scale. Se a grade for uma instância local, insira ou atualize diretamente os dados nos mapas do ObjectGrid.

O seguinte trecho de código de amostra mostra um carregamento de cliente simples.

```
// 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", "custPU", null,
    null, null, null, true, null);

// Set ObjectGrid state back to ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

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.

Os parâmetros usados no método ClientLoader.load são:

1. **objectGrid** : A instância do ObjectGrid. É uma instância do ObjectGrid do lado do cliente.
2. **"CUSTOMER"** : O mapa a ser carregado. Como o Customer possui uma relação cascade-all com entidades Order, as entidades Order também serão carregadas.
3. **"custPU"** : O nome da unidade de persistência JPA para as entidades Customer e Order.
4. **null** : O mapa persistenceProps é nulo, o que significa que as propriedades de persistência padrão configuradas no persistence.xml serão utilizadas.
5. **null** : A entityClass é configurada como nula. Ela será configurada como a classe de entidade configurada no XML descritor de metadados de entidade do ObjectGrid para o mapa "CUSTOMER", neste caso, Customer.class.
6. **null** : O loadSql é nulo, o que significa que o "select o from CUSTOMER o" padrão será utilizado para consultar as entidades JPA.
7. **null** : O mapa do parâmetro de consulta é nulo.
8. **true** : Isto indica que o modo de carregamento de dados é pré-carregado. Portanto, as operações clear serão chamadas para ambos os mapas CUSTOMER e ORDER para limpar todos os dados antes do carregamento devido à relação cascade-remove entre eles.
9. **null** : O ClientLoaderCallback é nulo.

Para obter mais informações sobre os parâmetros necessários, consultea API do ClientLoader na Documentação da API.

Exemplo de Recarregamento

Recarregar um mapa é o mesmo que pré-carregar um mapa, a não ser que o argumento isPreload seja configurado para false no método ClientLoader.load.

No modo de recarregamento, o fluxo dos dados é o seguinte:

1. Execute a consulta fornecida no mapa ObjectGrid e invalide todos os resultados. No caso de 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 fornecida para o JPA para consultar as entidades JPA em lote. O tamanho do batch é 1000.
3. Para cada batch, crie uma lista de chaves e uma lista de valores para cada partição.
4. Para cada partição, chame o agente da grade de dados para inserir ou atualizar os dados no lado do servidor diretamente se ele for um cliente do eXtreme Scale. Se a grade é uma configuração do eXtreme Scale local, insira ou atualize diretamente os dados nos mapas do ObjectGrid.

A seguir há uma amostra de recarregamento:

```
// 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);
```

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 Customer com id entre 1000 e 2000.

Note que essa cadeia de consulta observa a sintaxe de consulta JPA e a sintaxe de consulta da entidade eXtreme Scale. Essa cadeia de consulta é importante porque ela é executada duas vezes, uma vez para o ObjectGrid invalidar as entidades ObjectGrid correspondidas e depois para o JPA carregar as entidades do JPA correspondidas.

Chamando um Utilitário de Carga do Cliente em uma Implementação do Utilitário de Carga

Na interface Loader, há um método preload:

```
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 utilitários de carga do JPA na *Visão Geral do Produto*.

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. É importante chamar o utilitário de carga do cliente em uma e apenas uma partição.

```
ObjectGrid og = session.getObjectGrid();
int partitionId = backingMap.getPartitionId();
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();

// Only call client loader data in one partition
if (partitionId == preloadPartition) {

    ClientLoader loader = ClientLoaderFactory.getClientLoader();

    // Call the client loader to load the data
    try {

        loader.load(og, backingMap.getName(), txCallback.getPersistenceUnitName(),
     null, entityClass, null, null, true, null);
    } catch (ObjectGridException e) {
        LoaderException le = new LoaderException("Exception caught in ObjectMap "
 + ogName + "." + mapName);
        le.initCause(e);
        throw le;
    }
}
```

Nota: Configure 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.

Carregamento Manual do Cliente

O método ClientLoader.load fornece uma função de carregamento do cliente que atende à maioria dos cenários. Entretanto, se desejar carregar os dados sem o método ClientLoader.load, poderá implementar seu próprio pré-carregamento.

A seguir há um modelo de carregamento de cliente manual:

```
// Get the StateManager
StateManager stateMgr = StateManagerFactory.getStateManager();

// Set ObjectGrid state to PRELOAD before calling ClientLoader.loader
stateMgr.setObjectGridState(AvailabilityState.PRELOAD, objectGrid);


// Load the data
```

```
...

// Set ObjectGrid state back to ONLINE
stateMgr.setObjectGridState(AvailabilityState.ONLINE, objectGrid);
```

Se você estiver carregando os dados no lado do cliente, carregar os dados usando 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 designar seu aplicativo para certificar-se de que os agentes DataGrid em várias partições sejam executados em paralelo para aumentar ainda mais o desempenho.

Para implementar o pré-carregamento de dados com um agente DataGrid, consulte o seguinte exemplo.

Depois de criar a implementação de pré-carregamento de dados, poderá criar um Utilitário de Carga genérico para concluir as seguintes tarefas:

1. Consultar os dados do banco de dados nos lotes.
2. Criar uma lista de chaves e uma lista de valores para cada partição.
3. 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: Pré-carregamento de Dados com um Agente DataGrid

Se você estiver carregando os dados no lado do cliente, carregar os dados usando 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 designar seu aplicativo para certificar-se de que os agentes DataGrid em várias partições sejam executados em paralelo para aumentar ainda mais o desempenho.

A seguir há um exemplo de como carregar os dados com um agente DataGrid:

```
import java.io.Externalizable;
import java.io.IOException;
import java.io.ObjectInput;
import java.io.ObjectOutput;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

import com.ibm.websphere.objectgrid.NoActiveTransactionException;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TransactionException;
import com.ibm.websphere.objectgrid.datagrid.ReduceGridAgent;
import com.ibm.websphere.objectgrid.em.EntityManager;

public class InsertAgent implements ReduceGridAgent, Externalizable {

    private static final long serialVersionUID = 6568906743945108310L;

    private List keys = null;

    private List vals = null;
```

```
protected boolean isEntityMap;

public InsertAgent() {
}

public InsertAgent(boolean entityMap) {
    isEntityMap = entityMap;
}

public Object reduce(Session sess, ObjectMap map) {
    throw new UnsupportedOperationException(
        "ReduceGridAgent.reduce(Session, ObjectMap)");
}

public Object reduce(Session sess, ObjectMap map, Collection arg2) {
    Session s = null;
    try {
        s = sess.getObjectGrid().getSession();
        ObjectMap m = s.getMap(map.getName());
        s.beginNoWriteThrough();
        Object ret = process(s, m);
        s.commit();
        return ret;
    } catch (ObjectGridRuntimeException e) {
        if (s.isTransactionActive()) {
            try {
                s.rollback();
            } catch (TransactionException e1) {
            } catch (NoActiveTransactionException e1) {
            }
        }
        throw e;
    } catch (Throwable t) {
        if (s.isTransactionActive()) {
            try {
                s.rollback();
            } catch (TransactionException e1) {
            } catch (NoActiveTransactionException e1) {
            }
        }
        throw new ObjectGridRuntimeException(t);
    }

}

public Object process(Session s, ObjectMap m) {
    try {

        if (!isEntityMap) {
            // 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;
    }
}
```

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 mudança 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.

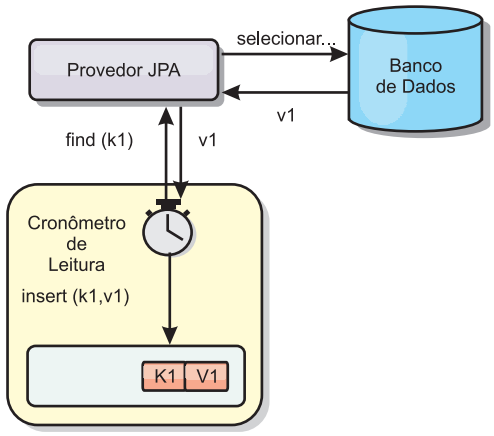


Figura 16. Atualização Periódica

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 mudança 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 as informações deno *Guia de Administração*.

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 as informações 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 316.

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 mudança 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 mudança 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 mudança. Além disso, um número de mudança mais recente é controlado para esta tabela. Sempre que um registro for inserido ou atualizado, o número de mudança 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;
}
```

Capítulo 8. Programação para Integração de Spring

Aprenda como integrar seus aplicativos eXtreme Scale com o Spring Framework popular.

Visão Geral de Integração da Estrutura Spring

O Spring é uma estrutura popular para desenvolvimento de aplicativos Java. O WebSphere eXtreme Scale fornece suporte para permitir que o Spring gerencie as transações do eXtreme Scale e configure clientes e servidores que compõem a grade de dados de 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 se conectar em 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 as 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. Consulte Beans de extensão Spring e suporte a espaço de nomes para obter mais detalhes sobre esses tópicos e uma amostra de como iniciar um servidor de contêiner do eXtreme Scale usando configurações Spring.

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. Todos os pedidos para beans com um ID ou IDs correspondentes a essa definição de bean no mesmo shard resultarão no retorno dessa instância de bean específica 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 o estado da sua sessão na Sessão HTTP pelo padrão. Se um aplicativo da Web for configurado para usar o eXtreme Scale para o gerenciamento de sessão, então ele será usado automaticamente pelo Spring para armazenar este estado e se tornará tolerante a falhas da mesma forma que a sessão.

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 JEE que está em execução em um WebSphere Extended Deployment alterado WebSphere Application Server Network Deployment, então o aplicativo deve colocar o arquivo `spring.jar` e seus arquivos associados nos módulos EAR (Enterprise Archive). Você também deve colocar o arquivo `ogspring.jar` no mesmo local.

Transações Gerenciadas pelo Spring

Spring é uma estrutura popular para desenvolver 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.

Transações Nativas com o WebSphere eXtreme Scale

O Spring fornece transações gerenciadas por contêiner junto com o estilo de um servidor de aplicativos Java Platform, Enterprise Edition, 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.

Criando um Gerenciador de Transações

O eXtreme Scale fornece uma implementação de um Spring PlatformTransactionManager. Este gerenciador pode fornecer sessões gerenciadas do eXtreme Scale para POJOs gerenciados pelo Spring. Pelo 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 um Objeto Session do ObjectGrid para a Transação Spring Atual

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.

Um POJO de Amostra Utilizando Anotações

A seguir, está um POJO que utiliza anotações para declarar suas intenções transacionais para o Spring. É possível visualizar que a classe possui uma anotação de nível de classe indicando que todos os métodos, por padrão, devem utilizar a semântica de transação REQUIRED. A classe implementa uma interface com um método para todos os métodos na classe. Isso é necessário para que o AOP Spring funcione quanto ele não puder realizar o bytecode weaving. A classe possui um txManager da variável da instância que vinculamos ao gerenciador de transações do ObjectGrid utilizando o arquivo XML do Spring. Cada método simplesmente chama o método txManager.getSession para obter o objeto Session para utilizar o método. O método queryNewTx é anotado para indicar uma semântica REQUIRES_NEW. Isto significa que qualquer transação existente será suspensa e uma nova transação independente criada para tal método.

```
@Transactional(propagation=Propagation.REQUIRED)
public class TestService implements ITestService
{
    SpringLocalTxManager txManager;

    public TestService()
    {

    }

    public void initialize()
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        ObjectMap m = s.getMap("TEST");
        m.insert("Hello", "Billy");
    }

    public void update(String updatedValue)
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        System.out.println("Update using " + s);
        ObjectMap m = s.getMap("TEST");
        String v = (String)m.get("Hello");
        m.update("Hello", updatedValue);
    }

    public String query()
        throws ObjectGridException
    {
        Session s = txManager.getSession();
```

```
        System.out.println("Query using " + s);
        ObjectMap m = s.getMap("TEST");
        return (String)m.get("Hello");
    }

    @Transactional(propagation=Propagation.REQUIRES_NEW)
    public String queryNewTx()
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        System.out.println("QueryTX using " + s);
        ObjectMap m = s.getMap("TEST");
        return (String)m.get("Hello");
    }

    public void testRequiresNew(ITestService bean)
        throws ObjectGridException
    {
        update("1");
        String txValue = bean.query();
        if(!txValue.equals("1"))
        {
            System.out.println("Requires didnt work");
            throw new IllegalStateException("requires didn't work");
        }
        String committedValue = bean.queryNewTx();
        if(committedValue.equals("1"))
        {
            System.out.println("Requires new didnt work");
            throw new IllegalStateException("requires new didn't work");
        }
    }

    public SpringLocalTxManager getTxManager() {
        return txManager;
    }

    public void setTxManager(SpringLocalTxManager txManager) {
        this.txManager = txManager;
    }
}
```

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.

Autoinicialização Simples para Testes

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 tem 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().

eXtreme Scale para Transações JTA e Globais

O eXtreme Scale não suporta o JTA ou o 2 phase commit por vários motivos, principalmente em relação 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.

Beans de Extensão Gerenciados pelo Spring

É possível declarar que 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 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, então, normalmente, tais POJOs possuem um requisito para serem conectados ao restante do aplicativo.

Um aplicativo pode registrar uma instância do Spring Bean Factory para usar para um ObjectGrid específico nomeado. O aplicativo deve criar uma instância do BeanFactory ou um contexto de aplicativo Spring e, então, registrá-la com o ObjectGrid utilizando o seguinte método estático:

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object springBeanFactory)
```

Este método especifica se o ObjectGrid localiza um bean de extensão (como um ObjectTransformer, Loader, TransactionCallback, etc.) cujo className inicia com o prefixo {spring} e, em seguida, utiliza o restante do nome como um nome Spring Bean e obtém a instância de bean utilizando o Spring Bean Factory. O ObjectGrid também pode criar um Spring Bean Factory a partir de um arquivo de configuração XML do Spring padrão. Se nenhum bean factory tiver sido registrado para um determinado ObjectGrid, o ObjectGrid tentará localizar um arquivo xml chamado 'ObjectGridName'_spring.xml. Por exemplo, se sua grade for denominada GRID, então o arquivo xml será denominado '/GRID_spring.xml' e deverá estar no caminho da classe no pacote-raiz. Se este arquivo for localizado, então, o ObjectGrid constrói um ApplicationContext utilizando tal arquivo e constrói beans a partir de tal bean factory. Um nome de classe de exemplo poderia ser:

```
"{spring}MyLoaderBean"
```

Isto faria com que o ObjectGrid solicitasse ao Spring um bean denominado "MyLoaderBean". Esta abordagem pode ser utilizada para especificar POJOs gerenciados pelo Spring para qualquer ponto de extensão no ObjectGrid contanto que o bean factory tenha sido registrado de antemão. As extensões spring do ObjectGrid estão no arquivo ogspring.jar. Este arquivo jar deve estar no caminho de classe para que o suporte ao spring funcione adequadamente. Se um aplicativo JavaEE estiver em execução em um ND acrescido por XD, então, o aplicativo deve colocar o arquivo spring.jar e seus arquivos associados nos módulos EAR. O ogspring.jar também deve ser colocado no mesmo local.

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 casos, você deve usar o Spring para configurar determinados objetos do plug-in. Use a configuração a seguir como exemplo:

```
<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 TransactionCallback integrada, 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 bena 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 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
- register
- server
- 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 as informações sobre o arquivo descritor Spring no *Guia de Administração*.

Continue usando 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 aque que em vez de especificar a classe "com.ibm.websphere.objectgrid.jpa.JPATxCallback" como no exemplo anterior, foi usado diretamente o bean "objectgrid:JPATxCallback" predefinido. Como pode ser visto, esta configuração é menos detalhada e mais amigável para verificação de erro.

Início com Servidor de Contêineres com Spring Extension Beans

Neste exemplo, será mostrado como iniciar um servidor ObjectGrid usando beans de extensão gerenciados Spring do ObjectGrid e suporte a espaço de nomes.

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>
```

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>
```

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 arquivo XML do Spring é nomeado como "Grid_spring.xml". Observe que estão incluídos dois esquemas no arquivo XML: spring-beans-2.0.xsd é para uso com benas gerenciados do Spring, e objectgrid.xsd é para uso com beans predefinidos no espaço de nomes 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>
```

Há 6 beans definidos neste arquivo XML do Spring:

1. *objectgrid:register*: Isto registra o bean factory padrão para o "Grid" do ObjectGrid.
2. *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.
3. *objectgrid:catalog*: Isto define um terminal de serviço de catálogo ObjectGrid, que está configurado como "localhost:2809".

4. *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.
5. *objectgrid:LRUEvictor*: Isto define um LRUEvictor com a quantidade de filas LRU para usar configurada como 31.
6. *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.

Início do 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();
    }
}
```

Capítulo 9. Programação para Segurança

Use as interfaces de programação para tratar vários aspectos da segurança em um ambiente do eXtremeScale.

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.

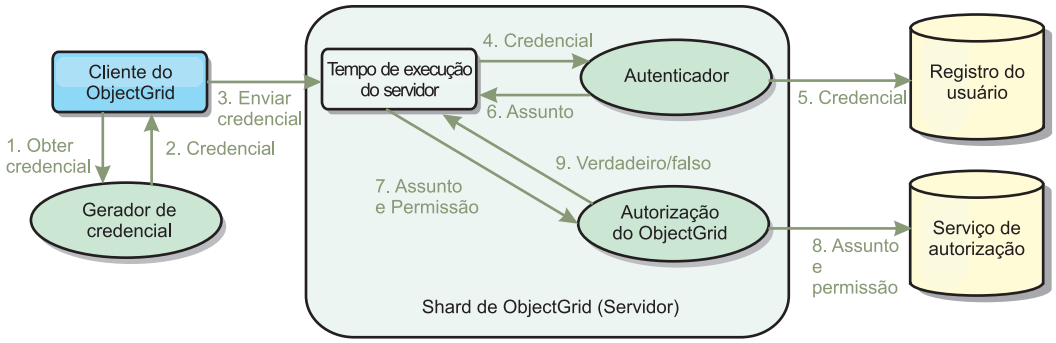


Figura 17. Fluxo de Autenticação e Autorização do Cliente

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 discute como é possível usar o SecureTokenManager 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. Consulte as informações sobre a API de Credencial na documentação de API para obter mais detalhes. 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 de credencial 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).

Programação de Autorização de Cliente

O WebSphere eXtreme Scale suporta autorização JAAS (Java Authentication and Authorization Service) pronta para o uso e também suporta Autorização customizada usando a interface ObjectGridAuthorization.

O plug-in ObjectGridAuthorization é usado para autorizar acessos a ObjectGrid, ObjectMap e JavaMap aos Principals representados por um objeto Subject de uma forma customizada. 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) 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 13. 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)

Tabela 13. Lista de Métodos e a MapPermission Necessária (continuação)

Permissão	ObjectMap/JavaMap
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 14. Lista de Métodos e a ObjectGridPermission Necessária

Ação da permissão	Métodos
consultar	com.ibm.websphere.objectgrid.Session.createObjectQuery(String)
consultar	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 15. 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 1.4 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 é composto de um conjunto de proprietários e cada Proprietário representa uma identidade desse 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 a esses mapas para 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 OS local for utilizado como registro do usuário, o nome da máquina é MACH1, o ID do usuário é user1 e o principal_name é 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 possuir.

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 trecho 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;
}
```

Autenticação de Grade

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.

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.

Ativando a Segurança

A lista a seguir fornece as duas maneiras pelas quais a segurança local é ativada:

- **Configuração XML** É possível utilizar o arquivo XML do ObjectGrid para definir um ObjectGrid e ativar a segurança para tal ObjectGrid. O arquivo a seguir é o arquivo secure-objectgrid-definition.xml que é utilizado na amostra do aplicativo corporativo ObjectGridSample. Nesse arquivo XML, a segurança é ativada ao configurar o atributo securityEnabled como true.

```
<objectGrids>
    <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
        authorizationMechanism="AUTHORIZATION_MECHANISM_JASS">
    ...
</objectGrids>
```

- **Programação** Se desejar criar um ObjectGrid utilizando o ObjectGrid.setSecurityEnabled() do método da API, chame o método a seguir na interface ObjectGrid para ativar a segurança.

```
/**
 * Enable the ObjectGrid security
 */
void setSecurityEnabled();
```

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) {
    // get the RunAs subject
    s = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
  }
  else if (finalType == CALLER_SUBJECT) {
    // get the 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. É possível 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. É possível 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

É possível 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 por meio 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

É possível 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 10. Considerações sobre Desempenho para Desenvolvedores de Aplicativos

Para melhorar o desempenho do espaço de processamento da grade de dados ou do banco de dados de memória, é possível examinar diversas considerações, como ajuste das configurações de sua Java Virtual Machine e o uso de boas práticas para recursos de produto, como bloqueio, serialização e desempenho de consulta.

Ajuste da JVM

Você deve levar em conta vários aspectos específicos do ajuste da Java Virtual Machine (JVM) para melhorar o desempenho do WebSphere eXtreme Scale.

A recomendação é de heaps de 1 a 2 Gb com um JVM para cada 4 núcleos. Os tamanhos de heap dependem da natureza dos objetos que estão sendo armazenados nos servidores, o que é discutido posteriormente neste documento.

Recomendações de Tamanho de Heap e Coleta de Lixo

O melhor número de tamanho de heap depende de três fatores:

1. Quantidade de objetos ativos no heap.
2. Complexidade dos objetos ativos no heap.
3. Quantidade de núcleos disponíveis para a JVM.

Por exemplo, um aplicativo que armazena matrizes de 10 Kbytes pode executar um heap muito maior do que um aplicativo que usa gráficos complexos de POJOs.

Todas as JVMs modernas usam algoritmos de coleta de lixo paralelos, o que significa que usar mais núcleos pode reduzir as pausas na coleta de lixo. Assim, caixas com 8 núcleos serão coletadas mais rapidamente do que uma caixa com 4 núcleos.

Uso de Memória Real Contra a Especificação do Heap

Uma JVM com heap de 1 Gb usa aproximadamente 1,3 Gb de memória real. Em nosso laboratório, não conseguimos executar dez JVMs de 1 Gb em uma caixa com 16 Gb de RAM. Depois de os heaps JVM serem preenchidos com 800 MB adicionais, a caixa começou a efetuar a paginação.

Coleta de Lixo

Para JVMs IBM, use o coletor avgoptpause para cenários com alta taxa de atualização (100% de entradas de modificação de transações). O coletor gencon funciona muito melhor que o coletor avgoptpause nos cenários em que os dados são atualizados relativamente com pouca freqüência (10% do tempo ou menos). Experimente ambos os coletores para ver qual funciona melhor no seu cenário. Em caso de algum problema de desempenho, ative a coleta de lixo detalhada para verificar a porcentagem de tempo que está sendo consumida pela coleta. Ocorreram cenários em que 80% do tempo foi gasto na coleta de lixo até que um que o ajuste resolvesse o problema.

Desempenho da JVM

O WebSphere eXtreme Scale pode executar em diferentes versões de J2SE (Java 2 Platform, Standard Edition). O ObjectGrid Versão 6.1 suporta J2SE Versão 1.4.2 e posterior. Para produtividade e desempenho de desenvolvedor aprimorados, utilize o J2SE 5 ou mais recente para obter vantagem das anotações e da coleta de lixo aprimorada. O ObjectGrid trabalha em JVMs de 32 ou 64 bits.

Os clientes do ObjectGrid Versão 6.0.2 podem se conectar a uma grade do ObjectGrid Versão 6.1. Utilize os clientes do ObjectGrid Versão 6.1 para o J2SE Versão 1.4.2 ou clientes melhores. O único motivo para utilizar um cliente do ObjectGrid Versão 6.0.2 é o suporte para o J2SE Versão 1.3.

O WebSphere eXtreme Scale é testado com um subconjunto das máquinas virtuais disponíveis, todavia, a lista suportada não é exclusiva. É possível executar WebSphere eXtreme Scale em qualquer Versão 1.4.2 ou posterior mas, se for identificado um problema na JVM, você deve entrar em contato com o fornecedor da JVM para obter suporte. Se possível, use a JVM a partir do tempo de execução do WebSphere em qualquer plataforma que o WebSphere Application Server suporta.

A Java Platform, Standard Edition 6 é a melhor JVM. A Java 2 Platform, Standard Edition, v 1.4 é executada de maneira deficiente, especialmente para cenários onde o coletor gencon faz uma diferença. A Java Platform Standard Edition 5 é executada bem, mas a Java Platform, Standard Edition 6 é executada melhor.

Ajuste de orb.properties

A recomendação é usar o seguinte arquivo orb.properties para produção. Em nosso laboratório, usamos este arquivo em grades de até 1500 JVMs. O arquivo orb.properties está na pasta lib do JRE que está sendo usado.

```
# IBM JDK properties for ORB
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton

# WS Interceptors
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer

# WS ORB & Plugins properties
com.ibm.CORBA.ForceTunnel=never
com.ibm.CORBA.RequestTimeout=10
com.ibm.CORBA.ConnectTimeout=10

# Needed when lots of JVMs connect to the catalog at the same time
com.ibm.CORBA.ServerSocketQueueDepth=2048

# Clients and the catalog server can have sockets open to all JVMs
com.ibm.CORBA.MaxOpenConnections=1016

# Thread Pool for handling incoming requests, 200 threads here
com.ibm.CORBA.ThreadPool.IsGrowable=false
com.ibm.CORBA.ThreadPool.MaximumSize=200
com.ibm.CORBA.ThreadPool.MinimumSize=200
com.ibm.CORBA.ThreadPool.InactivityTimeout=180000

# No splitting up large requests/responses in to smaller chunks
com.ibm.CORBA.FragmentSize=0
```

Contagem de Encadeamentos

A contagem de encadeamentos depende de poucos fatores. Há um limite de quantos encadeamentos um único shard pode gerenciar. Com mais shards para cada JVM, pode haver mais encadeamentos e mais simultaneidade. Cada shard adicional fornece mais caminhos simultâneos para os dados. Cada shard é um concorrente possível mas, mesmo assim, existe um limite.

Boas Práticas de CopyMode

WebSphere eXtreme Scale faz uma cópia do valor com base nas seis configurações de CopyMode disponíveis. Determine qual configuração funciona melhor para seus requisitos de implementação.

É possível usar o método da API BackingMap setCopyMode(CopyMode, valueInterfaceClass) para configurar o modo de cópia para um dos seguintes campos estáticos finais que foram definidos na classe com.ibm.websphere.objectgrid.CopyMode.

Quando um aplicativo usar a interface ObjectMap para obter uma referência para uma entrada de mapa, use tal referência somente dentro da transação WebSphere eXtreme Scale 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 você esteja usando o modo de cópia COPY_TO_BYTES. 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 tem uma referência para ser 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. Usando a configuração COPY_TO_BYTES, é possível reduzir a quantidade de memória que um gráfico grande de Objetos pode consumir. Consulte “Mapas de Matriz de Byte” na página 40 para obter informações adicionais.

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.

Mapas de Matriz de Byte

É possível armazenar os pares de chave-valor 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 “Desempenho de Serialização” na página 231.

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" />
```

Consulte o tópico sobre o arquivo XML do descritor ObjectGrid no *Guia de Administração* para obter mais informações.

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.

Boas Práticas de Desempenho do Evictor de Plug-in

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 evictores least frequently used (LFU) e least recently 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:

```
//Use ObjectGridManager to create/get the ObjectGrid. Refer to
// the ObjectGridManger section
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");

//Set properties assuming 50,000 map entries
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");

//Set properties assuming 50,000 map entries
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.

Boas Práticas de Desempenho de Bloqueio

Estratégias de bloqueio e configurações de isolamento de transação afetam o desempenho dos seus aplicativos.

Recuperar uma Instância Armazenada em Cache

Consulte as informações sobre o bloqueio de entrada de mapa no *Guia de Administração* para obter mais informações.

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.

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. Delegar

os dados é a operação mais dispendiosa e deve ser endereçada pelo desenvolvedor de aplicativos ao projetar o esquema, configurar a grade 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.

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.

Para obter mais informações, consulte Plug-ins para Serializar e Copiar Objetos em Cache e Boas Práticas da Interface de ObjectTransformer.

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. Consulte Para obter mais informações, consulte o tópico sobre o uso de um utilitário de carga com os mapas e tuplas de entidade no *Guia de Programação*. 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 nos quais os objetos devem ser modificados para utilizar a serialização customizada, tais como a implementação da interface java.io.Externalizable ou pela implementação dos 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 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 por meio 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.

Boas Práticas da 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.

Visão Geral

Para obter detalhes sobre a interface ObjectTransformer, consulte “Plug-in ObjectTransformer” na página 227. A partir de um ponto de vista de desempenho, e das informações do método CopyMode que estão no tópico de boas práticas do método CopyMode, o eXtreme Scale claramente copia os valores para todos os casos, 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 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.

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.

Capítulo 11. Resolução de Problemas

Além dos logs e do rastreio, de mensagens e notas sobre o release discutidos nesta seção, é possível usar ferramentas de monitoramento para descobrir problemas como o local de dados no ambiente, a disponibilidade de servidores na grade 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 de amostra xsAdmin para exibir informações de texto sobre o ambiente.

Logs e Rastreio

É possível utilizar logs e rastreio para monitorar e solucionar problemas em seu ambiente. Os logs estão em locais diferentes dependendo da sua configuração. Pode ser necessário fornecer rastreio para um servidor quando você trabalha com o suporte IBM.

Logs com o WebSphere Application Server

Consulte o WebSphere Application ServerCentro de Informações para obter mais informações.

Logs com o WebSphere eXtreme Scale em um Ambiente Independente

Com os servidores de catálogos e contêineres independentes, é possível configurar o local dos logs e qualquer especificação de rastreio. Os logs do servidor de catálogo estão no local onde você executou o comando do servidor de início.

Configurando o local de log para os servidores de contêiner

Por padrão, os logs para um contêiner estão no diretório onde o comando do servidor foi executado. Se você iniciar os servidores no diretório *<eXtremeScale_home>*/bin, os arquivos de log e de rasteio estão nos diretórios logs/*<server_name>* no diretório bin. Para especificar um local alternativo para os logs do servidor de contêineres, como o arquivo server.properties, com os seguintes conteúdos:

```
workingDirectory=<directory>
traceSpec=
systemStreamToFileEnabled=true
```

A propriedade workingDirectory é o diretório-raiz para os logs e o arquivo de rastreio opcional. O WebSphere eXtreme Scale cria um diretório com o nome do servidor de contêineres com um arquivo SystemOut.log, um arquivo SystemErr.log e um arquivo de rastreio se o rastreio foi ativado com a opção traceSpec. 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.

As mensagens de informações comuns a serem procuradas no arquivo SystemOut.log são o início das mensagens de confirmação. Para obter mais informações sobre uma mensagem específica, consulte “Mensagens” na página 388.

Rastreio com o WebSphere Application Server

Consulte o WebSphere Application ServerCentro de Informações para obter mais informações.

Rastreio no Serviço de Catálogo

É possível configurar o rastreio em um serviço de catálogo usando os parâmetros **-traceSpec** e **-traceFile** durante a inicialização do serviço de catálogo. Por exemplo:

```
startOgServer.sh catalogServer -traceSpec
ObjectGridPlacement=all=enabled -traceFile
/home/user1/logs/trace.log
```

Se você iniciar o serviço de catálogo no diretório *<eXtremeScale_home>*/bin, os arquivos de log e de rasteio estarão no diretório logs/*<catalog_service_name>* no diretório bin. Consulte informações sobre o início do processo de serviço de catálogo em um ambiente independente no *Guia de Administração*.

Rastreio em um servidor de contêineres Independente

É possível ativar o rastreio em um servidor de contêiner de duas formas. É possível criar um arquivo de propriedades do servidor conforme explicado na seção de logs ou é possível ativar o rastreio utilizando a linha de comandos na inicialização. Para ativar o rastreio de contêiner com um arquivo de propriedades do servidor, atualize a propriedade **traceSpec** com a especificação de rastreio necessária. Para ativar o rastreio de contêiner utilizando parâmetros de início, utilize os parâmetros **-traceSpec** e **-traceFile**. Por exemplo:

```
startOgServer.sh c0 -objectGridFile ../xml/myObjectGrid.xml
-deploymentPolicyFile ../xml/myDepPolicy.xml -catalogServiceEndpoints
server1.rchland.ibm.com:2809 -traceSpec
ObjectGridPlacement=all=enabled -traceFile /home/user1/logs/trace.log
```

Se você iniciar o servidor no diretório *<eXtremeScale_home>*/bin, os arquivos de log e de rasteio estarão nos diretórios logs/*<server_name>* no diretório bin.

Consulte

Rastreio com a Interface ObjectGridManager

Outra opção é 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");
```

Ativando o Rastreio com o Utilitário xsadmin

Para ativar o rastreio com o utilitário xsadmin, use a opção **setTraceSpec**. Use o utilitário xsadmin para ativar o rastreio em um ambiente independente durante o tempo de execução e não durante a inicialização. É possível ativar o rastreio em todos os servidores e serviços de catálogo ou filtrar os servidores com base no nome do ObjectGrid, e assim por diante. Por exemplo, para ativar o rastreio ObjectGridReplication com acesso ao servidor de serviço de catálogo, execute:

```
<eXtremeScale_home>/bin>xsadmin.bat -setTraceSpec "ObjectGridReplication=all=enabled"
```

Também é possível desativar o rastreio ao configurar a especificação de rastreio para `*=all=disabled.`:

Consulte as informações sobre o utilitário xsAdmin no *Guia de Administração* para obter mais informações.

Diretório e Arquivos ffdc

Os arquivos FFDC servem para que o suporte IBM auxilie na depuração. Estes arquivos podem ser solicitados pelo suporte IBM se 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
```

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. Assim como o rastreio do WebSphere Application Server, é possível especificar o nível de rastreio a ser utilizado. 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.

7.1+ **ObjectGridClientInfo**
Informações do cliente do DB2.

7.1+ **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.

7.1+ **ObjectGridMonitor**
Infraestrutura de monitoramento histórico.

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.

ObjectGridStats
Estatísticas do ObjectGrid.

ObjectGridStreamQuery
API de Consulta do Fluxo.

ObjectGridWriteBehind
Atributo write-behind do ObjectGrid.

Projector
O mecanismo com a API do EntityManager.

QueryEngine
O mecanismo de consulta para a API de Consulta do Objeto e a API de Consulta do EntityManager.

QueryEnginePlan
Diagnósticos do plano de consulta.

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/.

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.

Localizando Mensagens

Ao encontrar uma mensagem em um log, copie o número da mensagem com seu prefixo de letra e número e procure no centro de informações (por exemplo, `CWOBJ1526I`). Ao procurar pela mensagem, será possível encontrar uma explicação adicional da mensagem e possíveis ações a serem tomadas para resolver o problema.

Consulte o centro de informações para obter um índice de mensagens do produto.

Notas sobre o Release

São fornecidos links para o Web site de suporte do produto, para documentação do produto e para atualizações de última hora, limitações e problemas conhecidos para o produto.

- "Acessando Últimas Atualizações, Limitações e Problemas Conhecidos"
- "Acessando Requisitos de Software e de Sistema" na página 389
- "Acessando a Documentação do Produto" na página 389
- "Acessando o Web Site de Suporte do Produto" na página 389
- "Entrando em Contato com o Suporte ao Software IBM" na página 389

Acessando Últimas Atualizações, Limitações e Problemas Conhecidos

As notas sobre o release estão disponíveis como notas técnicas no site de suporte do produto. Para ver uma lista de todas as notas técnicas do WebSphere eXtreme Scale, vá para a Página da Web de Suporte. Clicar nos links fornecidos aqui resultará em uma pesquisa, na página da Web de Suporte, das notas relevantes sobre o release, que serão retornadas como uma lista.

- **7.1+** Para visualizar uma lista das notas sobre o release para a Versão 7.1, vá para a Página da Web de Suporte.
- Para visualizar uma lista de notas sobre o release para a Versão 7.0, vá para a Página da Web de Suporte.
- Para visualizar uma lista das notas sobre o release para a Versão 6.1, vá para a página da wiki Notas sobre o Release.

Acessando Requisitos de Software e de Sistema

Os requisitos de hardware e software são documentados nas páginas a seguir:

- Requisitos do Sistema Detalhados

Acessando a Documentação do Produto

Para obter o conjunto de informações inteiro, acesse apágina da Biblioteca.

Acessando o Web Site de Suporte do Produto

Para procurar as notas técnicas, downloads, correções e outras informações relacionadas a suporte mais recentes, acesse apágina de Suporte.

Entrando em Contato com o Suporte ao Software IBM

Se você encontrar um problema com o produto, primeiro, tente as seguintes ações:

- Siga as etapas descritas na documentação do produto
- Procure a documentação relacionada na ajuda on-line
- Consulte as mensagens de erro na referência de mensagens

Se você não conseguir resolver o problema com nenhum dos métodos anteriores, entre em contato com o Suporte Técnico IBM.

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 lhe garante direito algum 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 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

Tais informações podem estar disponíveis, sujeitas a termos e condições apropriadas, 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

L

M

N

O

P

R

S

T

U

Impresso no Brasil