WebSphere® WebSphere Extended Deployment Versão 6.0.x

Guia de Modelo de Programação do ObjectGrid

Nota

Antes de utilizar estas informações, certifique-se de ler as informações gerais em “Avisos” na página 379.

Data da Compilação: 28 de Março de 2006

Índice

Como Enviar Seus Comentários

Seu feedback é importante para ajudar a fornecer informações mais precisas e de alta qualidade.

- Para enviar comentários sobre artigos no WebSphere Extended Deployment Information Center, disponível em: `http://www.ibm.com/software/webservers/appserv/extend/library/`
  1. Exiba o artigo em seu navegador da Web e mova o cursor para o fim do artigo.
  2. Preencha o link **Feedback** na parte inferior do artigo e envie.
- Para enviar comentários sobre este e outros manuais em PDF, você pode enviar seus comentários por e-mail para: **wasdoc@us.ibm.com**.

  Certifique-se de incluir o nome e número do documento e, se possível, o número da página, tabela ou figura específica sobre a qual você está enviando comentários.

Quando o Cliente envia seus comentários, concede direitos não-exclusivos à IBM para usá-los ou distribuí-los da maneira que achar conveniente, sem que isso implique em qualquer compromisso ou obrigação para com o Cliente.

Capítulo 1. Introdução ao ObjectGrid Executando o Aplicativo de Amostra

Utilize este tópico para introdução ao ObjectGrid, uma estrutura de computação distribuída que disponibiliza objetos para um conjunto de aplicativos.

O WebSphere Extended Deployment Versão 6.0 ou posterior e o WebSphere Application Server Versão 6.0.2 ou posterior deve estar instalado em pelo menos uma máquina em seu ambiente.

Restrição: Se estiver utilizando o ObjectGrid com o WebSphere Extended Deployment Versão 6.0, as organizações de licenciamento adicionais serão requeridas para também utilizar o ObjectGrid em um ambiente J2SE (Java 2 Platform, Standard Edition) Versão 1.4.2 ou superior ou em um ambiente WebSphere Application Server Versão 6.02 ou superior. Entre em contato com seu representante de vendas para obter detalhes.

Se quiser desenvolver aplicativos ObjectGrid sem acessar máquinas servidores com o WebSphere Extended Deployment instalado, você pode executá-los em sua máquina local. A máquina local requer a instalação de um IBM SDK (Software Developer Kit) ou Eclipse.

Para desenvolver aplicativos do ObjectGrid em sua máquina local, copie os seguintes diretórios de sua instalação para sua máquina local:

- Se estiver utilizando o WebSphere Extended Deployment Versão 6.0.1, copie o arquivo /lib/wsobjectgrid.jar e o arquivo /optionalLibraries/ObjectGrid/objectgridSamples.jar para seu diretório de trabalho.
- Se tiver instalado o ObjectGrid por meio de uma instalação mixed server environment, copie os arquivos /ObjectGrid/lib/objectgrid.jar e /ObjectGrid/samples/objectgridSamples.jar para seu diretório de trabalho.

Para obter informações adicionais sobre os arquivos JAR (Java Archive) que estão instalados com o ObjectGrid, consulte Pacote do ObjectGrid.

Utilize esta tarefa para executar e avançar com os aplicativos de amostra ObjectGrid. Você pode executar os aplicativos nesta tarefa em uma linha de comandos Java, no Eclipse ou em um ambiente J2EE (Java 2 Platform, Enterprise Edition).

- Para obter o aplicativo de amostra do ObjectGrid em execução na linha de comandos, consulte Executando o Aplicativo de Amostra do ObjectGrid na Linha de Comandos.
- Para executar o aplicativo de amostra do ObjectGrid no Eclipse, consulte Importando e Utilizando o Aplicativo de Amostra do ObjectGrid no Eclipse.
- Para executar o aplicativo de amostra do ObjectGrid no WebSphere Extended Deployment, consulte Carregando e Executando o Aplicativo de Amostra do ObjectGrid com o WebSphere Extended Deployment

Você fez uma introdução ao ObjectGrid executando o aplicativo de amostra e carregando a amostra para seu ambiente de desenvolvimento.

Executando o Aplicativo de Amostra ObjectGrid na Linha de Comandos

Utilize este tópico para executar aplicativos ativados por ObjectGrid em uma linha de comandos Java e teste a configuração do ObjectGrid.

Antes de iniciar esta tarefa, instale o Mixed Server Environment, incluindo o ObjectGrid independente.

É necessário ter um SDK (Software Development Kit) instalado. Também é necessário ter acesso aos aplicativos de amostra do ObjectGrid. Consulte Introdução ao ObjectGrid para obter informações adicionais.

Utilize esta tarefa para executar rapidamente um aplicativo com o ObjectGrid ativado.

1. Verifique a versão do SDK. O ObjectGrid requer um IBM SDK 1.4.2 ou superior. Para testar seu ambiente Java antes de executar o aplicativo de amostra ObjectGrid, execute as seguintes etapas:
   a. Abra um prompt da linha de comandos.
   b. Digite o seguinte comando:

      ```
      java -version
      ```

      Se o comando for executado corretamente, será exibido um texto semelhante ao seguinte exemplo:

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

      Nota: Também é possível executar estas amostras utilizando um SDK J2SE (Java 2 Platform, Standard Edition) Versão 1.3.x. Para obter informações adicionais, consulte Pacote do ObjectGrid.

      Se for exibido um erro, certifique-se de que o SDK esteja instalado e esteja em seu CLASSPATH.
2. Execute o aplicativo de amostra do ObjectGrid. O aplicativo de amostra ilustra um caso simples que envolve funcionários, escritórios e locais de trabalho. O aplicativo de amostra cria uma instância do ObjectGrid com mapas para cada tipo de objeto. Cada mapa tem entradas inseridas e manipuladas para demonstrar a função de armazenamento em cache do ObjectGrid.
   a. Abra uma linha de comandos e navegue para o diretório de trabalho. Copie os arquivos objectgrid.jar, asm.jar e cglib.jar da pasta /ObjectGrid/lib para um diretório de trabalho. Copie o /ObjectGrid/samples/objectgridSamples.jar para o diretório de trabalho.
   b. Emita o seguinte comando:

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

      O sistema exibe uma saída semelhante ao texto a seguir. Esta saída foi reduzida para fins de publicação:

      ```
      Initializing ObjectGridSample ...
      resourcePath: META-INF/objectgrid-definition.xml
      objectgridUrl:
      jar:file:/C:/temp/objg/objectgridSample.jar!/
      META-INF/objectgrid-definition.xml
      ```

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

3. Execute o aplicativo de amostra do ObjectGrid distribuído.

   O programa com.ibm.websphere.samples.objectgrid.basic.ObjectGridSample utiliza uma instância do ObjectGrid local como o cache de dados. Todos os objetos são armazenados em cache na JVM (Java Virtual Machine) local. Para utilizar um ObjectGrid distribuído implementado em um cluster do ObjectGrid, utilize o programa com.ibm.websphere.samples.objectgrid.distributed.DistributedObjectGridSample. O programa DistributedObjectGridSample está incluído no objectgridSamples.jar .

   a. Inicie um cluster do ObjectGrid. Para obter informações adicionais sobre como iniciar um cluster do ObjectGrid independente para utilização com a amostra do ObjectGrid distribuído, consulte Iniciando o Cluster do ObjectGrid de Amostra Independente.

b. Depois de iniciar o servidor do ObjectGrid, será possível executar o aplicativo de amostra do ObjectGrid distribuído com o seguinte comando:

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

Quando o cluster do ObjectGrid requerido for iniciado, o programa DistributedObjectGridSample terá uma saída semelhante ao programa ObjectGridSample.

Você executou o aplicativo de amostra do ObjectGrid em uma linha de comandos Java para testar a funcionalidade do ObjectGrid.

A origem para esta amostra está no arquivo objectgridSamples.jar, especificamente nos arquivos com\ibm\websphere\samples\objectgrid\basic\ObjectGridSample.java e com\ibm\websphere\samples\objectgrid\distributed\ DistributedObjectGridSample.java.

Iniciando o Cluster do ObjectGrid de Amostra Independente

Para executar a amostra do ObjectGrid distribuída, é necessário iniciar um cluster do ObjectGrid que hospeda o ObjectGrid requerido.

Verifique se o WebSphere Extended Deployment para Mixed Server Environment, Versão 6.0.x está instalado.

Utilize esta tarefa para iniciar um servidor do ObjectGrid baseado nos arquivos `cluster-config-1.xml` e `cluster-objectgrid-definition.xml`. Esta tarefa é requerida para executar a amostra do ObjectGrid distribuído. Consulte Executando o Aplicativo de Amostra do ObjectGrid na Linha de Comandos e Importando e Utilizando o Aplicativo de Amostra do ObjectGrid no Eclipse para obter informações adicionais. O `cluster-config-1.xml` tem apenas uma definição de servidor do ObjectGrid. Este servidor do ObjectGrid representa o cluster do ObjectGrid de amostra.

1. Localize o arquivo `objectgridSamples.jar` no diretório *mse_install_root*`/ObjectGrid/samples`.
2. Extraia o arquivo `META-INF/cluster-config-1.xml` e o arquivo `META-INF/cluster-objectgrid-definition.xml` do arquivo `objectgridSamples.jar` para o diretório *mse_install_root*`/ObjectGrid/samples`.
3. Verifique se a variável de ambiente JAVA_HOME está configurada e se a versão de Java atende o requisito. O Servidor do ObjectGrid requer um ambiente J2SE (Java 2 Platform, Standard Edition) Versão 1.4.2 ou posterior. Para verificar seu ambiente Java, desempenhe as seguintes etapas:
   a. Verifique a variável de ambiente JAVA_HOME. Em um prompt da linha de comandos, emita o seguinte comando:

      ```
      echo %JAVA_HOME%
      ```

      Este comando exibe o caminho para Java. Se precisar configurar a variável de ambiente JAVA_HOME, execute o seguinte comando:

      ```
      set JAVA_HOME=JDK_INSTALL_ROOT
      ```

      Configure o *JDK_INSTALL_ROOT* para seu diretório de instalação Java, por exemplo, `c:\java`.
   b. Verifique sua versão de Java. Execute o seguinte comando:

      ```
      java -version
      ```

Verifique se sua versão é J2SE (Java 2 Platform, Standard Edition) Versão 1.4.2 ou posterior.

4. Inicie o servidor do ObjectGrid. Em um prompt da linha de comandos, emita os seguintes comandos:

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

   Importante: É necessário especificar o arquivo `objectgridSamples.jar` no caminho de classe por meio da opção -jvmArgs. O arquivo `objectgridSamples.jar` contém classes requeridas pelo servidor do ObjectGrid de amostra para implementações do plug-in definidas no arquivo `cluster-objectgrid-definition.xml`. Este arquivo JAR também é utilizado para serializar e desserializar os objetos que estão armazenados em mapas.

   O sistema exibe uma saída semelhante ao texto a seguir. Esta saída foi reduzida para fins de publicação:

```
************ Start Display Current Environment ************
[1/17/06 14:04:34:144 CST] 7daee176 Launcher
I CWOBJ2501I: Launching ObjectGrid server server1.
:
[1/17/06 14:04:37:719 CST] 7daee176 ServerRuntime
I CWOBJ1001I: ObjectGrid Server server1 is ready to process requests.
```

Consulte Executando o Aplicativo de Amostra do ObjectGrid na Linha de Comandos ou Importando e Utilizando o Aplicativo de Amostra do ObjectGrid no Eclipse para executar o aplicativo de amostra do ObjectGrid distribuído. Para obter detalhes adicionais sobre como iniciar e parar o servidor ObjectGrid independente na linha de comandos, consulte Capítulo 8, “Suporte à Linha de Comandos”, na página 83.

Importando e Utilizando o Aplicativo de Amostra ObjectGrid no Eclipse

Utilize esta tarefa para importar e utilizar o aplicativo de amostra ObjectGrid no Eclipse.

Antes de iniciar esta tarefa, instale o Mixed Server Environment, incluindo o ObjectGrid independente.

Para este aplicativo de amostra, utilize o Eclipse Versão 3.1 ou posterior para importar e executar a amostra. Você pode obter o Eclipse a partir do Application Server Toolkit que está incluído no WebSphere Application Server, a partir da instalação do Rational Application Developer ou fazendo seu download diretamente do Eclipse.org.

Utilizando o Eclipse, você pode depurar facilmente seus aplicativos. É possível desempenhar uma demonstração passo a passo do aplicativo de amostra.

1. Importe o projeto para o Eclipse:
   a. Execute o programa Eclipse. Utilize o arquivo `eclipse.exe` no diretório de instalação do Eclipse.
   b. Utilizando o Eclipse, crie um novo projeto.
      1) Clique em **Arquivo > Novo > Projeto > Java > Projeto Java**. Clique em **Avançar**.

2) Digite um nome de projeto. Por exemplo, digite `ObjectGridSamples`.
3) Selecione **Criar Novo Projeto no Espaço de Trabalho**.
4) Na seção Layout de Projeto, clique em **Configurar Padrão**.
5) Para a pasta de origem e de saída, selecione **Projeto** e clique em **OK**.
6) Clique em **Avançar**.
7) Clique na guia **Bibliotecas**.
8) Clique em **Incluir JARs Externos.**
9) Navegue para a pasta `/ObjectGrid/lib` e selecione os arquivos **objectgrid.jar**, **asm.jar** e **cglib.jar**. Clique em **Abrir** no assistente de **seleção do JAR**.
10) Clique em **Concluir**.

2. Importe o arquivo `objectgridSamples.jar` para o Projeto Java.
   a. Clique com o botão direito no projeto Java e selecione **Importar**.
   b. Selecione **Arquivo Zip** em **Selecionar uma Origem de Importação**.
   c. Clique em **Avançar**.
   d. Clique em **Procurar** para abrir o assistente Importar de Arquivo Zip.
   e. Abra o arquivo **objectgridSamples.jar**. Navegue para o diretório `/ObjectGrid/samples`. Selecione o arquivo **objectgridSamples.jar** e clique em **Abrir**.
   f. Verifique se a caixa de opções da árvore de arquivos raiz está selecionada.
   g. Verifique se a opção **Para a Pasta** contém o projeto Java criado na etapa anterior, por exemplo, o projeto `ObjectGridSamples`.
   h. Clique em **Concluir**.
3. Verifique as propriedades do Projeto Java.
   a. Abra a Perspectiva Java. Clique em **Janela > Abrir Perspectiva > Java**.
   b. Vá para a visualização de console. Clique em **Janela > Mostrar Visualização > Console**.
   c. Verifique se a visualização Explorador de Pacotes está disponível e selecionada. Clique em **Janela > Mostrar Visualização > Explorador de Pacotes**.
   d. Clique com o botão direito no projeto Java e selecione **Propriedades**.
   e. Clique em **Caminho de Construção Java** no painel à esquerda.
   f. Clique na guia **Origem** no painel à direita.
   g. Verifique se a raiz do projeto está listada nas pastas de Origem no painel Caminho de Construção.
   h. Clique na guia **Bibliotecas** no painel à direita.
   i. Verifique se os arquivos `objectgrid.jar`, `asm.jar` e `cglib.jar` e uma Biblioteca do Sistema JRE estão listados nas pastas JAR e de classe no painel Caminho de Construção.
   j. Clique em **OK**.
4. Execute a amostra ObjectGrid.
   a. Na visualização Explorador de Pacotes, expanda o projeto Java.
   b. Expanda o pacote com.ibm.websphere.samples.objectgrid.basic.
   c. Clique com o botão direito no arquivo **ObjectGridSample.java**. Clique em **Executar > Aplicativo Java**.
   d. O console exibe saída semelhante quando você executa o aplicativo na linha de comandos Java. Para obter um exemplo da saída, consulte Executando o Aplicativo de Amostra ObjectGrid na Linha de Comandos.

5. Execute a amostra do ObjectGrid distribuído. Para executar a amostra do ObjectGrid distribuído, é necessário configurar um cluster do ObjectGrid. Para executar esta amostra, é possível utilizar os arquivos de configuração XML predefinidos fornecidos no arquivo objectgridSamples.jar. Consulte Iniciando o Cluster do ObjectGrid de Amostra Independente para obter informações adicionais.

   Quando o servidor do ObjectGrid tiver sido iniciado, será possível executar o aplicativo de amostra do ObjectGrid distribuído com as seguintes etapas:

   a. Na visualização Explorador de Pacotes, expanda o projeto Java.
   b. Expanda o pacote com.ibm.websphere.samples.objectgrid.distributed.
   c. Clique com o botão direito no arquivo **DistributedObjectGridSample.java**. Clique em **Executar > Aplicativo Java**.
   d. O console exibe saída semelhante à amostra do ObjectGrid.

As etapas para carregar o projeto e executar o depurador também estão no arquivo `SamplesGuide.htm`. O arquivo `SamplesGuide.htm` está no diretório `doc` no arquivo `objectgridSamples.jar`.

Referências relacionadas

Pacote do ObjectGrid
É possível acessar pacotes do ObjectGrid de duas maneiras: instalando o WebSphere Extended Deployment ou instalando o mixed server environment.

Carregando e Executando o Aplicativo de Amostra ObjectGrid com o WebSphere Extended Deployment

Utilize esta tarefa para carregar e executar a amostra ObjectGrid do J2EE (Java 2 Platform, Enterprise Edition) no WebSphere Extended Deployment.

O WebSphere Application Server e o WebSphere Extended Deployment devem ser instalados.

Utilize esta tarefa para entender e testar a integração do ObjectGrid com o WebSphere Extended Deployment.Para obter informações adicionais, consulte o Capítulo 10, “Integrando o ObjectGrid com o WebSphere Application Server”, na página 291.

1. Instale o arquivo `ObjectGridSample.ear`. Você pode instalar o arquivo EAR (Enterprise Archive) em um servidor de aplicativos único ou em um cluster. Para instalar o arquivo `ObjectGridSample.ear` no console administrativo, desempenhe as seguintes etapas:
   a. No console administrativo, clique em **Aplicativos > Instalar Novo Aplicativo**.
   b. Na página **Preparando para a Instalação do Aplicativo**, especifique o local do aplicativo de amostra do ObjectGrid. Por exemplo, navegue para: `<install_root>/installableApps/ObjectGridSample.ear`. Clique em **Avançar**.
   c. Na segunda página **Preparando para a Instalação do Aplicativo**, utilize as configurações padrão e clique em **Avançar**.
   d. Na página **Selecionar Opções de Instalação**, utilize as configurações padrão e clique em **Avançar**.
   e. Na página **Mapear Módulos para Servidores**, especifique destinos de implementação nos quais você deseja instalar os módulos que estão contidos em seu aplicativo. Selecione um servidor ou cluster de destino da

lista **Clusters e Servidores** para cada módulo. Selecione a caixa de opções **Módulo** para selecionar todos os módulos aplicativos ou para selecionar módulos individuais.

f. Nas páginas a seguir, utilize os valores padrão e clique em **Concluir**.

g. Clique em **Salvar na Configuração Principal** quando concluir a instalação do aplicativo.

h. Clique na opção **Sincronizar Alterações com Nós**. Na página **Aplicativos Corporativos > Salvar**, clique em **Salvar**.

i. Clique em **OK**.

2. Verifique a porta HTTP do default_host dos servidores e inclua um alias do host. Por padrão, os módulos da Web são ligados ao nome do host virtual default_host, a menos que você modifique o nome do host durante a instalação. Se você estiver instalando o aplicativo em um cluster, deverá configurar pelo menos um alias do host para a porta HTTP do default_host para cada membro de cluster. Também é necessário verificar a porta HTTP do default_host para cada membro de cluster e incluir o alias do host correspondente na lista Aliases do Host no console administrativo. Para verificar a porta HTTP do default_host de um servidor, desempenhe as seguintes etapas:

   a. No console administrativo, clique em **Servidores > Application Servers > *server_name***.

   b. Expanda as portas na seção Comunicação. A porta **WC_defaulthost** é a porta do host virtual default_host.

   Para incluir um alias de host, execute as seguintes etapas:

   a. No console administrativo, clique em **Ambiente > Hosts Virtuais > default_host > Aliases do Host > Novo**.

   b. Utilize o valor padrão do nome do host e especifique a porta.

   c. Clique em **OK**.

3. Inicie o aplicativo de amostra do ObjectGrid.

   - Para iniciar o aplicativo em um servidor, clique em **Servidores > Application Servers**. Selecione o servidor que possui o arquivo `ObjectGridSample.ear` instalado. Clique em **Iniciar**.
   - Para iniciar o aplicativo em um cluster, clique em **Servidores > Clusters**. Selecione o cluster que possui o arquivo `ObjectGridSample.ear` instalado. Clique em **Iniciar**.

   Após iniciar o aplicativo em um servidor ou cluster, você pode parar e iniciar o aplicativo independentemente do servidor host ou cluster. Para parar ou iniciar o aplicativo de amostra ObjectGrid, execute as etapas a seguir:

   a. No console administrativo, clique em **Aplicativos > Aplicativos Corporativos**.

   b. Selecione o aplicativo de amostra do ObjectGrid.

   c. Clique em **Iniciar** ou **Parar**.

4. Acesse a amostra do ObjectGrid. Depois de instalar o arquivo `ObjectGridSample.ear` em um único servidor ou cluster e iniciar o aplicativo, é possível acessar a amostra do ObjectGrid no seguinte endereço da Web:

   `http://`*`hostname`*`:`*`port`*`/ObjectGridSample`

   Por exemplo, se seu nome do host for `localhost` e o valor da porta for `9080`, utilize o endereço da Web `http://localhost:9080/ObjectGridSample`.

5. Teste a funcionalidade do ObjectGrid distribuído no ambiente do WebSphere Application Server. O arquivo `ObjectGridSample.ear` também contém o servlet DistributedObjectGridServlet que demonstra a utilização de um ObjectGrid distribuído no ambiente do WebSphere Application Server. O servidor de aplicativos que hospeda o servlet DistributedObjectGridServlet também deve hospedar o servidor do ObjectGrid, que é um membro do cluster do ObjectGrid requerido.
   - Para obter informações adicionais sobre como configurar um cluster do ObjectGrid para colocar o DistributedObjectGridServlet em execução, consulte Iniciando um Cluster do ObjectGrid de Amostra no Ambiente do WebSphere.
   - Para obter informações adicionais sobre como iniciar servidores do ObjectGrid em servidores de aplicativos, consulte Iniciando um Servidor do ObjectGrid em um Servidor de Aplicativos.

   Quando o servidor de aplicativos com o arquivo `ObjectGridSample.ear` instalado também hospedar o servidor do ObjectGrid requerido, o servlet DistributedObjectGridServlet se comportará da mesma maneira que outros servlets. É possível acessar o servlet no seguinte endereço da Web: `http://hostname:port/ObjectGridSample/DistributedObjectGridServlet`. Por exemplo, se seu nome do host for `localhost` e o valor da porta for 9080, utilize o endereço da Web `http://localhost:9080/ObjectGridSample/DistributedObjectGridServlet`.

   Você pode ativar o rastreio do ObjectGrid utilizando a seguinte cadeia de rastreio: `ObjectGrid*=all=enabled`.

Você instalou e configurou o aplicativo de amostra do ObjectGrid e o aplicativo de amostra do ObjectGrid distribuído em um servidor do WebSphere Extended Deployment.

Depois de instalar o aplicativo em um servidor ou cluster, será possível acessar a documentação de amostra depois de iniciar o aplicativo no seguinte endereço da Web:

`http://`*`hostname`*`:`*`port`*`/ObjectGridSample/docs/introduction.html`

Por exemplo, se seu nome do host for **localhost** e o valor da porta for **9080**, utilize o endereço da Web `http://localhost:9080/ObjectGridSample/docs/introduction.html`.

Iniciando um Cluster do ObjectGrid de Amostra no Ambiente do WebSphere

Utilize esta tarefa para iniciar um cluster simples do ObjectGrid para testar a funcionalidade do ObjectGrid distribuído no ambiente do WebSphere Application Server.

O WebSphere Extended Deployment deve estar instalado. É necessário ter o arquivo ObjectGridSample.ear instalado no servidor de aplicativos. Para obter informações adicionais sobre como instalar o arquivo ObjectGridSample.ear, consulte Carregando e Executando o Aplicativo de Amostra do ObjectGrid com o WebSphere Extended Deployment.

Utilize esta tarefa para configurar um servidor de aplicativos para hospedar um servidor do ObjectGrid baseado nos arquivos cluster-config-1.xml e cluster-objectgrid-definition.xml.

O arquivo cluster-config-1.xml possui apenas uma definição do servidor do ObjectGrid. Este servidor do ObjectGrid representa o cluster do ObjectGrid de amostra. É possível utilizar um servidor de aplicativos independente ou um cluster com um membro de cluster para hospedar o servidor do ObjectGrid de amostra.

1. Extraia os arquivos META-INF/cluster-config-1.xml e META-INF/cluster-objectgrid-definition.xml do arquivo /optionalLibraries/ObjectGrid/objectgridSamples.jar para o diretório /optionalLibraries/ObjectGrid.
2. Defina os argumentos de JVM genéricos necessários.
   a. No console administrativo, clique em **Servidores > Application servers > *server_name* > Definição de Processo > Java Virtual Machine**.
   b. No painel Argumentos Genéricos de JVM, digite o seguinte texto:

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

      O `INSTALL_ROOT` é o diretório raiz de instalação do WebSphere Application Server.
   c. Clique em **Salvar**.
   d. Clique em **Salvar na Configuração Principal**.
   e. Selecione a opção **Sincronizar Alterações com Nós**. Clique em **Salvar**.
3. Copie o arquivo /optionalLibraries/ObjectGrid/objectgridSamples.jar para o diretório /classes ou o diretório `lib/ext`. O objectgridSamples.jar contém classes requeridas pelo servidor do ObjectGrid de amostra para implementações de plug-in definidas no arquivo cluster-objectgrid-definition.xml. Este arquivo JAR também é utilizado para serializar e desserializar os objetos que estão armazenados em mapas.
4. Reinicie o servidor para que as alterações entrem em vigor.

Para obter detalhes adicionais sobre como iniciar e parar servidores do ObjectGrid nos servidores de aplicativos, consulte Iniciando um Servidor do ObjectGrid em um Servidor de Aplicativos.

Iniciando um Servidor do ObjectGrid em um Servidor de Aplicativos

Um servidor do ObjectGrid pode ser configurado para iniciar em um servidor de aplicativos. O WebSphere Application Server detecta o componente ObjectGrid e inicia automaticamente o servidor do ObjectGrid.

É possível configurar servidores do ObjectGrid no WebSphere Application Server Versão 6.0.2 e posterior, incluindo quando complementos como WebSphere Extended Deployment ou WebSphere Business Integration Server estão instalados. Versões anteriores do WebSphere Application Server, como WebSphere Application Server Versão 5.0.2, podem ter aplicativos que utilizam o ObjectGrid como clientes, mas a função do servidor do ObjectGrid não pode ser colocada com versões anteriores do servidor de aplicativos.

Se estiver utilizando configurações do cluster que ativam a replicação, será requerido o gerenciador de alta disponibilidade. Os servidores do ObjectGrid utilizam o gerenciador de alta disponibilidade de forma diferente dos servidores de aplicativos normais. Quando o servidor do ObjectGrid está em um servidor de aplicativos, ele não configura, inicializa nem cria o serviço do gerenciador de alta disponibilidade, mas utiliza o serviço de alta disponibilidade existente no servidor

de aplicativos. Para replicação entre servidores do ObjectGrid, os servidores do ObjectGrid devem estar em execução em servidores de aplicativos que são membros do mesmo grupo principal.

Todas as demais funções do servidor do ObjectGrid são iguais quando o servidor é executado no WebSphere Application Server. Se a especificação de cluster do ObjectGrid incluir três servidores, os três servidores de aplicativos em um único grupo principal poderão hospedar estes servidores do ObjectGrid. Os servidores de aplicativos também podem estender clusters, desde que os clusters pertençam ao mesmo grupo principal. A etapa mais importante é correlacionar informações de nome do host e porta TCP/IP do servidor no arquivo cluster.xml.

Utilize esta tarefa para executar servidores do ObjectGrid nos servidores de aplicativos no ambiente do WebSphere Application Server.

1. Inclua as propriedades customizadas requeridas na JVM (Java Virtual Machine). No console administrativo, clique em **Servidores > Application Servers > *server_name* > Gerenciamento Java e de Processo > Definição de Processo > Java Virtual Machine > Propriedades Customizadas**. Clique em **Novo**. Crie as seguintes propriedades customizadas:

Tabela 1. Propriedades Customizadas da JVM para Servidores do ObjectGrid

Nome da Propriedade Customizada	**Descrição**	**Valor de Exemplo**
`objectgrid.server.name`	Especifica o nome do servidor do ObjectGrid a ser utilizado neste servidor de aplicativos. O nome fornecido deve ser um dos nomes de servidores definidos no arquivo XML do cluster do ObjectGrid.	server1
`objectgrid.xml.url`	Especifica a URL (Universal Resource Locator) para o arquivo XML do ObjectGrid. Essa propriedade é requerida.	`file:///d:/was/etc/test/objectGridMatch.xml`
`objectgrid.cluster.xml.url`	Especifica a URL para o arquivo XML do cluster do ObjectGrid. Esta propriedade é requerida	`file:///d:/was/etc/test/csCluster0.xml`

Tabela 1. Propriedades Customizadas da JVM para Servidores do ObjectGrid (continuação)

Nome da Propriedade Customizada	Descrição	Valor de Exemplo
`objectgrid.security. server.props`	Especifica a URL para o arquivo de propriedades de segurança do servidor do ObjectGrid. Esta propriedade será requerida apenas se a segurança estiver ativada no arquivo xml do cluster do ObjectGrid. Para determinar se a segurança está ativada em seu arquivo XML do cluster, procure o seguinte texto: `<cluster name="cluster1" securityEnabled="true" ....` Se o atributo securityEnabled estiver configurado como `false`, não será necessário definir esta propriedade. Utilize o arquivo `security.ogserver.props` como um gabarito. Consulte "Segurança do ObjectGrid" na página 139 para obter o significado destas propriedades neste arquivo e como elas podem ser utilizadas.	`file:///d:/was/ optionalLibraries/ ObjectGrid/properties/ security.ogserver.props`

Também é possível definir estas propriedades da JVM no campo **Argumentos Genéricos de JVM** no painel **Java Virtual Machine** no console administrativo. A seguir está um valor de exemplo para o campo Argumentos Genéricos de JVM:

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

2. Salve as alterações e reinicie o servidor de aplicativos. O WebSphere Application Server detecta o componente ObjectGrid e inicia automaticamente o servidor do ObjectGrid.

   O ObjectGrid no servidor de aplicativos utiliza a estrutura do canal para interagir com clientes do ObjectGrid, especificamente chamada a porta do Client Access. Quando o servidor do ObjectGrid é iniciado, ele detecta a colocação com o WebSphere Application Server e utiliza a estrutura do canal que já está em execução no servidor de aplicativos. O servidor do ObjectGrid cria e inicia sua própria estrutura de canal apenas se uma estrutura do canal não tiver sido criada ou iniciada no servidor de aplicativos.
3. Pare o servidor do ObjectGrid. Pare o servidor do ObjectGrid parando o servidor de aplicativos associado. Não é possível parar o servidor do ObjectGrid utilizando os comandos de gerenciamento de sistemas do ObjectGrid.

Os servidores de aplicativos em seu ambiente do WebSphere Application Server estão executando servidores do ObjectGrid.

Capítulo 2. ObjectGrid

ObjectGrid é uma estrutura de armazenamento em cache de objetos transacionais extensível para aplicativos J2SE (Java 2 Platform, Standard Edition) e J2EE (Java 2 Platform, Enterprise Edition).

É possível utilizar a API do ObjectGrid ao desenvolver seus aplicativos para recuperar, armazenar, excluir e atualizar objetos na estrutura do ObjectGrid. Você também pode implementar plug-ins customizados que monitoram atualizações para o cache, recuperar e armazenar dados com origens de dados externas, gerenciar a evidência de entradas do cache e manipular a funcionalidade de cache em segundo plano para seu próprio ambiente de aplicativos ObjectGrid.

API Baseada em Mapa

O ObjectGrid fornece uma API baseada na interface java.util.Map. A API foi estendida para suportar o agrupamento de operações em blocos transacionais. Esta interface é um superconjunto da interface java.util.Map e inclui suporte para operações em batch, invalidação, associação de palavra-chave e inserção e atualização explícitas. A semântica do Mapa Java é aprimorada com pontos de extensão para que seja possível implementar os seguintes aprimoramentos:

- Evictors de cache para ajustar existências de entradas de cache
- Interfaces de retorno de chamada de transação para controlar cuidadosamente o gerenciamento de transações e, opcionalmente, integrar-se com o gerenciador de transações do WebSphere em ambientes J2EE
- Implementações do loader que recuperam automaticamente e colocam dados para e de um banco de dados quando um programador de aplicativos utiliza operações get e put do Mapa do ObjectGrid
- As interfaces do listener que podem fornecer informações sobre todas as transações confirmadas conforme elas ocorrem e são aplicadas em toda a estrutura do ObjectGrid ou aplicadas para instâncias do Mapa específicas.
- Interfaces do transformador de objetos que permitem cópia e serialização mais eficientes de chaves e valores.

O Ambiente ObjectGrid

É possível utilizar a estrutura do ObjectGrid instalando uma das ofertas existentes:

- O ObjectGrid é integrado ao WebSphere Extended Deployment Versão 6.0.1 e faz parte da instalação completa.
- O ObjectGrid independente faz parte da instalação do MSE (Mixed Server Environment).

Nas duas ofertas, o ObjectGrid suporta recursos de cliente/servidor. O tempo de execução do servidor suporta armazenamento em cluster completo, replicação e particionamento de caches de objetos distribuídos. O tempo de execução do cliente suporta o conceito de um near cache e da lógica de roteamento de gerenciamento de carga de trabalho para clusters remotos. O tempo de execução do cliente também suporta a criação de mapas de objetos locais.

O nível de suporte varia, dependendo de você estar executando o tempo de execução do cliente, o tempo de execução do servidor, o ObjectGrid integrado ou o ObjectGrid independente.

ObjectGrid Integrado à Oferta do WebSphere Extended Deployment

Tempo de Execução do Servidor: O tempo de execução do servidor é integrado. Para o WebSphere Extended Deployment Versão 6.0.1, o tempo de execução integrado não é suportado na plataforma z/OS.

Tempo de execução do cliente: O tempo de execução do cliente é suportado no J2SE e no J2EE no nível de JDK 1.3.1 e superior, incluindo o WebSphere Application Server Versão 5.0.2 e posterior. O tempo de execução do cliente é totalmente suportado na plataforma z/OS.

Oferta de ObjectGrid Independente

Tempo de execução do servidor: O tempo de execução do servidor pode ser executado em JVMs (Java Virtual Machines) independentes como um único servidor ou como um cluster de servidores. O servidor independente é suportado na maioria das plataformas J2SE e J2EE no nível de JDK 1.4.2 e superior. O servidor independente é suportado no WebSphere Application Server Versão 6.0.2 e posterior. O tempo de execução do servidor independente não é suportado na plataforma z/OS para WebSphere Extended Deployment Versão 6.0.1.

Tempo de execução do cliente: O tempo de execução do cliente é suportado nas plataformas J2SE e J2EE no nível de JDK 1.3.1 e superior, incluindo o WebSphere Application Server Versão 5.0.2 e posterior.

Gerenciamento de sessão

É fornecida uma implementação de gerenciamento de Sessões HTTP totalmente distribuída que armazena objetos de Sessão HTTP no ObjectGrid.

Instalação Simples

Você pode instalar e configurar o ObjectGrid em algumas etapas simples. Estas etapas incluem a cópia de arquivos JAR (Java Archive) para seu caminho de classe e a definição de novas diretivas de configuração.

Alterações Transacionais

Todas as alterações são feitas no contexto de uma transação para garantir uma interface programática robusta. A transação pode ser explicitamente controlada no aplicativo ou o aplicativo pode utilizar o modo de programação de confirmação automática. Estas alterações transacionais podem ser replicadas em um cluster do ObjectGrid nos modos assíncrono e síncrono para fornecer acesso escalável e tolerante a falhas.

É possível escalar o ObjectGrid de uma grade simples em execução em uma única JVM para uma grade que envolve um ou mais clusters do ObjectGrid de Java Virtual Machines. Estes servidores disponibilizam dados por meio de APIs do Mapa para um grande conjunto de clientes ativados por ObjectGrid. Os clientes do ObjectGrid utilizam as APIs de Mapa Java básicas. No entanto, o desenvolvedor de aplicativos não precisa desenvolver APIs Java TCP/IP e RMI (Chamada de Método Remoto), porque o cliente do ObjectGrid pode acessar os outros servidores do ObjectGrid que estão retendo informações na rede. Se seu conjunto de dados for muito grande para uma única JVM, será possível utilizar o ObjectGrid para particionar os dados.

O ObjectGrid também oferece recursos de alta disponibilidade incluídos da solução de aplicativo. O compartilhamento de objetos é baseado em um modelo de

replicação no qual existem um servidor principal, um ou mais servidores de replicação e um ou mais servidores de espera. Este cluster de servidores de replicação é referido como um grupo de replicação. Se o acesso ao grupo de replicação for uma operação de gravação, o pedido será roteado para o servidor principal. Se o acesso for uma operação de leitura ou se o mapa for um mapa de leitura, o pedido poderá ser roteado para os servidores principal ou de replicação. Os servidores de espera serão definidos como possíveis servidores de replicação se um servidor falhar. Se um servidor principal falhar, o servidor de replicação se tornará o servidor principal para minimizar qualquer interrupção. Este comportamento é configurável e extensível com base em suas necessidades.

Se desejar utilizar uma abordagem de propagação de objetos mais simples, também está disponível um modelo no mesmo nível de qualidade de serviço inferior, como estava no Extended Deployment Versão 6.0. Com este suporte transacional distribuído mais simples, os pares podem ser notificados de alterações utilizando um transporte de mensagens. O transporte de mensagens será construído se você estiver executando o WebSphere Application Server Versão 6.0.2 ou posterior. Se não estiver executando o WebSphere Application Server Versão 6.0.2 ou posterior, outro transporte de mensagens deverá ser fornecido, como um provedor JMS (Java Message Service).

APIs Compatíveis com Contêiner de Injeção

Configure o ObjectGrid utilizando um arquivo XML simples ou programaticamente utilizando APIs Java. As APIs Java foram projetadas para funcionar também em ambientes nos quais você está utilizando estruturas baseadas em injeção para configurar seus aplicativos. As APIs e interfaces dos objetos do ObjectGrid também podem ser chamadas por um contêiner IoC (Inversion of Control) e, em seguida, as referências a objetos-chave do ObjectGrid poderão ser injetadas no aplicativo.

Arquitetura Extensível

Você pode estender a maioria dos elementos da estrutura de ObjectGrid desenvolvendo plug-ins. É possível ajustar o ObjectGrid para permitir que um aplicativo tome decisões equilibradas entre consistência e desempenho. O código customizado pelo plug-in também pode suportar os seguintes comportamentos específicos de aplicativo:

- Atender a eventos de instância do ObjectGrid para inicialização, início de transação, término de transação e destruição.
- Chamar retornos de chamada de transações para ativar o processamento de transações específicas.
- Implementar políticas específicas de transações comuns com transações do ObjectGrid genéricas.
- Utilizar loaders para pontos de entrada e saída transparentes e comuns para data stores externos e outros repositórios de informações.
- Manipular objetos não seriáveis de uma maneira específica com interfaces ObjectTransformer.

Você pode implementar cada um desses comportamentos sem afetar o uso das interfaces básicas da API do cache de ObjectGrid. Com esta transparência, os aplicativos que estão utilizando a infra-estrutura de cache podem ter data stores e processamento de transações que são alterados sem afetar esses aplicativos.

Utilizar o ObjectGrid como uma API Primária ou Cache de Segundo Nível

As APIs do ObjectGrid podem ser utilizadas diretamente pelo aplicativo como um cache de repositório associado ou como um cache de gravação direta. No modo de gravação direta, o aplicativo conecta um objeto Loader para que o ObjectGrid possa aplicar alterações e buscar dados de forma direta e transparente para o aplicativo. O ObjectGrid também pode ser utilizado como um cache de segundo nível para mapeadores relacionais de objetos populares gravando um adaptador. O cache é invisível para o aplicativo neste modelo, porque o aplicativo utiliza as APIs do mapeador relacional de objetos como a API primária para acessar os dados.

Capítulo 3. Visão Geral do ObjectGrid

O ObjectGrid fornece um modelo de aceso a dados baseado em Mapa Java e uma tecnologia de armazenamento em cache distribuído. Com o ObjectGrid, é possível configurar um ambiente de armazenamento em cluster altamente disponível. Os clientes do ObjectGrid podem contactar muitos clusters diferentes do ObjectGrid simultaneamente para soluções de integração de grande escala. O ObjectGrid também fornece uma solução de particionamento de dados aprimorada, distribuída para grandes quantidades de informações padronizadas com dados em mais de uma Java Virtual Machine. Basicamente, o ObjectGrid é um conjunto de APIs Java padronizadas e de serviços de rede que permitem o armazenamento em cache local e distribuído. As escalas de solução de uma única JVM (Java Virtual Machine) na qual é requerida uma solução de Mapa Java mais aprimorada para uma ampla matriz de serviços de dados distribuídos e escaláveis são requeridas de vários clusters do ObjectGrid em toda uma corporação.

ObjectGrid em uma Única JVM (Java Virtual Machine)

O uso mais básico do ObjectGrid é em uma única JVM.

É possível utilizar o ObjectGrid para criar um conjunto de instâncias do ObjectGrid. Cada instância do ObjectGrid pode conter uma ou mais instâncias compatíveis com Mapa Java. As instâncias do Mapa Java fornecem as interfaces get e put com as quais os programadores Java estão acostumados, mais recursos adicionais que não são oferecidos pela interface e funcionalidade atuais do Mapa Java. O diagrama a seguir ilustra o uso mais básico do ObjectGrid.

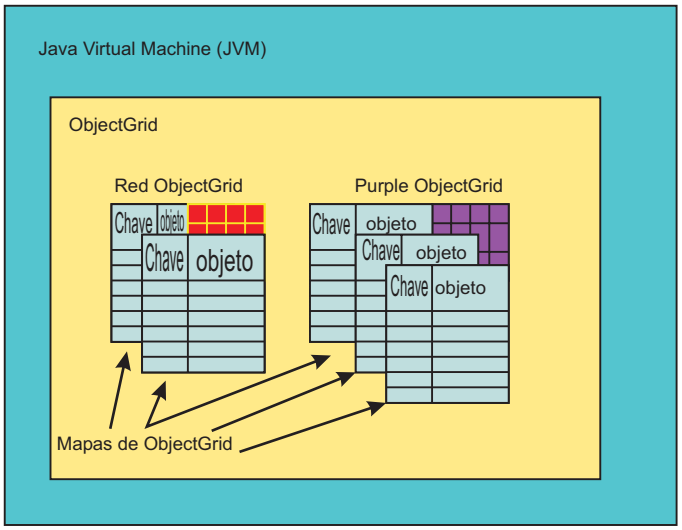


Figura 1. Uso de JVM do ObjectGrid

Um ObjectGrid e o Mapa do ObjectGrid incluem muitos recursos que não são fornecidos na interface do Mapa Java padrão. Estes recursos incluem acesso transacional, vários tipos de estratégias de bloqueio (Nenhum, Otimista e Pessimista), Eviction Plug and Play, interação total com bancos de dados como um efeito secundário da utilização de APIs get e put e muitos outros recursos. Também é possível desenvolver suas próprias extensões para o ObjectGrid. Por exemplo, é possível desenvolver um Listener do Mapa que fornece resultados para cada transação confirmada em uma determinada instância do Mapa. Os usuários podem

registrar as alterações, por exemplo, em um arquivo no local de um escritório filial para garantia contra perda de transações ou propagar as alterações com JMS (Java Message Service) ou alguma outra infra-estrutura.

No diagrama anterior, a JVM possui duas instâncias do ObjectGrid, uma com dois objetos semelhantes ao Mapa Java para utilização e a outra com três objetos do Mapa. Os objetos do Mapa são bidimensionais, permitindo que um par de chave e objeto seja manipulado como um Mapa Java normal. Uma única instância do ObjectGrid pode suportar muitas instâncias do mapa específicas.

A configuração a seguir é uma configuração do ObjectGrid básica para as instâncias do ObjectGrid Red e Purple:

```
<?xml version="1.0" encoding="UTF-8" ?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="Red">
<backingMap name="FirstRedMap" readOnly="false" />
<backingMap name="SecondRedMap readOnly="false" />
</objetGrid>
<objectGrid name="Purple">
<backingMap name="FirstPurpleMap readOnly="false" />
<backingMap name="SecondPurpleMap readOnly="false" />
<backingMap name="ThirdPurpleMap readOnly="false" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

ObjectGrid Distribuído

Além de utilizar o arquivo JAR (Java Archive) do ObjectGrid em uma única JVM, é possível utilizar o ObjectGrid em um ambiente distribuído. Neste ambiente, é possível criar um cluster do ObjectGrid. Um cluster do ObjectGrid é composto de um conjunto de servidores do ObjectGrid, cada um em sua própria JVM única.

O tópico “ObjectGrid em uma Única JVM (Java Virtual Machine)” na página 19 descreve que o ObjectGrid suporta o conceito de um Mapa Java. Este conceito também é suportado localmente em uma única JVM e em um cliente Java que conecta-se a um ou mais clusters de cache remoto do ObjectGrid. Os servidores do ObjectGrid permitem distribuir a funcionalidade básica já descrita acima no caso de JVM única. Por exemplo, vários clientes podem compartilhar o mesmo Mapa de instância do ObjectGrid, utilizando uma estratégia de bloqueio de Nenhum, Otimista e Pessimista. Além disso, um evictor nos servidores de cluster do ObjectGrid pode gerenciar a eviction para os dados de instância do Mapa do lado do servidor. Todos os clientes podem utilizar a semântica comum de get e put e o Loader configurado no servidor de cluster do ObjectGrid faz toda a interação com o banco de dados em vez de implementar e gerenciar drivers JDBC (Java Database Connectivity) em cada cliente.

No diagrama a seguir, a JVM possui duas instâncias de ObjectGrid: uma com dois objetos semelhantes ao Mapa Java para utilização e a outra com três objetos semelhantes ao Mapa Java. Cada um dos Mapas representa objetos bidimensionais que permitem uma chave e um objeto. Uma única instância do ObjectGrid pode suportar um grande número de Mapas, principalmente dependendo dos requisitos do aplicativo. Neste caso, a diferença é que os Mapas são hospedados em um servidor de cluster do ObjectGrid. Os clientes podem ser

um aplicativo Java normal ou servidores de aplicativos J2EE (Java 2 Platform, Enterprise Edition).

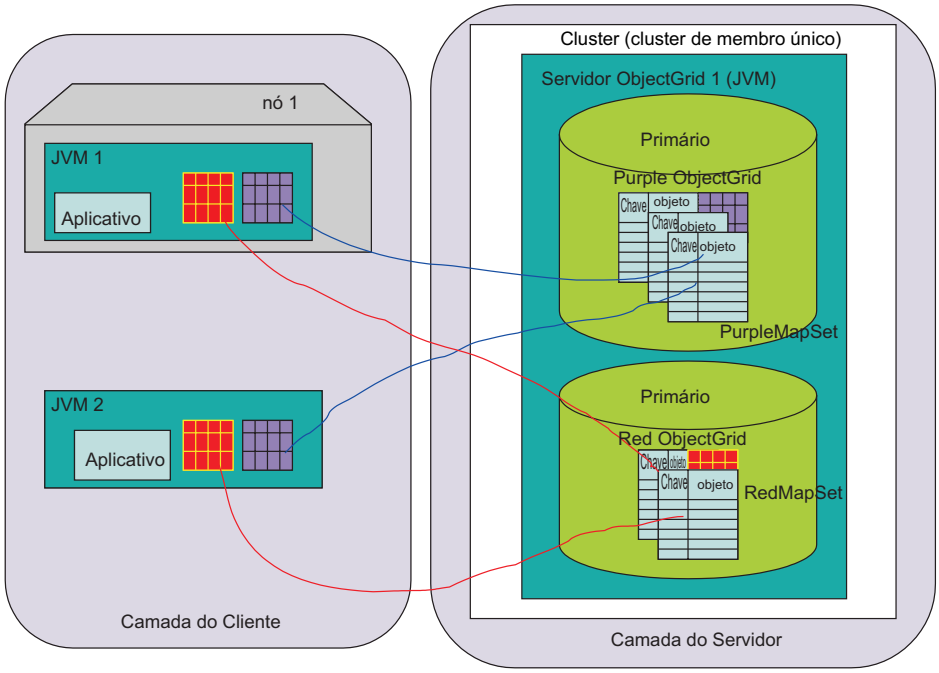


Figura 2. Topologia de Único Servidor do ObjectGrid Distribuído (com dois MapSets)

Clientes do ObjectGrid

Os clientes do ObjectGrid consistem em um conjunto de APIs para conexão com um cluster do ObjectGrid, auto-inicialização por meio da configuração de todo o cluster do ObjectGrid e, em seguida, desempenho de operações do mapa do ObjectGrid que são realmente distribuídas. Um cliente do ObjectGrid é qualquer aplicativo Java em sua própria instância de JVM que esteja utilizando o ObjectGrid de maneira distribuída. Um cliente do ObjectGrid distribuído também ainda pode utilizar a funcionalidade não distribuída na mesma Java Virtual Machine. O uso de um cliente do ObjectGrid pode ser tão complicado como todo um servidor de aplicativos com várias conexões paralelas do ObjectGrid, cada uma com a segurança ativada e agindo em nome de um usuário diferente.

Para ativar o comportamento distribuído, deve ser criado um cluster do ObjectGrid (serviços do lado do servidor da solução do ObjectGrid). A configuração adicional requerida é um arquivo XML do Cluster do ObjectGrid, além do arquivo de configuração do ObjectGrid.

A seguir está o XML do Cluster do ObjectGrid que configura a implementação de rede do ObjectGrid no diagrama anterior:

```
<?xml version="1.0" encoding="UTF-8" ?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!– single server ->
<serverDefinition name="server1" host="localhost" clientAccessPort="12053"
```

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

Esta configuração descreve um cluster único, ″cluster1″, que contém o servidor server1. O servidor server1 hospeda dois ObjectGrids, ″Red″ e ″Purple″. O arquivo de configuração especifica informações também para particionamento e replicação. O suporte ao servidor do cliente do ObjectGrid no ObjectGrid requer que o programador conecte-se a um servidor definido no cluster do ObjectGrid. Durante o processamento da conexão, a configuração do ObjectGrid e do cluster do ObjectGrid é dinamicamente transferida por download para o cliente, tornando muito simples a preparação do cliente para uso e tendo que gerenciar o conteúdo da configuração do lado cliente. Diferente do cliente do ObjectGrid que desempenha uma operação ″Conectar″, as APIs e conceitos de programação para utilizar um ObjectGrid com escopo definido para a JVM local e uma JVM que de fato esteja hospedada no cluster do ObjectGrid em geral são iguais.

Inicialização do Cluster do ObjectGrid

É possível iniciar servidores do ObjectGrid em um cluster com as ferramentas de linha de comandos fornecidas com o ObjectGrid. Um aplicativo do ObjectGrid pode incluir um cliente do ObjectGrid e ser integrado como qualquer outra Biblioteca da API Java seria integrada em sua estrutura de desenvolvimento de aplicativos. No entanto, nos dois casos, o uso do ObjectGrid deve ser inicializado.

Para funcionar no cenário de uso da JVM (Java Virtual Machine) local ou em um cluster do ObjectGrid distribuído, é necessário obter uma configuração válida para auto-inicialização por meio de uma abordagem gerenciável. Os clientes do ObjectGrid e os servidores do cluster do ObjectGrid devem utilizar uma configuração uniforme. Como programador, você pode iniciar com uma configuração muito simples, possivelmente limitada em um único aplicativo Java em uma JVM.

Em seguida, conforme você prepara-se para iniciar o teste de usuário simultâneo com vários clientes, crie seu primeiro cluster do ObjectGrid de único servidor. Após a conclusão do teste baseado em servidor de cliente inicial, é possível trabalhar com a equipe administrativa e experimentar a replicação e outros requisitos de solução de alta disponibilidade. Cada uma destas progressões normais em requisitos de desenvolvimento requer um arquivo de configuração mais aprimorado.

As alterações no arquivo de configuração para ativar cada um destes recursos avançados para cada um dos estágios de desenvolvimento descritos são relativamente modestas, mas cada estágio em seu desenvolvimento de solução requer uma versão diferente do arquivo de configuração. A intenção é que as alterações sejam baseadas umas nas outras. Você pode fazer o teste de unidade de uma solução replicada em uma única máquina se a quantidade de dados para desenvolver a solução não exceder um único sistema ou se ficar artificialmente restrita para fins de desenvolvimento.

Configuração do ObjectGrid com XML

Uma configuração do ObjectGrid distribuído, com um ou mais clientes e um ou mais servidores do ObjectGrid, requer a configuração XML. Além do arquivo de configuração XML base do ObjectGrid, é necessário criar uma descrição XML do cluster do ObjectGrid.

Uma única descrição de configuração XML do ObjectGrid e uma descrição da configuração XML do Cluster do ObjectGrid fornecem aos clientes e servidores em um único cluster do ObjectGrid as informações necessárias para que eles funcionem conforme o esperado. É possível ter qualquer quantidade de clusters do ObjectGrid em seu ambiente, no entanto, um documento XML do cluster do ObjectGrid específico de cluster deve descrever o cluster específico.

Os arquivos de configuração requeridos para que o ObjectGrid seja iniciado podem ser adquiridos por meio de qualquer abordagem de URL normal. Por exemplo, os clientes e servidores podem adquirir os arquivos XML com um arquivo físico ou uma URL HTTP.

Em um ambiente do ObjectGrid distribuído, conforme descrito nos diagramas a seguir, um conjunto inicial de servidores do ObjectGrid pode ser configurado com a linha de comandos para recuperar sua configuração por meio de uma URL ou, como a URL do arquivo pode ser complicada, um arquivo simples no sistema de arquivos. No entanto, uma melhor abordagem é iniciar os servidores seguintes no mesmo cluster do ObjectGrid, auto-inicializando-os a partir de outros servidores que já estejam operacionais no cluster. Esta abordagem é muito mais gerenciável, porque os administradores não precisam rastrear arquivos de configuração em cada máquina que esteja hospedando um cliente ou servidor do ObjectGrid. Além disso, um servidor que inicia por meio de auto-inicialização pode ter a garantia de que o XML já foi processado com êxito, reduzindo erros de configuração de XML.

Auto-inicialização

Auto-inicialização do Servidor do ObjectGrid

O diagrama a seguir descreve a auto-inicialização de um ambiente em cluster típico do ObjectGrid que hospeda a mesma configuração do ObjectGrid, mas oferece uma configuração de cluster de replicação mais aperfeiçoada. Neste caso, o primeiro servidor é auto-inicializado por meio de uma URL HTTP e o segundo e o terceiro servidores são iniciados a partir do primeiro. O segundo e o terceiro servidores também podem ser iniciados a partir da mesma URL que o primeiro servidor.

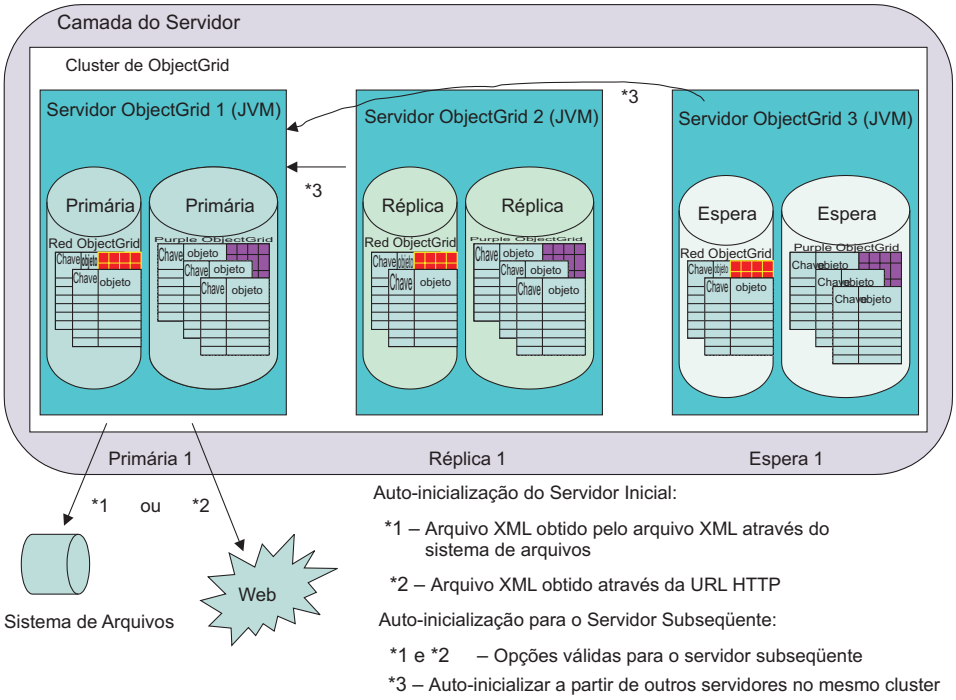


Figura 3. Auto-inicialização do Servidor Inicial por meio da Configuração do Arquivo XML ou a partir de um Servidor Existente

Conforme ilustrado no diagrama anterior, o servidor server1 no cluster cluster1 é o servidor inicial a ser auto-inicializado. O servidor server1 pode ser auto-inicializado por meio de um arquivo XML no sistema de arquivo ou por meio de uma URL para um arquivo local, servidor HTTP remoto ou outra opção de URL válida. O servidor server2 e o servidor server3 podem ser iniciados por estes meios ou direcionando o servidor server1 como um host de auto-inicialização de configuração. Em geral, a auto-inicialização de servidores subseqüentes a partir de outros servidores assegura que a configuração seja consistente nos membros de cluster.

Neste cenário específico, se o servidor server1 falhar, e server2 e server3 ainda estiverem operacionais, server1 poderá ser auto-inicializado a partir de server2 ou de server3, ou novamente por meio de abordagens de arquivo ou de URL. Consulte “APIs Connect do Cliente do ObjectGrid” na página 101 para obter detalhes adicionais sobre auto-inicialização e as opções de configuração específicas.

Auto-inicialização do Cliente do ObjectGrid

O cliente do ObjectGrid, para utilizar serviços de membros do servidor de cluster do ObjectGrid, devem ser auto-inicializados a partir de um dos servidores de ObjectGrid no cluster. Cada cliente pode se ″conectar″ a qualquer membro ativo do cluster. Os administradores podem configurar servidores específicos para desempenhar esse serviço. Para grandes implementações do cliente, a única finalidade de qualquer um dos servidores de cluster do ObjectGrid configurados é fornecer suporte à auto-inicialização do cliente. Esta abordagem será útil se o número de clientes for grande e eles se conectam e se desconectam freqüentemente. Depois que o cliente se ″conectar″, ele poderá obter uma referência distribuída aos ObjectGrids definidos na configuração de cluster. Consulte o “Interface ObjectGridManager” na página 93 para obter informações adicionais.

Os clientes adquirem sua configuração padrão do cluster do ObjectGrid para que o administrador não precise gerenciar o XML para a comunidade do cliente. O cliente do ObjectGrid pode utilizar uma URL remota assim como os servidores de cluster do ObjectGrid para poderem substituir configurações específicas que devem ser específicas do cliente.

Clientes do ObjectGrid em um Ambiente do ObjectGrid Distribuído

Os clientes do ObjectGrid podem conectar-se a mais de um cluster do ObjectGrid simultaneamente. Um único aplicativo Java em uma JVM (Java Virtual Machine) pode conectar-se ao mesmo cluster remoto várias vezes. Este aplicativo também pode conectar-se a diferentes clusters remotos ao mesmo tempo. Este recurso é importante porque permite que a funcionalidade do cliente acesse muitos recursos diferentes de informações que são exportadas por meio de um ou mais clusters do ObjectGrid.

O primeiro caso, no qual o mesmo cliente do ObjectGrid pode contactar o mesmo servidor do ObjectGrid é importante para ambientes seguros nos quais o cliente pode ser um servidor de aplicativos e cada conexão do servidor de aplicativos para o cluster do ObjectGrid remoto utiliza diferentes credenciais de segurança. Outro exemplo é um cliente do ObjectGrid que precisa correlacionar dados de vários clusters diferentes do ObjectGrid para uma única finalidade.

O diagrama a seguir descreve um cenário no qual o usuário cliente corporativo baseado na Web, por meio de um aplicativo da Web, está gerando um relatório a partir de três diferentes divisões corporativas. O mecanismo de servlet utiliza a funcionalidade do cliente do ObjectGrid do servidor de aplicativos para contactar três diferentes clusters do ObjectGrid, gerenciados por cada divisão corporativa. Em muitas corporações, os dados podem ser coletados e uma metachave do ObjectGrid é disponibilizar as informações de uma maneira fácil. Quando as informações forem externalizadas, outros usuários que têm o interesse e credenciais de segurança podem adquirir e utilizar as informações de novas maneiras. Os dados podem ser fornecidos em um modo de leitura ou, quando apropriado, para cenários de atualização de leitura/gravação.

Neste cenário, os dados podem ser adquiridos de maneira segura. O armazenamento em cache do ObjectGrid neste cenário não é apenas permitir o compartilhamento de dados flexível de maneira programática comum em cada divisão corporativa, mas também permitir o acesso a dados nas divisões para informações adquiridas por meio de um modelo de programação muito seguro, simples e claro utilizado freqüentemente por muitos desenvolvedores Java.

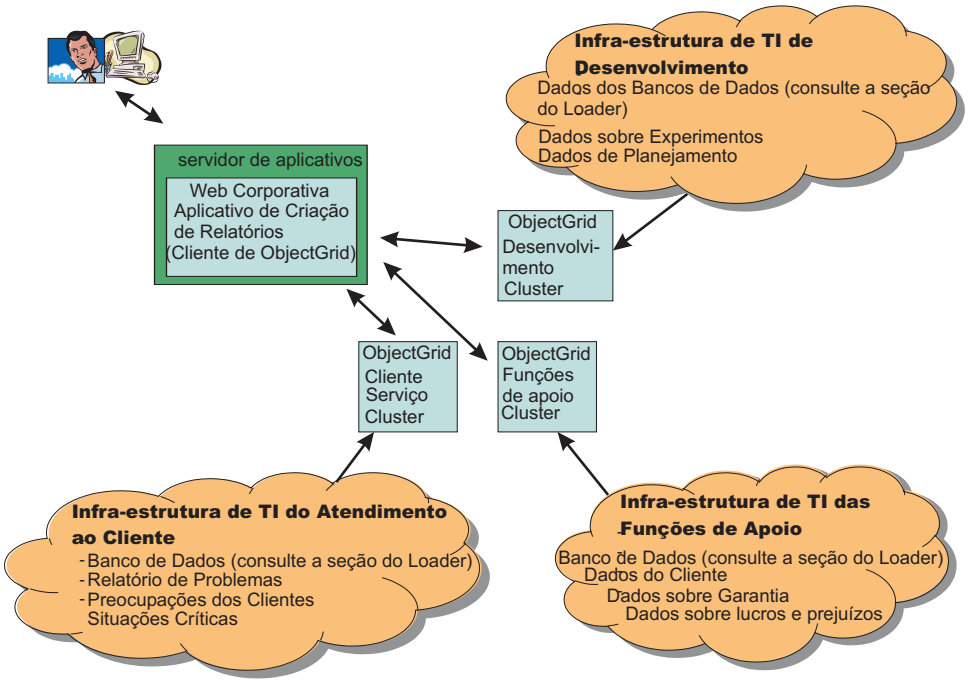


Figura 4. Um Usuário Cliente Baseado na Web Gera um Relatório a partir de Três Diferentes Divisões Corporativas.

Conceitos de Armazenamento em Cluster do ObjectGrid

O termo *ObjectGrid distribuído* inclui o conceito de que os clientes podem interagir com um ou mais clusters do ObjectGrid. Um cluster do ObjectGrid consiste em um ou muitos servidores do ObjectGrid.

Cliente do ObjectGrid

Um cliente do ObjectGrid pode ser imaginado de duas maneiras. Você pode imaginar um cliente como uma JVM (Java Virtual Machine) que utiliza a API do ObjectGrid para conectar-se a um cluster do ObjectGrid e desempenhar operações do Mapa Java nesse cluster. A segunda, uma maneira mais formal de imaginar um cliente é considerar o conceito de vários clientes na mesma JVM. Se você utilizar totalmente a função do ObjectGrid fornecida, poderá utilizar vários clientes na mesma JVM.

Sempre que um programador executar a operação de conexão do cliente do ObjectGrid em uma JVM, será retornado um contexto do cluster. Este contexto é de fato uma instância do cliente. Implicitamente, os encadeamentos assíncronos manipulam muitos aspectos do armazenamento em cache por contexto. Para cada contexto, o ObjectGridManager pode ser utilizado para adquirir ObjectGrids que estão hospedados nas instâncias do cluster do ObjectGrid remoto específico. Portanto, em geral, se conectar-se a três clusters remotos na mesma JVM, você implementa uma solução de três clientes na mesma JVM.

Considerações importantes para este cenário são as seguintes. Uma única sessão de transação não pode estender um conjunto de Mapas no mesmo cluster. Os usuários não podem ter uma única transação em clientes diferentes conectados ao mesmo cluster do ObjectGrid ou a um diferente. No entanto, para usuários que

tentam integrar silos de informações, os usuários podem utilizar uma transação para obter informações de cada um dos clusters remotos do ObjectGrid e para imprimir relatórios de consolidação ou juntar as informações e enviar por meio de dados de transação de um ObjectGrid para outro cluster do ObjectGrid, ou apenas atualizar os clusters individuais do ObjectGrid de maneira específica do cliente. Isto ocorre principalmente porque o ObjectGrid oferece suporte à transação de única fase, ao contrário do suporte à transação de duas fases, geralmente oferecido por gerenciadores de transações separados. Para obter informações adicionais sobre este tópico, consulte “Demarcação de Transação do ObjectGrid” na página 39.

Replicação

Você pode replicar entre servidores do ObjectGrid que estão no mesmo cluster do ObjectGrid. Com a replicação, é possível recuperar-se de uma falha mais rapidamente quando o servidor principal do ObjectGrid que tem as informações específicas requeridas pelo usuário falhar ou for encerrado para manutenção. No diagrama a seguir, o Red ObjectGrid e o Purple ObjectGrid estão em dois membros do grupo de replicação do ObjectGrid diferentes. No ObjectGrid, cada MapSet, um subconjunto de um ObjectGrid pode ser replicado como uma unidade. PartitionSets são uma exceção a esta regra, conforme discutido na seção seguinte. A única configuração do servidor descrita é modificada no diagrama a seguir para descrever a replicação.

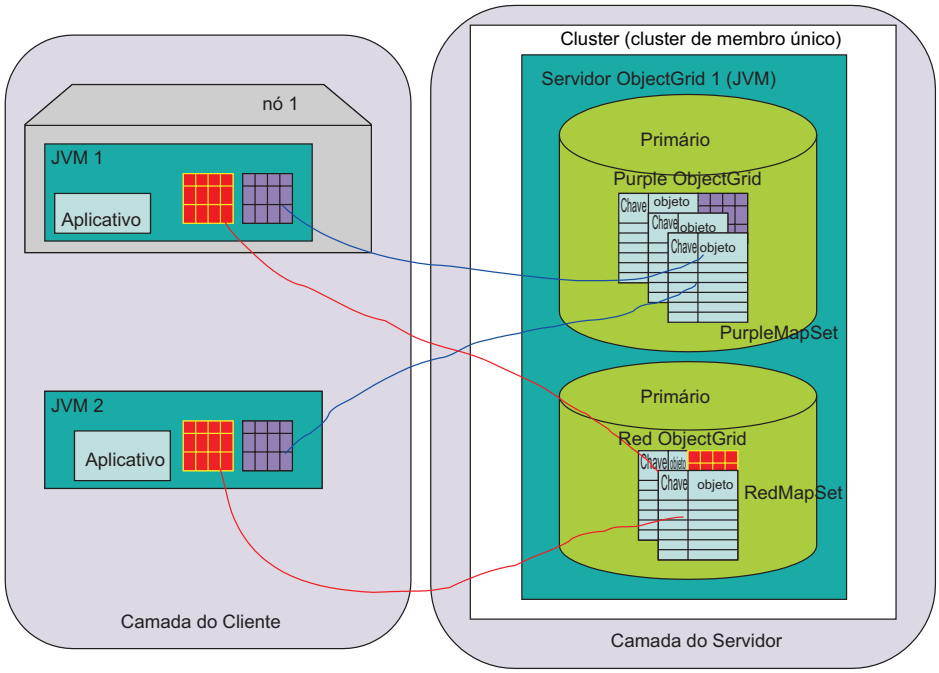


Figura 5. Topologia de Único Servidor do ObjectGrid Distribuído com Dois MapSets

O diagrama descreve um servidor de aplicativos como um aplicativo cliente e um aplicativo Java independente. Os dois clientes requerem acesso a duas instâncias do ObjectGrid, as instâncias Red e Purple em um cluster do ObjectGrid de único servidor. Cada uma destas instâncias de fato está contida em um membro do grupo de replicação. Um membro do grupo de replicação é um conceito-chave e é o limite para a demarcação de transação do ObjectGrid. Uma transação pode confirmar alterações apenas para um único membro do grupo de replicação.

Em um cluster do ObjectGrid, o cliente Java pode iniciar uma transação do ObjectGrid (uma Sessão) e atualizar dados em um único membro do grupo de replicação. Cada membro do grupo de replicação pode ser replicado como uma unidade, de maneira síncrona ou assíncrona, ou nem tudo dependendo de seus requisitos. Cada pedido de cliente do ObjectGrid é roteado para um membro do grupo de replicação específico nos servidores do Cluster do ObjectGrid. A instância do ObjectGrid no membro do grupo de replicação que está recebendo os pedidos processa o pedido e retorna um resultado ao cliente. Para replicação *síncrona*, cada pedido, antes de retornar ao cliente, é enviado à réplica ou servidor 2 do ObjectGrid no diagrama a seguir, para confirmar que o membro do grupo de replicação da réplica aplicou corretamente a atualização e, em seguida, retorna o resultado ao cliente. No modo *assíncrono*, o cliente do ObjectGrid pode aplicar uma alteração e o membro do grupo de replicação primário de servidores do ObjectGrid retorna o resultado ao cliente e não espera que a réplica confirme se as alterações foram recebidas e aplicadas corretamente. No modo assíncrono, a atualização será enviada ao membro do grupo de replicação de réplica do servidor remoto após a transação ter sido confirmada com êxito no membro do grupo de replicação primário.

O diagrama a seguir é uma versão diferente do exemplo de auto-inicialização. Neste caso, três servidores, cada um tendo uma função exclusiva na replicação das duas instâncias do ObjectGrid com as quais os usuários esperam interagir. O Cluster do ObjectGrid é formado por três servidores, cada um hospedando dois membros do grupo de replicação. O servidor server1 hospeda dois primários, o servidor server2 hospeda duas réplicas e o servidor server3 hospeda dois em espera.

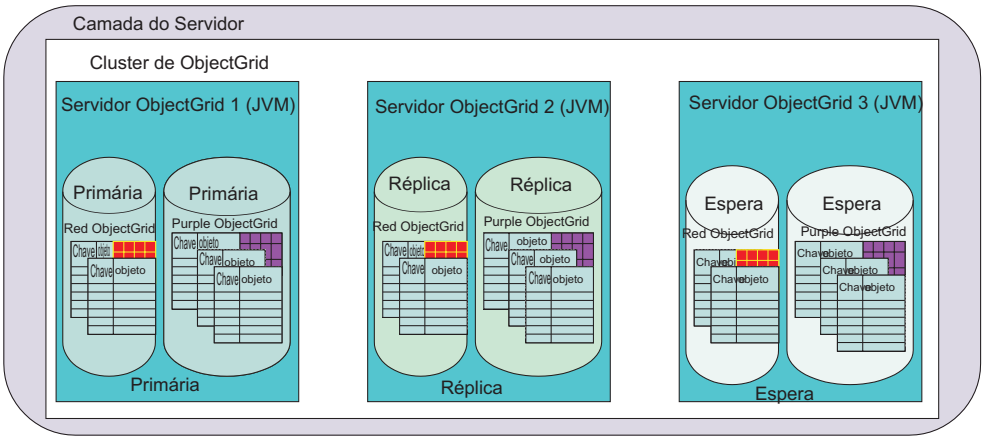


Figura 6. Replicando a Configuração de Amostra Básica

No entanto, a “Visão Geral de Alta Disponibilidade” na página 30 descreve estes conceitos, um conceito-chave a ser entendido são as diferenças de configuração requeridas para mover-se do servidor único para a solução de três servidores replicados descrita no diagrama anterior.

Visão Geral de Configuração de Replicação de Vários Servidores

A configuração a seguir é uma configuração do ObjectGrid básica para as instâncias de ObjectGrid Red e Purple.

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

```
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="Red">
<backingMap name="FirstRedMap" readOnly="false" />
<backingMap name="SecondRedMap" readOnly="false" />
</objectGrid>
<objectGrid name="Purple">
<backingMap name="FirstPurpleMap" readOnly="false" />
<backingMap name="SecondPurpleMap" readOnly="false" />
<backingMap name="ThirdPurpleMap" readOnly="false" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

A conversão desta configuração em um cluster do ObjectGrid distribuído requer um arquivo de configuração adicional, o arquivo XML do Cluster. Para converter a configuração original para as instâncias de Objetos Red e Purple em um único servidor são requeridas apenas as inclusões exibidas no exemplo a seguir. Especificamente, foram incluídas apenas duas referências do servidor. O grupo de replicação já estava presente no arquivo de configuração inicial, que fez referência cruzada ao grupo de replicação ColorMapsReplicationGroup, conforme ilustrado na amostra a seguir:

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

No exemplo anterior, os dois MapSets (descritos abaixo) referem-se ao ColorMapsReplicationGroup ReplicationGroup, que define os servidores a serem incluídos no grupo de replicação. O arquivo de configuração pode ter sido expandido para incluir outro ReplicationGroup, com cada um dos MapSets tendo os mesmos servidores em diferentes ordens ou diferentes servidores para atender os

requisitos dos clientes. A configuração de cluster do ObjectGrid suporta a reutilização de sub-rotinas. Por padrão, como os atributos de replicação do MapSet não estão configurados e o grupo de replicação tem mais de um servidor, a replicação será ativada e o modo será assíncrono.

Visão Geral de Alta Disponibilidade

A replicação permite alta disponibilidade em um cluster do ObjectGrid.

Para entender a replicação e alta disponibilidade, é necessário entender os tipos de membros do grupo de replicação do ObjectGrid. Os tipos de membros do grupo de replicação suportados pelo ObjectGrid incluem *primário*, *réplica* e *espera*. Cada um destes tipos tem uma função específica em configurações de alta disponibilidade.

Tipos de Membros do Grupo de Replicação do ObjectGrid

Membro do Grupo de Replicação Primário
O membro do grupo de replicação primário contém a visualização de dados mais recente do cliente em utilização. Conforme os dados são atualizados, eles são propagados para réplicas. O primário é a instância que se comunica com qualquer banco de dados de conexão por meio da interface do Loader do ObjectGrid, propaga confirmações de maneira síncrona, assíncrona ou nem tudo depende da configuração de replicação.

Membro do Grupo de Replicação de Réplica
Um membro do grupo de replicação de réplica contém uma versão dos dados que foram propagados do primário. O primário pode ser configurado para enviar as alterações de várias maneiras. O grupo de replicação deve ter pelo menos dois servidores listados para ter um primário e uma réplica; caso contrário, a replicação não será ativada.

Membro do Grupo de Replicação de Espera
Um membro do grupo de replicação de espera não recebe atualizações conforme são feitas alterações no primário da mesma forma que uma réplica. Ele apenas está configurado e pronto para receber atualizações se o primário ou réplica falhar. Se o primário falhar, a réplica se tornará o novo primário e a espera precisará ser convertida em uma réplica.

Cenário de Alta Disponibilidade

Em geral, a replicação permite alta disponibilidade em um cluster do ObjectGrid. As duas ilustrações a seguir descrevem cenários de falha de primário e recuperação. Se um membro do grupo de replicação primário for replicado, e ocorrer uma falha, uma das réplicas será escolhida para se tornar o novo primário. Neste cenário, existe uma réplica.

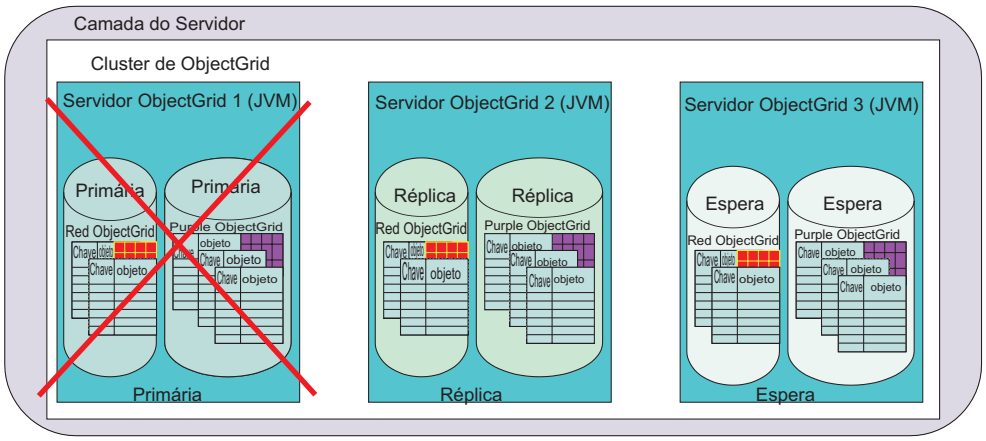


Figura 7. Cenário de Alta Disponibilidade do ObjectGrid

Quando é detectada uma falha, o primário se torna indisponível. A réplica se torna o primário. Se existir uma espera, ela se tornará uma Réplica, semelhante à recuperação de exemplo no diagrama a seguir:

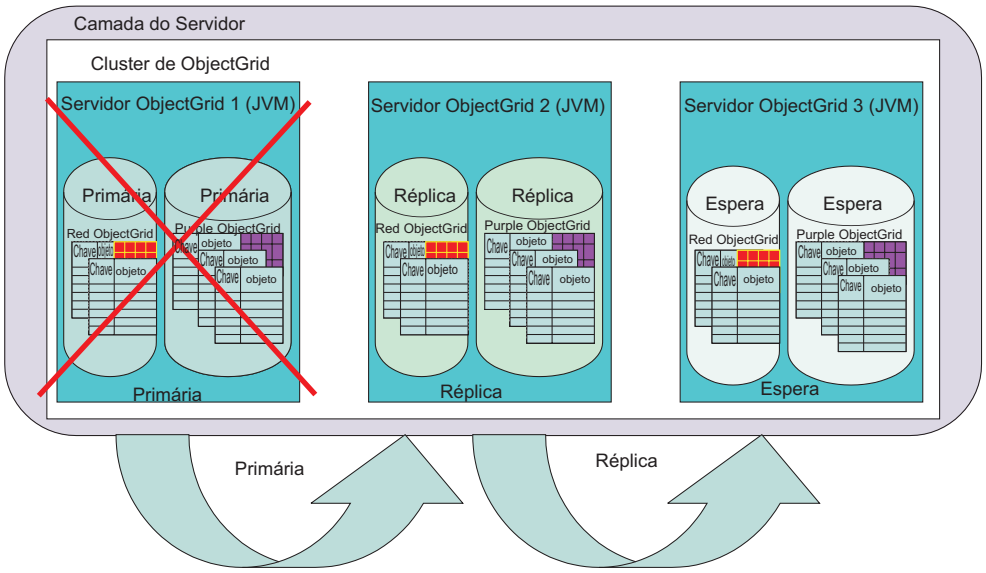


Figura 8. Failover do ObjectGrid

Os clientes do ObjectGrid se tornam cientes deste ajuste durante sua próxima conexão com qualquer um dos servidores afetados. Os clientes que contactam o servidor com falha utilizam sua configuração de tempo de execução e podem tentar os outros servidores no cluster dinamicamente. O cliente contacta o próximo servidor na configuração. Se o servidor estiver ativo e não operacional, o cliente esperará um tempo limite. Os clientes assumem que os membros do grupo de replicação replicados estejam se recuperando de uma falha. Após um período de tempo, os clientes tentam novamente e, depois que o grupo de replicação, agora com dois membros, estiver operacional, uma nova tabela de roteamento será fornecida para o cliente. A tabela de roteamento descreve onde o primário atual está localizado, seus locais de réplicas e quais membros do grupo de replicação deste grupo são esperas.

Conjuntos de Configurações de Armazenamento em Cluster do ObjectGrid

Ao configurar o armazenamento em cluster do ObjectGrid, é possível separar ObjectGrids em MapSets. Esta separação é importante, porque um ObjectGrid pode conter muitos mapas. Os MapSets podem ser particionados com um PartitionSet e replicados com um ReplicationGroup. Cada uma destas opções de configuração afeta a quantidade de membros do grupo de replicação criados durante a inicialização do servidor do ObjectGrid. Uma rápida visão geral de cada tipo de Conjunto ajuda a explicar a função de cada tipo.

MapSet do ObjectGrid

Cada mapa do ObjectGrid pode ter diferentes requisitos de uso e disponibilidade, no entanto, é correlacionado por uso do aplicativo típico. Por exemplo, um mapa pode ser de leitura sem alterações após a conclusão do pré-carregamento e outro pode ser de leitura/gravação e particionado para fins de escalabilidade. Neste caso, cada mapa está incluído em um MapSet exclusivo. No exemplo anterior, o PurpleMapSet e o RedMapSet continham todos os mapas para cada ObjectGrid especificado, que é a opção mais simples.

Um MapSet é uma unidade que pode ser replicada em vários Servidores do ObjectGrid e é correlacionado a um membro do grupo de replicação que não está particionado. Cada servidor do grupo de replicação associado a um MapSet por meio de um PartitionSet hospeda um membro do grupo de replicação, conforme apropriado, para suportar a configuração solicitada. Um membro do grupo de replicação é um nó de extremidade exclusivo no Cluster do ObjectGrid e hospeda todos os mapas indicados por um determinado MapSet na configuração.

Por exemplo, no diagrama anterior, o servidor server1 possui o primário, o servidor server2 possui a réplica e o servidor server3 possui uma unidade de espera que pode se tornar uma réplica ou um primário, dependendo dos cenários de recuperação de replicação. Os Mapsets estão correlacionados ao PartitionSet que descreve três servidores. Portanto, os dois MapSets, mesmo que tenham um número diferente de mapas em cada um, são mapeados para o mesmo servidor, porque as sub-rotinas de PartitionSet e de ReplicationGroup são iguais.

PartitionSet do ObjectGrid

Geralmente, o Mapset e os servidores do grupo de replicação enumerados determinam o número de servidores que suportam um MapSet específico em um cluster do ObjectGrid. Um membro do grupo de replicação é criado em cada servidor do grupo de replicação hosting. No entanto, o particionamento pode afetar o número de membros do grupo de replicação para um MapSet. O particionamento é gerenciado no arquivo de configuração por meio do relacionamento do PartitionSet entre o MapSet e o ReplicationGroup.

Um PartitionSet divide um MapSet em partes para que uma JVM (Java Virtual Machine) não contenha todo o MapSet em um único membro do grupo de replicação primário. Por exemplo, imagine um banco de dados com 1.000.000 de chaves. Se cada objeto referido por cada chave for grande, existe uma chance razoável de que uma única JVM de 32 bits não possa conter o mapa na memória em um único membro do grupo de replicação primário, de réplica ou de espera. No entanto, geralmente são requeridos grandes conjuntos de dados. Para evitar que você mesmo tenha que particionar os dados artificialmente, por exemplo, para evitar particionar manualmente o Purple ObjectGrid, primeiro mapeie a instância

para os mapas PurpleFirstMapMap1, PurpleFirstMap2, PurpleFirstMapN e coloque cada um em um Mapset diferente; o ObjectGrid pode fazer uma grande parte deste trabalho.

Posteriormente neste documento, o conceito de um PartitionKey é definido. Efetivamente, isto resulta em uma API que pode ser chamada pelo ObjectGrid para determinar qual é o código hash da chave para uma entrada específica durante a inserção. Se um MapSet tiver duas partições, serão criados dois membros do grupo de replicação para esse MapSet. Geralmente, porque os dados são grandes, estes membros do grupo de replicação estão localizados em diferentes servidores. Para desenvolvedores, estes membros podem estar no mesmo servidor durante um protótipo anterior. Cada membro do grupo de replicação contém chaves que se dividem para o mesmo valor nos membros do grupo de replicação particionados que estão disponíveis. Como um exemplo simples, suponha que o MapSet tivesse sido particionado de três maneiras: 0, 1 e 2. São estabelecidos três membros do grupo de replicação primários e um contém todas as chaves que dividem para um módulo do valor especificado o número dos membros do grupo de replicação primários. Por exemplo, se um valor de hash da chave for 7, 7 módulos 3 serão 1, portanto, o membro do grupo de replicação primário com o índice de partição 1 conteria a instância.

Exemplo de PartitionSet

O diagrama a seguir ilustra a separação do mapa roxo em duas partições. Cada chave no mapa é dividida para um inteiro e designada a um conjunto de partições específico durante a inserção no membro do grupo de replicação apropriado. Cada partição está em um membro do grupo de replicação diferente, porque pode existir em uma JVM diferente e o ObjectGrid permite que o programador em geral trate o PurpleFirstMap como uma única instância de mapa lógica, não-particionada. O suporte ao cliente e ao servidor do ObjectGrid gerencia o roteamento dos pedidos corretamente entre os membros do grupo de replicação.

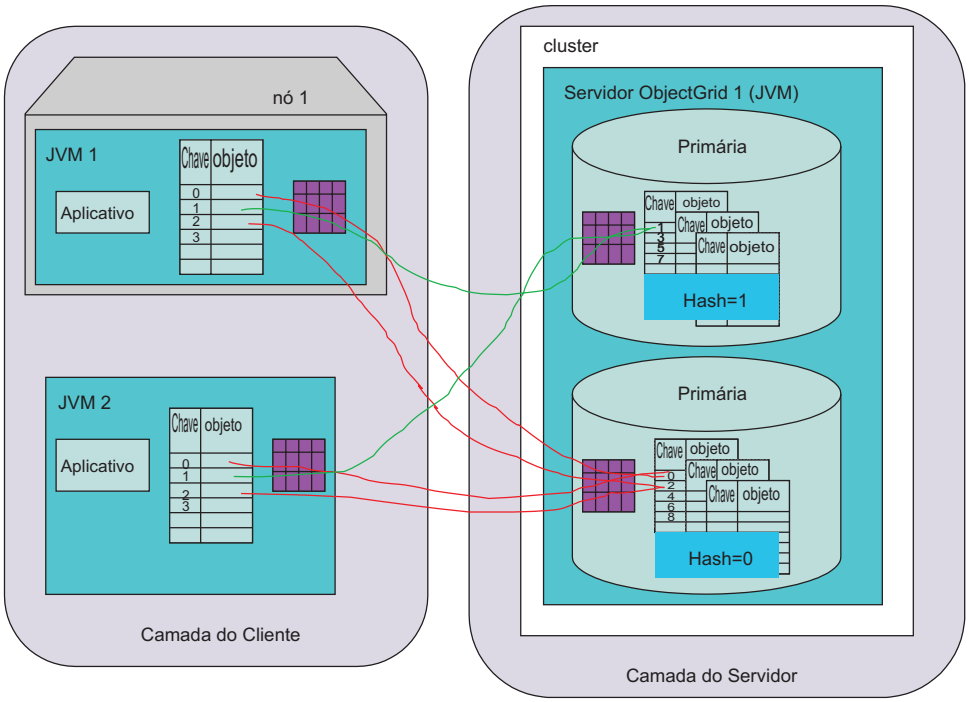


Figura 9. Topologia do ObjectGrid Distribuído: Servidor Único com Particionamento

A seguir está a alteração na configuração do conjunto de partições para ativar esta configuração:

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

A configuração de exemplo reflete como estabelecer um Purple ObjectGrid replicado, particionado. No caso a seguir existem três servidores e cada um dos membros do grupo de replicação primários é mapeado da mesma maneira para o conjunto de três servidores. Eles também podem ser facilmente mapeados de maneira diferente, se tiver sido utilizada outra sub-rotina de ReplicationGroup.

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

Observe no exemplo anterior que, novamente, apenas uma pequena alteração no arquivo configuração gera um novo nível de recurso e requer a modificação do aplicativo para atingir o resultado. O diagrama a seguir ilustra a visualização do cluster do ObjectGrid e como os membros do grupo de replicação são organizados para suportar a configuração de particionamento anterior:

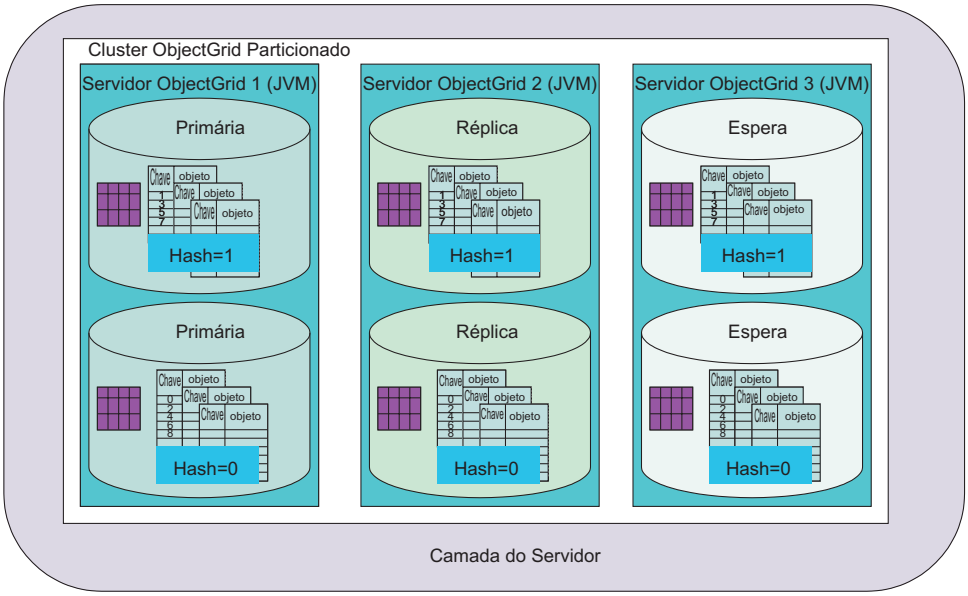


Figura 10. Topologia do ObjectGrid Distribuído: Vários Servidores com Particionamento

Para concluir a discussão de PartitionSet, a seguir está outra variação do exemplo anterior. Neste caso, foi criado um segundo ReplicationGroup. Cada PartitionSet agora está incluído em seu próprio grupo de replicação, em servidores separados.

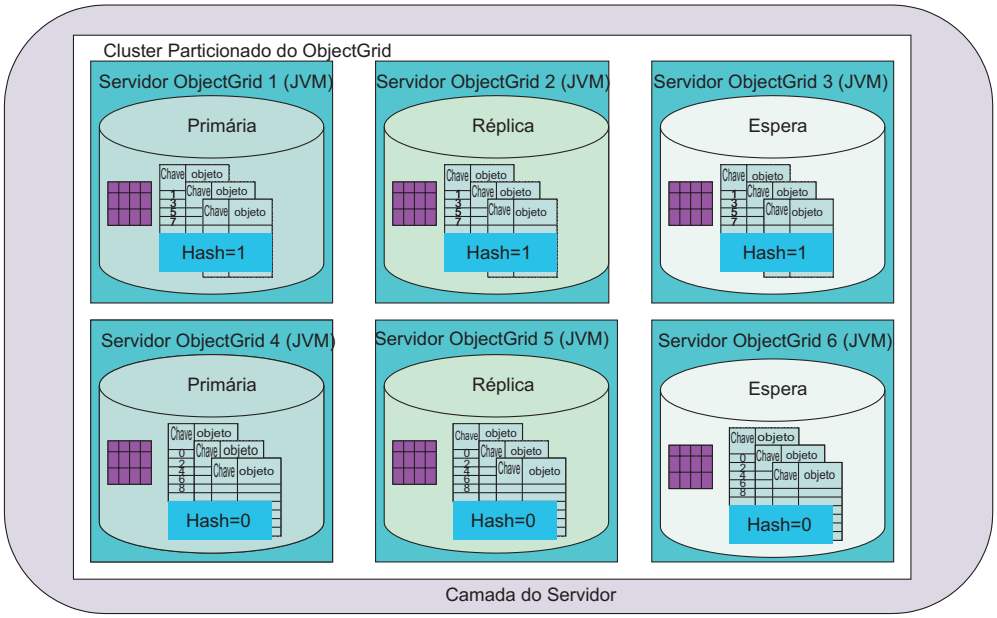


Figura 11. Topologia do ObjectGrid Distribuído: Vários Servidores com Particionamento

A configuração desta topologia é muito semelhante aos exemplos anteriores. As alterações incluem mais três instâncias do servidor e um novo ReplicationGroup que é referido pelo segundo PartitionSet.

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!- single server –>
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12504" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12506" />
<serverDefinition name="server3" host="localhost" clientAccessPort="12507"
peerAccessPort="12508" />
<serverDefinition name="server4" host="localhost" clientAccessPort="12513"
peerAccessPort="12514" /><!-*New*->
<serverDefinition name="server5" host="localhost" clientAccessPort="12514"
peerAccessPort="12516" /><!-*New*->
<serverDefinition name="server6" host="localhost" clientAccessPort="12517"
peerAccessPort="12518" /><!-*New*->
</cluster>
<objectGridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectGridBinding>
<partitionSet name="ColorMapsPartitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
<partition name="partition2" replicationGroupRef="ColorMapsReplicationGroup" />
<!-NEW->
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup" maxReplicas="1"
```

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

Esta configuração resulta no mesmo número de membros do grupo de replicação, mas força o segundo grupo de replicação a ser especificamente configurado por meio de sua própria sub-rotina XML e, ao mesmo tempo, atribui cada uma das instâncias a diferentes instâncias do servidor em vez de dispor conforme foi feito no exemplo anterior.

A vantagem desta configuração é que cada partição, neste caso, pode conter mais de 1.5+ gigabytes de dados, um total de 3 gigabytes ou mais, porque os dois membros do grupo de replicação estão em suas próprias instâncias de JVM. Por isso, é feita a melhor utilização de 2 gigabytes de JVM de 32 bits de espaço de memória endereçável.

Clientes do ObjectGrid que Contactam Vários Clusters do ObjectGrid

O ObjectGrid foi especificamente projetado não apenas para escalada em termos de suporte particionado em JVMs (Java Virtual Machines), mas também para estender o alcance no qual uma interface de Mapa Java normal pode ser obtida para adquirir informações. É possível contactar muitos clusters do ObjectGrid com um único cliente.

No cenário a seguir, uma camada do servidor contém dois clusters do ObjectGrid. Um dos clientes do aplicativo Java e um dos servidores de aplicativos precisam contactar vários clusters. Este é um poderoso recurso e permite uma grande escalabilidade.

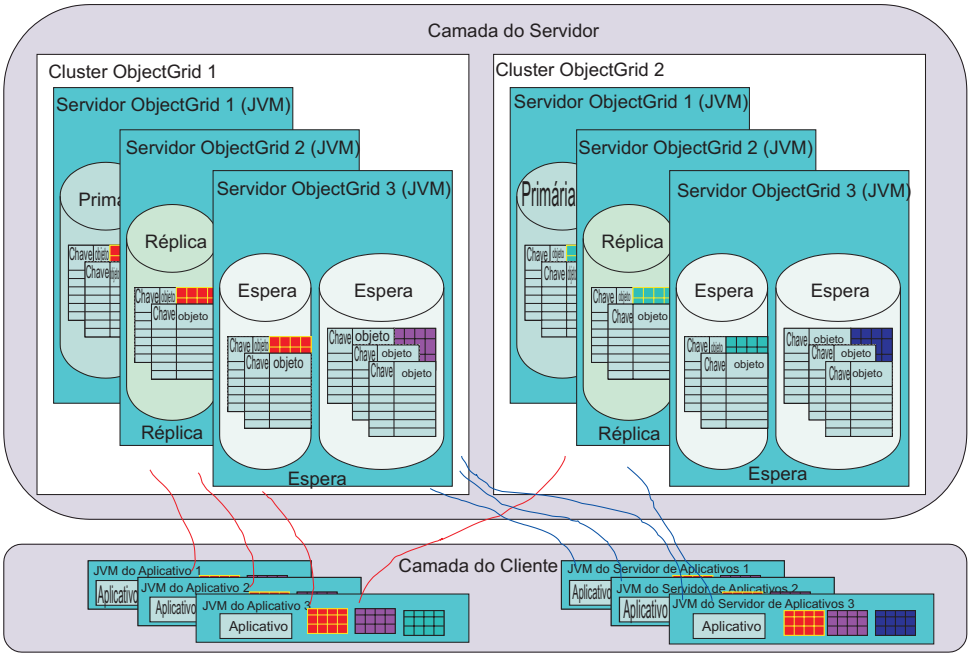


Figura 12. Clientes do ObjectGrid que Interagem com Vários Clusters do ObjectGrid

Um Cluster do ObjectGrid pode suportar muitos ObjectGrids, cada um contendo muitas configurações de MapSet e PartitionSet. Cada cluster do ObjectGrid pode ser gerado de uma ou mais JVMs - provavelmente de muitas. Para uma grande corporação, a capacidade de o Cliente do ObjectGrid contactar não apenas um mas vários clusters do ObjectGrid ao mesmo tempo é um recurso muito valioso.

No entanto, atenção: o Cliente do ObjectGrid não pode fazer referência a dados a partir de vários Clusters em uma única transação. O aplicativo Cliente do ObjectGrid deve executar uma ou mais transações para armazenar em cache os dados na instância da Java Virtual Machine e correlacionar as informações recuperadas como objeto Java. As atualizações baseadas nas informações também devem estar contidas em uma única transação para cada cluster. Consulte “Demarcação de Transação do ObjectGrid” na página 39 para obter detalhes adicionais sobre este assunto.

Suporte ao Near Caching do Cliente do ObjectGrid

O cliente do ObjectGrid é de fato uma camada de armazenamento em cache. É possível projetar seu aplicativo para alavancar o recurso de armazenamento em cache local se você souber que os dados adquiridos anteriormente do servidor remoto não são stale. Por exemplo, se os dados tiverem sido atualizados no cluster do ObjectGrid mas não neste cliente (isto utilizaria um novo pedido get(...)), o cliente deverá atualizar o cache local para que seja consistente (nem todos os aplicativos requerem isso) com o cluster do ObjectGrid.

Quando uma operação get é desempenhada, um pedido seguinte para a operação get para o mesmo par de chave e objeto resulta no cliente do ObjectGrid detectando que os dados já foram recuperados e utiliza a versão em cache ″na JVM (Java Virtual Machine)″ em vez de passar pela rede no cluster do ObjectGrid para acessar os dados. Quando os dados tiverem sido recuperados uma vez pela rede, eles continuarão sendo fornecidos no cache local até que a entrada local seja liberada, manualmente ou por meio de um evictor normal configurado.

Por exemplo, se você entender que os dados no servidor são atualizados uma vez a cada seis horas, poderá controlar quando ocorrerá a atualização do cliente do cache local ou do near cache. O usuário invalida a entrada de near cache e, em seguida, emite o pedido get. O pedido get contacta o servidor e adquire as informações se tudo correr bem. Suponha que o objeto seja um arquivo de imagem. Na primeira vez em que a imagem for transferida por download após a janela de atualização, cada pedido seguinte não resultará em uma chamada de procedimento remoto para o servidor para obter a imagem.

O suporte ao near caching não se aplica no modo pessimista, porque o cliente pode exigir um bloqueio dos dados no cluster do ObjectGrid para aplicar a estratégia de bloqueio solicitada. Reveja o método beginNoWriteThrough() para obter detalhes adicionais sobre como limpar ou invalidar entradas de near cache que devem ser removidas sem modificar a visualização das informações do cluster do ObjectGrid.

Demarcação de Transação do ObjectGrid

Um conceito-chave para se lembrar sendo um programador é o conceito de demarcação de transação. O ObjectGrid não suporta o protocolo de confirmação de duas fases para processamento de confirmação de transação nos membros do grupo de replicação em um cluster do ObjectGrid. Em uma única sessão baseada em cluster do ObjectGrid, as atualizações de leitura/gravação devem ser aplicadas a um único membro do grupo de replicação primário. Se vários membros do grupo de replicação estiverem envolvidos, não será possível fazer as atualizações atômicas durante o processamento de confirmação de transação. Isto é um pouco diferente do modelo de programação JAR do ObjectGrid local, que permite a confirmação em todos os mapas em um único ObjectGrid.

Um caso especial a ser notado é um PartitionSet no qual mais de uma partição está definida. Nestes cenários, a chave 1 poderia estar no servidor 1, a chave 2 no servidor 2 e assim por diante. Se, em uma transação, houver atualizações para a chave 1 e chave 2, essas atualizações falharão, porque uma sessão do ObjectGrid não pode ser confirmada em dois membros do grupo de replicação. Conforme observado anteriormente, um PartitionSet que suporta duas ou mais partições deve ser utilizado com cuidado no aplicativo. Tenha atenção para assegurar que a seqüência de transações não atualize dados de mais de uma única transação.

Relacionamento do ObjectGrid com Bancos de Dados

Esta visão geral não abordou especificamente os servidores do ObjectGrid que adquirem dados inicializados de locais diferentes dos outros clientes Java. Quando um ObjectGrid é iniciado, é inicializado um loader para cada Mapa em um MapSet. Este loader permite que os pedidos do usuário em forma de get ou put de um Mapa Java sejam recuperados ou gravados no banco de dados conforme apropriado. As operações do banco de dados ficam invisíveis para o usuário do Mapa Java e não é requerida nenhuma codificação especial em seu aplicativo. No entanto, a funcionalidade do loader deve ser desenvolvida por um programador e configurada para utilização no ObjectGrid antes que o programador usuário final possa utilizar esta função. Consulte “Loaders” na página 201 para obter mais detalhes.

Além disso, existe o suporte ao pré-carregamento para pré-carregamento a partir de um banco de dados após a inicialização de um Cluster do ObjectGrid, bem como um pré-carregamento particionado para assegurar que os dados corretos sejam lidos para a partição do membro do grupo de replicação específica para um determinado MapSet. Os usuários podem acessar e atualizar as informações de

leitura/gravação com várias estratégias de bloqueio, incluindo nenhum, otimista e pessimista. O acesso a dados de leitura é suportado e é o modelo mais rápido, pois podem ser fornecidas várias otimizações.

Capítulo 4. Tutorial do ObjectGrid: Modelo de Programação de Aplicativo

Utilize esta tarefa para aprender sobre o modelo de programação de aplicativo do ObjectGrid.

Prepare seu ambiente para executar aplicativos do ObjectGrid. Consulte o Capítulo 1, “Introdução ao ObjectGrid Executando o Aplicativo de Amostra”, na página 1 para aprender sobre locais do arquivo JAR (Java Archive), requisitos Java e como executar um arquivo simples para verificar se seu ambiente está configurado corretamente.

Decida qual ambiente de programação utilizar para esta tarefa. É possível utilizar um IDE (Ambiente de Desenvolvimento Integrado), como o Eclipse, mas o ambiente Java da linha de comandos também funciona. Incorpore o ObjectGrid em beans corporativos e servlets quando estiver mais familiarizado com o ObjectGrid. Os exemplos no Tutorial não assumem nenhum ambiente Java específico, portanto, é possível utilizar qualquer ambiente familiar.

Em sua definição mais básica, o ObjectGrid é um cache e um repositório na memória para objetos. A utilização de um mapa java.util.Map para armazenar e acessar objetos é semelhante à utilização do ObjectGrid. Ao mesmo tempo, o ObjectGrid é mais do que apenas um cache. Explorando os diversos recursos e plug-ins nesta tarefa, você descobre que o ObjectGrid é muito extensível e flexível. É possível utilizar o ObjectGrid como um cache de *repositório associado* simples ou um cache mais elaborado suportado por um gerenciador de recursos.

Os exemplos neste tutorial não são programas completos. Importações, processamento de exceções e até mesmo algumas das variáveis não estão totalmente declaradas em cada exemplo. É possível utilizar as amostras para gravar seus próprios programas.

Utilize esta tarefa para utilizar o ObjectGrid a partir de um programa Java.

1. Localize as APIs e exceções do ObjectGrid. Todas as APIs e Exceções públicas do ObjectGrid estão contidas no pacote `com.ibm.websphere.objectgrid`. Para tópicos de sistema ou de configuração mais avançados, consulte as APIs e Exceções adicionais no pacote `com.ibm.websphere.objectgrid.plugins`. Onde houver implementações de plug-ins fornecidas, localize estas classes no pacote `com.ibm.websphere.objectgrid.plugins.builtins`. Para os recursos de segurança do ObjectGrid, procure pacotes com **security** no nome, como `com.ibm.websphere.objectgrid.security`, `com.ibm.websphere.objectgrid.security.plugins` e `com.ibm.websphere.objectgrid.security.plugins.builtins`.

   Esta tarefa focaliza as APIs que estão no pacote `com.ibm.websphere.objectgrid`. O JavaDoc completo para ObjectGrid pode ser encontrado no seguinte local: `<install_root>/web/xd/apidocs`.

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

2. Obtenha ou crie uma instância do ObjectGrid. Utilize o ObjectGridManagerFactory para obter a instância do singleton do ObjectGridManager. Em seguida, crie uma instância do ObjectGrid com as seguintes instruções:

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

   A interface do ObjectGridManager possui vários métodos para criar, recuperar e remover instâncias do ObjectGrid. Consulte o tópico “Interface ObjectGridManager” na página 93 para escolher uma variação para sua situação. Também é possível definir configurações de rastreio com a interface ObjectGridManager. Se estiver em execução no WebSphere Extended Deployment ou no WebSphere Application Server, estes métodos não serão necessários porque o rastreio é gerenciado pelos recursos incluídos. Se estiver em execução fora do WebSphere Application Server, estes métodos poderão ser úteis. Consulte o tópico “ObjectGrid de Rastreio” na página 100 para obter informações mais completas para estes métodos.

3. Inicialize o ObjectGrid.
   a. Configure um nome para o ObjectGrid, se não tiver configurado o nome com os métodos `create`.
   b. Defina os BackingMaps, utilizando a configuração padrão para um BackingMap para seus aplicativos iniciais.
   c. Depois de definir seus BackingMaps, inicialize o ObjectGrid. A inicialização do ObjectGrid sinaliza que toda a configuração está concluída e que você deseja começar a utilizar o ObjectGrid.
   d. Após a inicialização do ObjectGrid, obtenha um objeto de Sessão. Consulte “Interface do ObjectGrid” na página 107 e o JavaDoc para obter informações adicionais.

   Utilize o seguinte exemplo como orientação nesta etapa:

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

4. Utilize sessões para gerenciar operações transacionais. Todo o acesso a um cache do ObjectGrid é transacional: vários acessos, inserções, atualizações e remoções de Objetos do cache estão contidos em uma única unidade de trabalho, referida como uma sessão. No final da sessão, é possível confirmar todas as alterações nesta unidade de trabalho ou efetuar rollback e esquecer todas as alterações na unidade de trabalho.

   Também é possível utilizar a confirmação automática para operações atômicas únicas no cache. Na ausência de um contexto de sessão ativa, os acessos individuais ao conteúdo do cache são contidos em suas próprias sessões confirmadas automaticamente.

   Outro aspecto importante da interface Session é obter acesso transacional, ou manipulação, para o BackingMap com a interface ObjectMap. Também é possível utilizar o método getMap para criar uma manipulação de ObjectMap para um BackingMap predefinido. Todas as operações no cache, como inserções, atualizações, exclusões, são concluídas com a instância do

ObjectMap. Consulte o tópico “Interface Session” na página 116 para obter informações adicionais. Utilize o seguinte exemplo para obter e gerenciar uma sessão:

```
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"); // confirmação automática
```

5. Utilize a interface ObjectMap para acessar e atualizar o cache. Conforme você consulta a interface ObjectMap, observe vários métodos para acessar e atualizar o cache. A interface ObjectMap é modelada como uma interface semelhante a um mapa. No entanto, as exceções verificadas são introduzidas como um auxílio no desenvolvimento de aplicativos do ObjectGrid com um IDE, como o Eclipse. Se desejar utilizar uma interface java.util.Map sem exceções verificadas, será possível utilizar o método getJavaMap. Consulte “Interfaces ObjectMap e JavaMap” na página 120 para obter maiores informações.

   Os métodos de inserção e atualização explícitos evitam a operação de entrada indefinida. Ainda é possível utilizar o método put, mas a utilização dos métodos de inserção e atualização explícitos transmitem sua intenção muito mais claramente. A utilização do método put é esclarecida definindo um método put sem uma operação get anterior como um método de inserção. Se houver tentativa de uma operação get anterior antes da operação de entrada, a operação de entrada será tratada como uma inserção ou atualização, dependendo de a entrada existir no cache.

   É possível desempenhar as seguintes operações básicas do ObjectMap: get, de entrada, de inserção, de atualização, de remoção, de toque, de invalidação e containsKey. Vários detalhes e variações podem ser localizados no tópico “Visão Geral do Modelo de Programação do Sistema” na página 45 ou na documentação da API do ObjectMap. O exemplo a seguir demonstra a utilização do ObjectMap para modificar o cache:

   ```
   ObjectGridManager objectGridManager =
   ObjectGridManagerFactory.getObjectGridManager();
   ObjectGrid objectGrid =
   objectGridManager.createObjectGrid("someGrid");
   objectGrid.defineMap("someMap");
   objectGrid.initialize();
   Session session = objectGrid.getSession();
   ObjectMap objectMap = session.getMap("someMap");
   // Iniciar uma transação/sessão...
   session.begin();
   objectMap.insert("key1", "value1");
   objectMap.put("key2", "value2");
   session.commit();
   // Verificar alterações confirmadas
   String value1 = (String)objectMap.get("key1");
   String value2 = (String)objectMap.get("key2");
   System.out.println("key1 = " + value1 + ", key2 = " + value2);
   //Iniciar uma nova transação/sessão...
   session.begin();
   objectMap.update("key2", "newValue2");
   objectMap.remove("key1");
   session.rollback();
   // Verificar alterações não confirmadas
   String newValue1 = (String)objectMap.get("key1");
   String newValue2 = (String)objectMap.get("key2");
   System.out.println("key1 = " + newValue1 + ", key2 = " + newValue2);
   ```

6. Utilize o índice para procurar objetos em cache. Utilizando o índice, seus aplicativos podem localizar objetos por um valor específico ou um intervalo de

valores. O mapa do BackingMap deve ter o plug-in do índice configurado antes de os aplicativos utilizarem a função de índice. Os aplicativos devem obter o objeto de índice do método getIndex() da interface ObjectMap e lançá-lo para a interface de índice correta, como a interface MapIndex, a interface MapRangeIndex ou uma interface de índice customizado.

No momento, o recurso de indexação é suportado apenas no cache local. O recurso de indexação não é suportado no cache distribuído. Se você tentar desempenhar qualquer operação de indexação em um cache distribuído, isto resultará na exceção `UnsupportedOperationException`.

O exemplo a seguir demonstra como utilizar o índice:

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

Conforme conclui a leitura desta seção e tenta utilizar o código de exemplo, você fica mais familiarizado com o modelo de programação essencial do ObjectGrid.

Para obter informações mais específicas, consulte o capítulo Capítulo 9, “Visão Geral da Interface de Programação de Aplicativo do ObjectGrid”, na página 93.

Introdução ao ObjectGrid Remoto

Coloque sua descrição resumida aqui; utilizada para o primeiro parágrafo e resumo.

O Capítulo 4, “Tutorial do ObjectGrid: Modelo de Programação de Aplicativo”, na página 41 geralmente lida com um ″local″ ou no uso do aplicativo de um ObjectGrid. O aplicativo criou uma instância de um ObjectGrid e utilizou essa instância. Quando a JVM (Java Virtual Machine) do aplicativo foi encerrada, o cache do ObjectGrid também foi encerrado. O ObjectGrid remoto, como o nome sugere, permite acesso a um ObjectGrid residente em uma JVM diferente. Vários clientes podem conectar-se ao ObjectGrid remoto e acessar o ObjectGrid utilizando a mesma API de forma transparente.

1. Reveja as seguintes seções para iniciar:
   - Capítulo 3, “Visão Geral do ObjectGrid”, na página 19
   - “Configuração do ObjectGrid” na página 260
   - “APIs Connect do Cliente do ObjectGrid” na página 101
   - Capítulo 8, “Suporte à Linha de Comandos”, na página 83
2. Para iniciar os servidores, é necessário definir o arquivo XML do ObjectGrid e o arquivo XML do Cluster. Consulte “Configuração do ObjectGrid Distribuído” na página 273. Este tópico se refere aos arquivos `university.xml` e `universityCluster.xml`. É possível utilizar estes arquivos como um exemplo, modificando o host e porta para ativar ou iniciar o servidor. Consulte Capítulo 8, “Suporte à Linha de Comandos”, na página 83 para obter detalhes de como ativar servidores do ObjectGrid.
3. Quando o servidor estiver em execução, um cliente poderá conectar-se a este servidor em execução. Consulte “APIs Connect do Cliente do ObjectGrid” na página 101 para obter detalhes de como um cliente pode conectar-se e desempenhar operações do ObjectGrid.

Visão Geral do Modelo de Programação do Sistema

O modelo de programação do sistema fornece vários recursos e pontos de extensão adicionais para o ObjectGrid.

O diagrama a seguir ilustra como o modelo de programação do sistema fornece vários recursos e pontos de extensão adicionais.

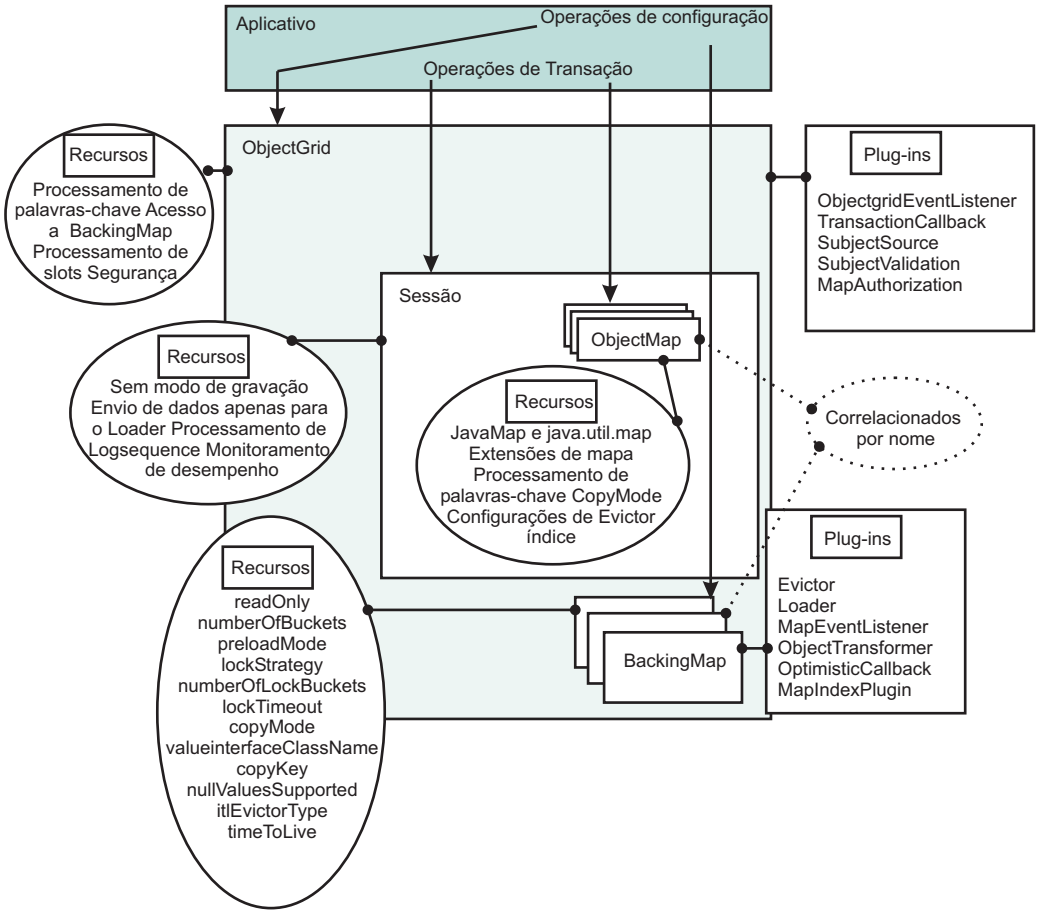


Figura 13. Visão Geral do ObjectGrid

Um plug-in no ObjectGrid é um componente que fornece um determinado tipo de função para os componentes conectáveis do ObjectGrid, que incluem o ObjectGrid e BackingMap. Um recurso representa uma função ou característica específica de um componente do ObjectGrid, incluindo ObjectGrid, Session, BackingMap, ObjectMap e outros. Se um recurso representa uma função, ele pode ser utilizado para atingir um objetivo de computação específico. Se um recurso é uma característica, ele pode ser utilizado para ajustar o comportamento dos componentes do ObjectGrid.

Cada uma das seções a seguir descreve alguns dos recursos e extensões ilustrados no diagrama anterior:

- "Visão Geral do Modelo de Programação do Sistema: Pontos de Conexão e Recursos de Interface do ObjectGrid" na página 46

  A interface do ObjectGrid possui vários pontos e recursos de conexão para interações mais extensíveis com o ObjectGrid.

- “Visão Geral do Modelo de Programação do Sistema: Pontos de Conexão e Recursos de Interface BackingMap” na página 48

  A interface BackingMap possui vários pontos de conexão e recursos opcionais para interações mais extensíveis com o ObjectGrid.
- “Visão Geral do Modelo de Programação do Sistema: Recursos da Interface Session” na página 56

  A interface Session possui vários recursos para interações mais extensíveis com o ObjectGrid. Cada uma das seções neste tópico descreve o recurso e fornece alguns breves trechos de código para o cenário de uso.
- “Visão Geral do Modelo de Programação do Sistema: Recursos da Interface ObjectMap” na página 58

  A interface ObjectMap possui vários recursos e interações mais extensíveis com o ObjectGrid. Cada uma das seções neste tópico descreve o recurso e fornece alguns breves trechos de código para o cenário de uso.

Para obter informações sobre os recursos e plug-ins individuais, consulte o Capítulo 9, “Visão Geral da Interface de Programação de Aplicativo do ObjectGrid”, na página 93.

Visão Geral do Modelo de Programação do Sistema: Pontos de Conexão e Recursos de Interface do ObjectGrid

A interface do ObjectGrid possui vários pontos e recursos de conexão para interações mais extensíveis com o ObjectGrid.

Cada uma das seções a seguir descreve o recurso e fornece alguns trechos de código breves para o cenário de uso. Onde apropriado, é fornecido um snippet XML para mostrar a configuração XML alternativa. Para obter informações mais abrangentes, consulte os tópicos “Interface do ObjectGrid” na página 107 e “Configuração do ObjectGrid” na página 260.

Processamento de Palavras-chave

A interface do ObjectGrid fornece um mecanismo de invalidação flexível baseado em palavras-chave. Uma palavra-chave é uma instância não nula de qualquer objeto seriável. Você pode associar palavras-chave a entradas do BackingMap de qualquer maneira. A maior parte do processamento de palavra-chave é desempenhada no nível do ObjectMap, mas a associação de uma palavra-chave à outra palavra-chave para fornecer uma árvore hierárquica de palavras-chave é desempenhada no nível do ObjectGrid.

O método associateKeyword(java.io.Serializable parent, java.io.Serializable child) vincula as duas palavras-chave em um relacionamento direcional. Se um pai for invalidado, o filho também será invalidado. A invalidação do filho não tem nenhum impacto sobre o pai. Por exemplo, este método é utilizado para incluir uma entrada do mapa New York como um filho da entrada do mapa USA para que, se USA for invalidada, todas as entradas New York também sejam invalidadas. Consulte a seguinte amostra de código:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
// associar várias cidades à palavra-chave "USA"
objectGrid.associateKeyword("USA", "New York");
objectGrid.associateKeyword("USA", "Rochester");
objectGrid.associateKeyword("USA", "Raleigh");
:
// inserir várias entradas com várias palavras-chave
objectMap.insert("key1", "value1", "New York");
objectMap.insert("key2", "value2", "Mexico");
objectMap.insert("key3", "value3", "Raleigh");
```

```
objectMap.insert("key4", "value4", "USA");
objectMap.insert("key5", "value5", "Rochester");
objectMap.insert("key6", "value6", "France");
:
// invalidar todas as entradas associadas à palavra-chave "USA", deixando
// as entradas "key2" e "key6"
objectMap.invalidateUsingKeyword("USA", true);
:
```

Para obter mais informações, consulte “Palavras-chave” na página 124.

Acesso a BackingMap

O ObjectGrid fornece acesso a objetos do BackingMap. É possível obter acesso a um BackingMap com os métodos defineMap ou getMap. Consulte “Interface BackingMap” na página 112 para obter maiores informações. O exemplo a seguir cria duas referências de BackingMap:

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

Processamento de Slots

É possível reservar um slot para armazenar objetos utilizados no decorrer da transação, como o objeto do ID da transação (TxID) ou um objeto de conexão com o banco de dados (Conexão). Estes objetos armazenados são então referidos com um índice específico, que é fornecido pelo método reserveSlot. É possível localizar informações adicionais sobre como utilizar slots nos tópicos “Loaders” na página 201 e “Plug-in TransactionCallback” na página 217. O trecho de código a seguir demonstra o processamento de slots:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
int index = objectGrid.reserveSlot
(com.ibm.websphere.objectgrid.TxID.SLOT_NAME);
:
// Utilizar o índice posteriormente ao armazenar ou recuperar objetos do
//objeto TxID ...
TxID tx = session.getTxID();
tx.putSlot(index, someObject);
:
Object theTxObject = tx.getSlot(index);
:
```

Processamento de Segurança

Os mapas podem ser protegidos utilizando mecanismos de segurança. Os métodos a seguir estão disponíveis em um ObjectGrid para configuração e utilização dos recursos de segurança.

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

Consulte “Segurança do ObjectGrid” na página 139 para obter informações adicionais sobre os mecanismos de segurança disponíveis.

ObjectGridEventListener

O listener ObjectGridEventListener permite que os aplicativos recebam

notificação em caso de início ou confirmação de uma transação. Uma instância de um ObjectGridEventListener pode ser configurada no ObjectGrid. Consulte o tópico “Listeners” na página 186 para obter informações adicionais. A seguir está um exemplo de como implementar a interface ObjectGridEventListener programaticamente:

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

Também é possível desempenhar a mesma configuração com XML:

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

Plug-in TransactionCallback

A chamada de métodos na sessão envia eventos correspondentes ao plug-in TransactionCallback. Um ObjectGrid pode ter zero ou um plug-in TransactionCallback. Os BackingMaps definidos em um ObjectGrid com um plug-in TransactionCallback devem ter um Loader correspondente. Consulte “Plug-in TransactionCallback” na página 217 para obter maiores informações. O trecho de código a seguir demonstra como implementar o plug-in TransactionCallback programaticamente:

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

É possível desempenhar a mesma configuração com XML:

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

Visão Geral do Modelo de Programação do Sistema: Pontos de Conexão e Recursos de Interface BackingMap

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

Cada uma das seções a seguir descreve o recurso e fornece alguns trechos de código breves para o cenário de uso. Onde apropriado, é fornecido um snippet XML para mostrar a configuração XML alternativa. As informações mais extensivas estão nos tópicos “Interface BackingMap” na página 112 e “Configuração do ObjectGrid” na página 260 ou na documentação da API.

Atributos de Configuração

Vários itens de configuração estão associados a BackingMaps:

- **ReadOnly** (padronizado como `false`): A configuração deste atributo como `true` torna o mapa de suporte de leitura. A configuração como `false` tornará o mapa de suporte um mapa de leitura e gravação. Se não for especificado um valor, isto resultará no padrão de leitura e de gravação.
- **NullValuesSupported** (padronizado como `true`): O suporte a valor nulo significa que um valor nulo pode ser colocado em um mapa. Se este atributo estiver configurado como true, os valores nulos serão suportados no ObjectMap; caso contrário, os valores nulos não serão suportados. Se forem suportados valores nulos, uma operação `get` que retorna nulo pode significar que o valor é nulo ou que o mapa não contém a chave transmitida.
- **NumberOfBuckets** (padronizado como `503`): Especifica o número de depósitos utilizados por este BackingMap. A implementação de BackingMap utiliza um mapa hash para sua implementação. Se existirem muitas entradas no BackingMap, mais depósitos resultarão em melhor desempenho porque o risco de colisões é menor conforme aumenta o número de depósitos. Mais depósitos também resultam em maior simultaneidade.
- **NumberOfLockBuckets** (padronizado como `383`): Especifica o número de depósitos de bloqueios utilizados pelo gerenciador de bloqueios para este BackingMap. Quando o atributo 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 hash, mais depósitos de bloqueios resultarão em melhor desempenho, pois o risco de colisões é menor conforme aumenta o número de depósitos. Mais depósitos de bloqueios também significam mais simultaneidade. Quando lockStrategy for NONE, nenhum gerenciador de bloqueios será utilizado por este BackingMap. Neste caso, a configuração do atributo numberOfLockBuckets não tem efeito.

Exemplo de Configuração Programática

O exemplo a seguir configura propriedades em um mapa de suporte:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// substituir padrão de leitura/gravação
backingMap.setReadOnly(true);
// substituir padrão de permissão de valores Nulos
backingMap.setNullValuesSupported(false);
// substituir padrão (números primos
funcionam melhor)
backingMap.setNumberOfBuckets(251);
// substituir padrão (números primos
funcionam melhor)
backingMap.setNumberOfLockBuckets(251);
:
```

Exemplo de Configuração XML

O exemplo de configuração XML a seguir configura as mesmas propriedades demonstradas na amostra programática anterior.

```
:
<objectGrids>
<objectGrid name="someGrid">
```

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

Estratégia de Bloqueio

Quando a estratégia de bloqueio estiver configurada como OPTIMISTIC ou PESSIMISTIC, será criado um gerenciador de bloqueios para o BackingMap. Para evitar que ocorram conflitos, o gerenciador de bloqueios tem um valor de tempo limite padrão para esperar que um bloqueio seja concedido. Se este tempo limite for excedido, isto resultará em uma exceção LockTimeoutException. O valor padrão de 15 segundos é suficiente para a maioria dos aplicativos mas, em um sistema altamente carregado, pode ocorrer um tempo limite quando não existir nenhum conflito real. Neste caso, o método setLockTimeout pode ser utilizado para aumentar o valor de tempo limite de bloqueio do padrão para qualquer valor requerido para evitar que ocorram falsas exceções de tempo limite. Quando a estratégia de bloqueio for NONE, nenhum gerenciador de bloqueios será utilizado por este BackingMap. Neste caso, a configuração do atributo lockTimeout não tem nenhum efeito. Para obter informações adicionais, consulte o tópico “Bloqueio” na página 130.

Exemplo de Configuração Programática

O exemplo a seguir configura a estratégia de bloqueio:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// substituir valor padrão de OPTIMISTIC
backingMap.setLockStrategy(LockStrategy.PESSIMISTIC);
backingMap.setLockTimeout(30); // configura o tempo limite de bloqueio como 30
segundos
:
```

Exemplo de Configuração XML

O exemplo a seguir configura a mesma estratégia de bloqueio definida no exemplo programático anterior.

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

Copiar Chaves e Valores

Fazer cópias de chaves e valores pode ser caro, de uma perspectiva de recurso e de desempenho. Sem o recurso para fazer estas cópias, podem ocorrer problemas desconhecidos e de difícil depuração. O ObjectGrid permite configurar se e quando fazer cópias de chaves ou valores. Normalmente, as chaves são consideradas imutáveis, portanto, não é necessário copiar os objetos de chaves. O modo padrão para objetos de chaves é não fazer cópias. Os objetos Value têm maior probabilidade de serem modificados pelo aplicativo. Quando fornecer uma cópia do objeto Value versus a referência real ao objeto Value é uma opção configurável.

Consulte o tópico Capítulo 11, “Boas Práticas de Desempenho do ObjectGrid”, na página 329 e o JavaDoc para obter detalhes adicionais sobe as configurações de CopyKey e CopyMode.

Exemplo de Configuração Programática

A seguir está um exemplo de configuração das configurações de modo de cópia e de chave de cópia:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.setCopyKey(true); // fazer uma cópia de cada nova chave
backingMap.setCopyMode(NO_COPY); // Mais eficiente - confiar no aplicativo
:
```

Exemplo de Configuração XML

O exemplo a seguir resulta na mesma configuração que está no exemplo de configuração programática anterior:

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

Evictors

Os evictores são utilizados para limpar periodicamente entradas desnecessárias no mapa. As entradas removidas são definidas pelo Evictor. Os Evictores internos são baseados em tempo, portanto, a estratégia de evicção é baseada na quantidade de tempo que uma entrada ficou ativa no mapa. Outras estratégias de evicção são baseadas no uso, tamanho ou uma combinação de fatores.

- **Evictor TTL (Time To Live) Interno**: O evictor Time To Live interno fornece alguns itens de configuração que estão configurados no BackingMap com os métodos setTtlEvictorType e setTimeToLive. Por padrão, este evictor TimeToLive interno não está ativo. É possível ativá-lo chamando o método setTtlEvictorType com um de três valores: CREATION_TIME, LAST_ACCESS_TIME ou NONE (padrão). Em seguida, dependendo do tipo de evictor TimeToLive selecionado, o valor para o método setTimeToLive é utilizado para configurar a existência de cada entrada do mapa.
- **Plug-ins do Evictor:** Além do Evictor Time To Live interno, um aplicativo pode fornecer seu próprio plug-in de implementação do Evictor. É possível utilizar qualquer algoritmo periodicamente para invalidar entradas do mapa.

Configuração Programática

A classe a seguir cria um evictor:

```
class MyEvictor implements com.ibm.websphere.objectgrid.plugins.Evictor { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// o cronômetro é iniciado quando a entrada é criada pela primeira vez
backingMap.setTtlEvictorType(CREATION_TIME);
```

```
// Permitir que cada entrada do mapa exista por 30 segundos antes da invalidação
backingMap.setTimeToLive(30);
// Os Evictors internos e customizados ficarão ativos
backingMap.setEvictor(new MyEvictor()); :
```

Configuração XML

O código XML a seguir cria uma configuração idêntica à configuração programática anterior:

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

Para obter mais informações, consulte “Evictors” na página 191.

Loaders

Um Loader do ObjectGrid é um componente conectável que permite que um mapa do ObjectGrid se comporte como um cache de memória para dados normalmente mantidos em um armazenamento persistente no mesmo sistema ou em outro sistema. Geralmente, um banco de dados ou sistema de arquivos é utilizado como o armazenamento persistente. Um loader tem a lógica para leitura e gravação de dados de e para um armazenamento persistente.

Um Loader é um plug-in opcional para um mapa de suporte do ObjectGrid. Apenas um Loader pode ser associado a um determinado mapa de suporte e cada mapa de suporte possui sua própria instância do Loader. O mapa de suporte solicita dados que ele não possui de seu Loader. As alterações feitas no mapa são enviadas para o Loader. O plug-in do Loader fornece uma maneira para que o mapa de suporte mova dados entre o mapa e seu armazenamento persistente.

Configuração Programática

A seguir está um exemplo de uma implementação do loader:

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

Configuração XML

A amostra XML a seguir resulta na mesma configuração que está no exemplo programático anterior:

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

Para obter informações adicionais, consulte o tópico “Loaders” na página 201.

Interface MapEventListener

A interface de retorno de chamada MapEventListener é implementada pelo aplicativo quando ele deseja receber eventos sobre um Mapa, como a evicção de uma entrada do mapa ou a conclusão do pré-carregamento de dados. O exemplo de código a seguir demonstra como configurar uma instância MapEventListener em uma instância do BackingMap:

Configuração Programática

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

Configuração XML

O exemplo a seguir resulta na mesma configuração que está no exemplo programático anterior:

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

Consulte o tópico “Listeners” na página 186 para obter informações adicionais.

Interface ObjectTransformer

O ObjectTransformer pode ser utilizado para serializar chaves e valores de entrada de cache que não estão definidos como seriáveis para que seja possível definir seu próprio esquema de serialização sem estender ou implementar a interface Serializable diretamente. Esta interface também fornece métodos para

desempenhar a função de cópia em chaves e valores. A seguir está uma classe que implementa a interface ObjectTransformer:

Configuração Programática

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

Configuração XML

O exemplo XML a seguir resulta na mesma configuração que está na amostra programática anterior:

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

Para obter informações adicionais, consulte o tópico “Plug-in ObjectTransformer” na página 213.

Interface OptimisticCallback

A interface OptimisticCallback pode ser utilizada para criar e processar um campo de versão que está associado a um objeto Value especificado. Em muitos casos, a utilização do objeto Value diretamente para determinar se outro cliente do cache modificou o valor desde que ele foi recuperado é muito ineficiente e está sujeita a erros. Uma alternativa é fornecer outro campo que representa o estado do objeto Value. A intenção da interface OptimisticCallback é fornecer um objeto Versioned Value alternativo que representa o objeto Value. A seguir está uma configuração de amostra da interface OptimisticCallback:

Configuração Programática

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

Configuração XML

O exemplo a seguir resulta na mesma configuração que está no exemplo programático anterior:

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

Índices

Um MapIndexPlugin, ou um Index para abreviar, é uma opção que foi utilizada pelo BackingMap para construir índice com base no atributo especificado do objeto armazenado. O índice permite que aplicativos localizem objetos por um valor específico ou um intervalo de valores. Para utilizar o índice, os Aplicativos precisam obter o objeto de índice do método getIndex() da interface ObjectMap e lançá-lo na interface de índice correta, como MapIndex ou MapRangeIndex ou uma interface de índice customizado.

No momento, o recurso de indexação é suportado apenas no cache local, não no cache distribuído. Se estiver tentando desempenhar uma operação de indexação em um cache distribuído, ocorrerá uma UnsupportedOperationException.

Existem dois tipos de índice: índice estático e dinâmico. Os índices estáticos podem ser criados por meio de configuração Programática e de configuração XML. Os índices dinâmicos podem ser criados apenas programaticamente.

Configuração Programática

O exemplo de código a seguir demonstra como incluir índice estático em uma instância do BackingMap:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("indexSampleGrid");
BackingMap personBackingMap= objectGrid.getMap("person");
//utilize a classe com.ibm.websphere.objectgrid.plugins.index.HashIndex
//interna como a classe de plugin do índice.
HashIndex mapIndexPlugin = new HashIndex();
mapIndexPlugin.setName("CODE");
mapIndexPlugin.setAttributeName("EmployeeCode");
mapIndexPlugin.setRangeIndex(true);
personBackingMap.addMapIndexPlugin(mapIndexPlugin); //Nota: a configuração de
Index anterior que armazenou o objeto possui
// um atributo denominado EmployeeCode e um método denominado getEmployeeCode()
//que retorna o valor do atributo EmployeeCode.
:
```

O exemplo de código a seguir demonstra como criar um índice dinâmico em uma instância de BackingMap:

```
class DynamicIndexCallbackImpl implements
com.ibm.websphere.objectgrid.plugins.index.DynamicIndexCallback { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("indexSampleGrid");
BackingMap personBackingMap= objectGrid.getMap("person");
objectGrid.initiallize();
:
//inserir, atualizar ou remover dados
//O índice dinâmico pode ser criado após a inicialização da instância do ObjectGrid
//de abrangência
//Se for necessário criar um índice dinâmico, crie-o sem
//DynamicIndexCallback
personBackingMap.createDynamicIndex("CODE2", true, "employeeCode", null);
:
//Outra opção é criar um índice dinâmico com DynamicIndexCallback
//Suponha que exista uma classe DynamicIndexCallbackImpl que implementa
//a interface DynamicIndexCallback
personBackingMap.createDynamicIndex("CODE3", true, "employeeCode",
new DynamicIndexCallbackImpl());
:
```

Configuração XML

O exemplo a seguir resulta na mesma configuração que está no exemplo programático anterior de índice estático:

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

Consulte o tópico Indexação para obter informações adicionais.

Visão Geral do Modelo de Programação do Sistema: Recursos da Interface Session

A interface Session possui vários recursos para interações mais extensíveis com o ObjectGrid. Cada uma das seções a seguir descreve o recurso e fornece alguns trechos de código breves para o cenário de uso.

Para obter informações adicionais sobre a interface Session, consulte “Interface Session” na página 116.

Modo Sem Gravação Direta

ÀS vezes, os aplicativos desejam apenas aplicar alterações no mapa base, mas não no Loader. O método beginNoWriteThrough da interface Session foi projetado

para alcançar este objetivo. O método isWriteThroughEnabled da interface Session pode ser utilizado para verificar se a sessão atual está gravando no Loader de backend. Isto pode ser útil para outros usuários do objeto de Sessão para saber qual tipo de sessão está sendo processada no momento. O exemplo a seguir ativa o modo sem gravação direta:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
boolean isWriteThroughEnabled = session.isWriteThroughEnabled();
// fazer atualizações no mapa ...
session.commit();
:
```

Enviar Dados apenas para o Loader

Os aplicativos podem aplicar alterações locais da sessão ao Loader sem confirmar estas alterações permanentemente, chamando o método flush, como no exemplo a seguir:

```
:
Session session = objectGrid.getSession();
session.begin();
// fazer algumas alterações...
session.flush(); // enviar estas alterações para o Loader, mas ainda não confirmar
// fazer mais algumas alterações...
session.commit();
:
```

Método processLogSequence

O método processLogSequence é utilizado para processar um LogSequence. Cada LogElement no LogSequence é examinado e a operação apropriada, como as operações de inserção, atualização, invalidação, é desempenhada no BackingMap identificado pelo LogSequence MapName. Uma Sessão do ObjectGrid deve estar ativa antes da chamada deste método. O responsável pela chamada 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.

A principal utilização deste método é processar um LogSequence que foi recebido por uma JVM remota. Por exemplo, utilizando o suporte à confirmação distribuída, os LogSequences associados a uma determinada sessão confirmada são, então, distribuídos para outros ObjectGrids de atendimento em outras JVMs (Java Virtual Machines). Depois de receber os LogSequences na JVM remota, o listener pode iniciar uma Sessão utilizando o método beginNoWriteThrough, chamar este método processLogSequence e, em seguida, desempenhar o método commit na Sessão. Este é um exemplo:

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

Monitoração do Desempenho

Opcionalmente, os mapas podem ser instrumentados para monitoramento de desempenho durante a execução no WebSphere Application Server. O método `setTransactionType` está disponível em uma Sessão para configuração e utilização dos recursos de monitoramento de desempenho. Consulte “Monitorando o Desempenho do ObjectGrid com a PMI (Performance Monitoring Infrastructure) do WebSphere Application Server” na página 295 para obter maiores informações.

Visão Geral do Modelo de Programação do Sistema: Recursos da Interface ObjectMap

A interface ObjectMap possui vários recursos e interações mais extensíveis com o ObjectGrid.

Para obter informações adicionais sobre a interface ObjectMap, consulte “Interfaces ObjectMap e JavaMap” na página 120.

A Interface JavaMap e a Interface java.util.Map

Para aplicativos que desejam utilizar a interface java.util.Map, o ObjectMap possui o método getJavaMap para que os aplicativos possam obter a implementação da interface java.util.Map suportada pelo ObjectMap. A instância do Mapa retornada pode então ser lançada para a interface JavaMap, que estende a interface java.util.Map. A interface JavaMap possui as mesmas assinaturas de método que ObjectMap, mas com diferente manipulação de exceção. A interface JavaMap estende a interface java.util.Map, portanto, todas as exceções são instâncias da classe java.lang.RuntimeException. Como a interface JavaMap estende a interface java.util.Map, é fácil utilizar rapidamente o ObjectGrid com um aplicativo existente que utiliza uma interface java.util.Map para o armazenamento em cache de objetos. O trecho de código é o seguinte:

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

Extensões do Mapa

A interface ObjectMap também fornece recursos funcionais adicionais além dos recursos de exceções verificados. Por exemplo, um usuário pode especificar que uma determinada entrada do mapa seja atualizada com o método getForUpdate, que indica ao tempo de execução do ObjectGrid e ao plug-in do Loader que a entrada foi bloqueada durante o processamento, se apropriado. O processamento de batch é outro recurso adicional com os métodos getAll, putAll e removeAll. Para obter informações adicionais sobre estes métodos, consulte a documentação da API.

Processamento de Palavras-chave

A maioria das operações do mapa possuem a versão do parâmetro de palavra-chave, como insert, get, getForUpdate, put, remove e invalidate. Para facilidade de utilização, o método setDefaultKeyword também é fornecido. Este método associa entradas a uma palavra-chave sem utilizar a versão de palavra-chave da operação do mapa. A seguir está um exemplo de palavra-chave:

```
:
// setDefaultKeyword
session.begin();
objectMap.setDefaultKeyword("New York");
```

```
Person p = (Person) objectMap.get("Billy"); // A entrada "Billy"
tem a  palavra-chave "
New York"
p = (Person) objectMap.get("Bob", "Los Angeles"); // A entrada "Bob"
//tem a palavra-chave "Los Angeles"
objectMap.setDefaultKeyword(null);
p = (Person) objectMap.get("Jimmy"); // A entrada "Jimmy" ;
não tem palavra-chave
session.commit();
:
// versão do parâmetro de palavra-chave da operação insert
session.begin();
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person, "Rochester"); // "BillyBob" tem
//a palavra-chave "Rochester"
session.commit();
:
```

Consulte “Palavras-chave” na página 124 para obter maiores informações.

Método do Modo de Cópia

O método setCopyMode permite que o modo de cópia para o Mapa seja substituído neste mapa apenas por esta sessão ou transação. Este método permite que um aplicativo utilize um modo de cópia adequado em uma base por sessão, conforme necessário. O modo de cópia não pode ser alterado durante uma sessão ativa. Existe um método clearCopyMode correspondente que reconfigura o modo de cópia de volta ao definido no BackingMap. É possível chamar este método apenas quando não existirem sessões ativas. A seguir está um exemplo de configuração do modo de cópia:

```
:
objectMap.setCopyMode(CopyMode.COPY_ON_READ, null);
session.begin();
// modificar objectMap ...
session.commit();
objectMap.clearCopyMode(); // reconfigurar CopyMode como a configuração do
BackingMap session.begin();
// modificar objectMap ...
session.commit();
:
```

Para obter informações adicionais, consulte os tópicos “Plug-in ObjectTransformer” na página 213 e Capítulo 11, “Boas Práticas de Desempenho do ObjectGrid”, na página 329.

Configurações do Evictor

É possível substituir o valor de tempo limite TimeToLive para o evictor TimeToLive interno no nível do ObjectMap. O método setTimeToLive estabelece o número de segundos de existência de qualquer entrada de cache especificada. Quando modificado, o valor TimeToLive anterior é retornado. Este valor TimeToLive é o tempo mínimo que uma entrada permanece no cache antes de ser considerada para evicção e indica ao evictor TimeToLive interno quanto tempo uma entrada deve permanecer após a hora do último acesso. O novo valor TimeToLive se aplica apenas às entradas do ObjectMap acessadas pela transação iniciada pelo objeto de Sessão que foi utilizado para obter a instância do ObjectMap. O novo valor TimeToLive se aplica a qualquer transação que esteja em andamento para a Sessão e futuras transações que são executadas pela Sessão. O novo valor TimeToLive não afeta entradas de uma instância do ObjectMap que são acessadas por uma transação iniciada por alguma outra Sessão. Chamando este método no

ObjectMap, qualquer valor anterior configurado pelo método setTimeToLive no BackingMap é substituído por esta instância do ObjectMap. Este é um exemplo:

```
:
session.begin();
int oldTTL = objectMap.setTimeToLive(60); // configurar TTL como 60 segundos
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person); // a entrada "BillyBob" terá um TTL
//de 60 segundos
session.commit();
:
objectMap.setTimeToLive(oldTTL); // reconfigurar TTL como o valor original
Person person2 = new Person("Angelina", "Jolie", "somewhere");
objectMap.insert("Brad", person2); // A entrada "Brad" utilizará o
valor TTL original
:
```

Para obter informações adicionais, consulte o tópico “Evictors” na página 191.

Capítulo 5. Amostras do ObjectGrid

Este tópico descreve as amostras do ObjectGrid que são fornecidas durante a instalação do produto WebSphere Extended Deployment.

Visão Geral

Várias amostras do ObjectGrid ilustram a integração com aplicativos J2EE (Java 2 Platform, Enterprise Edition) e o recurso de particionamento (WPF). Este tópico descreve cada uma das amostras, os recursos demonstrados por cada amostra, o local de cada amostra e os ambientes nos quais a amostra é executada.

Este tópico descreve as amostras fornecidas durante a instalação do WebSphere Extended Deployment. Serão fornecidas outras amostras relacionadas à utilização da integração de JMS (Java Message Service) e à integração do ObjectGrid com outras estruturas de código aberto no seguinte endereço da Web:http://www-1.ibm.com/support/docview.wss?uid=swg27006432.

Amostras

- **ObjectGridSamplesSA :** Esta amostra é um conjunto de exemplos J2SE (Java 2 Platform, Standard Edition) que estão empacotados no arquivo `objectgridSamples.jar` para demonstração das funções do ObjectGrid. Estas amostras do J2SE podem ser executadas em um ambiente do J2SE. O arquivo `objectgridSamples.jar` contém o arquivo `SamplesGuide.htm`, que possui instruções para a execução destas amostras.
- **ObjectGridSample :** Esta amostra é um exemplo do J2EE que demonstra como os servlets e beans corporativos de Sessão utilizam as funções do ObjectGrid. Esta amostra é fornecida no arquivo EAR (Enterprise Archive) `ObjectGridSample.ear`. O arquivo `ObjectGridSample.ear` contém o arquivo `readme.txt`, que possui instruções para configuração e execução desta amostra.
- **ObjectGridPartitionCluster :** Esta é uma amostra do J2EE para demonstrar como o WPF e o ObjectGrid funcionam juntos e como utilizar o ObjectGridEventListener para propagar alterações do objeto e como ativar o roteamento baseado em contexto para manter a integridade e consistência do ObjectGrid. Esta amostra é fornecida no arquivo EAR `D_ObjectGridPartitionClusterSample.ear`. O arquivo `D_ObjectGridPartitionClusterSample.ear` contém o arquivo `readme.txt`, que possui instruções para configuração e execução desta amostra.
- **ObjectGridJMSSamples:** Este é um conjunto de amostras do J2EE empacotadas no arquivo ObjectGridJMSSamples.zip que demonstram como utilizar a função do JMS para transmitir alterações em uma instância do ObjectGrid para outra instância do ObjectGrid em uma única JVM ou em um ambiente em cluster. Estas amostras do J2EE estão disponíveis apenas na Web no seguinte endereço da Web: http://www-1.ibm.com/support/docview.wss?uid=swg27006432.

Funcionalidade de Amostra

Tabela 2. Funcionalidades de Amostra

Área Funcional	Amostra SamplesSA do ObjectGrid	Amostra Sample do ObjectGrid	Amostra Cluster do ObjectGridPartition	Amostra JMSSamples do ObjectGrid
ObjectGrid EventListener			x	x
Retorno de Chamada de Transação	x	x	x	
Loader	x	x	x	
Listener MapEvent	x			
Transformador de Objeto	x	x	x	x
Retorno de Chamada Otimista	x	x	x	
Modo de Cópia de BackingMap	x	x		
Invalidação Distribuída			x	x
Atualização Distribuída			x	x
Processamento de LogSequence				x
Recurso de Particionamento (WPF)			x	
Java Message Service (JMS)				x
Índice do Mapa	x			
Cluster do ObjectGrid	x	x		
Contexto ClusterClient do ObjectGrid	x	x		
ObjectGrid Distribuído	x	x		
Gerenciamento do ObjectGrid	x			
Segurança do ObjectGrid	x		x	

Local

Após a instalação do WebSphere Extended Deployment, os seguintes arquivos .jar serão localizados nos seguintes diretórios:

Tabela 3. Locais de Amostra

Amostra	Local
ObjectGridSamplesSA	*install_root*\optionalLibraries\ObjectGrid\objectgridSamples.jar
ObjectGridSample	*install_root*\installableApps\ObjectGridSample.ear
ObjectGridPartitionCluster	*install_root*\installableApps\D_ObjectGridPartitionClusterSample.ear

As versões atualizadas das amostras fornecidas listadas e amostras adicionais como ObjectGridJMSSamples, podem ser localizadas na Web no seguinte endereço da Web: http://www-1.ibm.com/support/docview.wss?uid=swg27006432. Também é possível localizar artigos no IBM DeveloperWorks que descrevem tópicos de interesse no seguinte endereço da Web: http://www.ibm.com/developerworks. Procure **ObjectGrid**.

Ambientes de Amostra

Algumas amostras podem ser executadas em um ambiente do J2SE, mas algumas devem ser executadas em um ambiente do J2EE. Algumas podem ser executadas em uma única instância do servidor, outras devem ser executadas em um cluster. A tabela a seguir mostra o ambiente de execução das amostras.

Restrição: Se estiver utilizando o ObjectGrid em um ambiente do WebSphere Extended Deployment Versão 6.0, também poderá utilizar o ObjectGrid em um ambiente J2SE (Java 2 Platform, Standard Edition) Versão 1.4.2 ou superior ou em um ambiente do WebSphere Application Server Versão 6.02 ou superior com organizações de licença adicionais. Entre em contato com seu representante de vendas para obter detalhes.

Tabela 4. Ambientes de Execução de Amostra

		ObjectGrid SamplesSA	Amostra do ObjectGrid	Cluster de Partição do ObjectGrid	ObjectGrid JMSSamples
J2SE	Eclipse	x			
	linha de comandos	x			
WebSphere Application Server Versão 6.0.*x*	servidor único		x		x
	Cluster		x		x
	UTE (Unit Test Environment) do Rational Application Developer		x		
WebSphere Application Server Versão 5.0.2.*x* e Versão 5.1.*x*	servidor único		x		
	Cluster		x		

Tabela 4. Ambientes de Execução de Amostra (continuação)

		ObjectGrid SamplesSA	**Amostra do ObjectGrid**	**Cluster de Partição do ObjectGrid**	**ObjectGrid JMSSamples**
WebSphere Extended Deployment Versão 6.0.*x*	servidor único		x		x
	Cluster		x	x	x

Capítulo 6. Pacote do ObjectGrid

É possível acessar pacotes do ObjectGrid de duas maneiras: instalando o WebSphere Extended Deployment ou instalando o mixed server environment.

Pacote do ObjectGrid do WebSphere Extended Deployment Versão 6.0.1

Ao instalar o WebSphere Extended Deployment Versão 6.0.1 ou posterior, os seguintes arquivos de tempo de execução são instalados:

Tabela 5. Arquivos de tempo de execução do ObjectGrid do WebSphere Extended Deployment

Nome do Arquivo	Ambiente de Tempo de Execução	Descrição
/lib/asm.jar /lib/cglib.jar	Local, cliente e servidor	Estes jars servem para a função de utilitário cglib ao utilizar a cópia no modo de cópia de gravação.
/lib/wsobjectgrid.jar	Local, cliente e servidor	Este arquivo JAR (Java Archive) contém o tempo de execução local, de cliente e de servidor do ObjectGrid para utilização no ambiente do WebSphere Extended Deployment Versão 6.0.1 e posterior.

Pacote do ObjectGrid do WebSphere Extended Deployment para Mixed Server Environment Versão 6.0.1

Ao instalar o WebSphere Extended Deployment para Mixed Server Environment, os seguintes arquivos de tempo de execução são instalados:

Tabela 6. Arquivos de tempo de execução do ObjectGrid do WebSphere Extended Deployment para Mixed Server Environment

Nome do Arquivo	Ambiente de Tempo de Execução	Descrição
/ObjectGrid/lib/asm.jar /ObjectGrid/lib/cglib.jar	Local, cliente e servidor	Estes arquivos JAR servem para a função de utilitário cglib ao utilizar a cópia no modo de cópia de gravação. Inclua estes arquivos JAR em seu CLASSPATH se estiver utilizando a cópia no modo de cópia de gravação e desejar utilizar a função de proxy cglib. Estes arquivos JAR são automaticamente incluídos no tempo de execução do servidor. Inclua estes arquivos no tempo de execução do ObjectGrid do cliente ou local.
/ObjectGrid/lib/mx4j.jar /ObjectGrid/lib/mx4j-remote.jar /ObjectGrid/lib/mx4j-tools.jar	Cliente e Servidor Gateway de Gerenciamento	Estes arquivos JAR servem para a função de utilitário mx4j utilizada pelo servidor gateway de gerenciamento do ObjectGrid, bem como programas clientes gateways de gerenciamento. Inclua estes arquivos JAR no CLASSPATH do cliente gateway de gerenciamento quando estiver conectando-se ao servidor gateway de gerenciamento.
/ObjectGrid/lib/objectgrid.jar	Local, cliente e servidor	Este arquivo JAR é utilizado pelo tempo de execução do servidor independente para J2SE (Java 2 Platform, Standard Edition) Versão 1.4.2 e posterior. Também é possível utilizar este arquivo JAR para tempo de execução de cliente e local para o J2SE versão 1.3 e posterior.

Tabela 6. Arquivos de tempo de execução do ObjectGrid do WebSphere Extended Deployment para Mixed Server Environment (continuação)

Nome do Arquivo	Ambiente de Tempo de Execução	Descrição
/ObjectGrid/lib/ogclient.jar	Local e cliente	Este arquivo JAR contém apenas os tempos de execução do ObjectGrid local e de cliente durante execução fora de um processo do WebSphere. Pode ser recomendável utilizar este arquivo JAR sobre o objectgrid.jar devido à base menor. É possível utilizar este jar com o J2SE versão 1.3 e posterior.
/ObjectGrid/lib/wsobjectgrid.jar	Local, cliente e servidor	Utilize este arquivo JAR no WebSphere Application Server Versão 6.0.2 ou posterior. Este arquivo JAR é o mesmo arquivo JAR instalado com o WebSphere Extended Deployment.
/ObjectGrid/lib/wsogclient.jar	Local e cliente	Utilize este arquivo JAR para o WebSphere Application Server Versão 5.0.2 e posterior. Este arquivo JAR contém apenas os tempos de execução do ObjectGrid local e de cliente.

Considerações para Utilizar o ObjectGrid com o J2SE Versão 1.3

Ao utilizar o arquivo ogclient.jar ou objectgrid.jar em um ambiente J2SE Versão 1.3.x, é necessário incluir os seguintes requisitos em seu ambiente J2SE 1.3.x para torná-lo funcional com o ObjectGrid:

- **Implementação do JAAS (Java Authentication and Authorization Service)**. O J2SE Versão 1.3 não incluía o objeto javax.security.Subject, uma parte da especificação do JAAS. O ObjectGrid e as interfaces Session requerem este objeto. Coloque a implementação do JAAS no diretório jre/lib/ext de extensões Java.
- **Implementação de JAXP (Java API para XML Processing)**. Se estiver transmitindo arquivos XML para o tempo de execução do ObjectGrid, é necessária uma implementação de JAXP para que o tempo de execução do ObjectGrid analise o arquivo XML. O ObjectGrid utiliza a validação de sintaxe da definição de esquema XML, portanto, é requerida uma implementação que suporta a validação de esquema. O produto Apache Xerces é um exemplo de uma implementação que suporta validação de esquema.
- **Implementação de JSSE (Java Secure Socket Extension)**. Quando estiver utilizando o tempo de execução do cliente, é requerida uma implementação de JSSE. Verifique se a implementação do JSSE utilizada é compatível com a

implementação de JDK (Java Development Kit) do servidor do ObjectGrid se estiver em execução com a segurança ativada.

Se o tempo de execução do ObjectGrid local ou de cliente estiver incluído em um ambiente compatível com o J2EE Versão 1.3 que utiliza o J2SE Versão 1.3, todos estes requisitos serão atendidos, porque todas as implementações de especificação requeridas eram requeridas como parte do J2EE Versão 1.3.

Capítulo 7. Visão Geral do Gerenciamento de Sistemas

Com o release do WebSphere Extended Deployment Versão 6.0.1, o ObjectGrid fornece uma infra-estrutura de gerenciamento de sistemas para permitir que os usuários monitorem e administrem ambientes do ObjectGrid. A arquitetura de gerenciamento de sistemas é uma abordagem de três camadas: um cliente do usuário conecta-se ao servidor Gateway de Gerenciamento, que faz uma conexão do cliente do ObjectGrid com um cluster do ObjectGrid.

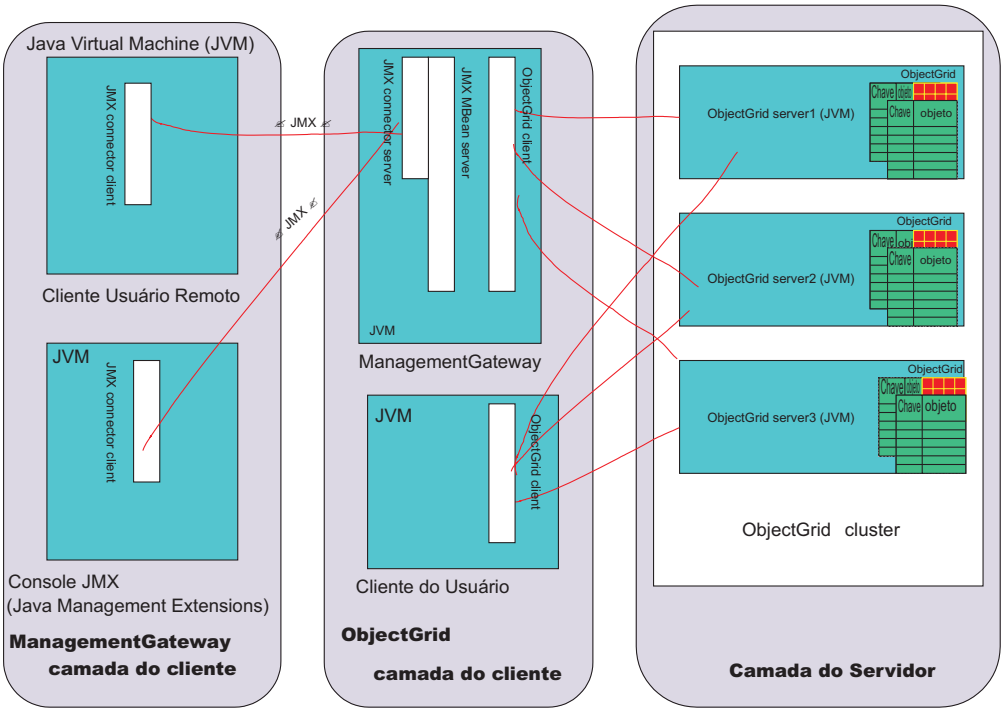


Figura 14. Diagrama de Gerenciamento de Sistemas

A *camada do cliente gateway de gerenciamento* contém qualquer programa que utiliza JMX (Java Management Extensions) para conectar-se ao servidor Gateway de Gerenciamento. Qualquer console JMX de terceiros, bem como um programa cliente que utiliza APIs MX4J são incluídos. A *camada do cliente do ObjectGrid* consiste no servidor gateway de gerenciamento. O gateway de gerenciamento age como um servidor para a camada do cliente gateway de gerenciamento e como um cliente para um cluster do ObjectGrid em funcionamento na *camada do servidor*. Além disso, o programa cliente pode chamar as mesmas APIs que o servidor gateway de gerenciamento chama se o usuário não desejar envolver o JMX. Por último, a Camada do Servidor consiste em um cluster do ObjectGrid.

O gateway de gerenciamento hospeda um conjunto de beans gerenciados (MBeans) e utiliza o JMX para administrar e monitorar o ambiente do ObjectGrid e é implementado pelo projeto de código aberto do MX4J. O MX4J é fornecido com o ObjectGrid.

O JMX e o modelo de administração do MBean do ObjectGrid foram criados para tirar vantagem dos vários consoles JMX que estão disponíveis para administrar ambientes do JMX. Você pode juntar painéis utilizando o console JMX de sua

escolha. Os consoles podem ser conectados aos MBeans em execução na JVM (Java Virtual Machine) do ManagementGateway e os painéis podem ser montados utilizando estes MBeans. Os consoles oferecem históricos gráficos ou gráficos de valores numéricos e de cadeia.

Existem duas opções para executar comandos de gerenciamento de sistemas.

- Chamar qualquer comando por meio da infra-estrutura de cliente/servidor adequado utilizando a interface ObjectGridAdministrator.
- Utilizar o JMX para chamar estes mesmos comandos, com os MBeans do ObjectGrid agindo como um wrapper para ObjectGridAdministrator.

Iniciar o Processo ManagementGateway

Após o início de um cluster (ou servidor único), o processo ManagementGateway pode ser iniciado. O ManagementGateway age como um servidor para pedidos do cliente usuário e um cliente do ObjectGrid para o cluster ao qual ele está conectado.

Opções

A seguir está uma lista de opções que podem ser transmitidas para o processo ManagementGateway:

- **connectorPort** (requerido) - Especifica o número da porta para o conector JMX.
- **clusterHost** (requerido) - Especifica o nome do host de um dos servidores no cluster do ObjectGrid.
- **clusterPort** (requerido) - Especifica a porta de acesso do cliente de um dos servidores no cluster do ObjectGrid.
- **clusterName** (requerido) - Especifica o nome do cluster do ObjectGrid.
- **traceEnabled** - Especifica se o rastreio está ativado para o processo ManagementGateway.
- **traceSpec** - Indica a especificação de rastreio do ManagementGateway.
- **traceFile** - Especifica o arquivo para o qual a saída de rastreio será impressa.
- **sslEnabled** - Especifica se o SSL está ativado no ManagementGateway.
- **csConfig** - Especifica o objeto ClientSecurityConfiguration para o ManagementGateway seguro.
- **refreshInterval** - Especifica o intervalo de tempo no qual o gateway de gerenciamento atualiza os atributos de MBean.

Interface ManagementGateway

O processo ManagementGateway precisa ser iniciado para disponibilizar os MBeans. A interface ManagementGateway mostra quais opções podem ser transmitidas ao iniciar o ManagementGateway.

```
public interface ManagementGateway {
/**
* Iniciar o servidor do conector MBean JMX
*/
void startConnector();
/**
* Parar o servidor do conector MBean JMX
*/
void stopConnector();
/**
* @param Porta do conector JMX
*/
```

```
void setConnectorPort(int port);
/**
* @return porta do conector JMX
*/
int getConnectorPort();
/**
* @param a {@link com.ibm.websphere.objectgrid.security.config.
ClientSecurityConfiguration} object.
*/
void setCsConfig(ClientSecurityConfiguration csConfig);
/**
* @return a {@link com.ibm.websphere.objectgrid.security.config.
ClientSecurityConfiguration} object.
*/
ClientSecurityConfiguration getCsConfig();
/**
* @param porta do servidor ao qual o cliente gateway se conecta
*/
void setPort(String port);
/**
* @return porta do servidor ao qual o cliente gateway se conecta
*/
String getPort();
/**
* @param host do servidor ao qual o cliente gateway se conecta
*/
void setHost(String host);
/**
* @return host do servidor ao qual o cliente gateway se conecta
*/
String getHost();
/**
* @param true booleano se o SSL estiver ativado no gateway
*/
void setSSLEnabled(boolean sslEnabled);
/**
* @return true booleano se o SSL estiver ativado no gateway
*/
boolean getSSLEnabled();
/**
* @param cluster ao qual o cliente gateway se conecta
*/
void setClusterName(String clusterName);
/**
* @return cluster ao qual o cliente gateway se conecta
*/
String getClusterName();
/**
* @param true se o rastreio estiver ativado no gateway
*/
void setTraceEnabled(boolean traceEnabled);
/**
* @return true se o rastreio estiver ativado no gateway
*/
boolean getTraceEnabled();
/**
* @param especificação de rastreio no gateway
*/
void setTraceSpec(String traceSpec);
/**
* @return especificação de rastreio no gateway
*/
String getTraceSpec();
/**
* @param arquivo de saída de rastreio para rastreio do gateway
*/
void setTraceFile(String traceFile);
```

```
/**
* @return arquivo de saída de rastreio para rastreio do gateway
*/
String getTraceFile();
/**
* @param intervalo (em segundos) para atualizar atributos de MBean do cluster
*/
void setRefreshInterval(int refreshInterval);
/**
* @return intervalo (em segundos) para atualizar atributos de MBean do cluster
*/
int getRefreshInterval();
}
```

Opções para Iniciar o Processo ManagementGateway

Utilizando o ManagementGatewayFactory Programaticamente

A seguir está o código de amostra para utilizar esta opção:

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

Este código deve estar em um programa de usuário executado após o início do cluster do ObjectGrid ao qual você está tentando conectar-se.

Na linha de comandos com o arquivo em batch startManagementGateway

Este é um exemplo:

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

Para obter informações adicionais sobre scripts startManagementGateway, consulte “Iniciar o Servidor Gateway de Gerenciamento” na página 88.

O ManagementGateway age como um servidor para um processo de cliente que deseja fazer chamadas JMX, mas também como um cliente do ObjectGrid para o cluster ao qual o usuário deseja conectar-se. Após o início do ManagementGateway, é estabelecida uma conexão com o cluster e o serviço Conector JMX se torna disponível. É possível então acessar o serviço Conector JMX por meio das APIs MX4J ou J2SE (Java 2 Platform, Standard Edition) Versão 5.

Exemplo:

A seguir está o código de amostra de como obter um MapStatsModule de um servidor chamado Server1 por meio de um ManagementGateway com a porta do conector 1.

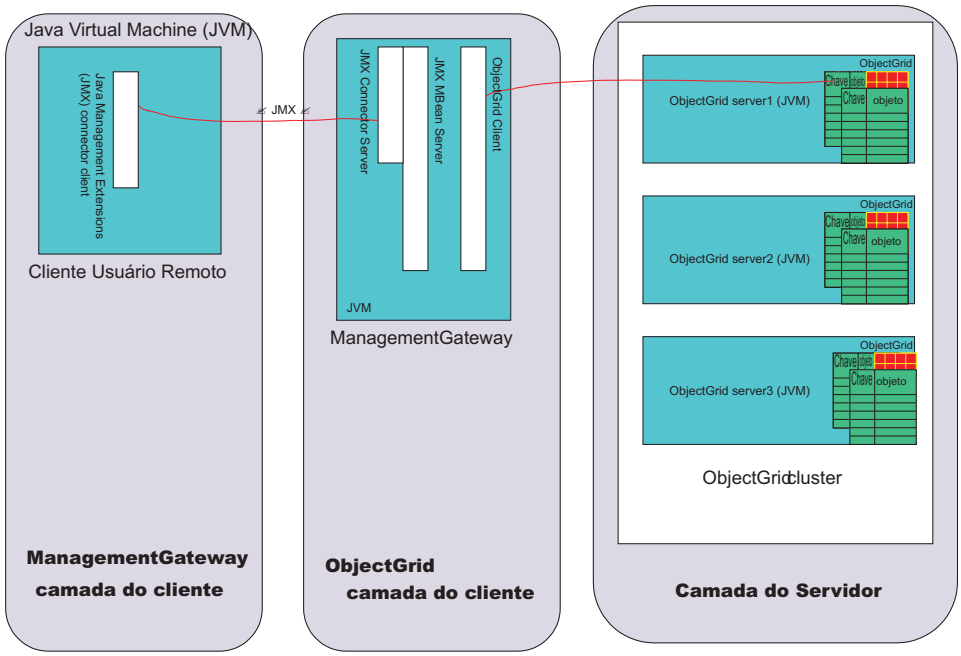


Figura 15. Obter Estatísticas do Mapa do Servidor server1

Execute o seguinte código em um programa de usuário executado na seção do cliente usuário remoto do diagrama anterior:

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

Para parar o servidor server1 por meio do ManagementGateway:

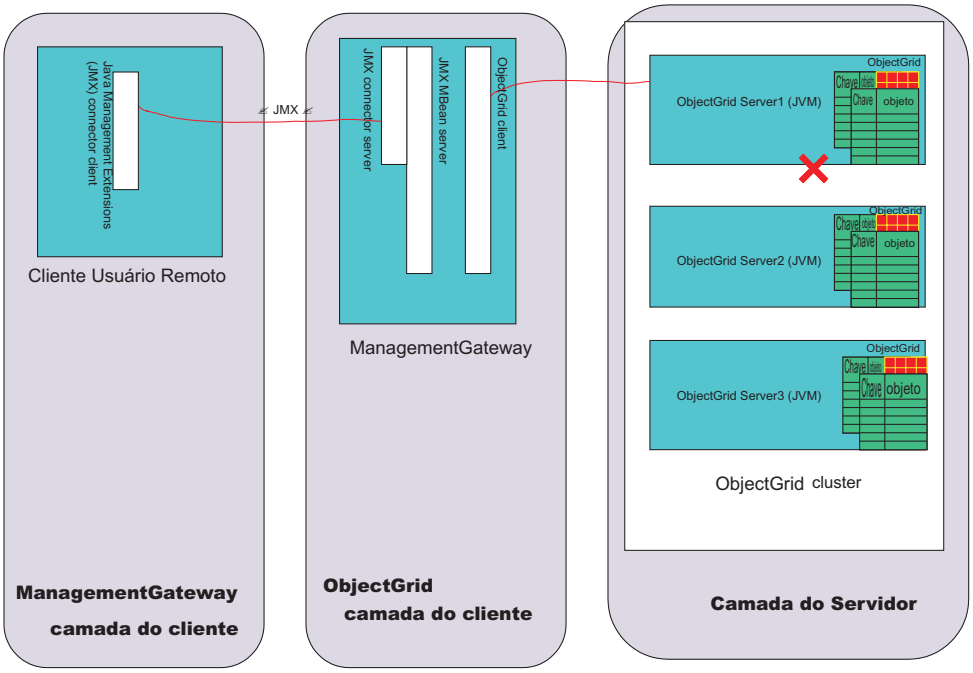


Figura 16. Parar o Servidor server1

Execute o seguinte código em um programa de usuário executado no cliente usuário remoto do diagrama anterior:

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

Depois de executar a amostra de código anterior, o servidor server1 é parado. Depois que o servidor server1 for parado, ele não poderá ser reiniciado com o ManagementGateway. O servidor pode ser reiniciado utilizando a linha de comandos. Consulte “Parar Servidores do ObjectGrid” na página 87 para obter maiores informações.

MBeans (Beans Gerenciados) do ObjectGrid

Existem cinco tipos de MBeans no ambiente do ObjectGrid. Cada MBean se refere a uma entidade específica, como um mapa, grade de objetos, servidor, grupo de replicação ou membro do grupo de replicação e possui atributos e operações.

Cada MBean no ObjectGrid tem métodos getxxx que representam valores de atributos. Estes métodos getxxx não podem ser chamados diretamente de um programa de usuário. Isto ocorre porque a especificação JMX (Java Management Extensions) trata atributos de maneira diferente de operações. Os atributos podem

ser visualizados por meio de qualquer console JMX de terceiros e as operações podem ser desempenhadas por meio de um programa de usuário ou de um console JMX de terceiros.

Mbean MapMbean

O MapMBean permite que o usuário monitore as estatísticas de cada mapa definido para o cluster. Cada mapa possui as seguintes estatísticas associadas a ele:

- Tempo de atualização de batch (mín/máx/médio/total)
- Count
- Taxa de acesso

Além disso, como os mapas podem ser particionados nos servidores, é possível definir o escopo de estatísticas do mapa para um servidor ou membro do grupo de replicação específico. Também é possível mapear estatísticas para todo o cluster. O ObjectName para um MapMBean pode ser especificado de várias maneiras:

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

Utilize uma configuração de exemplo com o ObjectGrid OG1, um mapa Map1 com dois servidores no grupo de replicação RG1, server1 e server2. Suponha também que o servidor server1 seja o primário e o servidor server2 seja uma réplica. Para obter as estatísticas para o mapa Map1 no primário, utilize qualquer um destes ObjectNames:

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

Em qualquer ObjectName para MBeans do ObjectGrid, quando IDX=0, ele se refere ao primário do grupo de replicação. IDX=1-10 se refere a réplicas para o grupo de replicação.

A seguir está uma listagem da interface MapMBean:

```
public interface MapMBean {
/**
* Operação para obter o MapStatsModule associado ao MBean.
*
* @return MapStatsModule
*/
MapStatsModule retrieveStatsModule();
/**
* A operação vai para o servidor para obter StatsModule apenas se
* StatsModule não estiver armazenado em cache no ObjectGridAdministrator.
*
*/
void refreshStatsModule();
/**
* Mapa.
*
* @return nome do mapa
*/
String getMapName();
/**
* ObjectGrid que contém o mapa.
*
* @return nome da grade de objetos
```

```
*/
String getObjectGridName();
/**
* Nome do servidor do membro do grupo de replicação para o mapa.
*
* @return nome do servidor do membro do grupo de replicação
*/
String getServerName();
/**
* Nome do grupo de replicação para o mapa.
*
* @return nome do grupo de replicação
*/
String getReplicationGroup();
/**
* Índice do membro do grupo de replicação para o mapa.
*
* @return índice do membro do grupo de replicação
*/
int getIndex();
/**
* Atributo MapStatsModule carregado pela
* chamada retrieveStatsModule.
*
* @return Formato de cadeia de MapStatsModule
*/
String getMapStatsModule();
/**
* Atributo de contagem de mapas carregado pela
* chamada retrieveStatsModule.
*
* @return número de entradas no mapa
*/
long getMapCountStatistic();
/**
* Atributo de taxa de acesso carregado pela
* chamada retrieveStatsModule.
*
* @return taxa de acesso para o mapa
*/
double getMapHitRateStatistic();
/**
* Atributo de tempo médio de atualização de batch carregado pela
* chamada retrieveStatsModule.
*
* @return tempo médio de atualização de batch para o mapa
*/
double getMapBatchUpdateMeanTime();
/**
* Atributo de tempo máximo de atualização de batch carregado pela
* chamada retrieveStatsModule.
*
* @return tempo máximo de atualização de batch para o mapa
*/
double getMapBatchUpdateMaxTime();
/**
* Atributo de tempo mínimo de atualização de batch carregado pela
* chamada retrieveStatsModule.
*
* @return tempo mínimo de atualização de batch para o mapa
*/
double getMapBatchUpdateMinTime();
/**
* Atributo de tempo total de atualização de batch carregado pela
* chamada retrieveStatsModule.
*
```

```
* @return tempo total de atualização de batch para o mapa
*/
double getMapBatchUpdateTotalTime();
}
```

MBean ObjectGridMBean

O MBean ObjectGridMBean permite que o usuário monitore as estatísticas para todos os mapas em cada ObjectGrid definido para o cluster. Cada ObjectGrid possui as seguintes estatísticas associadas a ele:

- Tempo de transação (mín/máx/médio/total)
- Count

Além disso, como os ObjectGrids podem ser particionados nos servidores, é possível definir o escopo de estatísticas do ObjectGrid para um servidor ou membro do grupo de replicação específico. Também é possível obter estatísticas do ObjectGrid para todo o cluster. O ObjectName para um ObjectGridMBean pode ser especificado de várias maneiras:

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

A seguir está uma listagem da interface ObjectGridMbean:

```
public interface ObjectGridMBean {
/**
* Operação para obter OGStatsModule associado ao MBean.
*
* @return OGStatsModule
*/
OGStatsModule retrieveStatsModule();
/**
* A operação vai para o servidor para obter StatsModule apenas se
* StatsModule não estiver armazenado em cache no ObjectGridAdministrator.
*
*/
void refreshStatsModule();
/**
* ObjectGrid.
*
* @return nome da grade de objetos
*/
String getObjectGridName();
/**
* Nome do servidor do membro do grupo de replicação para o ObjectGrid.
*
* @return nome do servidor do membro do grupo de replicação
*/
String getServerName();
/**
* Nome do grupo de replicação para o ObjectGrid.
*
* @return nome do grupo de replicação
*/
String getReplicationGroup();
/**
* Índice do membro do grupo de replicação para o ObjectGrid.
*
* @return índice do membro do grupo de replicação
*/
int getIndex();
```

```
/**
* Atributo OGStatsModule carregado pela chamada retrieveStatsModule.
*
* @return Formato de cadeia de OGStatsModule
*/
String getOGStatsModule();
/**
* Atributo de contagem de ObjectGrids carregado pela chamada retrieveStatsModule.
*
* @return número de transações
*/
long getOGCount();
/**
* Atributo de tempo máximo de transação carregado pela chamada retrieveStatsModule.
*
* @return tempo máximo de transação para o ObjectGrid
*/
long getOGMaxTranTime();
/**
* Atributo de tempo mínimo de transação carregado pela chamada retrieveStatsModule.
*
* @return tempo mínimo de transação para o ObjectGrid
*/
long getOGMinTranTime();
/**
* Atributo de tempo médio de transação carregado pela chamada retrieveStatsModule.
*
* @return tempo médio de transação para o ObjectGrid
*/
double getOGMeanTranTime();
/**
* Atributo de tempo total de transação carregado pela chamada retrieveStatsModule.
*
* @return tempo total de transação para o ObjectGrid
*/
long getOGTotalTranTime();
}
```

MBean ServerMBean

O Mbean ServerMBean permite que o usuário desempenhe operações em servidores no cluster. O ObjectName para um ServerMBean pode ser especificado da seguinte maneira:

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

A seguir está uma listagem da interface ServerMBean:

```
public interface ServerMBean {
/**
* Operação para carregar o status de replicação para o servidor.
*
*/
void retrieveReplicationStatus();
/**
* Retornar o nome do servidor.
*
* @return nome do servidor
*/
String getServerName();
/**
* Operação para obter o status do servidor.
*
* @return status do servidor (true se em execução; caso contrário, false)
*/
boolean retrieveServerStatus();
/**
```

```
* Operação para parar o servidor.
*
* @return true se o servidor foi parado; caso contrário, false
*/
boolean stopServer();
/**
* Operação para forçar a parada do servidor.
*
* @return true se o servidor foi parado; caso contrário, false
*/
boolean forceStopServer();
/**
* Operação para parar o cluster do qual o servidor faz parte.
*
* @param determina se os servidores foram parados com force
* @return true se o cluster foi parado; caso contrário, false
*/
boolean stopCluster(Boolean force);
/**
* Operação para modificar a especificação de rastreio para todos os servidores no
* cluster do qual o servidor faz parte.
*
* @param especificação de rastreio
*/
void modifyClusterTraceSpec(String spec);
/**
* Operação para modificar a especificação de rastreio para o servidor.
*
* @param especificação de rastreio
*/
void modifyServerTraceSpec(String spec);
}
```

Mbean ReplicationGroupMBean

O ReplicationGroupMBean permite monitorar o status de todos os membros do grupo de replicação associados a um grupo de replicação específico, incluindo qual servidor é o primário e até dez réplicas. O ObjectName para um ReplicationGroupMBean pode ser especificado:

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

A seguir está uma listagem da interface ReplicationGroupMBean:

```
public interface ReplicationGroupMBean {
/**
* Operação para carregar o status dos atributos do grupo de replicação.
*
*/
String[] retrieveReplicationGroupStatus();
/**
* Atributo ReplicationGroupName.
*
* @return nome do ReplicationGroup
*/
String getReplicationGroupName();
/**
* Atributo primário.
*
* @return nome do Primário
*/
String getPrimary();
/**
* Atributo Replica1.
*
* @return nome do servidor de Replica1
```

```
*/
String getReplica1();
/**
* Atributo Replica2.
*
* @return nome do servidor de Replica2
*/
String getReplica2();
/**
* Atributo Replica3.
*
* @return nome do servidor de Replica3
*/
String getReplica3();
/**
* Atributo Replica4.
*
* @return nome do servidor de Replica4
*/
String getReplica4();
/**
* Atributo Replica5.
*
* @return nome do servidor de Replica5
*/
String getReplica5();
/**
* Atributo Replica6.
*
* @return nome do servidor de Replica6
*/
String getReplica6();
/**
* Atributo Replica7.
*
* @return nome do servidor de Replica7
*/
String getReplica7();
/**
* Atributo Replica8.
*
* @return nome do servidor de Replica8
*/
String getReplica8();
/**
* Atributo Replica9.
*
* @return nome do servidor de Replica9
*/
String getReplica9();
/**
* Atributo Replica10.
*
* @return nome do servidor de Replica10
*/
String getReplica10();
/**
* Todas as réplicas para este grupo de replicação delimitado por vírgulas
*
* @return nomes do servidor para todas as réplicas
*/
String getReplicas();
}
```

Mbean ReplicationGroupMemberMBean

O ReplicationGroupMemberMBean permite monitorar as seguintes estatísticas para um membro do grupo de replicação:

- Status de um membro do grupo de replicação. É possível monitorar membros do primário ou de réplica.
- Proporção de peso de réplica. Esta estatística se aplica apenas aos membros do grupo de replicação que são réplicas. Esta proporção é uma quantificação de quão perto os mapas de uma réplica estão de serem sincronizados com os mapas do primário. Quanto maior a proporção, mais perto uma réplica está de ter as informações atualizadas do primário.

O ObjectName para um ReplicationGroupMemberMBean pode ser especificado das seguintes maneiras:

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

Especificar IDX=0 retorna o primário do grupo de replicação e IDX=1 até 10 são réplicas. A seguir está uma listagem da interface ReplicationGroupMBean:

```
public interface ReplicationGroupMemberMBean {
/**
* Operação para carregar o status dos atributos de membros do grupo de replicação.
*
*/
void retrieveReplicationGroupMemberStatus();
/**
* Operação para carregar o status dos atributos
* de membros do grupo de replicação.
* Utilizará o cache em oposição ao método retrieveReplicationGroupMemberStatus,
* que irá para o servidor para obter status.
*
*/
void refreshReplicationGroupMemberStatus();
/**
* Atributo ReplicationGroupName.
*
* @return nome do ReplicationGroup ao qual este membro pertence
*/
String getReplicationGroupName();
/**
* Status do ReplicationGroupMember: primário/réplica/espera.
*
* @return status do ReplicationGroupMember
*/
String getStatus();
/**
* Estatística que representa a porcentagem de quão perto uma réplica
* está de ficar atualizada com os mapas primários.
*
* @return Estatística de réplica do ReplicationGroupMember
*/
double getReplicaWeightRatio();
/**
* Nome do servidor no qual este ReplicationGroupMember reside.
*
* @return nome do servidor
*/
String getServerName();
/**
* Índice de ReplicationGroupMember.
```

```
*
* @return índice de réplica
*/
int getIndex();
}
```

Capítulo 8. Suporte à Linha de Comandos

Utilize scripts da linha de comandos para gerenciar seus servidores do ObjectGrid.

Um conjunto de arquivos de script é fornecido no diretório `/ObjectGrid/bin` de uma instalação do ambiente do servidor misto. Estes scripts podem ser utilizados para iniciar ou parar um servidor ObjectGrid, iniciar um servidor gateway de gerenciamento e codificar senhas em um arquivo de propriedades. Antes de tentar utilizar os scripts, verifique se a variável de ambiente JAVA_HOME está configurada e se seu valor é uma versão de Java suportada pelo ObjectGrid. É possível atualizar JAVA_HOME no arquivo `setupCmdLine.bat|sh` para apontar para uma versão apropriada de Java, se não desejar alterar sua variável de ambiente globalmente.

Consulte os seguintes tópicos para obter informações adicionais sobre os scripts da linha de comandos:

Iniciar Servidores do ObjectGrid

O script startOgServer é fornecido para iniciar um servidor do ObjectGrid.

Uso

Utilize o arquivo `startOgServer.bat` para iniciar um servidor em uma máquina do Windows. Utilize o arquivo `startOgServer.sh` para iniciar um servidor do ObjectGrid em plataformas Linux e Unix.

Utilizando Arquivos XML

Um arquivo XML válido do ObjectGrid deve ser emparelhado com um arquivo XML de cluster válido para iniciar com êxito um servidor do ObjectGrid. Os arquivos XML podem ser transmitidos para o script startOgServer utilizando um nome de arquivo regular ou uma URL (Localizador Uniforme de Recursos). A opção da URL permite diferentes protocolos além do protocolo de arquivos, por exemplo, http, ftp ou jarfile.

Os argumentos de script startOgServer para iniciar um servidor utilizando arquivos XML são os seguintes:

```
startOgServer.bat <servidor> -objectgridFile <arquivo XML>
| -objectgridUrl
<URL do arquivo XML> -clusterFile <arquivo XML> | -clusterUrl <URL do arquivo XML>
[options]
```

Exemplo:

A seguir estão alguns exemplos de início do servidor server1 do ObjectGrid. Estes exemplos utilizam o arquivo `startOgServer.bat`.

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

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

Os exemplos utilizam o arquivo `universityCluster3Servers.xml`. Como o servidor server1 é especificado como o servidor a ser iniciado, o arquivo `universityCluster3Servers.xml` deve ter um valor serverDefinition com o nome `server1`.

O arquivo `universityCluster3Servers.xml` é o seguinte. Observe serverDefinition de server1 e se seu host é lion.ibm.com. O servidor server1 deve ser iniciado no host lion.ibm.com. Este arquivo também define os servidores server2 e server3. Estes servidores devem ser iniciados nos hosts tiger.ibm.com e bear.ibm.com, respectivamente.

Arquivo *universityCluster3Servers.xml*

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

Auto-inicialização

Quando um servidor do ObjectGrid no cluster estiver disponível, outros servidores no cluster poderão auto-inicializar no servidor disponível. O script startOgServer deve ser fornecido com a porta de acesso ao host e ao cliente de um servidor já disponível para auto-inicialização nele.

Como o primeiro servidor em um cluster é ativado com XML, ele já possui as informações de configuração para todos os servidores no cluster. A auto-inicialização permite ativar o servidor para conectar-se ao servidor disponível e fazer download da configuração.

A seguir estão os argumentos de scripts startOgServer para iniciar um servidor auto-inicializando em um servidor disponível.

```
startOgServer.bat <servidor> -bootstrap <host:porta,host:porta> [options]
```

A seguir estão alguns exemplos de início de um servidor auto-inicializando em outro servidor. Para o primeiro exemplo, suponha que o servidor server1 do arquivo `universityCluster3Servers.xml` tenha sido iniciado utilizando arquivos XML e esteja disponível. Este exemplo mostra como auto-inicializar no servidor server1 para iniciar o servidor server2.

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

Para o próximo exemplo, suponha que o servidor server2 tenha sido iniciado com êxito, mas o servidor server1 tenha se tornado indisponível. Uma lista de combinações separadas por vírgulas de host:port pode ser utilizada durante a auto-inicialização em outro servidor. É feita uma tentativa de contactar cada host e porta na lista até que seja localizado um servidor disponível. No exemplo a seguir, o server3 tentar contactar o host e a porta do servidor server1. No entanto, como o servidor server1 está indisponível neste cenário, a conexão falhará. Utilizando o próximo item da lista, o servidor server3 tenta auto-inicializar no host e porta para o servidor server2. Esta tentativa de auto-inicialização deve ser bem-sucedida, porque o servidor server2 está disponível.

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

Argumentos Opcionais

Existem vários argumentos opcionais que podem ser transmitidos para o script startOgServer. Os argumentos válidos startOgServer são os seguintes.

Opções:

- -traceSpec <especificação de rastreio>
- -traceFile <arquivo de rastreio>
- -serverSecurityFile <arquivo de propriedades de segurança do servidor>
- -timeout <segundos>
- -script <nome do arquivo de script>
- -jvmArgs <argumentos de JVM>

-traceSpec

O argumento -traceSpec pode ser utilizado para configurar uma especificação de rastreio que entra em vigor quase imediatamente durante a inicialização do servidor. Durante a inicialização normal do servidor, a especificação de rastreio não será configurada até que possa ser lida a partir do arquivo XML do cluster ou a partir da configuração auto-inicializada. Se ocorrerem problemas durante a inicialização do servidor, pode ser útil configurar a especificação de rastreio anteriormente.

A seguir está um exemplo de como configurar a opção -traceSpec:

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

-traceFile

O argumento -traceFile pode ser utilizado para especificar um local para a saída de rastreio durante a inicialização do servidor. Após a leitura da configuração para este servidor, suas configurações de rastreio, conforme especificadas pelo arquivo XML do cluster, entram em vigor.

A seguir está um exemplo de como configurar a opção -traceFile:

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

-serverSecurityFile

O argumento -serverSecurityFile pode ser utilizado para transmitir para um servidor seu arquivo de propriedades de segurança. Esta opção é requerida quando a segurança está ativada no servidor. A seguir está um exemplo de como configurar a opção -serverSecurityFile:

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

-timeout

O argumento -timeout pode ser utilizado para especificar a quantidade de tempo, em segundos, em que há permissão para transmissão antes de a ativação do servidor ser abortada. Por padrão, o servidor tem 90 segundos para se tornar disponível a partir do momento em que foi ativado. Se este tempo for muito curto para um cenário específico, utilize o argumento -timeout para configurá-lo como um valor mais apropriado. A seguir está um exemplo de como utilizar o argumento timeout:

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

-script

O argumento -script pode ser utilizado para criar um script que ativa um processo do servidor do ObjectGrid e mantém sua saída no prompt de comandos atual. Em circunstâncias normais, quando um servidor ObjectGrid é ativado, o script startOgServer exibe saída do processo do servidor para o prompt de comandos até que o servidor esteja disponível. Quando o servidor estiver disponível, startOgServer parará de exibir a saída do processo do servidor e será encerrado. Em alguns casos, você pode ativar um processo do servidor que é enviado para o prompt de comandos atual.

Ao especificar um nome de arquivo para o script, não forneça um caminho para o arquivo. O arquivo está localizado no diretório `bin` do caminho OBJECTGRID_HOME. Forneça o nome do arquivo. O arquivo de script criado inclui os argumentos que foram transmitidos para o script startOgServer, portanto, não é necessário fornecer esses mesmos argumentos ao executar o script criado.

A seguir está um exemplo de como utilizar a opção -script:

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

Este exemplo cria um script `universityClusterServer1.bat` no diretório `OBJECTGRID_HOME/bin`. Para executar o script recém-criado, navegue para o diretório apropriado no prompt de comandos, digite o nome do script e pressione **Enter**.

-jvmArgs

O argumento -jvmArgs pode ser utilizado para enviar argumentos para a JVM (Java Virtual Machine) do servidor do ObjectGrid que está sendo ativado. Qualquer argumento que pode ser transmitido para a JVM normalmente pode ser transmitido para o servidor utilizando o argumento -jvmArgs.

O argumento -jvmArgs deve ser o último argumento opcional do ObjectGrid especificado como um argumento para o script startOgServer. Tudo o que vem após o argumento -jvmArgs é transmitido para a JVM do servidor como um argumento da JVM. A seguir está um exemplo de como configurar o argumento -jvmArgs:

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

Se o argumento -jvmArgs incluir um argumento -classpath ou -cp JVM, o caminho de classe especificado será anexado ao caminho de classe do ObjectGrid. A seguir está um exemplo de como utilizar o argumento -jvmArgs para incluir os arquivos Java Archive (JAR) Xerces no caminho de classe que são utilizados para ativar um servidor do ObjectGrid.

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

Parar Servidores do ObjectGrid

Utilize o script stopOgServer para parar servidores do ObjectGrid.

Uso

Utilize o arquivo `stopOgServer.bat` para parar um servidor em uma máquina do Windows. Utilize o arquivo `stopOgServer.sh` para parar um servidor do ObjectGrid em plataformas Linux e Unix. O script stopOgServer cria um cliente que pode parar um servidor, conectando-se a qualquer servidor disponível no cluster. O comportamento deste script é semelhante à auto-inicialização em um servidor disponível para iniciar outro servidor. A seguir estão os argumentos do script stopOgServer para parar um servidor.

```
stopOgServer.bat <servidor> -bootstrap <host:porta,host:porta> [options]
```

Exemplos

A seguir estão alguns exemplos de parada de diferentes servidores do ObjectGrid. Estes exemplos utilizam o arquivo `stopOgServer.bat`. Para estes exemplos, suponha que três servidores estejam ativos e em execução: os servidores server1, server2 e server3 conforme definido pelo arquivo `universityCluster3Servers.xml` em “Iniciar Servidores do ObjectGrid” na página 83.

Este primeiro exemplo pára o servidor server1 auto-inicializando em seu host e porta de acesso do cliente.

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

Suponha que o servidor server1 tenha sido parado com êxito. O próximo exemplo pára o server2 tentando primeiro auto-inicializar no servidor server1. Como o servidor server1 já foi parado, a auto-inicialização será malsucedida. O próximo host e porta na lista pertencem ao servidor server3. Como o servidor server3 está disponível, a auto-inicialização no servidor server3 será bem-sucedida e o server2 será parado.

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

Argumentos Opcionais

Existem alguns argumentos opcionais que podem ser transmitidos para o script stopOgServer. Esta seção mostrará como utilizar cada um destes argumentos opcionais. A seguir estão os argumentos válidos stopOgServer seguidos pelos argumentos opcionais.

```
stopOgServer.bat <servidor> -bootstrap <host:porta,host:porta> [options]
```

Opções:

- -traceSpec <especificação de rastreio>
- -traceFile <arquivo de rastreio>
- -clientSecurityFile <arquivo de propriedades de segurança do cliente>

-traceSpec

O argumento -traceSpec pode ser utilizado para configurar uma especificação de rastreio no cliente que tenta parar um servidor do ObjectGrid. A seguir está um exemplo de como configurar o argumento -traceSpec:

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

-traceFile

O argumento -traceFile pode ser utilizado para especificar um local para a saída rastreio do cliente durante o encerramento do servidor. A seguir está um exemplo de como configurar o argumento traceFile:

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

-clientSecurityFile

O argumento -clientSecurityFile pode ser utilizado para transmitir ao cliente seu arquivo de propriedades de segurança. Este argumento é requerido ao tentar conectar-se a um servidor com a segurança ativada.

A seguir está um exemplo de como configurar o argumento -clientSecurityFile:

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

Iniciar o Servidor Gateway de Gerenciamento

Para monitorar e administrar um cluster do ObjectGrid utilizando JMX (Java Management Extensions), o gateway de gerenciamento deve ser iniciado por meio do script da linha de comandos ou programaticamente.

Finalidade

Para iniciar o Gateway de Gerenciamento por meio da linha de comandos, utilize o script startManagementGateway. Utilize o arquivo `startManagementGateway.bat` para iniciar um servidor ManagementGateway em uma máquina do Windows. Utilize o arquivo `startManagementGateway.sh` para iniciar um servidor ManagementGateway em plataformas Linux e Unix. Para obter informações adicionais sobre a função do Gateway de Gerenciamento, MBeans ObjectGrid e JMX, consulte Capítulo 7, “Visão Geral do Gerenciamento de Sistemas”, na página 69.

O script startManagementGateway cria um servidor de conector JMX e um cliente do ObjectGrid que se conecta a um cluster do ObjectGrid para parar servidores, reunir status e estatísticas e desempenhar várias outras funções.

A seguir estão os argumentos de scripts startManagementGateway para iniciar um servidor gateway de gerenciamento.

```
startManagementGateway.bat -connectorPort <porta> -clusterHost <host>
-clusterPort <port> -clusterName <cluster> [options]
```

Argumentos Opcionais

Alguns argumentos opcionais podem ser transmitidos para o script startManagementGateway. Os argumentos startManagementGateway válidos são seguidos pelos argumentos opcionais.

```
startManagementGateway.bat -connectorPort <porta> -clusterHost <host>
-clusterPort <port> -clusterName <cluster> [options]
```

Opções

- -traceEnabled <rastreio ativado true/false>
- -traceSpec <especificação de rastreio>
- -traceFile <arquivo de rastreio>
- -refreshInterval <intervalo de atualização de atributo MBean>
- -sslEnabled <SSL ativado para gateway de gerenciamento true/false>
- -clientSecurityFile <caminho para arquivo de segurança do cliente>

-traceEnabled

O argumento -traceEnabled pode ser utilizado para configurar se o rastreio será ativado para o servidor Gateway de Gerenciamento. O padrão é false, portanto, a única maneira de ver o rastreio do ObjectGrid é ativá-lo configurando -traceEnabled como ″true″ e fornecendo valores válidos -traceSpec e -traceFile.

-traceSpec

O argumento -traceSpec pode ser utilizado para configurar uma especificação de rastreio para o servidor gateway de gerenciamento.

-traceFile

O argumento -traceFile pode ser utilizado para especificar um local para a saída de rastreio do Gateway de Gerenciamento. A seguir está um exemplo de como configurar os argumentos traceEnabled, traceSpec e traceFile.

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

-refreshInterval

O argumento -refreshInterval pode ser utilizado para transmitir a quantidade de tempo (em segundos) que o gateway de gerenciamento espera entre atualizações dos valores de atributo MBean. O valor padrão é de 120 segundos. A seguir está um exemplo de como configurar o argumento refreshInterval:

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

-sslEnabled

O argumento -sslEnabled pode ser utilizado para configurar se o SSL está ativado para o gateway de gerenciamento. Se o valor deste argumento for

`true`, qualquer cliente usuário que se conectar ao servidor gateway de gerenciamento precisará fornecer propriedades de SSL:

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

O valor padrão se o argumento -sslEnabled não for fornecido é ″false″.

-clientSecurityFile
O argumento -clientSecurityFile pode ser utilizado para transmitir o nome do arquivo que contém as propriedades de segurança do cliente para acesso de cliente seguro entre o servidor Gateway de Gerenciamento e o cluster do ObjectGrid. Este argumento é requerido ao tentar conectar-se a um cluster com a segurança ativada. O ObjectGrid fornece o seguinte gabarito de arquivo de propriedades de segurança do cliente: `security.ogclient.props`.

A seguir está um exemplo de como configurar as propriedades sslEnabled e clientSecurityFile:

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

Codificação de Senha

A codificação de senha impede a observação casual de senhas nos arquivos de propriedades de segurança do ObjectGrid.

Uso

O ObjectGrid contém várias senhas codificadas que não estão criptografadas. O ObjectGrid fornece o utilitário FilePasswordEncoder, que pode ser utilizado para codificar estas senhas. Utilize o arquivo `FilePasswordEncoder.bat` para codificar senhas em uma máquina do Windows. Utilize o arquivo `FilePasswordEncoder.sh` para codificar senhas em plataformas Linux e Unix.

A sintaxe do comando é a seguinte:

```
FilePasswordEncoder.bat file_name password_properties_list [file_type]
```

Opções

As opções a seguir estão disponíveis para o comando FilePasswordEncoder:

file_name
O file_name é utilizado para especificar o nome do arquivo que possui senhas a serem codificadas. Por exemplo, `security.ogserver.props`.

password_prop_list
A password_prop_list é uma lista de nomes de propriedades de senhas separados por vírgulas, por exemplo, ″trustStorePassword,keyStorePassword″.

file_type
Este argumento é opcional. O file_type pode ser um xml ou valor da propriedade, indicando se o arquivo fornecido é um arquivo de propriedades ou um arquivo XML. O valor padrão é `property`. No momento, o ObjectGrid não armazena senhas em um arquivo XML, portanto, esta opção não é requerida. Os exemplos a seguir demonstram a sintaxe correta:

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

Este utilitário FilePasswordEncoder não é fornecido com o WebSphere Extended Deployment. É possível utilizar o utilitário PropFilePasswordEncoder fornecido pelo WebSphere Application Server para codificar estas senhas. Consulte a Referência ao Comando PropFilePasswordEncoder para obter detalhes adicionais.

Capítulo 9. Visão Geral da Interface de Programação de Aplicativo do ObjectGrid

Esta seção discute como configurar o ObjectGrid com XML ou por meio de interfaces programáticas. Além disso, as informações são incluídas para implementar as interfaces externas fornecidas pelo ObjectGrid. Em todos os casos, são descritos uma visão geral, interfaces da API e exemplos.

Documentação da API

O JavaDoc para o ObjectGrid é a fonte de informações definitiva sobre as APIs. Localize o JavaDoc no seguinte diretório da instalação do WebSphere Extended Deployment: *install_root*\web\xd\apidocs

Interface 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.

Métodos createObjectGrid

Utilize este tópico para aprender sobre os sete métodos createObjectGrid que estão na interface ObjectGridManager.

Métodos createObjectGrid

A interface ObjectGridManager possui sete métodos createObjectGrid. A seguir está um cenário simples:

Caso simples com configuração padrão

A seguir está um caso simples de criação de um ObjectGrid para ser compartilhado 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 anterior cria e armazena em cache o ObjectGrid Employees. O ObjectGrid Employees é inicializado com a configuração padrão e está pronto para ser utilizado. O segundo parâmetro no método createObjectGrid está configurado como `true`, que instrui o ObjectGridManager a armazenar em cache a instância do ObjectGrid criada. Se este parâmetro for configurado como

`false`, a instância não será armazenada em cache. Cada instância do ObjectGrid possui um `nome` e a instância pode ser compartilhada entre muitos clientes ou usuários com base nesse nome.

Se a instância do objectGrid for utilizada no compartilhamento de mesmo nível, o armazenamento em cache deverá ser configurado como `true`. Para obter informações adicionais sobre compartilhamento de mesmo nível, consulte o Capítulo 12, “Distribuindo Alterações entre Java Virtual Machines no Mesmo Nível”, na página 341.

Configuração de XML

O ObjectGrid é altamente configurável. O exemplo anterior demonstra como criar um ObjectGrid simples sem nenhuma configuração. Com este exemplo, é possível criar uma instância do ObjectGrid pré-configurada que seja 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 tiver sido validado anteriormente e nenhuma alteração tiver sido 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 JDK (Java Developer Kit) 1.4.2 tem 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; // ativar a validação XML
boolean cacheInstance = true; // Armazenar a instância em cache
String objectGridName="Employees"; // Nome da URL do Object Grid
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees = oGridManager.createObjectGrid(objectGridName,
allObjectGrids,
validateXML,
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 de XML, consulte “Configuração do ObjectGrid” na página 260.

Existem sete métodos createObjectGrid. Os métodos estão documentados no seguinte bloco de código.

```
/**
* 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
* @ibm-api
*/
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 uma instância do ObjectGrid
* @this o nome já foi armazenado em cache ou
* qualquer erro durante a criação do ObjectGrid.
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, boolean
cacheInstance) throws ObjectGridException;
/**
* Crie uma instância do ObjectGrid com o nome do ObjectGrid especificado.
* A instância do ObjectGrid criada será armazenada em cache.
* @param objectGridName o Nome da instância do ObjectGrid a ser criada.
* @return uma instância do ObjectGrid
* @throws ObjectGridException se um ObjectGrid com este nome já tiver sido
* armazenado em cache ou algum erro encontrado durante a criação do
* ObjectGrid @ibm-api
*/
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 uma instância do ObjectGrid
```

```
* @see ObjectGrid
* @ibm-api
*/
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 uma instância do ObjectGrid
* @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
* @ibm-api
*/
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.
* @ibm-api
*/
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.
```

```
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
throws ObjectGridException;
```

Métodos getObjectGrid

Utilize os métodos getObjectGrid para recuperar instâncias armazenadas em cache.

Recuperar uma Instância Armazenada em Cache

Como a instância Employees do ObjectGrid foi armazenada em cache pela interface ObjectGridManager, qualquer outro usuário pode acessá-la com o seguinte trecho de código:

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

A seguir estão os dois métodos do getObjectGrid que retornam instâncias do ObjectGrid armazenadas em cache.

```
/**
* Obter uma Lista das instâncias do ObjectGrid que foram armazenadas em cache
* anteriormente.
* Retorna nulo se nenhuma das instâncias do ObjectGrid tiver sido armazenada
* em cache.
* @return uma Lista de instâncias do ObjectGrid que foram armazenadas em
* cache anteriormente
* @ibm-api
*/
public List getObjectGrids();
/**
* Utilize isto se já existir um ObjectGrid. Retorna uma instância do
* ObjectGrid armazenada em cache por nome. Este método retorna nulo se
* nenhum ObjectGrid com este objectGridName tiver sido armazenado em
* cache.
*
* @param objectGridName o nome do objectgrid armazenado em cache.
* @return um ObjectGrid armazenado em cache existente.
*
* @since WAS XD 6.0
* @ibm-api
*
*/
public ObjectGrid getObjectGrid(String objectGridName);
```

Métodos removeObjectGrid

Este tópico descreve como utilizar os dois métodos removeObjectGrid.

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 estiver configurado como **true**, o método destroy será chamado no ObjectGrid. A chamada para o método destroy no ObjectGrid encerra o ObjectGrid e libera recursos que ele está utilizando. A seguir está uma descrição de como utilizar os dois métodos removeObjectGrid:

```
/**
* 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
* @ibm-api
*/
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
* @ibm-api
*/
public void removeObjectGrid(String objectGridName, boolean destroy)
throws ObjectGridException;
```

Método getObjectGridAdministrator

Retornar uma Instância ObjectGridAdministrator para o Cluster

```
public ObjectGridAdministrator getObjectGridAdministrator
(ClientClusterContext ctx)
/**
* Retornar uma instância ObjectGridAdministrator para este cluster. Cada
* cluster exigirá a utilização de um ObjectGridAdministrator diferente.
*
* @param clientClusterContext. Um contexto de cluster exclusivo, com o
* qual o cliente precisa interagir.
* @param objectGridName o nome do objectgrid armazenado em cache.
* @return um ObjectGrid
*
* @since WAS XD 6.0.1
* @ibm-api
*
*/
public ObjectGridAdministrator getObjectGridAdministrator
(ClientClusterContext
ontext);
```

Consulte o Capítulo 7, “Visão Geral do Gerenciamento de Sistemas”, na página 69 para obter informações adicionais sobre este método.

Utilize a interface ObjectGridManager para controlar o ciclo de vida de uma instância do ObjectGrid

Este tópico demonstra como a interface ObjectGridManager pode ser utilizada para controlar o ciclo de vida de uma instância do ObjectGrid utilizando beans de inicialização e um servlet.

Gerenciar o Ciclo de Vida de uma Instância do ObjectGrid em um Bean de Inicialização

Um bean de inicialização pode ser utilizado 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` pode

ser utilizado para criar instâncias do ObjectGrid. O método `stop` pode ser utilizado para destruir instâncias do ObjectGrid. A seguir está um trecho de código que demonstra o gerenciamento de ciclo de vida deste ObjectGrid em um bean de inicialização.

```
public class MyStartupBean implements javax.ejb.SessionBean {
private ObjectGridManager objectGridManager;
/*
* Os métodos na interface SessionBean ficaram de fora
* deste exemplo por motivo de brevidade
*/
public boolean start(){
// Iniciando o bean de inicialização
// Este método é chamado quando o aplicativo é iniciado
objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
try {
// criar 2 ObjectGrids e armazenar estas instâncias em cache
ObjectGrid bookstoreGrid =
objectGridManager.createObjectGrid("bookstore", true);
bookstoreGrid.defineMap("book");
ObjectGrid videostoreGrid =
objectGridManager.createObjectGrid("videostore", true);
// na JVM,
// estes ObjectGrids agora podem ser recuperados do
//ObjectGridManager utilizando o método getObjectGrid(String)
} catch (ObjectGridException e) {
         e.printStackTrace();
                                  return false;
}
              return true;}
  public void stop()  {
// Parando o bean de inicialização
// Este método é chamado quando o aplicativo é parado
try {
// remover os ObjectGrids armazenados em cache e destruí-los
objectGridManager.removeObjectGrid("bookstore", true);
objectGridManager.removeObjectGrid("videostore", true);
} catch (ObjectGridException e) {
         e.printStackTrace();
}
}
}
```

Quando o método `start` tiver sido chamado, as instâncias do ObjectGrid recém-criadas poderão ser recuperadas do ObjectGridManager. Por exemplo, se um servlet estiver incluído no aplicativo, o servlet poderá acessar estes ObjectGrids 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 de um ObjectGrid em um Servlet

Um método para gerenciar o ciclo de vida de um ObjectGrid em um Servlet é criar a instância do ObjectGrid no método `init` e destruir o ObjectGrid no método `destroy`. Se a instância do ObjectGrid for armazenada em cache, ela poderá ser recuperada e manipulada no código do servlet. A seguir está algum código de amostra que demonstra a criação, manipulação e destruição de ObjectGrids 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");
BackingMap bookMap = bookstoreGrid.getMap("book");
// desempenhar operações no ObjectGrid armazenado em cache
// ...
}
public void destroy() {
super.destroy();
try {
// remover e destruir o ObjectGrid bookstore armazenado em cache
objectGridManager.removeObjectGrid("bookstore", true);
} catch (ObjectGridException e) {
         e.printStackTrace();
}
}
}
```

ObjectGrid de Rastreio

Este tópico explica como configurar o rastreio para o ObjectGrid.

Ambiente J2SE (Java 2 Platform, Standard Edition)

Quando for necessário enviar informações de depuração para a IBM, utilize o mecanismo de rastreio para obter o rastreio de depuração. A seguir está um exemplo de como obter o rastreio de depuração em um ambiente J2SE:

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

O exemplo anterior não inclui o rastreio de plug-ins do evictor internos para o ObjectGrid. Se estiver utilizando um ou mais dos plug-ins do evictor fornecidos pelo ObjectGrid e se tiver problemas que podem estar relacionados à evicção, ative o rastreio para o ObjectGrid mais os evictores do ObjectGrid, conforme ilustra o exemplo a seguir:

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

Ambiente do WebSphere Application Server

Não é necessário utilizar o ObjectGridManager para configurar o rastreio em um ambiente do WebSphere Application Server. É possível utilizar o console administrativo para configurar a especificação de rastreio.

APIs Connect do Cliente do ObjectGrid

Informações Básicas

Um cliente conecta-se a um processo do servidor ativo ou em execução em um cluster. Um cliente tem uma necessidade mínima do nome do host e do número da porta do servidor ao qual ele se conecta. As informações de nome do host e de porta estão disponíveis no arquivo XML da definição do cluster que foi inicialmente utilizado para iniciar o servidor. Consulte “Configuração do ObjectGrid” na página 260 para obter detalhes da configuração XML. A seguir está um snippet da definição XML do cluster, utilizado como uma amostra para esta seção. Utilizando as APIs documentadas nesta seção, o cliente conecta-se a um ObjectGrid remoto que está configurado para receber e processar pedidos do cliente. Observe que o cliente ″faz download″ das definições do ObjectGrid e XML do Cluster para auto-inicializar-se. A configuração do cliente é baseada na configuração do servidor ao qual ele se conecta.

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

Esta definição de cluster está incompleta, mas é suficiente para este exemplo. No cluster cluster1 estão três servidores: os servidores server1, server2 e server3. O atributo clientAccessPort especifica a porta listener na qual o servidor está atendendo e a porta com a qual o cliente estabelece uma conexão pela primeira vez. No snippet XML anterior, as portas para os servidores server1, server2 e server3 são `12503`, `12504` e `12504`, respectivamente.

APIs Connect

A interface ObjectGridManager possui métodos de conexão que estão documentados na amostra a seguir. As seguintes APIs connect estão disponíveis na interface ObjectGridManager. Consulte a documentação da API para obter uma descrição destes métodos.

```
/**
* Permite que o cliente conecte-se a um ObjectGrid remoto
* O ObjectGrid remoto é hospedado conforme especificado pelos parâmetros:
* @param clusterName: O nome do cluster ao qual este cliente
```

```
* conecta-se
* @param host: O host para conexão
* @param port: A porta clientAceess que está atendendo
* @param ClientSecurityConfiguration: A configuração de segurança pode ser nula
* se a segurança não estiver configurada
* @param overRideObjectGrid xml. Este parâmetro pode ser nulo. Se não for nulo,
* a configuração do lado cliente do plug-in do ObjectGrid será substituída.
* Nem todos os plug-ins podem ser substituídos. Para obter detalhes, consulte os
* documentos do ObjectGrid
* @throws ConnectException
*
*/
public ClientClusterContext connect(String clusterName,
String host,
String port,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
/**
*
* @param clusterName
* @param attributes Os atributos do par Host e Porta nos quais há tentativa
* de conexão em ordem seqüencial. Se uma tentativa de conexão com um
* servidor falhar, o próximo par de atributos host e porta será coletado
* para tentar novamente a conexão.
* @param ClientSecurityConfiguration: Configuração de segurança. Pode ser
* nula se a segurança não estiver configurada.
* @param overRideObjectGrid xml. Este parâmetro pode ser nulo. Se não for
* nulo, a configuração do lado cliente do plug-in do ObjectGrid será
* substituída.
* Nem todos os plug-ins podem ser substituídos. Para obter detalhes,
* consulte os documentos  do ObjectGrid.
* @return ClientClusterContext
* @throws ConnectException
*
*
*/
public ClientClusterContext connect(String clusterName,
HostPortConnectionAttributes[] attributes,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
/**
* Este método poderá ser utilizado apenas se um cliente for colocado com
* um servidor do ObjectGrid, especificamente em um ambiente J2EE (Java 2
* Platform, Enterprise Edition) com o IBM WebSphere
* Application Server, que suporta o
* servidor do ObjectGrid incorporado.
* Este método conecta o cliente ao Servidor que está em execução
* na mesma JVM (Java Virtual Machine).
* @param securityProps. Pode ser nulo se não estiver em execução no modo
* seguro.
* @param overRideObjectGrid xml. Este parâmetro pode ser nulo. Se for
* não for nulo, a configuração do lado cliente do plug-in do ObjectGrid
* será substituída.
* Nem todos os plug-ins podem ser substituídos. Para obter detalhes,
* consulte os documentos do ObjectGrid
* @return ClientClusterContext
* @throws ConnectException
*
*/
public ClientClusterContext connect(ClientSecurityConfiguration
securityProps, URL overRideObjectGrid) throws ConnectException;
/**
* Permite que o cliente conecte-se a um ObjectGrid Remoto
* @param clusterConfigFile Uma URL para o Arquivo clusterConfig.
* Esse é o mesmo arquivo utilizado para iniciar servidores.
* É utilizado para recuperar informações de porta do host. Ele não
* pode ser nulo. Se for nulo, o resultado será uma exceção
```

```
* IllegalArgumentException.
* @param serverName Uma Cadeia, o nome do servidor específico para
* conexão.
* Se o nome do servidor não estiver na configuração, o resultado será
* IllegalArgumentException.
* Este parâmetro pode ser nulo e, neste caso, será feita uma tentativa
* de conectar-se
* a um dos servidores especificados no arquivo
* XML do cluster. Se falhar uma tentativa de conectar-se a um, outro
* servidor será selecionado.
* Isto será feito até que a lista se esgote.
* @param securityProps
* @param overRideObjectGrid xml. Este parâmetro pode ser nulo. Se não for
* nulo, a configuração do lado cliente do plug-in do ObjectGrid será
* substituída.
* Nem todos os plug-ins podem ser substituídos. Para obter detalhes, consulte
* os documentos  do ObjectGrid
* @return ClientClusterContext
* @throws ConnectException
*
* @ibm-api
*/
public ClientClusterContext connect(URL clusterConfigFile,
String serverName,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
```

Exemplo de Utilização dos Parâmetros de Host e de Porta

O código a seguir utiliza o XML de cluster documentado em “Informações Básicas” na página 101. Este cliente conecta-se ao host `s1.myco.com` na porta `12503`.

```
import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ConnectException;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
public class C1 {
/**
* @param args
*/
public static void main(String[] args) {
final ObjectGridManager oGridManager=ObjectGridManagerFactory.
getObjectGridManager(); //etapa 1
ClientClusterContext ctx = null;
try {
ctx=oGridManager.connect("cluster1","s1.myco.com","12503",null,null);
//etapa 2
ObjectGrid employees = oGridManager.getObjectGrid(ctx,"employees");
// etapa 3
// Executar operações do objectGrid
// get
// update
// commit...etc..
} catch (ConnectException e) {
//falha na conexão
e.printStackTrace();
//terminate
}finally {
if(ctx !=null) {
oGridManager.disconnect(ctx); // etapa 4
}
}
}
}
```

1. Obtenha o Objeto Singleton do ObjectGridManager do ObjectGridManagerFactory.
2. Chame a API connect.
3. Supondo que existem funcionários do objectGrid no ObjectGrid remoto, chame o método do getObjectGrid transmitindo o parâmetro ClientClusterContext.
4. Chame o método disconnect. Como última etapa, todos os clientes devem chamar disconnect, se o trabalho estiver concluído. Esta é uma etapa muito importante.

Fornecendo Vários Hosts para uma Nova Tentativa Automática de Conexão, no Caso de uma Exceção ConnectException

Neste exemplo, os atributos HostPortConnectionAttributes são utilizados para fornecer uma matriz de atributos de porta do host, à qual um cliente pode conectar-se. A API utiliza estes atributos do par de host e porta em ordem seqüencial de conexão. Se uma tentativa de conexão com um servidor falhar, o próximo par de atributos host e porta será selecionado para uma nova tentativa de conexão.

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

Cliente e Servidor no Mesmo Processo

Se o cliente estiver na mesma JVM que o servidor, o seguinte método connect poderá ser utilizado.

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

Especificar XML do Cluster

Se o cliente tiver acesso ao arquivo XML do Cluster, não será necessário especificar o nome do host e o número da porta. Esta API recupera o nome do servidor e o número da porta e os utiliza para conexão. O nome do servidor é opcional e pode ser nulo, neste caso, a API tentar conectar-se a um dos servidores definidos no arquivo XML do cluster.

```
ctx=oGridManager.connect(urlToClusterxml,"server1",null,null);
// conectar-se a server1
// ou
ctx=oGridManager.connect(urlToClusterxml,null,null,null);
//conectar-se a qualquer servidor no cluster
```

Segurança do Cliente com a API Connect

Em todos os exemplos, o parâmetro ClientSecurityConfiguration era nulo. A transmissão do valor nulo implica na desativação da segurança. Se a segurança estiver ativada, transmita o objeto ClientSecurityConfiguration como um argumento. Consulte o “Segurança do ObjectGrid” na página 139 para obter informações adicionais.

Substituir Configuração XML do ObjectGrid

O cliente ″faz download″ das definições do ObjectGrid a partir do servidor para configurar-se. Todos os plug-ins que estão definidos no ObjectGrid são disponibilizados para o cliente. Basicamente, existe um ObjectGrid local no lado cliente que se comunica com o ObjectGrid do lado do servidor. Fornecendo um arquivo XML de substituição na API connect, é possível ″substituir″ a configuração de plug-in, que é específica para utilização apenas de um cliente. Estes plug-ins são:

Plug-ins do **ObjectGrid**:

- Plug-in TransactionCallback
- Plug-in ObjectGridEventListener

Plug-ins do **BackingMap**:

- Plug-in Evictor
- Plug-in MapEventListener

Todos os demais plug-ins que estão definidos no XML de substituição serão ignorados.

Exemplo

Suponha que o cliente precise substituir a configuração do Evictor para um BackingMap específico. Ou seja, no lado cliente, o Evictor precisa ser diferente do que está configurado no lado do servidor.

Suponha que o Evictor do lado do servidor seja utilizado da seguinte forma. Ele utiliza o evictor LFUEvictor interno:

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

Os requisitos do lado cliente são diferentes. É requerido que seja utilizado no lugar um usuário definido, Evictor myco.og.MyEvictor. O XML de substituição pode incluir o snippet mostrado a seguir. Todos os backingMaps configurados para utilizar LFUEvictor utilizam o usuário definido:

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

XML Completo

O código XML a seguir exibe dois arquivos XML: um utilizado para o lado do servidor e o segundo para o cliente. Esta configuração permite que um cliente substitua a configuração do Evictor para o dow BackingMap.

Arquivo XML do ObjectGrid do Lado do Servidor

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="market">
<backingMap name="dow" ttlEvictorType="NONE" readOnly="false"
pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.
builtins.LRUEvictor">
<property name="maxSize" type="int" value="2"
description="set max size for LRU Evictor" />
<property name="numberOfLRUQueues" type="int" value="1"
description="set number of LRU queues" />
<property name="sleepTime" type="int" value="2" description="evictor
thread sleep time" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Arquivo XML do ObjectGrid do Lado Cliente a Ser Substituído durante a Conexão

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="market">
<backingMap name="dow" pluginCollectionRef="default" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="default">
<bean id="Evictor" className="myco.og.MyEvictor">
<property name="name" type="java.lang.String" value="MyEvictor"
description="" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Considerações de Design do Aplicativo

A API connect() do cliente é uma operação cara. Dependendo do trabalho, um cliente estabelece uma ou mais conexões físicas com um único servidor. O número de conexões do cliente varia entre os valores especificados pelos atributos tcpMinConnections e tcpMaxConnections que estão definidos no elemento de cluster da definição XML da configuração de cluster. Esta é a mesma definição XML do cluster que foi utilizada para iniciar o servidor. O gerenciador de conexões agrupa estas conexões físicas e o ObjectGrid reutiliza-as conforme necessário. Os atributos tcpMinConnections e tcpMaxConnections especificam o número de conexões do cliente apenas com um único servidor. Se um cliente conectar-se a mais de um servidor, o número máximo de conexões do cliente será menor ou igual ao atributo tcpMaxConnections vezes o número de servidores aos quais o cliente se conecta. Por exemplo, se o cliente conectar-se a três servidores e tcpMaxConnections estiver especificado para ser cinco, o cliente terá um máximo de `(5*3)=15` conexões e um mínimo de três conexões, assumindo que a configuração de tcpMinConnection seja `1`. As conexões são compartilhadas entre todos os clientes.

O atributo **threadsPerClientConnect** especifica o número de worker threads. Estes worker threads efetuam dispatch do trabalho por meio de conexões físicas. Eles processam dados de configuração, pedidos do cliente, respostas do servidor e pedidos de administração do sistema. Seu valor padrão é `5`. Este atributo está disponível no primeiro iFix. Se o iFix não estiver disponível, utilize a propriedade de sistema -Dthreads da JVM (Java Virtual Machine) para especificar o número de worker threads. Dependendo de seu aplicativo, aumentar este número pode ajudar no desempenho. Como regra, este número não deve ser menor que o número de conexões físicas utilizadas pelo cliente.

Se o design de seu aplicativo permitir, um cliente poderá criar vários encadeamentos para concluir o trabalho com uma chamada de conexão reutilizando o método *ClientClusterContext*.

Interface do ObjectGrid

Utilize este tópico para fazer referência aos métodos necessários para modificar um ObjectGrid.

Apresentação

ObjectGrid é uma estrutura de armazenamento em cache de objetos transacional, extensível, baseada na interface do Mapa Java. As operações da API do ObjectGrid estão agrupadas em uma unidade de trabalho transacional e permite extensibilidade por meio de suporte ao plug-in projetado customizado. O ObjectGrid é um contêiner lógico denominado que contém vários BackingMaps. Para obter informações adicionais sobre mapas de suporte, consulte “Interface BackingMap” na página 112.

Criar e Inicializar

Consulte o tópico da interface ObjectGridManager para obter as etapas requeridas para criar uma instância do ObjectGrid. Existem dois métodos distintos para criar um ObjectGrid: programaticamente ou com arquivos de configuração XML. Consulte o “Interface ObjectGridManager” na página 93 para obter informações adicionais.

Métodos Get ou Set e Factory

Atenção: Os métodos *set* devem ser chamados antes da inicialização da instância do ObjectGrid. Se você chamar um método set depois de chamar o método initialize, isto resultará em `java.lang.IllegalStateException`. Cada um dos métodos getSession da interface do 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, inclusão e remoção dos objetos EventListener. Estes objetos podem ser processados após a conclusão do processamento de ″initialize″.

A interface do ObjectGrid contém os seguintes métodos:

Tabela 7. Métodos da Interface do ObjectGrid

Método	Descrição
BackingMap defineMap(String name);	*defineMap*: é um método do depósito de informações do provedor para definir um BackingMap denominado exclusivamente. Para obter informações adicionais sobre mapas de suporte, consulte “Interface BackingMap” na página 112.
BackingMap getMap(String name);	*getMap*: Retorna um BackingMap definido anteriormente chamando *defineMap*. Utilizando este método, é possível configurar o BackingMap, se ainda não estiver configurado por meio da configuração XML.
BackingMap createMap(String name);	*createMap*: Cria um BackingMap, mas não o armazena em cache para utilização por este ObjectGrid. Utilize este método em série com o método setMaps(List) da interface do ObjectGrid, que armazena BackingMaps em cache para utilização com este ObjectGrid. Utilize estes métodos quando estiver configurando um ObjectGrid com Spring Framework.
void setMaps(List mapList);	*setMaps*: Limpa os BackingMaps que foram definidos anteriormente neste ObjectGrid e os substitui pela lista de BackingMaps fornecida.
public Session getSession() throws ObjectGridException, TransactionCallbackException;	*getSession*: Retorna uma Sessão, que fornece a funcionalidade de início, confirmação e rollback para uma Unidade de Trabalho. Para obter informações adicionais sobre objetos de Sessão, consulte “Interface Session” na página 116.
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 a utilização de um objeto Subject específico em vez do configurado no ObjectGrid para obter uma sessão.

Tabela 7. Métodos da Interface do ObjectGrid (continuação)

Método	Descrição
void initialize() throws ObjectGridException;	*initialize*: O ObjectGrid é inicializado e fica disponível para utilização 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 depois que este método for chamado.
void setTxTimeout(int timeout);	*setTxTimeout*: Utilize este método para configurar a quantidade de tempo, em segundos, que uma transação iniciada por uma Sessão criada por esta instância do ObjectGrid tem permissão para conclusão. Se uma transação não for concluída dentro da quantidade de tempo especificada, a Sessão que iniciou a transação será marcada como ″tempo limite excedido″. Marcar uma Sessão como tempo limite excedido faz o próximo método ObjectMap chamado pela Sessão com tempo limite excedido resultar em uma exceção `com.ibm.websphere.objectgrid.TransactionTimeoutException`. A Sessão é marcada como *apenas rollback*, que faz com que seja efetuado rollback da transação, mesmo que o aplicativo chame o método commit em vez do método rollback após a exceção `TransactionTimeoutException` ter sido capturada pelo aplicativo. Um valor de tempo limite de `0` indica que a transação tem uma quantidade de tempo ilimitada para ser concluída. O tempo limite da transação não será excedido se um valor de tempo limite de `0` for utilizado. Se este método não for chamado, qualquer Sessão que seja retornada pelo método getSession desta interface terá um valor de tempo limite de transação configurado como `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.
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 tiver sido chamado, o método retornará `0` para indicar que a transação tem um valor de tempo ilimitado para ser concluída.

Tabela 7. Métodos da Interface do ObjectGrid (continuação)

Método	Descrição
//Palavras-chave.	
void associateKeyword(Serializable parent, Serializable child);	*associateKeyword*: A palavra-chave do ObjectGrid fornece um mecanismo de invalidação flexível baseado em palavras-chave. Para obter informações adicionais sobre palavras-chave, consulte “Palavras-chave” na página 124. 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.
//Segurança	
void setSecurityEnabled()	*setSecurityEnabled*: Ativa a segurança. A segurança é desativada por padrão.
void setPermissionCheckPeriod(long period);	*setPermissionCheckPeriod*: Este método utiliza um único parâmetro que indica a freqüência de verificação da permissão utilizada para permitir o acesso do cliente. Se o parâmetro for `0`, todos os métodos solicitarão o mecanismo de autorização, após a autorização JAAS ou autorização customizada, para verificar se o subject 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. Como alternativa, se o parâmetro for menor que `0`, ele indicará o número de milissegundos para armazenar em cache um conjunto de permissões antes de retornar ao mecanismo de autorização para atualização. 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*: Configura o mecanismo de autorização. O padrão é `SecurityConstants.JAAS_AUTHORIZATION`.
setMapAuthorization(MapAuthorization ma);	*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.

Tabela 7. Métodos da Interface do ObjectGrid (continuação)

Método	Descrição
setSubjectSource(SubjectSource ss);	*setSubjectSource*: Configura o plugin 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 plugin 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.

Interface do ObjectGrid: Plug-ins

A interface do ObjectGrid possui vários 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);
// Plug-ins relacionados à segurança
void setSubjectValidation(SubjectValidation subjectValidation);
void setSubjectSource(SubjectSource source);void setMapAuthorization
(MapAuthorization mapAuthorization);
```

- *ObjectGridEventListener*: Uma interface ObjectGridEventListener é utilizada para receber notificações quando ocorrem eventos importantes 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 “Listeners” na página 186 e “Interface Session” na página 116 para obter informações adicionais.
- *TransactionCallback*: Uma interface de listener TransactionCallback permite que eventos transacionais, como sinais de início, confirmação e rollback sejam enviados para esta interface. Geralmente, uma interface do listener TransactionCallback é utilizada com um Loader. Para obter informações adicionais, consulte “Plug-in TransactionCallback” na página 217 e “Loaders” na página 201

página 201. Estes eventos podem então ser utilizados para coordenar transações com um recurso externo ou em vários loaders.

- *reserveSlot*: Permite que plug-ins neste ObjectGrid reservem slots para utilização em instâncias do objeto que possuem slots como TxID.
- *SubjectValidation*. Se a segurança estiver ativada, este plug-in poderá ser utilizado para validar uma classe `javax.security.auth.Subject` transmitida 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 poderá ser utilizado para obter um objeto Subject que representa o cliente do ObjectGrid. Este subject é então utilizado para autorização do ObjectGrid.

Interface BackingMap

Cada instância do ObjectGrid contém uma coleta de objetos de BackingMap.

Cada BackingMap é denominado e incluído em uma instância do ObjectGrid utilizando o método defineMap ou o método createMap da interface do ObjectGrid. Estes métodos retornam uma instância de BackingMap que é então utilizada para definir o comportamento de um Mapa individual. Consulte o “Interface do ObjectGrid” na página 107 para obter informações adicionais.

A interface de Sessão é utilizada para iniciar uma transação e para obter o ObjectMap ou o JavaMap requerido para desempenhar interação transacional entre um aplicativo e um objeto de 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 confirmados para um Mapa individual. Para obter informações adicionais sobre a interface de Sessão, consulte Interface de Sessão.

A interface com.ibm.websphere.objectgrid.BackingMap fornece métodos para a configuração de atributos de BackingMap. Alguns dos métodos set permitem a extensibilidade de um BackingMap por meio de vários plug-ins projetados customizados. A seguir está uma lista dos métodos set para a configuração de atributos e fornecimento de suporte a plug-ins projetados customizados:

```
// Para a configuração de atributos de BackingMap.
public void setReadOnly(boolean readOnlyEnabled);
public void setNullValuesSupported(boolean nullValuesSupported);
public void setLockStrategy( LockStrategy lockStrategy );
public void setCopyMode(CopyMode mode, Class valueInterface);
public void setCopyKey(boolean b);
public void setNumberOfBuckets(int numBuckets);
public void setNumberOfLockBuckets(int numBuckets);
public void setLockTimeout(int seconds);
public void setTimeToLive(int seconds);
public void setTtlEvictorType(TTLType type);
// Para a configuração de um plug-in customizado opcional fornecido pelo aplicativo.
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:

- Atributo *ReadOnly*. Este atributo indica se o Mapa é um Mapa de 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.
- Atributo *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.
- Atributo *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” na página 130 para obter detalhes sobre as estratégias de bloqueio suportadas.
- Atributo *CopyMode*. Este atributo determina se uma cópia de um objeto de valor será feita pelo BackingMap quando um valor for lido a partir do mapa ou será colocado no BackingMap durante o ciclo de confirmação de uma transação. São suportados vários modos de cópia para permitir que o aplicativo faça o equilíbrio entre desempenho e integridade de dados. Se este atributo não for configurado, será utilizado o modo de cópia de `COPY_ON_READ_AND_COMMIT`. Este modo de cópia não tem o melhor desempenho, mas tem a maior proteção contra problemas de integridade de dados. Para obter informações adicionais sobre os modos de cópia, consulte Boas Práticas do Método copyMode.
- Atributo *CopyKey*. Este atributo determina se o BackingMap fará uma cópia de um objeto de chave quando uma entrada for 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.
- Atributo *NumberOfBuckets*. Este atributo indica o número de depósitos 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 de `503` será utilizado se o aplicativo não configurar o atributo NumberOfBuckets.
- Atributo *NumberOfLockBuckets*. Este atributo indica o número de depósitos de bloqueios que serão 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 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, um valor padrão de 383 será utilizado.

- Atributo *LockTimeout*. Este atributo será utilizado quando o BackingMap estiver 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 for configurado, será utilizado o valor de 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.
- Atributo *TtlEvictorType*. Cada BackingMap possui seu próprio evictor time to live interno que utiliza um algoritmo baseado em tempo para determinar quais entradas do mapa serão liberadas. Por padrão, o evictor time to live interno não está ativo. É possível ativar o evictor time to live chamando o método setTtlEvictorType com um de três valores: `CREATION_TIME`, `LAST_ACCESS_TIME` ou `NONE`. Um valor de `CREATION_TIME` indica que o evictor inclui o atributo TimeToLive na hora da criação da entrada do mapa no BackingMap para determinar quando o evictor deve liberar a entrada do mapa do BackingMap. Um valor de `LAST_ACCESS_TIME` indica que o evictor inclui o atributo TimeToLive durante o último acesso à entrada do mapa por alguma transação na qual o aplicativo está em execução para determinar quando o evictor deve liberar a entrada do mapa. A entrada do mapa será liberada apenas se uma entrada do 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 do 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.
- Atributo *TimeToLive*. Este atributo é utilizado para especificar o número de segundos que o evictor time to live interno precisa incluir na hora da criação ou 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 ilustra a definição de BackingMap de someMap na instância someGrid do ObjectGrid e a configuração de vários 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 ); // substituir padrão de leitura/gravação
bm.setNullValuesSupported(false); // substituir padrão de permissão de valores Nulos
bm.setLockStrategy( LockStrategy.PESSIMISTIC ); // substituir padrão de OPTIMISTIC
```

```
bm.setLockTimeout( 60 ); // substituir padrão de 15 segundos.
bm.setNumberOfBuckets(251); // substituir padrão (números primos
funcionam melhor)
bm.setNumberOfLockBuckets(251); // substituir padrão (número primos
funcionam melhor)
...
```

Plug-ins de 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 aprimorar 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 o “Plug-in ObjectTransformer” na página 213 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 o “Evictors” na página 191 para obter informações adicionais.
- **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, utilizando o método addMapEventListener ou o método removeMapEventListener. Consulte o “Interface MapEventListener” na página 189 para obter informações adicionais.
- **Plug-in do Loader**: Um BackingMap é um cache de memória de um Mapa. Um plug-in do Loader é uma opção utilizada pelo BackingMap para mover dados entre a memória e é utilizado para armazenamento persistente para o BackingMap. Por exemplo, um Loader JDBC (Java Database Connectivity) pode ser utilizado para mover dados para dentro e 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 Loader também pode ser utilizado para mover dados entre um BackingMap e um arquivo, entre um BackingMap e um mapa Hibernate, entre um BackingMap e um bean de entidade J2EE (Java 2 Platform, Enterprise Edition), 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 “Loaders” na página 201 para obter informações adicionais sobre este plug-in.

- **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 o “Interface OptimisticCallback” na página 224 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” na página 238 para obter informações detalhadas.

Interface Session

Esta seção descreve como os aplicativos iniciam e encerram transações utilizando a interface Session. A interface Session também fornece acesso ao aplicativo com base nas interfaces ObjectMap e JavaMap.

Apresentação

Cada instância de ObjectMap ou de JavaMap está diretamente ligada a um objeto de Sessão específico. Cada encadeamento que deseja acesso a um ObjectGrid primeiro deve obter uma Sessão do objeto do ObjectGrid. Uma instância de Sessão não pode ser compartilhada simultaneamente entre encadeamentos. O ObjectGrid não utiliza nenhum armazenamento local de encadeamento, mas as restrições de plataforma podem limitar a oportunidade de transmissão de 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; ObjectGrid getObjectGrid();
TxID getTxID()
throws NoActiveTransactionException;
boolean isWriteThroughEnabled();
void setTransactionType(String tranType);
public void processLogSequence(LogSequence logSequence)
throws NoActiveTransactionException, UndefinedMapException, ObjectGridException;

public ObjectGrid getObjectGrid();
public void setTransactionTimeout(int timeout);
```

```
public int getTransactionTimeout();
public boolean transactionTimedOut();
public boolean isCommitting();
public boolean isFlushing();
public void markRollbackOnly(Throwable t) throws NoActiveTransactionException;
public boolean isMarkedRollbackOnly();
}
```

Método Get

Um aplicativo obtém uma instância de Sessão de um objeto do ObjectGrid utilizando o método ObjectGrid.getSession. O trecho de código 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.

Transações e Métodos de Sessões

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 ObjectGrid também suporta a capacidade de confirmar automaticamente as transações, também conhecidas como confirmação automática. 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"); // confirmação automática
```

Método Session.flush

O método Session.flush faz sentido apenas quando um Loader está associado a um BackingMap. O método flush chama o Loader com o conjunto atual de alterações no cache de transações. O Loader aplica as alterações ao backend. Estas alterações não são confirmadas quando o flush é chamado. Se uma transação de Sessão for confirmada após uma chamada de flush, apenas as atualizações que ocorrem após a chamada do flush serão aplicadas ao Loader. Se uma transação de Sessão receber rollback após uma chamada de flush, as alterações limpas serão descartadas com todas as demais alterações pendentes na transação. Utilize o método Flush com moderação, pois ele limita a oportunidade de operações de batch em um Loader. A seguir está um exemplo do uso do método Session.flush:

```
Session session = objectGrid.getSession();
session.begin();
// fazer algumas alterações
...
session.flush(); // enviar estas alterações para o Loader, mas ainda não confirmar
// fazer mais algumas alterações
...
session.commit();
```

Método No Write Through

Alguns mapas do ObjectGrid são suportados por um Loader, que fornece armazenamento persistente para os dados no mapa. Às vezes, é útil confirmar dados apenas no mapa do ObjectGrid e não enviar dados para o Loader. 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 ObjectGrid e não são confirmados no armazenamento persistente fornecido pelo Loader. Este método é muito útil ao desempenhar o pré-carregamento de dados no mapa.

Ao utilizar uma instância do ObjectGrid distribuído, o método beginNoWriteThrough é útil para fazer alterações apenas no near cache, sem modificar o far cache no servidor. Se os dados forem considerados stale no near cache, a utilização do método beginNoWriteThrough pode permitir que entradas sejam invalidadas no near cache sem invalidá-las também no servidor.

A interface Session também fornece o método isWriteThroughEnabled para determinar qual o tipo de transação que está ativa no momento.

```
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
// fazer algumas alterações...
session.commit(); // estas alterações não serão enviadas ao Loader
```

Obter o Método de Objeto TxID

O objeto TxID é um objeto opaco que identifica a transação ativa. Utilize o objeto TxID para as seguintes finalidades:

- Para comparação quando estiver procurando uma transação específica.
- Para armazenar dados compartilhados entre os objetos TransactionCallback e Loader.

Consulte “Plug-in TransactionCallback” na página 217 e “Loaders” na página 201 para obter informações adicionais sobre o recurso de slot do Objeto.

Configurar o Tipo de Transação para Método de Monitoramento de Desempenho

Se estiver utilizando o ObjectGrid em um servidor de aplicativos do WebSphere Application Server, pode ser necessário reconfigurar o tipo de transação para monitoramento de desempenho. É possível configurar o tipo de transação com o método setTransactionType. Consulte “Monitorando o Desempenho do ObjectGrid com a PMI (Performance Monitoring Infrastructure) do WebSphere Application Server” na página 295 para obter informações adicionais sobre o método setTransactionType.

Processar um Método LogSequence Completo

O ObjectGrid pode propagar conjuntos de alterações do mapa para outros listeners do ObjectGrid como um meio de distribuir mapas de uma JVM (Java Virtual Machine) para outra. 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 ativa 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 de recebimento na JVM remota seria iniciar uma Sessão utilizando o método beginNoWriteThrough, que impede a propagação infinita de alterações, seguida por uma chamada para este método processLogSequence e, em seguida, a confirmação ou rollback da transação.

```
// Utilizar o objeto de Sessão transmitido durante
//ObjectGridEventListener.initialization...
session.beginNoWriteThrough();
// processar o LogSequence recebido
try {
session.processLogSequence(receivedLogSequence);
            } catch (Exception e) {
session.rollback(); throw e;
}
// confirmar as alterações
session.commit();
```

Método markRollbackOnly

Este método é utilizado para marcar a transação atual como ″apenas rollback″. Marcar uma transação como ″apenas rollback″ assegura que, mesmo que o método commit seja chamado pelo aplicativo, a transação receba rollback. Este método geralmente é utilizado pelo próprio ObjectGrid ou pelo aplicativo quando ele sabe que pode ocorrer danos nos dados se a transação tiver permissão para ser confirmada.

Quando este método tiver sido chamado, o objeto Throwable transmitido para este método será encadeado para a exceção `com.ibm.websphere.objectgrid.TransactionException` que resulta no método commit se ele for chamado em uma Sessão que foi anteriormente marcada como ″apenas rollback″. 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 limpas do plugin do Loader como resultado do método flush da interface Session que está sendo chamada. Um plugin do Loader pode achar este método útil quando precisar saber por que 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.

Interfaces ObjectMap e JavaMap

Este tópico descreve como os aplicativos interagem com o ObjectGrid utilizando as interfaces ObjectMap e JavaMap. Estas duas interfaces são utilizadas para interação transacional entre aplicativos e BackingMaps.

Interface ObjectMap

Uma instância do ObjectMap é obtida do objeto de Sessão que corresponde ao encadeamento atual. A interface do ObjectMap é o principal veículo utilizado pelos aplicativos para fazer alterações em entradas em um BackingMap.

Obter uma Instância do ObjectMap

Um aplicativo obtém uma instância do ObjectMap de um objeto de Sessão utilizando o 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 de Sessão específico com o mesmo nome de BackingMap sempre retorna a mesma instância do ObjectMap.

Autoconfirmar Transações

Conforme mencionado anteriormente, as operações em BackingMaps que utilizam ObjectMaps e JavaMaps são mais eficientemente desempenhadas em uma transação de Sessão. O ObjectGrid fornece suporte à autoconfirmação quando os métodos nas interfaces ObjectMap e JavaMap são chamados fora de uma transação de Sessão. Os métodos iniciam uma transação implícita, desempenham a operação solicitada e confirmam a transação implícita.

Semântica de Métodos

A seguir está uma explicação da semântica atrás 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” na página 124. O método setTimeToLive é discutido no tópico “Evictors” na página 191. Consulte a documentação da API para obter informações adicionais sobre estes métodos.

Método containsKey
: Determina se uma chave tem um valor no BackingMap ou Loader. 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 Loader não contêm a chave.

Método flush
: A semântica deste método 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, apenas as alterações neste ObjectMap são limpas no loader.

método get
: Busca a entrada do BackingMap. Se a entrada não for localizada no BackingMap mas um Loader estiver associado ao BackingMap, ele tentará buscar a entrada do Loader. O método getAll é fornecido para permitir o processamento de busca de batch.

Método getForUpdate
: O método getforUpdate é igual ao método get, mas a utilização do método getForUpdate informa ao BackingMap e ao Loader que a intenção é atualizar a entrada. Um Loader pode utilizar esta sugestão para emitir uma consulta `SELECT for UPDATE` para um backend de banco de dados. Se uma LockingStrategy Pessimista estiver definida para o BackingMap, o gerenciador de bloqueios bloqueará a entrada. O método getAllForUpdate é fornecido para permitir o processamento de busca de batch.

Método insert
: Insere uma entrada no BackingMap e no Loader. A utilização deste método informa ao BackingMap e ao Loader que você deseja inserir uma entrada

inexistente 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** transmitido para o método. O método invalidateAll é fornecido para permitir o processamento de invalidate de batch.

A invalidação local é especificada quando o valor *false* é transmitido 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 Loader. Quando uma transação é confirmada, as entradas que estão marcadas como estando invalidadas localmente não têm nenhum impacto no BackingMap. As alterações que foram limpas no Loader ainda serão confirmadas, mesmo que a entrada tenha sido invalidada.

A invalidação global é especificada quando *true* é transmitido 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 desempenhadas na entrada. Quando uma transação é confirmada, as entradas que estão marcadas como invalidadas globalmente são liberadas 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 Loader. 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 Loader e o banco de dados. O banco de dados configura a coluna de versão para o próximo número na seqüência, que desatualiza o objeto Person na transação. Para atualizar o objeto, o aplicativo desempenha uma invalidação global. O próximo método get emitido obtém a entrada do Loader 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 depende se um método get anterior foi chamado na transação para a chave. Se o aplicativo emitir uma operação get que retorna uma entrada existente no BackingMap ou Loader, a

chamada de método put será interpretada como uma atualização e retornará o valor anterior na transação. Uma chamada de método put sem uma chamada de método get anterior ou uma chamada de método get anterior que não localizou uma entrada é interpretada como uma inserção. A semântica dos métodos insert e update se aplica quando a operação put é confirmada. O método putAll é fornecido para ativar o processamento de inserção e atualização de batch.

Método remove
Remove a entrada do BackingMap e do Loader, se houver uma conectada. 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 de batch sem os valores de retorno.

Método setCopyMode
Especifica um CopyMode para este ObjectMap. Com este método, um aplicativo pode substituir o CopyMode especificado no BackingMap. O CopyMode especificado permanece em vigor até que o método clearCopyMode seja chamado. Os dois métodos são chamados fora dos limites transacionais. Um CopyMode não pode ser alterado no meio de uma transação.

Método touch
Atualiza a hora 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
Atualiza explicitamente uma entrada no BackingMap e no Loader. A utilização deste método indica ao BackingMap e ao Loader 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
Tenta obter um índice denominado construído no BackingMap. O índice não pode ser compartilhado entre encadeamentos e funciona de acordo com as mesmas regras que uma Sessão. O objeto de índice retornado deve ser lançado para a interface de índice do aplicativo correta, como a interface MapIndex, a interface MapRangeIndex ou uma interface de índice customizada.

Interface JavaMap

Uma instância JavaMap é obtida de um objeto ObjectMap. A interface JavaMap possui as mesmas assinaturas de método que ObjectMap, mas com diferente manipulação de exceção. JavaMap estende a interface java.util.Map, portanto, todas as exceções são instâncias da classe java.lang.RuntimeException. Como JavaMap estende a interface java.util.Map, é fácil utilizar rapidamente o ObjectGrid com um aplicativo existente que utiliza uma interface java.util.Map para o armazenamento em cache de objetos.

Obter uma Instância JavaMap

Um aplicativo obtém uma instância JavaMap 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 getJavaMap várias vezes utilizando um ObjectMap específico sempre retorna a mesma instância JavaMap.

Métodos Suportados

A interface JavaMap suporta apenas um subconjunto dos métodos na interface java.util.Map. A interface java.util.Map suporta os seguintes métodos:

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

Todos os demais métodos herdados da interface java.util.Map resultam na exceção `java.lang.UnsupportedOperationException`.

Palavras-chave

O ObjectGrid fornece um mecanismo de invalidação flexível baseado em palavras-chave. Uma *palavra-chave* é uma instância não nula de qualquer objeto serializado. É possível associar palavras-chave a entradas do BackingMap de qualquer maneira.

Associar Palavras-chave a Entradas

Um conjunto de entradas pode ser associado a zero ou mais palavras-chave. Os métodos no ObjectMap e no JavaMap que manipulam entradas, incluindo os métodos `get`, `update`, `put`, `insert` e `touch` possuem versões que permitem que uma única palavra-chave seja associada a todas as entradas alteradas pelo método. Novas associações de palavras-chave são visíveis apenas na transação atual até que a transação seja confirmada. Após uma confirmação, a nova associação é aplicada ao BackingMap e fica visível para outras transações. Se ocorrer um erro durante o processamento de confirmação resultando em um rollback ou se um usuário efetuar rollback de uma transação ativa, será efetuado rollback das novas associações de palavras-chave. O código a seguir demonstra como uma nova entrada é associada a uma palavra-chave:

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

O código de exemplo anterior insere uma nova entrada no BackingMap e a associa à palavra-chave `"New York"`. Um aplicativo que insere entradas também deve

associar palavras-chave quando as entradas forem recuperadas. O aplicativo deve associar palavras-chave a entradas sempre que as obtiver. Considere a seguinte amostra de código:

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

O código de exemplo anterior assegura que a entrada recuperada seja associada à palavra-chave ″New York″. Um aplicativo pode associar várias palavras-chave a uma entrada, mas apenas uma palavra-chave por chamada de método. Para associar mais palavras-chave, emita outra chamada de método, como a amostra a seguir:

```
sess.begin();
Person p = (Person)map.get("Billy", "New York");
map.touch("Billy", "Another keyword");
map.get("Billy", "Yet another keyword");
sess.commit();
```

Palavras-chave Padrão

O método `setDefaultKeyword` nas interfaces ObjectMap e JavaMap fornece uma maneira de associar entradas a uma palavra-chave específica sem utilizar a versão de palavra-chave dos métodos `get`, `insert`, `put`, `update`, ou `touch`. Se versão de palavra-chave de um método for utilizada, a palavra-chave padrão será ignorada e o objeto de palavra-chave fornecido será utilizado.

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

No exemplo anterior, `Billy` está associado à palavra-chave padrão, ″New York″. `Bob` não está associado à palavra-chave padrão, porque uma palavra-chave explícita foi transmitida para a chamada de get para recuperar a entrada Bob. Nenhuma palavra-chave está associada a ″Jimmy″ porque a palavra-chave padrão foi reconfigurada e nenhum argumento de palavra-chave explícito foi transmitido para a chamada de método `get`.

Invalidar Entradas com Palavras-chave

A utilização do método `invalidateUsingKeyword` nas interfaces ObjectMap e JavaMap invalida todas as entradas associadas a uma palavra-chave no BackingMap correspondente. Com esta abordagem, é possível invalidar de forma eficiente entradas relacionadas a uma única operação.

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

No exemplo anterior, a entrada para ″Billy″ é invalidada e não é inserida no BackingMap. A entrada para ″Bob″ não é invalidada, porque foi inserida após a chamada de método `invalidateUsingKeyword`. O método `invalidateUsingKeyword` invalida entradas com base nas associações de palavras-chave quando o método é chamado.

Agrupamento de Palavras-chave

As palavras-chave também podem ser agrupadas em um relacionamento pai-filho. Uma palavra-chave pai pode ter vários filhos e uma palavra-chave filha pode ter vários pais. Por exemplo, se um aplicativo utilizar as palavras-chave ″Dublin″, ″Paris″, ″New York″ e ″Los Angeles″, ele poderá incluir os seguintes agrupamentos de palavras-chave:

- ″USA″ agrupa ″New York″ e ″Los Angeles″
- ″Europe″ agrupa ″Dublin″ e ″Paris″
- ″World″ agrupa ″USA″ e ″Europe″

A invalidação da palavra-chave ″USA″ invalida todas as entradas que estão associadas às palavras-chave ″New York″ e ″Los Angeles″. A invalidação da palavra-chave ″World″ invalida todas as entradas que estão associadas aos agrupamentos ″USA″ e ″Europe″. As associações de palavras-chave são definidas utilizando o método `associateKeyword` na interface do ObjectGrid. A inclusão de palavras-chave filhas em uma palavra-chave pai após uma chamada de método `invalidateUsingKeyword` não causa a invalidação das entradas associadas à palavra-chave filha. O código de exemplo a seguir define o conjunto de associações de palavras-chave descritas:

```
ObjectGrid objectGrid = ...;
objectGrid.associateKeyword("USA", "New York");
objectGrid.associateKeyword("USA", "Los Angeles");
objectGrid.associateKeyword("Europe", "Dublin");
objectGrid.associateKeyword("Europe", "Paris");
objectGrid.associateKeyword("World", "USA");
objectGrid.associateKeyword("World", "Europe");
```

Objetos LogElement e LogSequence

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, existe um LogElement correspondente para fornecer os detalhes da alteração. Os Loaders recebem um objeto LogSequence para um mapa específico sempre que um aplicativo solicita uma limpeza ou confirmação da transação. O Loader itera sobre os LogElements na LogSequence e aplica cada LogElement ao backend.

ObjectGridEventListeners registrados em um ObjectGrid também utilizam 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.

Este tópico descreve quatro interfaces ou classes relacionadas a registros que são fornecidas pela estrutura do ObjectGrid:

- `com.ibm.websphere.objectgrid.plugins.LogElement`
- `com.ibm.websphere.objectgrid.plugins.LogSequence`
- `com.ibm.websphere.objectgrid.plugins.LogSequenceFilter`
- `com.ibm.websphere.objectgrid.plugins.LogSequenceTransformer`

Interface LogElement

Um LogElement representa uma operação em uma entrada durante uma transação. Um objeto LogElement tem os seguintes atributos. Os atributos comumente mais utilizados são os atributos *type* e *current value*:

Atributo *type*
Um elemento de registro *type* indica o tipo de operação que este elemento de registro representa. O *type* pode ser uma das seguintes constantes que estão definidas na interface `LogElement`: `INSERT`, `UPDATE`, `DELETE`, `EVICT`, `FETCH` ou `TOUCH`.

Atributo *undo type*
Retorna a operação que deve ser desempenhada para ″desfazer″ uma alteração anterior feita pela transação na entrada do mapa.

Atributo *current value*
current value representa o novo valor para a operação `INSERT`, `UPDATE` ou `FETCH`. Se a operação for `TOUCH`, `DELETE` ou `EVICT`, current value será nulo. Este valor pode ser lançado no ValueProxyInfo quando um ValueInterface estiver sendo utilizado.

Atributo *CacheEntry*
É possível obter uma referência ao objeto CacheEntry do LogElement e utilizar os métodos definidos no objeto CacheEntry para recuperar as informações necessárias.

Atributo *pending state*
Se *pending state* for `true`, a alteração representada por este elemento de registro ainda não foi aplicada ao loader. Se for `false`, a alteração foi aplicada ao loader, muito provavelmente pela operação de limpeza.

Atributo *versioned value*
Versioned value é um valor que pode ser utilizado para controle de versões.

Atributo *new keywords*
A coleta de new keyword contém as novas palavras-chave que foram associadas a esta entrada.

Atributo *last access time*
Representa a hora do último acesso para a entrada.

Atributos *before image / after image*
Os métodos getter estão disponíveis para obter a imagem do objeto de valor antes ou depois da aplicação das alterações no mapa.

Interface LogSequence

Na maioria das transações, ocorrem operações em mais de uma entrada em um mapa, portanto, são criados vários objetos LogElement. Faz sentido ter um objeto que atua como uma composição de vários objetos LogElement. A interface LogSequence atende esta finalidade contendo uma lista de objetos LogElement. A interface LogSequence possui os seguintes métodos:

Método size
Retorna o número de objetos LogElement na seqüência especificada.

Método getAllChanges
Retorna um iterador de todas as alterações na seqüência de registro especificada.

Método getPendingChanges
Retorna um iterador de todas as alterações pendentes. Provavelmente, será utilizado por um loader para aplicar apenas alterações pendentes no armazenamento persistente.

Método getChangesByKeys
Retorna um iterador dos objetos LogElement que possuem a chave de destino, com base no parâmetro de entrada.

Método getChangesByTypes
Retorna um iterador dos objetos LogElement que são do tipo LogElement especificado.

Método getMapName
Retorna o nome do mapa de suporte ao qual as alterações se aplicam. O responsável pela chamada pode utilizar este nome como entrada para o método `Session.getMap(string)`.

Método isDirty
Retorna se este LogSequence possui LogElements que *sujariam* um Mapa. Ou seja, se o LogSequence contiver objetos LogElement que são de qualquer tipo diferente de Fetch ou Get, LogSequence será considerado ″sujo″.

Método isRollback
Retorna se este LogSequence foi gerado para efetuar rollback de uma transação.

Método getObjectGridName
Retorna o nome do ObjectGrid que hospeda o mapa ao qual estas alterações se aplicam.

LogElement e LogSequence são amplamente utilizados no ObjectGrid 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`

Para obter detalhes adicionais sobre estas interfaces, consulte a documentação da API.

Exemplo de Loader

Esta seção demonstra como os objetos LogSequence e LogElement são utilizados em um Loader. Um *Loader* é utilizado para carregar dados de e para persistir dados para um armazenamento persistente. O método `batchUpdate` da interface do Loader utiliza LogSequence:

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

O método `batchUpdate` é chamado sempre que um ObjectGrid precisa aplicar todas as alterações atuais no Loader. O Loader recebe uma lista de objetos LogElement

para o mapa, encapsulados em um objeto LogSequence. A implementação do método `batchUpdate` deve iterar sobre as alterações e aplicá-las ao backend. O trecho de código a seguir mostra como um Loader utiliza um objeto LogSequence. O snippet itera sobre o conjunto de alterações e constrói até três instruções JDBC (Java Database Connectivity) de batch: uma que tem inserções, uma que tem atualizações e uma terceira instrução que tem exclusões:

```
public void batchUpdate(TxID tx, LogSequence sequence)
throws LoaderException
{
// Obter uma conexão SQL para utilizar.
Connection conn = getConnection(tx);
try
{
// Processar a lista de alterações e construir um conjunto de instruções preparadas
// para executar uma operação SQL update, insert ou delete
// de batch. As instruções são armazenadas em cache em stmtCache.
Iterator iter = sequence.getPendingChanges();
while ( iter.hasNext() )
{
LogElement logElement = (LogElement)iter.next();
Object key = logElement.getCacheEntry().getKey();
Object value = logElement.getCurrentValue();
switch ( logElement.getType().getCode() )
{
case LogElement.CODE_INSERT:
buildBatchSQLInsert( key, value, conn );
break;
case LogElement.CODE_UPDATE:
buildBatchSQLUpdate( key, value, conn );
break;
case LogElement.CODE_DELETE:
buildBatchSQLDelete( key, conn );
break;
}
}
// Executar as instruções de batch que foram construídas pelo loop acima.
Collection statements = getPreparedStatementCollection( tx, conn );
iter = statements.iterator();
while ( iter.hasNext() )
{
PreparedStatement pstmt = (PreparedStatement) iter.next();
pstmt.executeBatch();
}
}
catch (SQLException e)
{
LoaderException ex = new LoaderException(e);
throw ex;
}
}
```

A amostra anterior ilustra a lógica de alto nível de processamento do argumento LogSequence e os detalhes de como uma instrução SQL `insert`, `update` ou `delete` é construída não são ilustrados. Este exemplo ilustra que o método `getPendingChanges` é chamado no argumento LogSequence para obter um iterador de objetos LogElement que um Loader precisa processar e o método `LogElement.getType().getCode()` é utilizado para determinar se um LogElement destina-se a uma operação SQL `insert`, `update` ou `delete`.

Amostra de Evictor

Este exemplo explora como LogSequence e LogElement são utilizados em 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 suporte. Também relatará as entradas que foram
* liberadas com êxito.
*
* @param sequence LogSequence de alterações no mapa
*/
void apply(LogSequence sequence);
```

Para obter informações sobre como o método apply utiliza LogSequence, consulte a amostra de código no tópico “Evictors” na página 191.

Interfaces LogSequenceFilter e LogSequenceTransformer

Às vezes, é necessário filtrar os objetos LogElement para que apenas os objetos LogElement com alguns critérios sejam aceitos e rejeitar outros objetos. Por exemplo, você pode serializar um determinado LogElement com base em algum critério. LogSequenceFilter resolve este problema com o seguinte método:

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

Este método retorna true se o LogElement especificado tiver que ser utilizado na operação e retorna false se o LogElement especificado não tiver que ser utilizado.

LogSequenceTransformer é uma classe que utiliza a função LogSequenceFilter descrita acima. 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 seguinte:

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

Este método lê o formulário serializado de seqüência de registro, que foi criado pelo método `serialize` a partir do fluxo de entrada de objetos fornecido.

Bloqueio

Este tópico descreve a estratégia de bloqueio suportada por um BackingMap do ObjectGrid.

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

- Bloqueio Pessimista

- Bloqueio Otimista
- Nenhum

A seguir está um exemplo de 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:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm = og.defineMap("map2");
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

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

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 rastrear as entradas que estão bloqueadas por uma ou mais transações. Se existirem muitas entradas no mapa hash, mais depósitos de bloqueios significam melhor desempenho. O risco de colisões de sincronização Java é menor conforme o número de depósitos aumenta. Mais depósitos de bloqueios também resultam em maior simultaneidade. O exemplo a seguir mostra como um aplicativo pode configurar o número de depósitos de bloqueios a serem utilizados para um BackingMap especificado:

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

Mais uma vez, 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 o número de depósitos de bloqueios a serem utilizados. Utilizar um número primo assegura uma distribuição uniforme de entradas do mapa sobre os depósitos de bloqueios. Um bom ponto de partida para melhor desempenho é configurar o número de depósitos de bloqueios como aproximadamente dez por cento do número esperado de entradas do BackingMap.

Bloqueio Pessimista

Utilize a estratégia de bloqueio pessimista para ler e gravar mapas quando outras estratégias de bloqueio não forem possíveis.

Quando um Mapa do ObjectGrid é configurado para utilizar a estratégia de bloqueio PESSIMISTIC, um bloqueio de transação pessimista para uma entrada do mapa é obtido quando uma transação obtém pela primeira vez a entrada do BackingMap. O bloqueio pessimista fica retido até que o aplicativo conclua a transação. Geralmente, a estratégia de bloqueio pessimista é utilizada nas seguintes situações:

- O BackingMap é configurado com ou sem um loader e as informações de controle de versões não estão disponíveis.

- O BackingMap é utilizado diretamente por um aplicativo que precisa de ajuda do ObjectGrid para controle de simultaneidade.
- 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, falhas de atualização otimistas ocorrem com freqüência ou a recuperação de falha otimista é difícil de ser tratada por um aplicativo.

Métodos ObjectMap e Modos de Bloqueio

Quando um aplicativo utiliza os métodos da interface ObjectMap, o ObjectGrid tenta automaticamente um bloqueio pessimista para a entrada do mapa que está sendo acessada. O ObjectGrid utiliza os seguintes modos de bloqueio com base em qual método o aplicativo chama na interface ObjectMap:

- Os métodos `get` e `getAll` adquirem um *bloqueio S*, ou um modo de bloqueio compartilhado para a chave de uma entrada do mapa. O bloqueio S fica retido até que a transação seja concluída. 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.
- `put`, `putAll`, `remove`, `removeAll`,`insert`, `update` e `touch` adquirem um modo de *bloqueio X* ou de bloqueio exclusivo para a chave de uma entrada do 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 resultante de um modo de bloqueio X é assegurar que apenas uma transação possa inserir, atualizar ou remover uma

entrada do mapa e impedir a perda de atualizações quando mais de uma transação estiver tentando atualizar a mesma entrada do mapa.

A tabela a seguir é uma matriz de compatibilidade de modo de bloqueio que resume os modos de bloqueio descritos e é utilizada para determinar quais modos de bloqueio são compatíveis entre si. 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 for exibido **Sim** na coluna, o modo de bloqueio solicitado pela outra transação é concedido, porque é compatível com o modo de bloqueio já concedido. **Não** 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 8. Compatibilidade e Força de Modos de Bloqueio

bloqueio	**bloqueios compatíveis**			**força**
	S (compartilhado)	U (para upgrade)	X (exclusivo)	
S (Compartilhado)	Sim	Sim	Não	mais fraco
U (Para Upgrade)	Sim	Não	Não	normal
X (Exclusivo)	Não	Não	Não	mais forte

Tempo Limite de Espera de Bloqueio

Cada BackingMap do ObjectGrid possui um valor de tempo limite de espera de bloqueio padrão. O valor de tempo limite é utilizado para assegurar que um aplicativo não esperará para sempre que um modo de bloqueio seja concedido devido a uma condição de conflito que ocorre devido a um erro do 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 suporte map1 como 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 o método setLockStrategy e o método setLockTimeout antes de chamar os métodos initialize ou getSession na instância do ObjectGrid. O parâmetro do método setLockTimeout é um inteiro primitivo Java que especifica o número de segundos que o ObjectGrid espera 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. Consulte Capítulo 14, “Resolução de Problemas”, na página 349 e Capítulo 11, “Boas Práticas de Desempenho do ObjectGrid”, na página 329 para obter informações adicionais.

Conflitos

Considere a seguinte seqüência de pedidos de modo de bloqueio:

```
O bloqueio X é concedido à transação 1 para key1.
O bloqueio X é concedido à transação 2 para key2.
O bloqueio X é solicitado pela transação 1 para key2.
(A transação 1 bloqueia a espera do bloqueio pertencente à transação 2.)
O bloqueio X é solicitado pela transação 2 para key1.
(A transação 2 bloqueia a espera do bloqueio pertencente à transação 1.)
```

A seqüência anterior é o exemplo de conflito clássico de duas transações que tentam adquirir mais de um único bloqueio 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 confirmação, o ObjectGrid determina as chaves para as entradas do mapa que precisam ser bloqueadas e solicita os modos de bloqueio na seqüência de chaves. Com este método, o ObjectGrid evita a grande maioria dos conflitos clássicos. No entanto, o ObjectGrid não pode evitar todos os cenários de conflito possíveis. Existem alguns 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 no qual o ObjectGrid pode detectar um conflito sem ter que esperar que ocorra um tempo limite de espera de bloqueio. 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("Billy");
// Billy fez aniversário, ele ficou um ano mais velho.
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
sess.commit();
```

Nesta situação, a esposa de Billy deseja torná-lo mais velho do que ele é, e Billy e sua esposa executam esta transação simultaneamente. Nesta situação, as duas transações possuem um modo de bloqueio S na entrada **Billy** do mapa PERSON como resultado da chamada de método person.get(″Billy″). Como resultado da chamada de método person.put (″Billy″, 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 ObjectGrid emite uma exceção `LockDeadlockException` em vez de uma exceção `LockTimeoutException`. Consulte “LockDeadlockException” na página 353 para obter maiores informações.

O aplicativo pode evitar a exceção `LockDeadlockException` para o exemplo anterior utilizando a estratégia de bloqueio OPTIMISTIC em vez da estratégia de bloqueio PESSIMISTIC. A utilização da estratégia de bloqueio OPTIMISTIC é a solução preferida quando o mapa é lido em sua maior parte e as atualizações no mapa não são freqüentes. Consulte “Bloqueio Otimista” na página 137 para obter detalhes adicionais sobre a estratégia otimista. Se a estratégia de bloqueio PESSIMISTIC tiver que ser utilizada, o método getForUpdate poderá ser utilizado em vez do método get no exemplo acima. Se isso ocorrer, a primeira transação para chamar o método getForUpdate adquirirá um modo de bloqueio U em vez de um modo de bloqueio S. Este modo de bloqueio causa o bloqueio da segunda transação quando 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 Billy. A primeira transação não é bloqueada quando tenta fazer upgrade do modo de bloqueio U para um modo de bloqueio X como resultado da chamada de método put a partir da primeira transação. Este recurso demonstra por que o modo de bloqueio U é chamado de modo de bloqueio ″para upgrade″. 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 em vez do método get quando uma estratégia de bloqueio PESSIMISTIC estiver sendo utilizada.

Importante: Esta solução não impede que transações de leitura leiam uma entrada do 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.

É necessário ter atenção quando uma transação chama o método getForUpdate em mais de uma entrada do mapa para assegurar que os bloqueios U sejam adquiridos na mesma ordem por 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 seqüência 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 em vez de no tempo de confirmação, o ObjectGrid não pode ordenar os pedidos de bloqueio como faz durante o ciclo de confirmação. 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, portanto, a condição de espera de bloqueio e o conflito podem ocorrer no backend em vez de ocorrer no gerenciador de bloqueios do ObjectGrid. Considere a seguinte transação:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
sess.begin();
activeTran = true;
Person p = (IPerson)person.get("Billy");
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
person.flush();
...
p = (IPerson)person.get("Tom");
p.setAge( p.getAge() + 1 );
sess.commit();
activeTran = false;
}
finally
{
if ( activeTran ) sess.rollback();
}
```

Suponha que alguma outra transação também tenha atualizado a pessoa Tom, chamado o método flush e, em seguida, atualizado a pessoa Billy. Se esta situação tiver ocorrido, a seguinte intercalação das duas transações resultará em uma condição de conflito do banco de dados:

```
O bloqueio X é concedido à transação 1 para "Billy" quando a limpeza é executada.
O bloqueio X é concedido à transação 2 para "Tom" quando a limpeza é executada.
O bloqueio X é solicitado pela transação 1 para "Tom" durante o processamento
de confirmação.
(A transação 1 bloqueia a espera do bloqueio pertencente à transação 2.)
O bloqueio X é solicitado pela transação 2 para "Billy" durante o processamento
de confirmação.
(A transação 2 bloqueia a espera do bloqueio pertencente à transação 1.)
```

Este exemplo demonstra que a utilização do método flush pode causar um conflito no banco de dados em vez de causar no ObjectGrid. 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 Loader for conectado ao BackingMap. O exemplo anterior também ilustra outra razão pela qual o ObjectGrid tem um mecanismo de tempo limite de espera de bloqueio. Uma transação que está esperando um bloqueio do banco de dados pode esperar enquanto possui um bloqueio de entrada do mapa do ObjectGrid. Por isso, os problemas no nível do banco de dados podem causar tempos de espera excessivos para um modo de bloqueio do ObjectGrid e resultar em uma exceção `LockTimeoutException`.

Tratamento de exceções

Os exemplos neste tópico não possuem uma manipulação de exceção. Para evitar que os bloqueios sejam retidos por quantidades de tempo excessivas quando ocorre uma exceção `LockTimeoutException` ou uma `LockDeadlockException`, um aplicativo precisa assegurar que capturará exceções inesperadas e chamará o método rollback quando ocorrer 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("Billy");
// Billy fez aniversário, ele ficou um ano mais velho.
p.setAge( p.getAge() + 1 );
person.put( "Billy", 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.

Bloqueio Otimista

A estratégia de bloqueio otimista acredita que duas transações não podem tentar atualizar a mesma entrada do mapa durante a execução simultânea. Acreditando nisso, não é necessário manter um modo de bloqueio para a existência da transação, porque é improvável que mais de uma transação possa atualizar a entrada do mapa simultaneamente.

A estratégia de bloqueio otimista geralmente é utilizada quando:

- Um BackingMap é configurado com ou sem um loader e as informações de controle de versões estão disponíveis.
- Um BackingMap é lido em sua maior parte. Ou seja, as transações lêem freqüentemente entradas do mapa e, apenas ocasionalmente, inserem, atualizam ou removem uma entrada do mapa.
- Um BackingMap é inserido, atualizado ou removido mais freqüentemente do que é lido, mas as transações raramente colidem na mesma entrada do mapa.

Assim como a estratégia de bloqueio pessimista, os métodos na interface ObjectMap determinam como o ObjectGrid tenta automaticamente adquirir um modo de bloqueio para a entrada do mapa que está sendo acessada. No entanto, a seguir estão algumas diferenças muito importantes entre as estratégias pessimistas e otimistas:

- Assim como a estratégia de bloqueio pessimista, um *modo de bloqueio S* é adquirido pelos métodos `get` e `getAll` quando o método é chamado. No entanto, com o bloqueio otimista, 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. A finalidade da aquisição do modo de bloqueio é que o ObjectGrid pode assegurar que apenas dados confirmados de outras transações fiquem visíveis para a transação atual. Após o ObjectGrid ter verificado que os dados foram confirmados, o modo de bloqueio S é liberado. No tempo de confirmação, é desempenhada uma verificação de controle de versões otimista para assegurar 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 buscada no mapa antes de ser atualizada, invalidada ou excluída, o tempo de execução do ObjectGrid implicitamente buscará a entrada 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 manipulado 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 ao aplicativo.
- Todos os demais métodos ObjectMap são manipulados exatamente como são manipulados para a estratégia de bloqueio pessimista. 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.

Esta estratégia de bloqueio é chamada de otimista porque existe uma perspectiva otimista. A estratégia de bloqueio otimista significa que duas transações não podem tentar atualizar a mesma entrada do mapa durante execução simultânea. Acreditando nisso, não é necessário manter um modo de bloqueio para a existência da transação, porque é improvável que mais de uma transação possa atualizar a entrada do mapa simultaneamente. No entanto, como um modo de bloqueio não foi retido, outra transação simultânea provavelmente poderia atualizar a entrada do mapa após a transação atual ter liberado seu modo de bloqueio S. Para lidar com esta possibilidade, o ObjectGrid obtém um bloqueio X no tempo de confirmação e desempenha uma verificação de controle de versões otimista para verificar se nenhuma outra transação alterou a entrada do mapa desde que a transação atual leu a entrada do mapa a partir de BackingMap. Se outra transação alterar a entrada do mapa, a verificação de versão falhará e ocorrerá uma exceção `OptimisticCollisionException`. Esta exceção força o rollback da transação atual e toda a transação deve ser tentada novamente pelo aplicativo. A estratégia de bloqueio otimista é muito útil quando um mapa é lido em sua maior parte e é improvável que ocorram atualizações na mesma entrada do mapa.

Nenhuma Estratégia de Bloqueio de BackingMap

Quando um BackingMap é configurado para utilizar uma estratégia de bloqueio de NONE, não são obtidos bloqueios de transação para uma entrada do mapa. Um cenário no qual esta estratégia é útil é quando um aplicativo é um gerenciador de persistência, como um contêiner EJB (Enterprise JavaBeans) J2EE (Java 2 Platform, Enterprise Edition) ou utiliza Hibernate para obter dados persistentes. Neste cenário, o BackingMap é configurado sem um loader e está sendo utilizado como um cache de dados pelo gerenciador de persistência. O gerenciador de persistência neste cenário fornece controle de simultaneidade entre transações que estão acessando as mesmas entradas de Mapa do ObjectGrid. O ObjectGrid não precisa obter nenhum bloqueio de transação com a finalidade de controle de simultaneidade. Isto está assumindo que o gerenciador de persistência não libera seus bloqueios de transação antes de atualizar o mapa do ObjectGrid com alterações confirmadas. Se este não for o caso, deverá ser utilizada uma estratégia de bloqueio PESSIMISTIC ou OPTIMISTIC. 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 da liberação dos bloqueios de transação do gerenciador de persistência, poderá ser utilizada uma estratégia de bloqueio NONE. Se ocorrer a atualização do mapa do

ObjectGrid após a liberação dos bloqueios de transação do gerenciador de persistência, será requerida a estratégia de bloqueio OPTIMISTIC ou PESSIMISTIC.

Outro cenário em que a estratégia de bloqueio NONE pode ser utilizada é quando o aplicativo utiliza um BackingMap diretamente e um Loader é configurado para o mapa. Neste cenário, o loader utiliza o suporte ao controle de simultaneidade fornecido por um RDBMS (Relational Database Management System) utilizando JDBC (Java Database Connectivity) ou Hibernate para acessar dados em um banco de dados relacional. A implementação do loader pode utilizar uma abordagem otimista ou pessimista.

Um loader que utiliza um bloqueio otimista ou uma abordagem de controle de versões ajuda a obter a maior quantidade de simultaneidade e desempenho. Para obter informações adicionais sobre como implementar uma abordagem de bloqueio otimista, consulte a seção “OptimisticCallback” na página 211 no tópico “Considerações sobre o Loader” na página 207.

Um loader que utiliza o suporte ao bloqueio pessimista do backend subjacente pode utilizar o parâmetro **forUpdate** transmitido no método get na interface do Loader. Este parâmetro será configurado como *true* se o método `getForUpdate` da interface do ObjectMap tiver sido utilizado pelo aplicativo para obter os dados. O loader pode utilizar este parâmetro para determinar se solicitará um bloqueio para upgrade na linha que está sendo lida. Por exemplo, o DB2 obtém um bloqueio para upgrade quando uma instrução SQL select contém uma cláusula `for update`. Esta abordagem oferece a mesma prevenção de conflito descrita no tópico Bloqueio Pessimista.

Segurança do ObjectGrid

Utilize mecanismos de segurança do ObjectGrid para proteger o acesso a dados do mapa e tarefas de gerenciamento por meio de configuração ou de programação.

O ObjectGrid fornece mecanismos de segurança para proteger acessos a dados do mapa e tarefas de gerenciamento. A segurança do ObjectGrid é construída sobre o mecanismo JAAS (Java Authentication and Authorization Services). O JAAS é uma parte integral da Segurança do Java 2.

Esta seção descreve os mecanismos de segurança do ObjectGrid e como utilizar as APIs de segurança do ObjectGrid.

- “Visão Geral de Segurança de ObjectGrid” na página 140 fornece uma visão geral da segurança do ObjectGrid.
- “Segurança do Servidor do Cliente” na página 144 descreve a segurança do servidor do cliente para o modelo de programação do ObjectGrid distribuído.
- “Segurança do ObjectGrid Local” na página 164 descreve a segurança do ObjectGrid local.
- “Autorização” na página 170 descreve o mecanismo de autorização e plug-ins relacionados que se aplicam ao modelo de programação dos ObjectGrids distribuído e local.
- “Segurança do Cluster do ObjectGrid” na página 180 descreve o mecanismo de segurança e plug-ins relacionados ao cluster do ObjectGrid.
- “Segurança do Gateway” na página 183 discute a segurança do gateway.
- “Integração de Segurança com o WebSphere Application Server” na página 185 realça a integração com o WebSphere Application Server.

A maior parte do código de amostra mostrado nesta seção é proveniente das amostras fornecidas pelo ObjectGrid. É possível localizar a visão geral de amostra de segurança no Capítulo 5, “Amostras do ObjectGrid”, na página 61.

Visão Geral de Segurança de ObjectGrid

ObjectGrid é um sistema de armazenamento em cache distribuído. O acesso aos dados em cache pode ser seguro. Geralmente, a segurança é baseada em três conceitos principais:

- *Autenticação Confiável*: determina de maneira confiável a identidade do solicitante.
- *Autorização*: concede direitos de acesso ao solicitante com permissões.
- *Transporte Seguro*: transmite os dados com segurança por meio de redes.

ObjectGrid fornece segurança nos seguintes aspectos:

- “Segurança do Servidor do Cliente” cuida da segurança de autenticação e da comunicação de servidor cliente utilizando SSL (Secure Sockets Layer).
- Os mecanismos de “Autorização” na página 141 garantem que apenas os clientes autorizados podem acessar os dados de mapa e tarefas de gerenciamento de ObjectGrid.
- “Segurança do Cluster do ObjectGrid” na página 141 verifica se apenas os servidores autorizados podem juntar-se ao cluster de ObjectGrid.
- “Segurança do Gateway” na página 141 cuida da autenticação de cliente do gateway.
- “Segurança do ObjectGrid Local” na página 142 fornece um mecanismo de segurança quando o aplicativo instancia diretamente a instância de ObjectGrid.

A segurança de ObjectGrid é construída com base na arquitetura baseada em padrão aberto e fornece vários pontos de plug-in para customização. O mecanismo de plug-in tem uma função importante. O ObjectGrid também fornece alguma implementação interna para estes plug-ins. Algumas implementações servem para utilização de produção out-of-box e outras servem para fins de teste ou de amostra. Consulte “Plug-ins de Segurança” na página 142 para obter um resumo de plug-ins e de implementações internas.

Segurança do Servidor do Cliente

O ObjectGrid suporta estrutura de servidor do cliente distribuída. Uma infra-estrutura de segurança do servidor do cliente está disponível para proteger o acesso aos servidores do ObjectGrid.

Um cliente do ObjectGrid pode utilizar qualquer credencial desejada para autenticação no servidor do ObjectGrid. Deve ser estabelecido um contrato entre clientes e servidores para que esta credencial seja entendida pelo mecanismo de autenticação de servidor. Quando o SSL (Secure Sockets Layer) é utilizado, o cliente também pode utilizar certificados SSL para autenticação no servidor do ObjectGrid.

Para proteger a comunicação do servidor do cliente, o ObjectGrid suporta SSL. O protocolo SSL fornece segurança da camada de transporte com autenticidade, integridade e sigilo, para uma conexão segura entre um cliente e servidor do ObjectGrid. Alguns dos recursos de segurança fornecidos pelo SSL são: criptografia de dados para impedir a exposição de informações sigilosas durante o percurso dos dados, assinatura de dados para impedir modificação não autorizada de dados

durante o percurso dos dados e autenticação do cliente e do servidor para assegurar que você esteja falando com a pessoa ou máquina apropriada. SSL pode ser eficaz na proteção de um ambiente corporativo.

Consulte “Segurança do Servidor do Cliente” na página 144 para obter maiores informações.

Autorização

As autorizações de ObjectGrid são baseadas em subjects e permissões. No ObjectGrid, existem duas categorias de permissões: permissões para acesso a dados e permissões para tarefas de gerenciamento. É possível utilizar o JAAS (Java Authentication and Authorization Services) para autorizar o acesso ou conectar seus próprios mecanismos para manipular as autorizações.

Consulte “Autorização” na página 170 para obter maiores informações.

Segurança do Cluster do ObjectGrid

Em um ambiente seguro, um servidor deve poder verificar a autenticidade de outro servidor. O ObjectGrid utiliza um mecanismo de cadeia de chaves de segredo compartilhado para esta finalidade. Este mecanismo de chave secreta é semelhante a uma senha compartilhada. Todos os servidores do ObjectGrid estão de acordo com um segredo compartilhado. Quando um servidor se junta ao cluster, ele é desafiado a apresentar a cadeia secreta. Se a cadeia secreta do servidor de junção corresponder à do servidor master, o servidor de junção poderá juntar-se ao cluster; caso contrário, o pedido de junção será rejeitado.

Não é seguro enviar um segredo em texto sem formatação. A infra-estrutura de segurança do ObjectGrid fornece um plug-in SecureTokenManager para permitir que o servidor “proteja” este segredo antes de enviá-lo. A maneira de implementação da operação de “proteção” é aberta. O ObjectGrid fornece uma implementação out-of-box, na qual a operação de “proteção” é implementada para criptografar e assinar o segredo.

Consulte “Segurança do Cluster do ObjectGrid” na página 180 para obter informações adicionais

Segurança do Gateway

Um gateway do ObjectGrid serve como um ponto para delegar os pedidos de gerenciamento de clientes ao servidor do ObjectGrid. O gateway de gerenciamento acomoda um conjunto de mbeans. O cliente do gateway chama estes mbeans para administrar ou monitorar servidores do ObjectGrid.

A comunicação entre o gateway de gerenciamento e o servidor utiliza o mecanismo de comunicação do servidor do cliente do ObjectGrid, no qual o gateway é tratado como um cliente do ObjectGrid. A comunicação entre o cliente do gateway e o gateway (servidor MBean) pode ser protegida por SSL. Este recurso é fornecido pela camada do conector JMX, que é implementada pelo projeto de código aberto mx4j. O ObjectGrid requer o mx4j apropriado para fazer o gateway funcionar.

Para a autenticação, o gateway propaga a credencial apresentada pelo cliente do gateway para o servidor do ObjectGrid. A autenticação e autorização são aplicadas em servidores do ObjectGrid.

Consulte “Segurança do Gateway” na página 183 para obter maiores informações.

Segurança do ObjectGrid Local

No WebSphere Extended Deployment Server release 6.0, foi introduzido o modelo de programação do ObjectGrid local. Neste modelo, o aplicativo instancia e utiliza diretamente uma instância do ObjectGrid. Seu aplicativo e as instâncias do ObjectGrid estão na mesma JVM (Java Virtual Machine). Não existe nenhum conceito de cliente ou servidor neste modelo.

A autenticação não é suportada no modelo de programação do ObjectGrid local. Seus aplicativos devem gerenciar sua própria autenticação e, em seguida, transmitir o objeto Subject autenticado para o ObjectGrid.

É utilizado um mecanismo de autorização para o modelo de programação do ObjectGrid local igual ao utilizado para o modelo de servidor do cliente.

Consulte “Segurança do ObjectGrid Local” na página 164 para obter maiores informações.

Plug-ins de Segurança

A estrutura de segurança do ObjectGrid é complementada por plug-ins de segurança. Estes plug-ins podem ser implementados para estender ou customizar a estrutura de segurança.

Um exemplo é o plug-in CredentialGenerator, representado pela interface com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator. Quando um cliente do ObjectGrid conecta-se a um servidor do ObjectGrid, o método getCredential() deste plug-in é chamado para gerar um objeto de Credencial. Este objeto de Credencial é, então, enviado ao servidor. O servidor, então, utiliza este objeto de Credencial para autenticação utilizando o plug-in Authenticator, que é representado pela interface thecom.ibm.websphere.objectgrid.security.plugins.CredentialGenerator.Authenticator.

Os plug-ins têm uma função importante na estrutura de segurança do ObjectGrid. É possível implementar o CredentialGenerator para gerar uma credencial específica, por exemplo, um par de ID do usuário e senha, um registro do kerberos ou um token de segurança. É possível implementar o plug-in Authenticator para autenticar o cliente. Se desejar, é possível implementar o plug-in Authenticator para suportar a senha do usuário ou um token de segurança.

Todos os plug-ins de segurança que podem ser utilizados estão na tabela a seguir:

Tabela 9. Plug-ins de Segurança

Categoria	Nome da Classe do Plug-in	Instância
Autenticação	`com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator`	cliente
	`com.ibm.websphere.objectgrid.security.plugins.Credential`	cliente
	`com.ibm.websphere.objectgrid.security.plugins.Authenticator`	servidor

Tabela 9. Plug-ins de Segurança (continuação)

Categoria	Nome da Classe do Plug-in	Instância
Autorização	com.ibm.websphere.objectgrid.security.plugins.MapAuthorization	ObjectGrid
	com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization	Cluster
Segurança do Cluster do ObjectGrid	com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager	servidor
outro	com.ibm.websphere.objectgrid.security.plugins.SubjectSource	ObjectGrid local
	com.ibm.websphere.objectgrid.security.plugins.SubjectValidation	ObjectGrid local

O diagrama a seguir mostra os plug-ins e as instâncias aplicados. Por exemplo, o plug-in MapAuthorization se aplica a instâncias do ObjectGrid, mas o AdminAuthorization se aplica a instâncias do servidor.

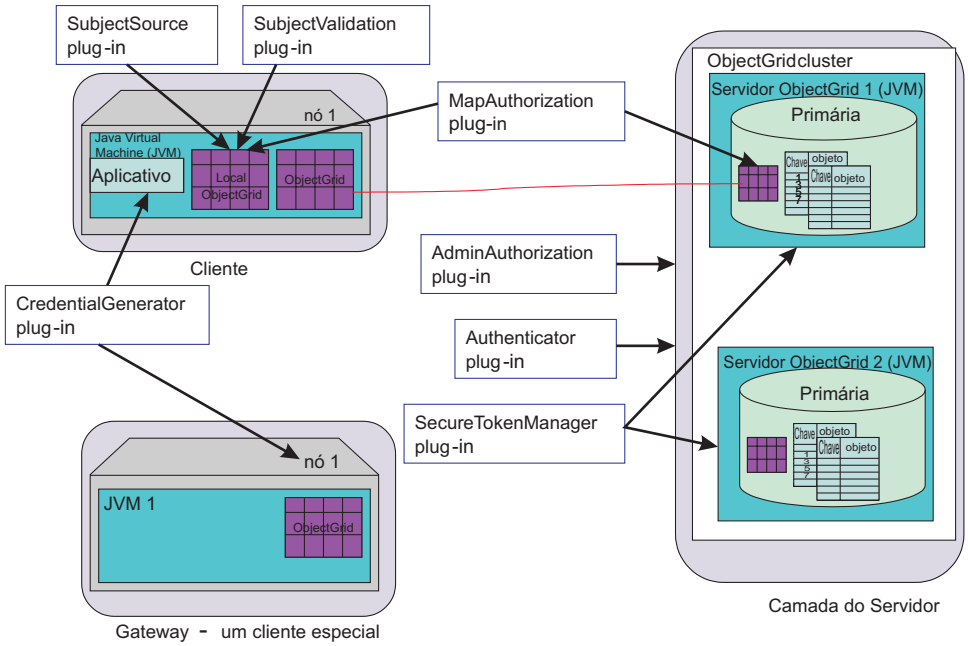


Figura 17. Plug-ins de Segurança

A tabela a seguir exibe a implementação interna. A coluna de finalidade exibe a finalidade. A finalidade pode ser para produção out-of-box ou para teste.

Tabela 10. Implementações de Segurança Interna

Plug-in	nome da classe interna	finalidade
Credencial do Gerador de Credenciais	com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator	produção
	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator	produção
	com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential	produção
	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential	produção
	com.ibm.websphere.objectgrid.security.plugins.builtins.ClientCertificateCredential	produção
Authenticator	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator	produção
	com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator	teste
	com.ibm.websphere.objectgrid.security.plugins.builtins.CertificateMappingAuthenticator	teste
	com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator	teste
Map Authorization	com.ibm.websphere.objectgrid.security.plugins.builtins.JAASMapAuthorizationImpl	produção
	com.ibm.websphere.objectgrid.security.plugins.builtins.TAMMapAuthorizationImpl	teste
SubjectSource	com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl	produção
Subject Validation	com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl	produção

Segurança do Servidor do Cliente

Este tópico descreve o mecanismo de autenticação e como proteger a comunicação do servidor do cliente.

A segurança do servidor do cliente possui os seguintes aspectos importantes:

- Como “Ativar a Segurança do Servidor do Cliente”
- Como obter uma credencial que representa o cliente com a “Credencial e Gerador de Credenciais ” na página 145
- Como configurar parâmetros utilizados para a configuração SSL utilizando “Comunicação Segura ” na página 162
- Como autenticar o cliente no lado do servidor utilizando um “Authenticator” na página 152

Ativar a Segurança do Servidor do Cliente

Ativar a Segurança do Cliente

Para ativar a segurança na segurança do cliente, configure a propriedade securityEnabled no arquivo `security.ogclient.props` como **true**. O ObjectGrid fornece um arquivo de gabarito de propriedade de segurança do cliente, o arquivo `security.ogclient.props`, no diretório `[WAS_HOME]/optionalLibraries/ObjectGrid/properties` para uma

instalação do WebSphere ou no diretório /ObjectGrid/properties para uma instalação de servidor mista. É possível modificar este arquivo de gabarito com valores apropriados.

A descrição da propriedade securityEnabled é a seguinte:

securityEnabled (true, false+)
Esta propriedade indica se a segurança está ativada. Quando um cliente se conecta a um servidor, os valores securityEnabled no lado 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.

A interface com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration representa o arquivo security.ogclient.props. É possível utilizar a API pública com.ibm.websphere.objectgrid.security.config. ClientSecurityConfigurationFactory para criar uma instância desta interface com valores padrão ou é possível criar uma instância transmitindo o arquivo de propriedades de segurança do cliente do ObjectGrid. O arquivo security.ogclient.props contém outras propriedades.

Ativar Segurança do Servidor
Para ativar a segurança no lado do servidor, é possível configurar a propriedade securityEnabled no XML do cluster como true. Eis um exemplo:

```
<cluster>
<objectGrid name="cluster" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300">
```

Credencial e Gerador de Credenciais

Ao conectar-se a um servidor, um cliente precisa apresentar sua própria credencial. Uma credencial de cliente é representada por uma interface com.ibm.websphere.objectgrid.security.plugins.Credential. A credencial pode conter um par de senha do usuário, um registro do Kerberos e outros.

A interface da credencial é a seguinte:

```
package com.ibm.websphere.objectgrid.security.plugins;
import java.io.Serializable;
/**
* Esta interface representa uma credencial utilizada por um cliente do ObjectGrid.
* Ela representa uma identidade do cliente. Esta credencial é enviada para o
* servidor do ObjectGrid para autenticação. Ela deve ser seriável.
*
* Uma credencial precisa implementar os métodos equals(Object) e
* hashCode(). Dois objetos de Credencial são considerados iguais
* apenas se representarem a mesma identidade e informações de segurança. Por
* exemplo, se a credencial contiver um ID do usuário e uma senha. Duas
* credenciais são iguais apenas se seus IDs do usuário e suas senhas forem
* iguais.
*
* O ObjectGrid fornece três implementações internas para esta interface:
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* ClientCertificateCredential:
* Uma credencial que contém uma cadeia de certificados SSL.
* com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential:
* Uma credencial que contém um par de ID do usuário e senha.
* com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential:
* Uma credencial que contém autenticação específica do WebSphere Application
```

```
* Server e tokens de autorização.
*
* Consulte a respectiva documentação da API para obter detalhes adicionais.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see CredentialGenerator
*/
public interface Credential extends Serializable {
/**
* Verifica dois objetos de Credencial por igualdade.
*
* Dois objetos de Credencial são considerados iguais apenas se
* representarem a mesma identidade e informações de segurança.
*
* @param o o objeto que está sendo testado para igualdade com este objeto.
*
* @return true se os dois objetos de Credencial forem equivalentes.
*/
boolean equals(Object o);
/**
* Retorna o código hash do objeto de Credencial
*
* @return o código hash do objeto de Credencial
*/
int hashCode();
}
```

Esta interface define explicitamente os métodos equals(Object) e hashCode(). Estes métodos são importantes para garantir o comportamento. Os objetos Subject autenticados são armazenados em cache com base nos objetos de Credencial no lado do servidor.

O ObjectGrid fornece três implementações padrão para as interfaces de Credencial:

1. A implementação com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential. Esta credencial contém um par de ID do usuário e senha.
2. A implementação com.ibm.websphere.objectgrid.security.plugins.builtins.ClientCertificateCredential. Esta credencial contém uma cadeia de certificados clientes. Esta credencial pode ser utilizada para autenticação do certificado cliente do ObjectGrid. Não é possível criar esta credencial no lado cliente. Ela tem que ser gerada pelo servidor como parte do protocolo de reconhecimento SSL.
3. A implementação com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential. Esta credencial 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.

Consulte a documentação da API para obter detalhes adicionais.

O ObjectGrid também fornece um plug-in para gerar uma credencial. Este plug-in é representado pela interface com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator. A seguir estão as interfaces CredentialGenerator:

```
/**
* Este plug-in é utilizado para obter uma Credencial que representa este
* cliente. É um depósito de informações do provedor para o objeto de
* Credencial.
* Uma implementação de exemplo é retornar um objeto de Credencial
* que contém um par ID do usuário e senha. A implementação da Credencial
* gerada por uma implementação desta classe devem ser entendida pelo
* plug-in Authenticator do servidor.
*
* Uma classe de implementação desta interface deve ter um construtor padrão.
* Ao ativar o cliente em um ambiente seguro, configure o nome da
* classe de implementação (credentialGeneratorClass) no arquivo de
* propriedades de configuração de segurança do cliente. O tempo de
* execução do cliente constrói um objeto desta
* classe de implementação e chama getCredential() para obter a Credencial
* para conectar-se a um cluster do ObjectGrid.
*
* Os usuários também podem especificar as propriedades adicionais para este
* depósito de informações do provedor utilizando
* a propriedade credentialGeneratorProps no arquivo de propriedades de
* configuração de segurança do servidor. Estas propriedades são
* transmitidas para este depósito de informações do provedor utilizando
* o método setProperties(String). Portanto, é possível
* customizar seu depósito de informações do provedor.
*
* Também é possível configurar o CredentialGenerator programaticamente
* chamando o método ClientSecurityCinfiguration.setCredentialGenerator
* (CredentialGenerator).
*
*
* Por exemplo, é possível ter as seguintes configurações no arquivo de
* propriedades de configuração de segurança do cliente:
* credentialGeneratorClass=com.myco.CredGenFactory
*
* credentialGeneratorProps=user1 password1
*
*
* , uma Cadeia "user1 password1" é transmitida para o método
* setProperties(String),
* com o "user1" indicando o nome do usuário e "password1"
* indicando a senha.
*
* O ObjectGrid fornece duas implementações internas para esta interface:
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* UserPasswordCredentialGenerator:
* Um gerador de credenciais que gera um UserPasswordCredential
* que contém um par ID do usuário e senha.
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* WSTokenCredentialGenerator:
* Um gerador de credenciais que gera um WSTokenCredential que contém
* tokens de autenticação e autorização específicos do WebSphere
* Application Server.
*
*
* O relacionamento entre CredentialGenerator e Credential pode ser um
* relacionamento de um para um ou um relacionamento de um para muitos.
* Por exemplo, UserPasswordCredentialGenerator tem um relacionamento
* de um para um com UserPasswordCredential, mas
* WSTokenCredentialGenerator tem um relacionamento de um para muitos
* com o WSTokenCredential, porque ele pode gerar um WSTokenCredential
* diferente, com base no Subject que está associado ao encadeamento
* atual.
*
* Consulte a respectiva documentação da API para obter detalhes
* adicionais.
*
* @ibm-api
```

```
* @since WAS XD 6.0.1
*
* @see Authenticator
* @see ClientSecurityConfiguration#setCredentialGenerator
* (Credentia Generator)
* @see Credential
* @see CredentialGeneratorFactory#getCredentialGenerator()
*/
public interface CredentialGenerator {
/**
* Obtém uma credencial que representa o cliente.
*
* @return a Credencial que representa o cliente
*
* @throws CannotGenerateCredentialException se ocorrer uma falha ao
* gerar a Credencial para o cliente.
*
* @see Credential
*/
Credential getCredential() throws CannotGenerateCredentialException;
/**
* Configurar as propriedades definidas pelo usuário para o depósito
* de informações do provedor
*
* Este método é utilizado para incluir propriedades adicionais de
* CredentialGenerator no objeto. Estas propriedades podem ser
* configuradas utilizando a propriedade credentialGeneratorProps
* no arquivo de propriedades de configuração de segurança do cliente.
* Portanto, é possível customizar seu depósito de informações do
* provedor.
*
* @param properties propriedades definidas pelo usuário
*/
void setProperties(String properties);
}
```

O ObjectGrid fornece duas implementações internas padrão:

1. A classe com.ibm.websphere.objectgrid.security.plugins.builtins. O construtor UserPasswordCredentialGenerator utiliza um ID do usuário e uma senha. Quando o método getCredential() é chamado, ele retorna um objeto UserPasswordCredential que contém o ID do usuário e a senha.
2. A classe com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator representa uma credencial (token de segurança) durante execução 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.

Consulte a documentação da API para obter detalhes adicionais.

Conectar

Se um cliente do ObjectGrid desejar conectar-se a um servidor de maneira segura, será possível utilizar qualquer método connect na interface ObjectGridManager. Utilize o seguinte método connect como exemplo:

```
/**
* Permite que o cliente conecte-se a um ObjectGrid Remoto
* O RemoteObject Grid é hospedado conforme especificado pelos parâmetros
```

```
* @param clusterName: O nome do cluster ao qual este cliente
* se conectará
* @param host: O host no qual será feita a conexão
* @param port: A porta clientAceess que está atendendo.
* @param ClientSecurityConfiguration: Configuração de segurança. Pode ser nula
* se a segurança não estiver configurada
* @param overRideObjectGrid xml. Este parâmetro pode ser nulo. Se não for
* nulo, a configuração do lado cliente do plug-in objectgrid será sobrescrita.
* Nem todos os plug-ins podem ser substituídos. Para obter detalhes, consulte
* os documentos do ObjectGrid
* @throws ConnectException
* @ibm-api
*/
public ClientClusterContext connect(String clusterName, String host, String port,
ClientSecurityConfiguration securityProps, URL overRideObjectGrid) throws
ConnectException ;
```

Este método utiliza um parâmetro de tipo ClientSecurityConfiguration entre outros. Esta interface representa uma configuração de segurança do cliente. É possível utilizar a API pública com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory para criar uma instância deste com valores padrão, ou é possível criar uma instância transmitindo o arquivo de propriedades de segurança do cliente do ObjectGrid. O arquivo `security.ogclient.props` contém as seguintes propriedades relacionadas à autenticação. O valor marcado com + é o valor padrão.

- `securityEnabled (true, false+)`: Esta propriedade indica se a segurança está ativada. Quando um cliente se conecta a um servidor, os valores securityEnabled no lado 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.
- `credentialAuthentication (Never, Supported+, Required)`: Esta propriedade indica se o cliente suporta autenticação de credencial.
  - Se o valor da propriedade for **Never**, nenhuma autenticação de credencial será suportada por este cliente.
  - Se o valor da propriedade for **Supported**, a autenticação de cliente será desempenhada durante a comunicação com qualquer servidor que suporta ou requer autenticação de credencial. A autenticação de credencial do cliente transmite uma credencial ou um token de SSO (Conexão Única)
  - Se o valor da propriedade for **Required**, o cliente deverá enviar uma credencial para o servidor para autenticação.
- `authenticationRetryCount (um valor inteiro, 0+)`. Esta propriedade determina quantas novas tentativas serão feitas para login quando uma credencial estiver expirada. Se o valor for `0`, não serão feitas novas tentativas. A nova tentativa de autenticação se aplicará apenas ao caso em que a credencial tiver expirado. Se a credencial não for válida, não haverá nenhuma nova tentativa. Seu aplicativo é responsável por tentar novamente a operação.
- `clientCertificateAuthentication (Never+, Supported, Required)`: Esta propriedade indica se o cliente suporta a autenticação do certificado cliente.
  - Se o valor da propriedade for **Never**, nenhuma autenticação de certificado cliente será suportada neste lado cliente.
  - Se o valor da propriedade for **Supported**, a autenticação de cliente da camada de transporte poderá ser desempenhada e o cliente enviará um certificado digital para o servidor durante o estágio de autenticação.

  - Se o valor da propriedade for **Required**, o cliente será autenticado apenas com servidores que suportam a autenticação de cliente da camada de transporte.
- `transportType (TCP/IP, SSL-Supported+, SSL-Required)`: Indica qual protocolo de transporte o cliente deseja para conectar-se ao servidor. Qual protocolo um cliente utiliza para conectar-se a um servidor também depende da configuração de transportType no lado do servidor. Consulte “Comunicação Segura ” na página 162 para obter mais detalhes..
  - Se o valor for **TCP/IP**, o cliente terá que utilizar o TCP/IP para conectar-se ao servidor.
  - Se o valor for **SSL-Supported**, o cliente poderá utilizar o TCP/IP ou SSL para conectar-se ao servidor. O cliente primeiro tenta utilizar o SSL para conectar-se ao servidor. Se a conexão SSL falhar, o cliente tentará utilizar o TCP/IP.
  - Se o valor for **SSL-Required**, o cliente deverá utilizar SSL para conectar-se ao servidor.
- `SSOEnabled`: Especifica se o cliente suporta a transmissão de tokens de conexão única para o servidor. Configure esta propriedade como `false` se o cliente autenticar-se em cada servidor. Configure esta propriedade como `true` se o cliente autenticar-se apenas em um servidor. Se você configurar SSOEnabled true no cliente, verifique se a propriedade ativada por conexão única na configuração XML do cluster também está configurada como `true`.

Também é possível configurar estas propriedades utilizando definidores na interface ClientSecurityConfiguration.

Depois de criar um objeto de tipo ClientSecurityConfiguration, configure o credentialGenerator no objeto 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);
```

Também é possível obter o CredentialGenerator no arquivo de propriedades de segurança do cliente do ObjectGrid. A seguir estão as propriedades:

- **credentialGeneratorClass**: o nome da implementação de classe para o 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 ObjectGrid 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.

Utilizar um CredentialGenerator Diferente por Sessão

Em alguns casos, um cliente do ObjectGrid representa apenas uma identidade do cliente; em outros casos, ele pode representar várias identidades. A seguir está um cenário para o último caso: Um cliente do ObjectGrid é criado e compartilhado em um servidor da Web. Todos os servlets neste servidor da Web utilizam este cliente do ObjectGrid. Como cada servlet representa um cliente Web diferente, utilize credenciais diferentes ao enviar pedidos para servidores do ObjectGrid.

O ObjectGrid fornece alteração da credencial no nível de sessão. Ou seja, cada sessão pode utilizar um CredentialGenerator diferente. Portanto, os cenários anteriores podem ser feitos permitindo que o servlet obtenha uma sessão com um CredentialGenerator diferente. A seguir está o método na interface ObjectGridManager.

```
/**
* Obter uma sessão com um CredentialGenerator. Este método pode ser chamado apenas
* pelo cliente do ObjectGrid em um ambiente do servidor do cliente.
*
* Se o ObjectGrid for utilizado em um modelo principal, ou seja, na mesma JVM sem
* um cliente ou servidor existente, getSession(Subject) deverá ser utilizado para
* proteger o ObjectGrid.
*
* @since WAS XD 6.0.1
*/
Session getSession(CredentialGenerator credGen) throws
ObjectGridException, TransactionCallbackException;
```

Eis um exemplo:

```
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
CredentialGenerator credGenManager = new UserPasswordCredentialGenerator
("manager", "xxxxxx");
CredentialGenerator credGenEmployee = new UserPasswordCredentialGenerator
("employee", "xxxxxx");
ObjectGrid og = ogManager.getObjectGrid(ctx, "accounting");
// Obter uma sessão com CredentialGenerator;
Session session = og.getSession(credGenManager );
// Obter o mapa employee
ObjectMap om = session.getMap("employee");
// iniciar uma transação.
session.begin();
Object rec1 = map.get("xxxxxx");
session.commit();
// Obter outra sessão com um CredentialGenerator diferente;
session = og.getSession(credGenEmployee );
// Obter o mapa employee
om = session.getMap("employee");
// iniciar uma transação.
session.begin();
Object rec2 = map.get("xxxxx");
session.commit();
```

Se você utilizar o método ObjectGird.getSession() para obter um objeto de Sessão, a sessão utilizará o CredentialGenerator configurado no objeto ClientConfigurationSecurity. Portanto, é possível tratar o CredentialGenerator transmitido para o método ObjectGird.getSession(CredentialGenerator) que substitui o CredentialGenerator configurado no objeto ClientConfigurationSecurity.

Se for possível reutilizar o objeto de Sessão, isto resulta em ganho de desempenho. No entanto, a chamada do método ObjectGrid.getSession(CredentialGenerator) não é muito cara; a principal sobrecarga é o tempo de coleta de lixo do objeto aumentado. Certifique-se de liberar a referência quando tiver concluído os objetos de Sessão. Em resumo, se seu objeto de Sessão puder compartilhar a identidade, tente reutilizar o objeto de Sessão; se seu objeto de Sessão não puder compartilhar a identidade, utilize o método ObjectGrid.getSession(CredentialGenerator).

Authenticator

Quando o cliente do ObjectGrid recuperar o objeto de Credencial utilizando o objeto CredentialGenerator, o objeto de Credencial será enviado junto com o pedido do cliente para o servidor do ObjectGrid. O servidor do ObjectGrid autentica o objeto de Credencial antes de processar o pedido. Se o objeto de Credencial for autenticado com êxito, um objeto Subject será retornado para representar este objeto de Credencial. Este objeto Subject é então utilizado para autorizar o pedido.

Este objeto Subject também é armazenado em cache. Ele expira depois que sua existência atinge o valor de tempo limite de 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″ faz o objeto Subject expirar em 300 segundos.

O servidor do ObjectGrid utiliza o plug-in Authenticator para autenticar o objeto de Credencial. A seguir está a interface Authenticator:

```
/**
* Este plug-in pode ser utilizado para autenticar um cliente do ObjectGrid para
* um servidor do ObjectGrid com base na credencial fornecida pelo cliente. Um
* objeto Subject é retornado como resultado de uma autenticação.
*
* Este plug-in é utilizado em um servidor do ObjectGrid. Ele pode ser configurado
* no arquivo XML de cluster do ObjectGrid.
*
* A Credencial transmitida no método authenticate(Credential)
* pode conter quaisquer informações de credenciais desejadas pelos usuários.
* Por exemplo, pode ser um objeto de Credencial contendo um par de senhas
* do usuário.
*
* O ObjectGrid fornece várias implementações internas para esta interface:
* * com.ibm.websphere.objectgrid.security.plugins.builtins.
* CertificateMappingAuthenticator:
* Um autenticador que apenas mapeia um certificado SSL para um Subject.
* com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator:
* Um autenticador que autentica um ID do usuário e senha para um arquivo de
* chave.
* com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator:
* Um autenticador que autentica um ID do usuário e uma senha para um servidor
* LDAP.
* com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator:
* Um autenticador que autentica um token de segurança do WebSphere Application
* Server.
*
* Consulte a respectiva documentação da API para obter detalhes adicionais.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see Credential
*/
public interface Authenticator {
/**
```

```
* Autentica um usuário representado pelo objeto de credencial.
*
* @param credential a Credencial do usuário
*
* @return um objeto Subject que representa o usuário
*
* @throws InvalidCredentialException se a credencial for inválida
* @throws ExpiredCredentialException se a credencial tiver expirado
*
* @see Credential
*/
Subject authenticate(Credential credential)
throws InvalidCredentialException, ExpiredCredentialException;
}
```

Este é o local em que a implementação obtém o objeto de Credencial e, em seguida, autentica-o para um registro do usuário, por exemplo, um servidor LDAP (Lightweight Directory Access Protocol) e outros. O ObjectGrid não fornece uma configuração de registro do usuário out-of-box. A conexão com um registro do usuário e a autenticação nele devem ser implementadas neste plug-in.

Por exemplo, a implementação de um 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 utilizar módulos de login JAAS. Um objeto Subject é retornado como resultado de autenticação.

Observe que este método emite 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. No entanto, o tempo de execução do cliente lida como duas exceções de maneira diferente:

- Se a exceção for uma InvalidCredentialException, o tempo de execução do cliente exibirá esta exceção. É esperado que seu aplicativo trate a exceção. É possível corrigir o CredentialGenerator, por exemplo e, em seguida, tentar a operação novamente.
- Se a exceção for uma ExpiredCredentialException e a contagem de novas tentativas não for 0, o tempo de execução do cliente chamará o método CredentialGenerator.getCredential() novamente e enviará o novo objeto de Credencial 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 de autenticação atingir o valor máximo permitido e o cliente ainda obtiver uma ExpiredCredentialException, o resultado será ExpiredCredentialException. Seu aplicativo deve tratar a exceção.

A interface Authenticator oferece uma grande flexibilidade. É possível implementar a interface Authenticator de qualquer maneira. Por exemplo, é possível implementar esta interface para fazer a autenticação de credencial e a autenticação do certificado cliente para suportar as duas autenticações. Ou é possível implementar a interface para suportar dois registros do usuário diferentes.

O ObjectGrid suporta dois tipos de autenticações: autenticação de credencial e autenticação do certificado cliente. Qual o mecanismo utilizar depende da configuração da propriedade de segurança do lado cliente e do servidor. Estas propriedades são apresentadas a seguir:

- credentialAuthentication no arquivo `security.ogclient.props`
- credentialAuthentication no arquivo `security.ogserver.props`
- clientCertificateAuthentication no arquivo `security.ogclient.props`
- clientCertificateAuthentication no arquivo `security.ogserver.props`

Lembre-se de que também é possível configurar estas propriedades utilizando APIs de programação.

As duas tabelas a seguir exibem qual mecanismo de autenticação será utilizado em diferentes configurações.

Tabela 11. Autenticação de Credencial nas Configurações do Cliente e o Servidor

credentialAuthentication do Cliente	**credentialAuthentication do Servidor**	**Resultado**
Não	Nunca	desativado
	Suportado	desativado
	Requerido	Error case
Suportado	Nunca	desativado
	Suportado	ativado
	Requerido	ativado
Requerido	Nunca	Error case
	Suportado	ativado
	Requerido	ativado

Quando não existir nenhuma autenticação de credencial (o resultado é desativado), poderá ocorrer a autenticação do certificado cliente.

A tabela a seguir mostra se a autenticação do certificado cliente é utilizada em diferentes configurações. Observe que a autenticação do certificado cliente é possível apenas se o SSL for utilizado como o protocolo de comunicação e a autenticação de credencial não for utilizada.

Tabela 12. Autenticação do Certificado Cliente em Configurações do Cliente e do Servidor.

Autenticação clientCertificate do Cliente	**Autenticação clientCertificate do Servidor**	**Resultado**
Não	Nunca	desativado
	Suportado	desativado
	Requerido	Error case
Suportado	Nunca	desativado
	Suportado	enabled*
	Requerido	enabled*
Requerido	Nunca	Error case
	Suportado	enabled*
	Requerido	enabled*

* ClientCertificateAuthentication ocorre apenas quando o SSL é utilizado como o protocolo e CredentialAuthentication não é utilizado.

Observe que existe sutileza: Quando a autenticação da credencial e a autenticação do certificado cliente são utilizadas, mas as credencial enviada do cliente é nula, a autenticação do certificado cliente é utilizada.

O autenticador pode ser configurado no arquivo XML do cluster. Este é um exemplo:

```
<cluster name="cluster1" securityEnabled="true" singleSignOnEnabled="true"
loginSessionExpirationTime="300" statisticsEnabled="true"
statisticsSpec="map.all=enabled">
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12500" workingDirectory="" traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12501" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<authenticator
className ="com.ibm.websphere.objectgrid.security.plugins.builtins.
WSTokenAuthenticator">
</authenticator>
</cluster>
```

O ObjectGrid fornece quatro implementações internas de autenticação padrão para o seguinte: autenticação de ID do usuário e senha para um registro do usuário do arquivo de chave, autenticação de ID do usuário e senha para um servidor LDAP, autenticação de mapeamento simples de certificado cliente SSL e mecanismo de Segurança do WebSphere Application Server. Exceto a implementação do Autenticador para o mecanismo de segurança do WebSphere Application Server, as implementações internas servem apenas para fins de teste. A finalidade principal destas duas internas é permitir fazer o teste simples sem gravar nenhum código. A implementação do Autenticador do WebSphere Application Server é uma implementação out-of-box que pode ser conectada quando os servidores e clientes do ObjectGrid estão no mesmo domínio de segurança.

Para servidores do ObjectGrid que desejam utilizar registros do usuário do WebSphere Application Server, é possível utilizar APIs do WebSphere Application Server para obter o registro do usuário configurado no servidor de aplicativos e, em seguida, utilizá-lo em sua implementação do Autenticador. No entanto, esta implementação está fora de escopo deste guia de programação.

Implementação do Autenticador do Registro de Arquivo de Chave

É possível armazenar ID do usuário e senha em um arquivo chamado arquivo de armazenamento de chaves. É possível utilizar a ferramenta keytool para criar um arquivo de armazenamento de chaves e entradas. Por exemplo, o comando a seguir cria uma entrada com um alias user1:

```
keytool -genkey -v -keystore ./keys.jks -storepass password -alias user1
-keypass password -dname CN=user1,O=MyCompany,L=MyCity,ST=MyState
```

Para fins de teste, o ObjectGrid fornece a implementação padrão com.ibm.websphere.objectgrid.security.plugins.builtins. KeyStoreLoginAuthenticator para este plug-in manipular a autenticação de nome do usuário e senha. Esta implementação utiliza o nome de login KeyStoreLogin para efetuar login do usuário em um arquivo de armazenamento de chaves.

A seguir está um trecho de código que mostra a implementação do método authenticate(Credential) na classe KeyStoreLoginAuthenticator.

```
public Subject authenticate(Credential credential)throws
InvalidCredentialException,
ExpiredCredentialException {
UserPasswordCredential cred = (UserPasswordCredential) credential;
LoginContext lc = null;
lc = new LoginContext("KeyStoreLogin",
new UserPasswordCallbackHandlerImpl(cred.getUserName(),
cred.getPassword().toCharArray()));
lc.login();
Subject subject = lc.getSubject();
```

Este snippet primeiro lança a Credencial para um UserPasswordCredential, que é uma implementação da interface Credential, porque ela possui um contrato com o cliente de que o cliente pode transmitir apenas um objeto de tipo UserPasswordCredential. Em seguida, chama o módulo de login KeyStoreLogin para efetuar login.

O ObjectGrid fornece um módulo de login com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule para esta finalidade. É necessário fornecer um arquivo de armazenamento de chaves que contenha o par nome do usuário e senha para cada usuário. O arquivo de armazenamento de chaves é configurado como uma opção para o módulo de login.

A seguir está o trecho de código que mostra como o modelo de login efetua login no arquivo de chave.

```
/**
* Autentica um usuário com base no arquivo de armazenamento de chaves.
*
* @see javax.security.auth.spi.LoginModule#login()
*/
public boolean login() throws LoginException {
if (debug) {
System.out.println("[KeyStoreLoginModule] login: entry");
}
String name = null;
char pwd[] = null;
if (keyStore == null || subject == null || handler == null) {
throw new LoginException("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) {
// trata uma senha NULL como uma senha vazia
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);
}
if (ObjectGridManagerImpl.isTraceEnabled && TC.isDebugEnabled())
Tr.debug(TC, "login", "userName="+name);
// Validar o nome do usuário e senha
```

```
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;
}
/**
* Validar o nome do usuário e senha com base no armazenamento de chaves.
*
* @param userName nome do usuário
* @param password senha
* @throws SecurityException se for encontrada alguma exceção
*/
protected void validate(String userName, char password[])
throws SecurityException {
PrivateKey privateKey = null;
// Obter a chave privada do armazenamento de chaves
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);
}
// Verificar os certificados
Certificate certs[] = null;
try {
certs = keyStore.getCertificateChain(userName);
}
catch (KeyStoreException kse) {
SecurityException se = new SecurityException();
se.initCause(kse);
throw se;
}
if (certs != null && certs.length > 0) {
// Se o primeiro certificado for um X509Certificate
if (certs[0] instanceof X509Certificate) {
try {
// Obter o primeiro certificado que representa o usuário
X509Certificate certX509 = (X509Certificate) certs[0];
// Criar um proprietário
X500Principal principal = new X500Principal(certX509
```

```
.getIssuerDN()
.getName());
principals.add(principal);
if (debug) {
System.out.println(" Principal added: " + principal);
}
}
catch (CertificateException ce) {
SecurityException se = new SecurityException();
se.initCause(ce);
throw se;
}
}
}
}
```

É necessário criar um nome de login ″KeyStoreLogin″ no arquivo de configuração de autenticação JAAS. Se não estiver familiarizado com o arquivo de configuração de autenticação JAAS, consulte o JAAS Authentication Tutorial para obter detalhes adicionais.

```
KeyStoreLogin {
com.ibm.websphere.objectgrid.jaas.KeystoreLoginModule required
keyStoreFile="${user.dir}${/}security${/}.keystore";
};
```

Esta implementação serve apenas para fins de teste.

Implementação do Autenticador LDAP

O ObjectGrid fornece a implementação padrão com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator para este plug-in manipular a autenticação de nome do usuário e senha para um servidor LDAP. Esta implementação utiliza o módulo de login LDAPLogin para efetuar login do usuário em um servidor LDAP.

O snippet 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);
}
}
```

O ObjectGrid fornece um módulo de login com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule

para esta finalidade. É necessário 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 provedor de contexto de LDAP

O LDAPLoginModule chama o método com.ibm.websphere.objectgrid.security.plugins.builtins. LDAPAuthentcationHelper.authenticate. O trecho de código a seguir mostra como o método authenticate do LDAPAuthentcationHelper é implementado:

```
/**
* Autenticar o usuário no diretório LDAP.
* @param user o ID do usuário, por exemplo, uid=xxxxxx,c=us,
* ou=bluepages,o=ibm.com
* @param pwd a senha
*
* @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);
// Procurar o usuário
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("");
// Verificar o 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. Em seguida, o módulo de login obtém as informações do UID e as informações do departamento 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.

A seguir está um exemplo de 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 o servidor `ldap://directory.acme.com:389/`. Altere esta configuração para seu servidor LDAP. Este módulo de login utiliza o ID do usuário e senha fornecidos para conectar-se ao servidor LDAP. Esta implementação serve apenas para fins de teste.

Implementação do Autenticador do WebSphere Application Server

O ObjectGrid também fornece a implementação interna decom.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator para utilizar a infra-estrutura de segurança do WebSphere Application Server. Esta implementação interna pode ser utilizada quando existirem as seguintes condições:

- A segurança global do WebSphere Application Server está ativada.
- Os clientes e servidores do ObjectGrid são ativados nas Java Virtual Machines do WebSphere Application Server.
- Estes servidores de aplicativos estão no mesmo domínio de segurança.
- O cliente do ObjectGrid já está autenticado no WebSphere Application Server.

O cliente do ObjectGrid pode utilizar a classe com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator para gerar uma credencial e o servidor do ObjectGrid utiliza esta classe de implementação do Autenticador para autenticar a credencial. Se o token for autenticado com êxito, será retornado um objeto Subject.

Este cenário tira vantagem do fato de que o cliente do ObjectGrid já foi autenticado. Como os servidores de aplicativos que possuem os servidores do ObjectGrid no mesmo domínio de segurança que os servidores de aplicativos que hospedam clientes do ObjectGrid, os tokens de segurança podem ser propagados do cliente do ObjectGrid para o servidor do ObjectGrid, portanto, o mesmo registro do usuário não precisa ser reautenticado.

Implementação do Autenticador de Mapeamento de Certificado Simples

O ObjectGrid também fornece uma implementação interna com.ibm.websphere.objectgrid.security.plugins.builtins.CertificateMappingAuthenticator para mapear o certificado para um objeto Subject. A implementação extrai o DN (Nome Distinto) do primeiro certificado na cadeia e cria um proprietário com esse nome. Esta implementação serve apenas para fins de teste.

Implementação do Autenticador do Tivoli Access Manager

O Tivoli Access Manager tem sido amplamente utilizado como um servidor de segurança. Também é possível implementar o Autenticador utilizando módulos de login fornecidos pelo Tivoli Access Manager.

Para autenticar um usuário utilizando o Tivoli Access Manager, o LoginModule fornecido pelo Tivoli, com.tivoli.mts.PDLoginModule, requer que o aplicativo de chamada forneça o seguinte:

- Um nome de proprietário, especificado como um nome abreviado ou um nome X.500 (DN)

- Uma senha

O LoginModule autentica o proprietário e retorna a credencial do Tivoli Access Manager. O LoginModule espera que o aplicativo de chamada forneça as seguintes informações:

- O nome do usuário, por meio de um javax.security.auth.callback.NameCallback
- A senha, por meio de um javax.security.auth.callback.PasswordCallback.

Quando a credencial do Tivoli Access Manager for recuperada com êxito, o LoginModule JAAS criará um Subject e um PDPrincipal. Não é fornecida nenhuma autenticação interna para o TAM, porque ela é muito comum com o PDLoginModule. Consulte o IBM Tivoli Access Manager Authorization Java Classes Developer Reference para obter detalhes adicionais.

Conexão Única

Quando um cliente do ObjectGrid for autenticado com êxito em um servidor, o servidor do ObjectGrid criará um objeto Subject. Se o cliente e o servidor suportarem conexão única (SSO), este objeto Subject será convertido em um token SSO. Este token é transmitido de volta para o lado cliente associado ao soquete. Este token SSO pode ser transmitido para um novo servidor para autenticação, portanto, não é necessário reautenticar-se em um servidor diferente.

O token SSO é implementado utilizando o mecanismo do gerenciador de tokens seguros do ObjectGrid. Para obter detalhes adicionais sobre o gerenciador de tokens seguros, consulte “Segurança do Cluster do ObjectGrid” na página 180. Basicamente, o mecanismo de token seguro utiliza chaves criptográficas (chaves secretas) para criptografar e decriptografar dados do usuário que são transmitidos entre os servidores e chaves públicas/privadas para assinar os dados.

O token SSO também contém um tempo de expiração. Todos os servidores do produto que participam em um domínio de proteção precisam ter seus horários, datas e fusos horários sincronizados. Caso contrário, os tokens SSO aparecerão prematuramente expirados e causarão falhas de autenticação ou de validação. (Isto não será necessário se a hora universal for utilizada).

Quando um cliente do ObjectGrid conecta-se a um servidor diferente, este token SSO pode ser transmitido para o novo servidor. Este servidor validará o token SSO para assegurar que ele não tenha sido violado pela remoção de assinatura e decriptografia. Ele também verifica seu time stamp para assegurar que não tenha expirado. Se o token for válido, o cliente não precisará autenticar-se neste servidor.

Se um token SSO estiver expirado, o servidor terá que reautenticar o cliente. O servidor solicita que o cliente forneça a credencial novamente.

Ativar a Conexão Única para o Cliente

A ativação da conexão única do cliente pode ser feita de duas maneiras:

- **Configuração**. Utilize a propriedade SSOEnabled no arquivo `security.ogclient.props` para ativar a conexão única no lado cliente.
- **Programação**. Utilize ClientSecurityConfiguration para ativar a conexão única com o método a seguir.

```
/**
* Configurar se a conexão única está ativada.
* @param enabled se a conexão única está ativada para este
```

```
* cliente ou não.
*/
void setSingleSignOnEnabled(boolean enabled);
```

Ativar a Conexão Única para o Servidor

Para ativar a conexão única no lado do servidor, configure o atributo singleSignOnEnabled como true no arquivo XML do cluster. Este é um exemplo:

```
<cluster>
<objectGrid name="cluster" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300">
```

Observe que a conexão única será ativada apenas se a segurança estiver ativada.

Comunicação Segura

O ObjectGrid suporta TCP/IP e SSL para comunicação segura. O SSL fornece uma comunicação segura entre cliente e servidor. O mecanismo de comunicação a ser utilizado depende das configurações das seguintes propriedades:

- A propriedade transportType no arquivo security.ogclient.props
- A propriedade transportType no arquivo security.ogserver.props

Tabela 13. Protocolo de Transporte a Ser Utilizado nas Configurações de Transporte do Cliente e Transporte do Servidor

transportType do Cliente	transportType do Servidor	Protocolo Resultante
TCP/IP	TCP/IP	TCP/IP
	SSL supported	TCP/IP
	SSL required	Error
SSL supported	TCP/IP	TCP/IP
	SSL supported	SSL (se o SSL falhar, TCP/IP)
	SSL required	SSL
SSL required	TCP/IP	Error
	SSL supported	SSL
	SSL required	SSL

Quando o SSL é utilizado, a configuração SSL deve ser fornecida no lado cliente e do servidor.

Configurar Parâmetros SSL para Clientes do ObjectGrid

Os parâmetros SSL no lado cliente podem ser configurados das seguintes maneiras:

- Criar um objeto com.ibm.websphere.objectgrid.security.config.SSLConfiguration utilizando a classe do depósito de informações do provedor com.ibm.websphere.objectgrid.security.config. ClientSecurityConfigurationFactory. Para obter detalhes adicionais, consulte a documentação da API.
- Configurar os parâmetros no arquivo `security.ogclient.props` e, em seguida, utilizar o método ClientSecurityConfigurationFactory.getClientSecurityConfiguration(String) para ocupar a instância do objeto.

As propriedades a seguir destinam-se a configurações no arquivo `security.ogclient.props`.

- **provider**: Especifica o provedor SSL JSSE. Os valores possíveis são IBMJSSE+, IBMJSSE2, SunJSSE e outros. Configure este valor com base no JDK (Java Development Kit) utilizado.
- **protocol**: Especifica o protocolo SSL. Os valores possíveis são SSL+, SSLV2, SSLV3, TLS, TLSv1 e outros. Configure este valor de protocolo com base no provedor JSSE (Java Secure Socket Extension) utilizado.
- **alias**: A cadeia que representa o alias no armazenamento de chaves. Não existe nenhum valor padrão. Esta propriedade será utilizada se o armazenamento de chaves tiver vários certificados de pares de chaves e você desejar selecionar um dos certificados.
- **keyStoreType**: Especifica o tipo de armazenamento de chaves SSL. Os valores possíveis são JKS+, JCEK, PKCS12 etc. Configure este valor com base no provedor JSSE (Java Secure Socket Extension) utilizado.
- **keyStore**: Especifica o nome do arquivo do caminho de armazenamento de chaves que possui os certificados públicos e chaves privadas do cliente. Por exemplo, `[OBJECTGRID_HOME]/properties/DummyClientKeyFile.jks`. Neste release, o suporte ao hardware não é suportado.
- **keyStorePassword**: Especifica a senha para proteger o caminho do armazenamento de chaves. A senha é codificada utilizando apenas o algoritmo ″xor″ pelo ObjectGrid. Utilize a ferramenta PropFilePasswordEncoder para codificar este arquivo de propriedades. A seguir está um exemplo de senha codificada: `{x0r}CDo9Hgw\\`.
- **trustStoreType**: Especifica o tipo de trust store. Os valores possíveis são JKS+, JCEK, PKCS12 etc. É possível configurar este valor com base no provedor JSSE utilizado.
- **trustStore**: Especifica o nome do arquivo do caminho de trust store que possui os certificados públicos do servidor. Por exemplo, `[OBJECTGRID_HOME]/properties/DummyClientTrustFile.jks`
- **trustStorePassword**: Especifica a senha para proteger o caminho de trust store. A senha é codificada utilizando apenas o algoritmo xor pelo ObjectGrid. Utilize a ferramenta PropFilePasswordEncoder para codificar este arquivo de propriedades. A seguir está um exemplo de senha codificada: `{x0r}CDo9Hgw\`
- **certReqSubjectDN**: Esta é a cadeia requerida no DN do subject do certificado do servidor. Um cliente tem permissão para conectar-se ao servidor apenas se o DN do certificado do servidor contiver esta cadeia. Se o valor for nulo, o cliente não precisará de um DN de subject específico no certificado do servidor. Por exemplo, se o DN do subject do certificado for ″CN=Server1, OU=Your Organizational Unit, O=Your Organization, S=Your State,C=Your Country″, ″CN=server1″, ″O=Your Organization″, ″OU=Your Organizational Unit, O=Your Organization, S=Your State,C=Your Country″ resultará em uma correspondência, mas ″CN=server2″ e ″OU=Your Organizational Unit, L=smething, O=Your Organization, S=Your State,C=Your Country″ não corresponderão. A correspondência de curingas não é suportada.

Configurar Parâmetros SSL para Servidores de Objetos

Os parâmetros SSL no lado cliente podem ser configurados no arquivo `security.ogserver.props`. Este arquivo de propriedades pode ser transmitido como um parâmetro quando você ativar um servidor do ObjectGrid.

Exceto as propriedades SSL anteriores, a configuração SSL do lado do servidor possui uma propriedade adicional:

- **clientAuthentication** (true+, false). Se esta propriedade estiver configurada como true, o cliente SSL deverá ser autenticado. Isto é diferente da autenticação do certificado cliente. A autenticação do certificado cliente significa autenticar um cliente em um registro do usuário com base na cadeia de certificados, enquanto esta propriedade assegura que o servidor se conectará ao cliente correto.

Segurança do ObjectGrid Local

Este tópico descreve a segurança do modelo de programação do ObjectGrid local. No modelo de programação do ObjectGrid local, a principal função de segurança é a autorização. O modelo de programação do ObjectGrid local não suporta nenhuma autenticação. É necessário fazer a autenticação fora do ObjectGrid. No entanto, o ObjectGrid fornece plug-ins para obter e validar objetos Subject.

A ativação de segurança do ObjectGrid pode ser feita de duas maneiras:

- **Configuração**. É possível utilizar o arquivo XML do ObjectGrid para definir um ObjectGrid e ativar a segurança para esse ObjectGrid. A seguir está o arquivo `secure-objectgrid-definition.xml` que é utilizado na amostra de aplicativo corporativo ObjectGridSample. Neste arquivo XML a segurança é ativada configurando o atributo securityEnabled como `true`.

  ```
  <objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_JASS">
  <bean id="TransactionCallback"
  classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- **Programação**. Se desejar criar um ObjectGrid utilizando APIs, chame o seguinte método na interface do ObjectGrid para ativar a segurança:

  ```
  /**
  * Ativar a Segurança do ObjectGrid
  */
  void setSecurityEnabled();
  ```

  No modelo de programação do ObjectGrid local, não existe nenhuma autenticação. Ao configurar a segurança com este método, você está configurando a autorização. Esta condição é consistente com o modelo do servidor do cliente. A ativação de segurança para um ObjectGrid no modelo de servidor do cliente ativa apenas a autorização nessa instância do ObjectGrid.

Autenticação

No modelo de programação do ObjectGrid local, o ObjectGrid não fornece nenhum mecanismo de autenticação. O ObjectGrid depende do ambiente, de servidores de aplicativos ou de aplicativos para autenticações. Quando um ObjectGrid é utilizado no WebSphere Application Server ou no WebSphere Extended Deployment, os aplicativos podem utilizar o mecanismo de autenticação de segurança do WebSphere Application Server. Quando um ObjectGrid está em execução em um ambiente J2SE (Java 2 Platform, Standard Edition), o aplicativo deve gerenciar autenticações com a 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. A seguir está a API ObjectGrid.getSession(Subject):

```
/**
* Esta API permite que o cache utilize um subject específico em vez de um
* configurado no ObjectGrid para obter uma sessão.
* @param subject
* @return Uma instância de Sessão
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws InvalidSubjectException o subject transmitido é inválido com base
* no mecanismo SubjectValidation.
*/
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException,
InvalidSubjectException;
```

O método getSession na interface do ObjectGrid também pode ser utilizado para obter um objeto de Sessão:

```
/**
* Retorna um objeto de Sessão que pode ser utilizado por um único encadeamento
* em determinado momento.
* Não é permitido compartilhar este objeto de Sessão entre encadeamentos sem
* colocar uma seção crítica em torno dele. Embora a estrutura principal
* permita que o objeto se mova entre encadeamentos, TransactionCallback e
* Loader podem impedir este uso, principalmente em ambientes J2EE.
* Quando a segurança for ativada, será utilizado SubjectSource para obter
* um objeto Subject.
*
* Se o método initialize() não tiver sido chamado antes da primeira chamada
* de getSession, ocorrerá uma inicialização implícita. Isto assegura que
* toda a configuração seja concluída antes de ser requerido qualquer uso de
* tempo de execução.
*
* @see #initialize()
* @return Uma instância de Sessão
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws IllegalStateException se este método for chamado após
* a chamada do método destroy().
*/
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 definidos no ObjectGrid para suportar a propagação de objetos Subject. Consulte “Plug-ins Relacionados à Segurança” na página 166 para obter maiores informações.

O método getSession(Subject) pode ser chamado apenas na instância do ObjectGrid local. Se você chamar o método getSession(Subject) em um lado cliente em uma configuração do ObjectGrid distribuído, isso resultará em uma exceção.

Plug-ins Relacionados à Segurança

O ObjectGrid fornece dois plug-ins de segurança que estão relacionados ao mecanismo de transmissão de subject: 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 utilizado para obter um objeto Subject de um ambiente em execução do ObjectGrid. Este ambiente do ObjectGrid pode ser um aplicativo que utiliza o ObjectGrid ou um servidor de aplicativos que hospeda o aplicativo. A interface é a seguinte:

```
/**
* 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. O método
* 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 um objeto Subject autenticado e, em seguida, transmitir
* a instância do ObjectGrid. Portanto, não é necessária uma outra
* autenticação.
*
* Por exemplo, utilize
* Subject.getSubject(AccessControlContext)
* para obter o subject associado ao AccessControlContext e,
* em seguida, retorne-o na implementação de getSubject.
*
* Este plug-in pode ser utilizado apenas em um domínio seguro, como em
* um servidor do ObjectGrid.
*
* @ibm-api
* @since WAS XD 6.0
*/
public interface SubjectSource {
/**
* Obter um objeto Subject que pode representar o cliente do ObjectGrid.
*
* @return um objeto Subject
* @throws ObjectGridSecurityException qualquer exceção durante a
* recuperação de subject
*/
Subject getSubject() throws ObjectGridSecurityException;
}
```

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 ObjectGrid que 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.

Este plug-in SubjectSource pode ser utilizado para obter um objeto Subject que representa um cliente do ObjectGrid que é utilizado para autorização do ObjectGrid. Quando o método ObjectGrid.getSession() for chamado, o

método Subject getSubject() throws ObjectGridSecurityException() será chamado pelo tempo de execução do ObjectGrid, se a segurança estiver ativada.

O ObjectGrid fornece uma implementação padrão deste plug-in: com.ibm.websphere.objectgrid.security.plugins.builtins. WSSubjectSourceImpl. Esta implementação pode ser utilizada para recuperar um subject do responsável pela chamada ou um subject RunAs do encadeamento quando um aplicativo está em execução no WebSphere Application Server. É possível configurar esta classe como a classe de implementação de SubjectSource ao utilizar o ObjectGrid no WebSphere Application Server. A seguir está um trecho de código que mostra o fluxo principal do WSSubjectSourceImpl.getSubject():

```
Subject s = null;
     try {
if (finalType == RUN_AS_SUBJECT) {
// obter o subject RunAs
s = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
}
else if (finalType == CALLER_SUBJECT) {
// obter o callersubject
s = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
}
}
catch (WSSecurityException wse) {
throw new ObjectGridSecurityException(wse);
}
return s;
```

Para obter outros detalhes, consulte a Documentação da API para o plug-in SubjectSource e a implementação do WSSubjectSourceImpl.

Plug-in SubjectValidation

O plug-in SubjectValidation, 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 se um javax.security.auth.Subject, transmitido para o ObjectGrid ou recuperado pelo plug-in SubjectSource, é um Subject válido que não foi violado. A seguir está a interface.

```
/**
* Este plug-in pode ser utilizado para validar que um
* 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, o criador de um subject pode não saber
* se o Subject foi violado. Neste
* caso, este plug-in não deve ser utilizado.
*
* Este plug-in pode ser utilizado apenas em um domínio seguro, como em um
* servidor de aplicativos. Não coloque este plug-in no lado cliente; ele
* será ignorado.
*
* @ibm-api
*
* @since WAS XD 6.0
*/
public interface SubjectValidation {
/**
* Validar se o Subject não foi violado.
```

```
* @param subject um subject a ser validado
* @return o objeto Subject validado
* @throws InvalidSubjectException
*/
Subject validateSubject(Subject subject) throws
InvalidSubjectException;
}
```

O método Subject validateSubject (Subject subject) throws InvalidSubjectException; na interface SubjectValidation obtém um objeto Subject e retorna um objeto Subject. Se um objeto Subject é considerado válido e qual objeto Subject será retornado dependem de suas implementações. Se o objeto Subject não for válido, isso resultará em uma `InvalidSubjectException`.

É possível utilizar este plug-in se você não confiar no objeto Subject transmitido para este método. Este caso é raro, considerando que confiamos nos desenvolvedores de aplicativos que desenvolvem o código para recuperarem 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 ObjectGrid fornece uma implementação padrão de SubjectValidation: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl. Esta implementação pode ser utilizada para validar o subject autenticado do WebSphere. Os usuários podem configurar esta classe como a classe de implementação SubjectValidation ao utilizar o ObjectGrid no WebSphere Application Server. A implementação do WSSubjectValidationImpl considera um objeto Subject válido apenas se o token de credencial associado a este Subject não tiver sido violado. Em outras palavras, você pode alterar outras partes do objeto Subject. A implementação de WSSubjectValidationImpl solicita do WebSphere Application Server o Subject original correspondente ao token de credencial e retorna o objeto 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):

```
// Criar um LoginContext com o esquema WSLogin e
// transmitir uma rotina de tratamento de Retorno de Chamada.
LoginContext lc = new LoginContext("WSLogin",
new WSCredTokenCallbackHandlerImpl(subject));
// Quando este método for chamado, os métodos da rotina de tratamento
de retorno de chamada
// serão chamados para efetuar login do usuário.
         lc.login();
// Obter o subject de LoginContext
return lc.getSubject();
```

No trecho de código anterior um objeto de rotina de tratamento de retorno de chamada do token de credencial, WSCredTOkenCallbackHandlerImpl, foi criado com o objeto Subject a ser validado. Em seguida, um LoginContext foi criado com o esquema de login ″WSLogin″. Quando o método lc.login() for chamado, a segurança do WebSphere Application Server recuperará o token de credencial do objeto Subject e, em seguida, retornará o Subject correspondente como o objeto Subject validado.

Para obter outros detalhes, consulte a documentação da API de SubjectValidation e de WSSubjectValidationImpl.

Configuração do Plug-in

O plug-in SubjectValidation e o plug-in SubjectSource podem ser configurados de duas maneiras:

- **Configuração**. 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 desejar criar um ObjectGrid por meio de APIs, será possível chamar os seguintes métodos para configurar os plug-ins SubjectSource ou SubjectValidation.

```
/**
* Configure o plug-in SubjectValidation para esta instância do
* ObjectGrid. Um plug-in SubjectValidation pode ser utilizado para
* validar se o objeto Subject transmitido é um Subject válido.
* Consulte {@link SubjectValidation}
* para obter detalhes adicionais.
* @param subjectValidation o plug-in SubjectValidation
*/
void setSubjectValidation(SubjectValidation subjectValidation);
/**
* Configure o plug-in SubjectSource. Um plug-in SubjectSource pode ser
* utilizado para obter um objeto Subject do ambiente para representar
* o cliente do ObjectGrid.
*
* @param source o plug-in SubjectSource
*/
void setSubjectSource(SubjectSource source);
```

Gravar Seu Código de Autenticação JAAS

É possível gravar seu código de autenticação JAAS 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, podem ser um ID do usuário e senha, certificado cliente e outros. Depois de receber as informações, o módulo de login verifica se elas representam um subject válido e, em seguida, cria um objeto Subject. No momento, várias implementações de módulos de login estão disponíveis para o público.

Após a gravação de um módulo de login, configure este módulo de login para que ele possa ser utilizado pelo tempo de execução. Um arquivo de configuração do módulo de login JAAS deve ser configurado. 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 este login ou o esquema falhará totalmente.

A configuração do arquivo de configuração do módulo de login JAAS pode ser feita de uma das seguintes maneiras:

- Configure o arquivo de configuração do módulo de login JAAS no **login.config.url** no arquivo `java.security`, por exemplo, `login.config.url.1=file:${java.home}/lib/security/file.login`
- Configure o arquivo de configuração do módulo de login JAAS a partir da linha de comandos utilizando argumentos da JVM **-Djava.security.auth.login.config**, por exemplo, `-Djava.security.auth.login.config ==$JAVA_HOME/lib/security/file.login`

Para obter informações adicionais sobre como gravar e configurar módulos de login, consulte o Tutorial de Autenticação JAAS.

Se seu código estiver em execução no WebSphere Application Server, será necessário configurar o login JAAS no console administrativo e armazenar esta configuração de login na configuração do servidor de aplicativos. Consulte Configuração de Login para Java Authentication and Authorization Service para obter detalhes.

Autorização

Após autenticação do cliente, é possível utilizar os mecanismos de autorização do ObjectGrid para autorizar o acesso aos dados do mapa e às tarefas de gerenciamento do ObjectGrid. A autorização do ObjectGrid é baseada no objeto Subject. O ObjectGrid suporta dois tipos de mecanismos de autorização: autorização JAAS (Java Authentication and Authorization Service) e autorização customizada.

Classe de Permissão

Existem dois tipos diferentes de autorização do ObjectGrid: autorização para os dados no mapa e autorização para as tarefas de gerenciamento. Cada autorização utiliza uma classe de Permissão. A permissão para acessar o mapa é representada pela classe MapPermission e a permissão para executar as tarefas de gerenciamento é representada pela classe AdminPermission.

Classe MapPermission

No ObjectGrid, a classe pública com.ibm.websphere.objectgrid.security.MapPermission representa permissões para os recursos do ObjectGrid, especificamente os métodos de interfaces ObjectMap ou JavaMap. O ObjectGrid define as seguintes cadeias de permissão para acessar os métodos de ObjectMap e de JavaMap:

- **read**: Concede permissão para ler os dados a partir do mapa. A constante do inteiro é definida como MapPermission.READ
- **write**: Concede permissão para atualizar os dados no mapa. A constante do inteiro é definida como MapPermission.WRITE.
- **insert**: Concede permissão para inserir os dados no mapa. A constante do inteiro é definida como MapPermission.INSERT.
- **remove**: Concede permissão para remover os dados do mapa. A constante do inteiro é definida como MapPermission.REMOVE.
- **invalidate**: Concede permissão para invalidar os dados do mapa. A constante do inteiro é definida como MapPermission.INVALIDATE.
- **all**: Concede todas as permissões: read, write, insert, remote e invalidate. A constante do inteiro é definida como MapPermission.ALL.

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. A cadeia de permissão pode ser uma cadeia delimitada por vírgulas das cadeias de permissões acima, como ″read, insert″, ou pode ser ″all″ que significa que todas as permissões são concedidas. Um valor inteiro de permissão pode ser quaisquer constantes de inteiro de permissão acima ou um valor matemático ″or″ de várias constantes de permissão de inteiro, como DGMapPermission.GET|DGMapPermission.PUT.

A autorização ocorre quando um cliente chama um método de ObjectMap ou JavaMap. O tempo de execução do ObjectGrid verifica diferentes permissões para diferentes métodos. Se as permissões requeridas não forem concedidas ao cliente, isso resultará em um AccessControlException.

Tabela 14. Lista de Métodos e Suas Permissões Requeridas

	com.ibm.websphere.objectgrid.ObjectMap com.ibm.websphere.objectgrid.JavaMap
read	boolean containsKey(Object)
	boolean equals(Object)
	Object get(Object)
	Object get(Object, Serializable)
	List getAll(List)
	List getAll(List keyList, Serializable)
	List getAllForUpdate(List, Serializable)
	Object getForUpdate(Object)
	Object getForUpdate(Object, Serializable)
write	Object put(Object key, Object value)
	void put(Object, Object, Serializable)
	void putAll(Map)
	void putAll(Map, Serializable)
	void update(Object, Object)
	void update(Object, Object, Serializable)

Tabela 14. Lista de Métodos e Suas Permissões Requeridas (continuação)

	com.ibm.websphere.objectgrid.ObjectMap **com.ibm.websphere.objectgrid.JavaMap**
inserir	public void insert (Object, Object)
	void insert(Object, Object, Serializable)
	remove Object remove (Object)
	void removeAll(Collection)
invalidar	public void invalidate (Object, boolean)
	void invalidateAll(Collection, boolean)
	void invalidateUsingKeyword(Serializable)
	int setTimeToLive(int)

A autorização é baseada apenas em qual método é utilizado, em vez de basear-se no que o método realmente faz. Por exemplo, um método put pode inserir ou atualizar um registro com base se o registro existe. No entanto, o caso de inserção ou atualização não está distinto neste momento.

Observe também que 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. Lembre-se disso ao projetar suas políticas de autorização.

AdminPermission

A permissão de administração é representada pela classe com.ibm.websphere.objectgrid.security.AdminPermission. O ObjectGrid define duas ações de permissão para permissões de administração:

- **admin**: Concede permissões para quaisquer tarefas de administração.
- **monitor**: Concede permissões para ações que sejam apenas tarefas de administração de acesso de leitura.

As operações detalhadas concedidas a usuários com diferentes permissões estão listadas na tabela a seguir. Estas operações correspondem aos métodos na interface ManagementMBean:

Tabela 15. Relacionamento entre Tarefas de Gerenciamento e Permissões Admin

operations	**admin**	**monitor**
startServer	S	N
stopServer	S	N
forceStopServer	S	N
setServerTrace	S	N
retrieveServerStatus	S	S
getMapStats	S	S
getOGStats	S	S
getReplicationStats	S	S

Se o cliente tiver permissão admin, ele poderá executar a tarefa startServer; se o cliente tiver permissão de monitor, ele não poderá executar a tarefa startServer.

Mecanismos de Autorização

O ObjectGrid suporta dois tipos de mecanismos de autorização: autorização JAAS e autorização customizada. Isto se aplica à autorização de acesso a dados do mapa e à autorização admin. A autorização JAAS aumenta as políticas de segurança Java com controles de acesso centrais do usuário. As permissões podem ser concedidas com base não apenas em qual código está em execução, mas também em quem (proprietário) está executando-o. Ele faz parte do JDK 1.4.

O ObjectGrid também suporta a autorização customizada com o plug-in com.ibm.websphere.objectgrid.security.plugins.MapAuthorization e o plug-in com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization. É possível implementar seu próprio mecanismo de autorização, se não desejar utilizar a autorização JAAS. Utilizando o mecanismo de autorização customizado, é possível utilizar o banco de dados de política, o servidor de política ou o Tivoli Access Manager para gerenciar autorizações do ObjectGrid.

O mecanismo de autorização do ObjectGrid pode ser configurado de duas maneiras:

- **Configuração**. 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 na amostra do aplicativo corporativo ObjectGridSample.

  ```
  <objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
  <bean id="TransactionCallback"
  classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- **Programação**. Se desejar criar um ObjectGrid utilizando APIs, será possível chamar o seguinte método para configurar o mecanismo de autorização. Isto se aplica apenas ao modelo de programação do ObjectGrid local quando você instancia diretamente a instância do ObjectGrid.

  ```
  /**
  * Configure o Mecanismo de autorização. O padrão é
  * com.ibm.websphere.objectgrid.security.SecurityConstants.
  * AUTHORIZATION_MECHANISM_JAAS.
  * @param authMechanism o mecanismo de autorização
  */
  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 proprietário de nome (″Joe Smith″) e um proprietário de grupo (″manager″).

Utilizando a política de autorização JAAS, as permissões podem ser concedidas a Proprietários específicos. O ObjectGrid associa o Subject ao contexto de controle de acesso atual. Para cada chamada de método para o ObjectMap ou Javamap, o tempo de execução Java determina automaticamente se a política concederá a permissão requerida apenas

para um Proprietário específico e, neste caso, a operação será permitida apenas se o Subject associado ao contexto de controle de acesso contiver o Proprietário designado.

É necessário estar familiarizado com a sintaxe de política do arquivo de políticas. Para obter uma descrição detalhada da autorização JAAS, consulte o Tutorial de Autorização JAAS.

O ObjectGrid possui uma base de código especial utilizada para verificar a autorização JAAS nas chamadas de métodos ObjectMap e JavaMap. A 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 ao arquivo JAR (Java Archive) do ObjectGrid foram concedidas todas as permissões.

O gabarito da política para conceder 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 Proprietário é semelhante ao seguinte:

```
Principal Principal_class "principal_name"
```

Ou seja, é a palavra ″Principal″ seguida pelo nome completo de uma classe de Proprietário e de um nome de proprietário. O map_name é o nome completo do mapa no formato de [ObjectGrid Name].[Map Name], por exemplo, ″secureClusterObjectGrid.employees″. A ação é uma cadeia delimitada por vírgulas das cadeias de permissões definidas na classe MapPermission, como ″read, insert″ ou ″all″.

A função de curinga limitada é suportada. É possível substituir o nome do ObjectGrid ou o nome do mapa por ″*″ para indicar ″any″ (qualquer). No entanto, o ObjectGrid não suporta a substituição de parte do nome do ObjectGrid ou de nome do mapa por ″*″. Therefore, “*.employees”, “clusterObjectGrid.*”, and “*.*” are all valid names, but “cluster*.employees” is not valid

Por exemplo, no aplicativo de amostra `ObjectGridSample.ear`, são definidos dois arquivos de políticas de autorização: `fullAccessAuth.policy` e `readInsertAccessAuth.policy`. O conteúdo de `readInsertAccessAuth.policy` é o seguinte:

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/
PrivilegedAction"
Principal com.ibm.ws.security.common.auth.WSPrincipalImpl
"principal_name" {
permission com.ibm.websphere.objectgrid.security.MapPermission
"secureClusterObjectGrid.employees", "read,insert";
permission com.ibm.websphere.objectgrid.security.MapPermission
"secureClusterObjectGrid.offices", "read,insert";
permission com.ibm.websphere.objectgrid.security.MapPermission
```

```
"secureClusterObjectGrid.sites", "read,insert";
permission com.ibm.websphere.objectgrid.security.MapPermission
"secureClusterObjectGrid.counters", "read,insert";
};
```

Nesta política, apenas as permissões ″insert″ e ″read″ são concedidas para estes quatro a um determinado proprietário. O outro arquivo de políticas, `fullAccessAuth.policy`, concede as permissões ″all″ para estes mapas a um proprietário. Antes de executar o aplicativo, altere o principal_name e a classe do proprietário para os valores apropriados. O valor de principal_name depende do registro do usuário. Por exemplo, se o S.O. local for utilizado como registro do usuário, o nome da máquina será MACH1, e o ID do usuário será user1, o principal_name será ″MACH1/user1″.

A política de autorização JAAS pode ser diretamente colocada no arquivo de políticas Java ou pode ser colocada em um arquivo de autorização JAAS separado e, em seguida, configurada utilizando o argumento da JVM `-Djava.security.auth.policy=file:[JAAS_AUTH_POLICY_FILE]` ou utilizando `auth.poliyc.url.x=file:[JAAS_AUTH_POLICY_FILE]` no arquivo `java.security`.

A descrição da autorização JAAS também se aplica quando você desejar gravar e configurar as políticas para autorizar o acesso a tarefas de gerenciamento. A única diferença será que, em vez de utilizar o formato ″com.ibm.websphere.objectgrid.security.MapPermission map name, actions;″, você utilizará ″com.ibm.websphere.objectgrid.security.AdminPermission action;″. A ação pode ser ″admin″ ou ″monitor″.

Autorização de Mapa Customizada

O ObjectGrid também suporta a autorização de mapa customizada pelo plug-in MapAuthorization. A interface é a seguinte:

```
/**Este plugin pode ser utilizado para autorizar acessos de
* ObjectMap/JavaMap
* aos proprietários representados pelo objeto Subject.
*
* Uma implementação típica deste plug-in é recuperar o
* proprietários do objeto Subject e, em seguida, verificar se
* as permissões especificadas serão concedidas aos proprietários.
*
*
* @ibm-api
* @since WAS XD 6.0
*/
public interface MapAuthorization {
/**
* Verifique se os proprietários representados pelo objeto Subject
* no subject possuem a MapPermission especificada. Se
* as permissões forem concedidas, será retornado true; caso contrário,
* será retornado false.
*
* @param subject o subject
* @param permission a permissão para acessar ObjectMap
*
* @return true se a permissão for concedida; caso contrário, false.
*/
boolean checkPermission(Subject subject, MapPermission permission);
}
```

Este plug-in pode ser utilizado para autorizar acessos de ObjectMap e de JavaMap aos proprietários contidos no objeto Subject. O método a seguir:

```
boolean checkPermission(Subject subject, MapPermission permission)
```

A interface MapAuthorization é chamada pelo tempo de execução do ObjectGrid para verificar se o objeto subject transmitido tem a permissão passed-in. Neste caso, a implementação da interface MapAuthorization retornará true e, caso contrário, false.

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, em um arquivo simples ou em um servidor de política do Tivoli Access Manager.

O ObjectGrid fornece duas implementações padrão para este plug-in. A classe com.ibm.websphere.objectgrid.security.plugins.builtins. JAASMapAuthorizationImpl é uma implementação de MapAuthorization que utiliza o mecanismo JAAS para autorização. Outra classe de implementação é a com.ibm.websphere.objectgrid.security.plugins.builtins. TAMMapAuthorizationImpl. Ela mostra como o Tivoli Access Manager pode ser utilizado para gerenciar as autorizações do ObjectGrid. A seguir está um trecho de código que mostra o fluxo básico de JAASMapAuthorizationImpl.checkPermission(Subject, MapPermission):

```
// Cria um PrivilegedExceptionAction para verificar as permissões.
PrivilegedExceptionAction action =
MapPermissionCheckAction.getInstance(permission);
Subject.doAsPrivileged(subject, action, null);
```

Consulte o IBM Tivoli Access Manager Authorization Java Classes Developer Reference para obter detalhes adicionais.

Não utilize este plug-in TAMMapAuthorizationImpl em um cenário out-of-box. Utilize este plug-in apenas para fins de teste. Ele requer algumas condições prévias restritivas:

- O objeto Subject contém um proprietário com.tivoli.mts.PDPrincipal.
- O servidor de política do TAM definiu as seguintes permissões para o objeto de nome ObjectMap ou JavaMap. O objeto definido no servidor de políticas deve ter o mesmo nome que o ObjectMap ou JavaMap no formato de [ObjectGrid_name].[ObjectMap_name]. A permissão é o primeiro caractere das cadeias de permissões definidas no MapPermission. Por exemplo, a permissão ″r″ definida no servidor de política representa a permissão ″read″ para o ObjectMap.

O snippet 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,
MapPermission permission) {
String[] str = permission.getParsedNames();
StringBuffer pdPermissionStr = new StringBuffer(5);
```

```
for (int i=0; i<str.length; i++) {
pdPermissionStr.append(str[i].substring(0,1));
}
PDPermission pdPerm = new PDPermission(permission.getName(),
pdPermissionStr.toString());
Set principals = subject.getPrincipals();
Iterator iter= principals.iterator();
while(iter.hasNext()) {
try {
PDPrincipal principal = (PDPrincipal) iter.next();
if (principal.implies(pdPerm)) {
return true;
}
}
catch (ClassCastException cce) {
// Handle exception
}
}
return false;
}
```

O plug-in MapAuthorization pode ser configurado das seguintes maneiras:

- **Configuração**. É possível utilizar o arquivo XML do ObjectGrid para definir um plug-in MapAuthorization. Eis um exemplo:

  ```
  <objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
  ...
  <bean id="MapAuthorization"
  className="com.ibm.websphere.objectgrid.security.plugins.builtins.
  JAASMapAuthorizationImpl" />
  </objectGrids>
  ```

- **Programação**. Se desejar criar um ObjectGrid utilizando APIs, será possível chamar o seguinte método para configurar o plug-in de autorização. Isto se aplica apenas ao modelo de programação do ObjectGrid local quando você instanciar diretamente a instância do ObjectGrid.

  ```
  /**
  * Configura o plug-in MapAuthorization para esta instância do
  * ObjectGrid.
  *
  * Um plug-in {@link MapAuthorization} pode ser utilizado para autorizar
  * o acesso aos mapas. Consulte {@link MapAuthorization}
  * para obter detalhes adicionais.
  * @param mapAuthorization o plug-in MapAuthorization
  */
  void setMapAuthorization(MapAuthorization mapAuthorization);
  ```

Autorização Admin Customizada

Assim como o suporte à autorização de acesso a dados do mapa customizado, o ObjectGrid suporta a autorização admin customizada. O plug-in é com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization.

```
/**
* Este plug-in pode ser utilizado para autorizar operações de gerenciamento
* para os
* proprietários contidos no objeto Subject. As permissões para as operações
* de gerenciamento são representadas por objetos
* AdminPermission.
*
* Este plug-in é utilizado em um servidor do ObjectGrid. Ele pode ser
* configurado no arquivo XML de cluster do ObjectGrid.
*
* Uma implementação típica deste plug-in é recuperar o
```

```
* conjunto de Proprietários do objeto Subject e, em seguida,
* verificar se as permissões especificadas são concedidas a estes
* proprietários.
*
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see AdminPermission
*/
public interface AdminAuthorization {
/**
* Verifica se o usuário representado pelo objeto Subject possui ou não
* um AdminPermission especificado.
*
* Se as permissões forem concedidas, será retornado true;
* caso contrário,
* false é retornado.
*
* @param subject o objeto Subject que representa o usuário
* @param permission a permissão de administração para verificação
*
* @return true se a permissão for concedida; caso contrário, false.
*
* @see AdminPermission
*/
boolean checkPermission(Subject subject, AdminPermission permission);
}
```

Este plug-in pode ser utilizado para autorizar acessos admin aos proprietários contidos no objeto Subject. O método

```
boolean checkPermission(Subject subject, AdminPermission permission)
```

na interface AdminAuthorization é chamado pelo tempo de execução do ObjectGrid para verificar se o objeto Subject transmitido tem a permissão passed-in admin. Neste caso, a implementação da interface AdminAuthorization deve retornar true; caso contrário, false.

É possível implementar esta interface com base nos requisitos de segurança. O ObjectGrid não fornece uma classe de implementação para esta interface.

É possível configurar o plug-in AdminAuthorization no nível do cluster no XML do cluster. Este é um exemplo:

```
<cluster name="cluster1" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300"
statisticsEnabled="true"
statisticsSpec="map.all=enabled">
<serverDefinition name="server1" host="localhost"
clientAccessPort="12503" peerAccessPort="12500" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="localhost"
clientAccessPort="12504" peerAccessPort="12501" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<authenticator className ="com.ibm.websphere.objectgrid.security.plugins.
builtins.WSTokenAuthenticator"></authenticator>
<adminAuthorization className= "com.ibm.ws.objectgrid.test.security.util.
TestAdminAuthorization"></adminAuthorization>
</cluster>
```

Período de Verificação de Permissão

O ObjectGrid suporta o armazenamento em cache dos resultados da verificação de permissão do mapa por motivo de desempenho. Sem este mecanismo, quando um método listado na Tabela 14 na página 171 é chamado, o tempo de execução do ObjectGrid chama o mecanismo de autorização configurado para autorizar o acesso. Com este período de verificação de permissão sendo configurado, o mecanismo de autorização é chamado periodicamente com base no período de verificação de permissão.

Armazenamos em cache as informações de autorização de permissão com base no objeto Subject. Quando um cliente tentar acessar os métodos, o tempo de execução do ObjectGrid consultará o cache com base no objeto Subject. Se não puder ser localizado no cache, o tempo de execução verificará as permissões concedidas para este objeto Subject e, em seguida, armazenará as permissões em um cache.

O período de verificação de permissão deve ser definido antes da inicialização do ObjectGrid. O período de verificação de permissão pode ser configurado de duas maneiras:

- **Configuração**. É possível utilizar o arquivo XML do ObjectGrid para definir um ObjectGrid e configurar o período de verificação de permissão. A seguir está um exemplo para configurar o período de verificação de permissão como 45 segundos.

```
<objectGrids>
<objectGrid name="secureClusterObjectGrid" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS"
permissionCheckPeriod="45">
<bean id="bean id="TransactionCallback"
className="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
...
</objectGrids>
```

- **Programação**. Se desejar criar um ObjectGrid com APIs, chame o seguinte método para configurar o período de verificação de permissão. Este método pode ser chamado apenas antes da inicialização da instância do ObjectGrid. Este método se aplica apenas ao modelo de programação do ObjectGrid local quando você instancia a instância do ObjectGrid diretamente.

```
/**
* Este método utiliza um único parâmetro indicando a freqüência com que
* o cliente deseja verificar a permissão utilizada para permitir o
* acesso de um cliente. Se o parâmetro for 0, cada chamada
* get/put/update/remove/evict única solicitará
* o mecanismo de autorização, a autorização JAAS ou a autorização customizada
* para verificar se o subject atual tem permissão. Isto pode ser
* proibitivamente caro de um ponto de vista de desempenho, dependendo da
* implementação de autorização mas, se for requerido, poderá ser feito.
* Como alternativa, se o parâmetro for > 0, isto indica o número
* de segundos para armazenar em cache um conjunto de permissões antes
* de retornar ao mecanismo de autorização para atualizá-las. Fornece
* um desempenho muito melhor, mas você corre o risco, se as permissões
* de backend forem alteradas durante este período, o ObjectGrid
* provavelmente permitirá ou impedirá o acesso mesmo que o provedor de
* segurança de backend tenha sido modificado.
*
* @param period o período de verificação de permissão em segundos.
*/
void setPermissionCheckPeriod(int period);
```

Segurança do Cluster do ObjectGrid

A segurança do cluster do ObjectGrid assegura que um servidor de junção tem a credencial correta, portanto, um servidor malicioso não pode juntar-se ao cluster. O ObjectGrid utiliza um mecanismo de cadeia de segredo compartilhado para esta finalidade.

Todos os servidores do ObjectGrid estão de acordo com uma cadeia de segredo compartilhado. Quando um servidor se junta ao cluster, ele é desafiado a apresentar a cadeia secreta. Se a cadeia secreta do servidor de junção corresponder a um no servidor principal, o servidor de junção poderá juntar-se ao cluster; caso contrário, o pedido de junção será rejeitado.

Não é seguro enviar um segredo em texto sem formatação. A infra-estrutura de segurança do ObjectGrid fornece um plug-in do gerenciador de token seguro para permitir que o servidor ″proteja″ este segredo antes de enviá-lo. Você deve decidir como implementar a operação de ″proteção″. O ObjectGrid fornece uma implementação out-of-box, na qual a operação de “proteção” é implementada para criptografar e assinar o segredo.

A cadeia de segredo (authenticationSecret) está configurada no arquivo `security.ogserver.props`:

- `authenticationSecret`: a cadeia de segredo que contestará o servidor. Quando um servidor é inicializado, ele precisa apresentar esta cadeia para o servidor principal. Se a cadeia de segredo corresponder ao que está no servidor principal, este servidor poderá juntar-se ao cluster.

Plug-in SecureTokenManager

Um plug-in do gerenciador de tokens seguros é representado pela interface com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager. A interface é a seguinte:

```
package com.ibm.websphere.objectgrid.security.plugins;
import com.ibm.websphere.objectgrid.security.ObjectGridSecurityException;
import com.ibm.websphere.objectgrid.security.SecurityConstants;
/**
* Esta interface é utilizada por servidores do ObjectGrid para transformar um
* objeto em um token seguro e vice-versa. Um token seguro é uma matriz de byte.
* A seguir está um exemplo de um possível uso: Quando um servidor se junta ao
* cluster, o servidor de junção precisará apresentar uma senha para o servidor
* principal no cluster. Antes de enviar a senha, o servidor de junção chama
* o método generateToken(Object) para gerar um token para esta
* senha. O token deve ser difícil de ser interrompido, portanto, a senha pode
* ser protegida seguramente. O token será então enviado pela ligação.
* Geralmente, o token está associado a um time stamp, portanto, será difícil
* um ataque de reprodução maliciosa.
* No lado de recebimento, o servidor chama o método verifyToken(byte[])
* para verificar o token e reconstruir o objeto correspondente do
* token.
*
* O ObjectGrid utiliza o JCE para fornecer uma implementação padrão desta
* interface. Nesta implementação, ao gerar o token, o objeto é
* criptografado com um time stamp e, em seguida, assinado. Para verificar um
* token, a assinatura do token é verificada e, em seguida, decriptografada.
* Esta implementação precisará de um armazenamento de chaves configurado nos
* servidores do ObjectGrid para suportar a criptografia e decriptografação
* de dados e assinatura e verificação de assinatura. Utilize
* security.ogserver.props para as configurações de chave do token seguro.
*
* Uma classe de implementação deve ter um construtor padrão. Os usuários
* podem configurar a propriedade CustomSecureTokenManagerProps no arquivo
```

```
* de propriedades de configuração de segurança do servidor. Esta
* propriedade será configurada no objeto utilizando o método
* setProperties(String).
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see SecurityConstants#SECURE_TOKEN_MANAGER_CUSTOM_STRING
* @see SecurityConstants#SECURE_TOKEN_MANAGER_DEFAULT_STRING
*/
public interface SecureTokenManager {
/**
*
Configure as propriedades definidas pelo usuário para o depósito de
informações do provedor
*
*
Este método é utilizado para configurar propriedades adicionais de
SecureTokenManager no objeto. Estas propriedades podem ser configuradas
utilizando a propriedade SecureTokenManagerProps
* no arquivo de propriedades de configuração de segurança do servidor.
* Portanto, é possível customizar seu depósito de informações do provedor.
*
* @param properties propriedades definidas pelo usuário
*/
void setProperties(String properties);
/**
* Gera o token para o objeto especificado.
*
* O token gerado deve ser difícil de ser interrompido.
*
* @param o o objeto a ser protegido
*
* @return um token que representa o objeto a ser protegido
*
* @throws ObjectGridSecurityException se qualquer exceção ocorrer durante
* a geração da matriz de byte do token
*/
byte[] generateToken(Object o) throws ObjectGridSecurityException;
/**
* Verifica o token e reconstrói o objeto.
*
* @param bytes a matriz de byte do token que representa o objeto protegido.
*
* @return o objeto protegido
*
* @throws ObjectGridSecurityException se qualquer exceção ocorrer durante
* a verificação da matriz de byte do token
*/
Object verifyToken(byte[] bytes) throws ObjectGridSecurityException;
}
```

O método generateToken(Object) utiliza um objeto a ser protegido e, em seguida, gera um token que não pode ser entendido por outros. O método verifyTokens(byte[]) faz o processo inverso: converte o token de volta ao objeto original.

Uma implementação simples de SecureTokenManager é utilizar um algoritmo de codificação simples (como o algoritmo XOR) para codificar o objeto em formulário serializado e, em seguida, utilizar o algoritmo de decodificação correspondente para decodificar o token. Esta implementação não é segura e é fácil de ser interrompida.

O ObjectGrid fornece uma implementação out-of-box para esta interface. A implementação não é uma API pública e é transparente para você.

A implementação padrão utiliza um par de chaves para assinar e verificar a assinatura e utiliza uma chave secreta para criptografar o conteúdo. Para isso, cada servidor tem um armazenamento de chaves de tipo JCKES para armazenar o par de chaves (chave privada e 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, portanto, ele expira após uma determinada quantidade de tempo. Na extremidade de recebimento, os dados são verificados, decriptografados e comparados com a cadeia de segredo do receptor. Os protocolos de comunicação semelhantes a SSL não são requeridos entre um par de servidores para autenticação, porque as chaves privadas e chaves públicas servem para a mesma finalidade. No entanto, se a comunicação do servidor não for criptografada, os dados poderão ser roubados por violação da 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 ObjectGrid precisam gerar chaves e transportá-las para todos os servidores, o que pode causar problemas de segurança.

Configurações

Para utilizar o gerenciador de tokens seguros, as seguintes propriedades devem ser configuradas no arquivo `security.ogserver.props`:

- Propriedade **secureTokenManagerType**: Esta propriedade indica o gerenciador de tokens seguros a ser utilizado.
  - Se o valor for **none**, nenhum gerenciador de tokens seguros será utilizado.
  - Se o valor for **default**, será utilizado o gerenciador de tokens seguros padrão fornecido out-of-box.
  - Se o valor for **custom**, será utilizado o gerenciador de tokens seguros fornecidos pelo usuário.
- Propriedade **customSecureTokenManagerClass**: Esta propriedade especifica a classe de implementação SecureTokenManager. Ela será utilizada apenas se o valor de secureTokenManagerType for ″custom″. A classe de implementação deve ter um construtor padrão a ser instanciado.
- Propriedade **customSecureTokenManagerProps**: Esta propriedade especifica as propriedades de SecureTokenManager customizado. Ela será utilizada apenas se o valor de secureTokenManagerType for ″custom″. O valor é configurado como Objeto SecureTokenManager com o método setProperties(String).
- Se o valor de secureTokenManagerType estiver configurado como default, as seguintes configurações para chaves de assinatura e de cifras serão requeridas:
  - `secureTokenKeyStore`: Especifica o nome do caminho do arquivo para o armazenamento de chaves que armazena o par de chaves pública/privada e a chave secreta.
  - `secureTokenKeyStoreType`: Especifica o tipo de armazenamento de chaves, por exemplo, JCKES. É possível configurar este valor com base no provedor

JSSE (Java Secure Socket Extension) utilizado. No entanto, este armazenamento de chaves deve poder suportar chaves secretas.

- `secureTokenKeyStorePassword`: Especifica a senha para proteger o armazenamento de chaves.
- `secureTokenKeyPairAlias`: Especifica o alias do par de chaves pública/privada utilizado para assinatura e verificação.
- `secureTokenKeyPairPassword`: Especifica a senha para proteger o alias do par de chaves utilizado para assinatura e verificação.
- `secureTokenSecretKeyAlias`: Especifica o alias da chave secreta utilizado para cifra.
- `secureTokenSecretKeyPassword`: Especifica a senha para proteger a chave secreta.
- `secureTokenCipherAlgorithm`: Especifica o algoritmo utilizado para a cifra. É possível configurar este valor com base no provedor JSSE utilizado.
- `secureTokenSignAlgorithm`: Especifica o algoritmo utilizado para assinatura do objeto. É possível configurar este valor com base no provedor JSSE utilizado.

Segurança do Gateway

O gateway de gerenciamento do ObjectGrid serve como um ponto para delegar os pedidos de administração do cliente ao servidor do ObjectGrid. Este tópico descreve como proteger o acesso ao gateway.

O diagrama a seguir é um exemplo. Se o cliente do ObjectGrid desejar obter as estatísticas de um cluster, ele primeiro enviará um pedido ao gateway. O gateway envia este pedido para os dois servidores para obter as estatísticas e, em seguida, combina as estatísticas. As estatísticas combinadas são enviadas de volta ao cliente.

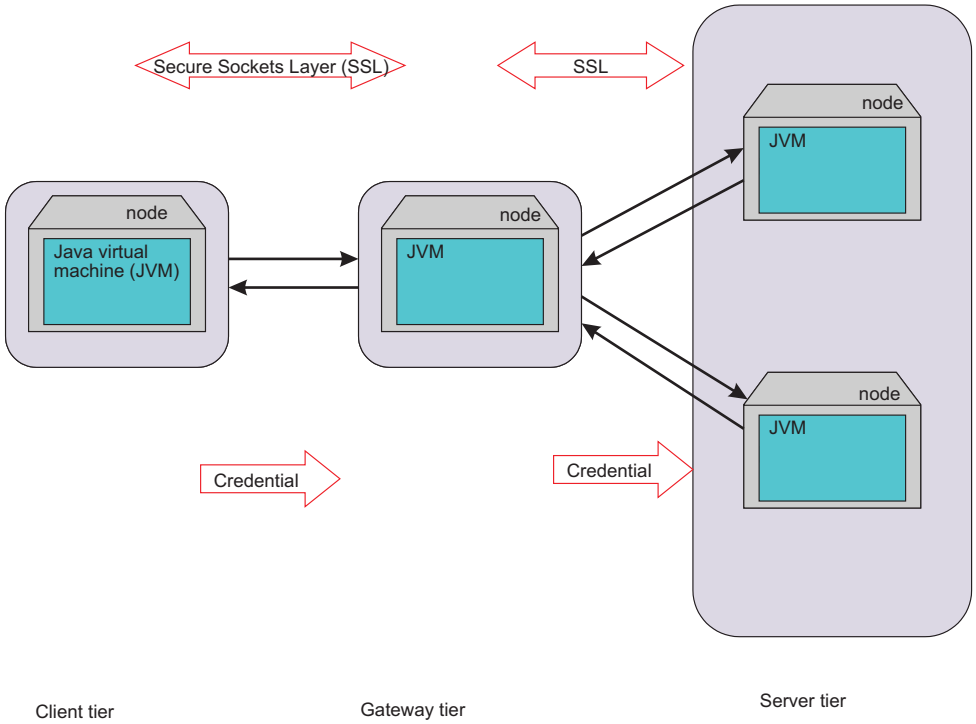


Figura 18. Segurança do Gateway

A comunicação entre o gateway e o servidor utiliza o mecanismo de comunicação do servidor do cliente do ObjectGrid. O gateway é tratado como um cliente do ObjectGrid. A comunicação do cliente e do gateway pode ser protegida por SSL. Este recurso é fornecido pela camada do conector JMX, que é o projeto de código aberto mx4j. O ObjectGrid requer o mx4j apropriado para fazer o gateway funcionar.

Para a autenticação, o gateway propaga a credencial, por exemplo, um ID do usuário e uma senha, que é apresentada pelo cliente ao servidor. A autenticação e autorização são aplicadas em servidores do ObjectGrid.

A autenticação do certificado cliente para o cliente gateway não é suportada.

Segurança do Servidor Gateway

Um servidor gateway é um cliente do ObjectGrid. Todos os aspectos de segurança são iguais aos de um cliente do ObjectGrid. Consulte “Iniciar o Servidor Gateway de Gerenciamento” na página 88 para obter detalhes adicionais sobre como iniciar um servidor gateway a partir de uma linha de comandos.

O trecho de código a seguir demonstra como iniciar o gateway seguro programaticamente:

```
// Obter o ClientSecurityConfiguration do arquivo de propriedades de segurança
do cliente
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration("etc/test/security/security.client.props");
CredentialGenerator creGen = new UserPasswordCredentialGenerator("admin",
"xxxxxx");
csConfig.setCredentialGenerator(credGen);
// Inicializar o gateway
ManagementGateway gateway = ManagementGatewayFactory.getManagementGateway();
gateway.setConnectorPort(namingPort);
gateway.setClusterName("cluster1");
gateway.setHost("localhost");
gateway.setPort("12503");
gateway.setTraceEnabled(true);
gateway.setTraceSpec("ObjectGrid=all=enabled");
gateway.setTraceFile("logs/GatewayTrace.log");
// Configurar o objeto ClientSecurityConfiguration
gateway.setCsConfig(csConfig);
// Iniciar o gateway
gateway.startConnector();
```

No código anterior, um objeto ClientSecurityConfiguration é criado e configurado na instância ManagementGateway.

Segurança do Cliente Gateway

O cliente gateway precisa transmitir uma credencial para um servidor gateway no tempo de conexão. O trecho de código a seguir demonstra como transmitir uma credencial:

```
/**
* recuperar o status do servidor do gateway
*/
public boolean retrieveServerStatus()
throws Exception {
String serverProtocol = "rmi";
String serverHost = "host";
String namingHost = "localhost";
String jndiPath = "/jmxconnector";
```

```
JMXServiceURL url = new JMXServiceURL("service:jmx:" + serverProtocol + "://"
+ serverHost + "/jndi/rmi://" + namingHost + ":" + namingPort + jndiPath);
// Criar JMXCconnectorServer
JMXConnector cntor = JMXConnectorFactory.newJMXConnector(url, null);
// O mapa do ambiente de conexão
Map environment = new HashMap();
// criar uma credencial
UserPasswordCredential gatewayClientCred =
new UserPasswordCredential("admin", "admin1");
environment.put(JMXConnector.CREDENTIALS, gatewayClientCred);
// Conectar e chamar uma operação no MBeanServer remoto
try {
cntor.connect(environment);
}
catch (SecurityException x) {
// Uh-oh! Credenciais inválidas!
throw x;
}
// Obter um stub para o MBeanServer remoto
mbsc = cntor.getMBeanServerConnection();
Iterator it = mbsc.queryMBeans(
new ObjectName("ManagementServer:type=ObjectGrid,S=server1"),
null).iterator();
ObjectInstance oi = (ObjectInstance) it.next();
server1MBean = oi.getObjectName();
boolean status = ((Boolean) mbsc.invoke(
server1MBean,
"retrieveServerStatus",
new Object[] {},
new String[] {})).booleanValue();
return status;
}
```

Neste trecho de código, um objeto gatewayClientCred é criado e colocado no ambiente. Este ambiente é então utilizado para conectar-se ao servidor gateway.

Se desejar utilizar o SSL para conectar-se do cliente gateway ao servidor gateway, será necessário utilizar as propriedades de sistema para armazenar o truststore e a senha do truststore. Por exemplo, você pode transmitir as seguintes propriedades quando iniciar um cliente gateway.

- `-Djavax.net.ssl.trustStore=etc/test/security/client.public`
- `-Djavax.net.ssl.trustStorePassword=public`

Consulte o Web site MX4J - Open Source Java Management Extensions para obter informações adicionais.

Integração de Segurança com o WebSphere Application Server

O ObjectGrid fornece vários recursos de segurança para integrar-se com a infra-estrutura de segurança do WebSphere Application Server.

Integração de Segurança do ObjectGrid Distribuído com o WebSphere Application Server

Para o modelo de ObjectGrid distribuído, a integração de segurança pode ser feita utilizando as seguintes classes:

- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator.`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator`

- com.ibm.websphere.objectgrid.security.plugins.builtins.
  WSTokenCredential

  .

Estas classes são discutidas em “Segurança do Servidor do Cliente” na página 144. A seguir está um exemplo de como utilizar a classe WSTokenCredentialGenerator.

```
/**
* conectar-se ao ObjectGrid Server.
*/
protected ClientClusterContext connect() throws ConnectException {
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration(proFile);
CredentialGenerator gen = getWSCredGen();
csConfig.setCredentialGenerator(gen);
return objectGridManager.connect(csConfig, null);
}
/**
* Obter um WSTokenCredentialGenerator
*
private CredentialGenerator getWSCredGen() {
WSTokenCredentialGenerator gen = new WSTokenCredentialGenerator(
WSTokenCredentialGenerator.RUN_AS_SUBJECT);
return gen;
}
```

No lado do servidor, o WSTokenAuthentication pode ser utilizado como o autenticador para autenticar o objeto WSTokenCredential.

Integração de Segurança do ObjectGrid Local com o WebSphere Application Server

Para o modelo de ObjectGrid local, a integração de segurança pode ser feita utilizando as duas classes a seguir:

- com.ibm.websphere.objectgrid.security.plugins.builtins.
  WSSubjectSourceImpl
- com.ibm.websphere.objectgrid.security.plugins.builtins.
  WSSubjectValidationImpl

Para obter informações adicionais sobre estas classes, consulte “Segurança do ObjectGrid Local” na página 164. É possível configurar a classe WSSubjectSourceImpl como o plug-in SubjectSource e a classe WSSubjectValidationImpl como o plug-in SubjectValidation.

Listeners

O ObjectGrid fornece duas interfaces de tipo de listener que podem ser estendidas. As extensões podem avisá-lo por meio da interface de extensão e descrever operações que são executadas em uma instância do ObjectGrid ou em uma instância do mapa.

Interface ObjectGridEventListener

Utilize a interface ObjectGridEventListener para receber notificações quando ocorrerem eventos importantes em um 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.

A Interface ObjectGridEventListener

A interface ObjectGridEventListener possui os seguintes métodos. Estes métodos são chamados quando alguns eventos importantes ocorrem no ObjectGrid.

```
/**
* Este método é chamado quando o próprio ObjectGrid é inicializado.
* Uma instância de Sessão utilizável é transmitida para este Listener
* para permitir a reprodução opcional de LogSequence recebido em um Mapa.
*
* @param session A instância da Sessão à qual este Listener está
* associado.
*/
void initialize(Session session);
/**
* Este evento sinaliza o início de uma transação (sessão).
* Uma versão stringified do TxID é fornecida para
* correlação com o encerramento da transação
* (sessão), se você desejar utilizar esta versão. O tipo de
* transação (sessão) também é
* fornecido por meio do parâmetro booleano isWriteThroughEnabled.
*
* @param txid Versão Stringified do TxID
* @param isWriteThroughEnabled sinalizador Booleano que indica se a
* Sessão foi iniciada por meio de beginNoWriteThrough
*/
void transactionBegin(String txid, boolean isWriteThroughEnabled);
/**
* Isto sinaliza o encerramento de uma transação (sessão). Uma versão
stringified
* do TxID é fornecida para correlação com o início
* da transação
* (sessão), se necessário. As alterações também são relatadas. As
* utilizações típicas deste evento são para invalidação no mesmo
* nível customizada ou para envio de confirmação no mesmo nível.
* Este listener de eventos exibe as alterações. As chamadas
* para este método são feitas
* após uma confirmação e são seqüenciadas para que sejam entregues
* uma por uma, não em paralelo. A ordem dos eventos é a ordem de
* confirmação.
*
* @param txid Versão Stringified do TxID
* @param isWriteThroughEnabled um sinalizador booleano que indica se
* a Sessão foi
* iniciada por meio de beginNoWriteThrough
* @param committed um sinalizador booleano que indica que a Sessão
* foi confirmada
* (true) ou se recebeu rollback (false)
* @param changes Uma Coleta de LogSequences que foram processados
* para a Sessão atual.
*/
void transactionEnd(String txid, boolean isWriteThroughEnabled,
boolean committed, Collection /*/* <LogSequence> */*/ changes);
/**
* Este método será chamado quando o ObjectGrid for destruído. É o
* oposto de initialize. Quando este método é chamado, o
* ObjectGridEventListener pode liberar qualquer recurso utilizado.
*/
void destroy();
```

Incluir e Remover Objetos ObjectGridEventListeners

Um ObjectGrid pode ter vários ObjectGridEventListeners. Existem dois métodos no ObjectGrid que permitem a inclusão de

ObjectGridEventListeners. Os ObjectGridEventListeners que foram incluídos também podem ser removidos de um ObjectGrid.

O método addEventListener pode ser utilizado para incluir um ObjectGridEventListener em um ObjectGrid.

```
/**
* Inclua um EventListener na Sessão. Os eventos importantes
* serão comunicados para os listeners interessados por meio deste retorno
* de chamada.
* Vários listeners de eventos podem ser registrados, sem uma ordenação
* implícita de notificações de eventos.
*
* Observe, este método é permitido para ser chamado antes e depois do
* método {@link ObjectGrid#initialize()}.
*
* @param cb Uma instância do ObjectGridEventListener
*/
void addEventListener(ObjectGridEventListener cb);
```

Para incluir uma lista de ObjectGridEventListeners, utilize o método setEventListeners:

```
/**
* Isto sobrescreve a lista atual de retornos de chamada e a substitui
* pela lista fornecida de retornos de chamada.
*
* Observe, este método é permitido para ser chamado antes e depois do
* método {@link ObjectGrid#initialize()}.
* @param callbacks
*/
void setEventListeners(List callbacks);
```

Para remover um ObjectGridEventListener de um ObjectGrid e utilizar o método removeEventListener:

```
/**
* Remove um EventListener da Sessão. Se o EventListener desejado
*não for localizado na Sessão, nenhum erro será retornado.
*
* Observe, este método é permitido para ser chamado antes e depois do
* método {@link ObjectGrid#initialize()}.
* @param cb Uma instância do ObjectGridEventListener
*/
void removeEventListener(ObjectGridEventListener cb);
```

Criar um Listener de Evento do ObjectGrid Customizado

Para utilizar um listener de evento do ObjectGrid customizado, primeiro crie uma classe que implementa a interface com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener. Inclua o listener customizado em um ObjectGrid para receber notificação de eventos importantes. Um ObjectGridEventListener pode ser configurado programaticamente ou com XML:

- **Programaticamente.** Suponha que o nome da classe do listener de evento do ObjectGrid seja a classe com.company.org.MyObjectGridEventListener. Esta classe implementa a interface ObjectGridEventListener. O trecho de código a seguir cria o 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);
  ```

- **Com XML.** Um ObjectGridEventListner também pode ser configurado utilizando XML. O XML a seguir cria uma configuração que é equivalente ao listener de evento do ObjectGrid criado pelo programa 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" />
  </objectGrid>
  </objectGrids>
  </objectGridConfig>
  ```

  Forneça este arquivo para o ObjectGridManager para facilitar a criação desta configuração. O trecho de código a seguir demonstra como criar um ObjectGrid utilizando este arquivo XML. O ObjectGrid criado possui um ObjectGridEventListener configurado no ObjectGrid myGrid.

  ```
  ObjectGridManager objectGridManager =
  ObjectGridManagerFactory.getObjectGridManager();
  ObjectGrid myGrid =
  objectGridManager.createObjectGrid("myGrid", new URL(
  "file:etc/test/myGrid.xml"), true, false);
  ```

Observar Alterações em um Mapa

O método transactionEnd na interface ObjectGridEventListener é muito útil para aplicativos que estão interessados em observar entradas nos Mapas locais. Um aplicativo pode incluir um destes listeners e, em seguida, utilizar o método transactionEnd para ver quando as entradas são alteradas. Por exemplo, se o ObjectGrid estiver funcionando em um modo distribuído, um aplicativo poderá observar alterações que chegam. Suponha que as entradas replicadas foram para preços de estoque mais recentes. Este listener pode observar estas alterações que chegam e atualizar um segundo Mapa que mantém o valor de uma posição em um portfólio. O listener deve fazer todas as alterações utilizando a Sessão fornecida para o listener no método initialize na interface ObjectGridEventListener. O listener pode distinguir entre alterações locais e alterações remotas que chegam, geralmente verificando se a transação é de gravação direta. As alterações que chegam de ObjectGrids no mesmo nível são sempre de gravação direta.

Interface MapEventListener

Utilize a interface MapEventListener para receber eventos importantes sobre um mapa. Os eventos são enviados para o MapEventListener quando uma entrada é liberada a partir do mapa e quando o pré-carregamento de um mapa é concluído.

Interface MapEventListener

A interface MapEventListener possui os seguintes métodos. Implementar a interface com.ibm.websphere.objectgrid.plugins.MapEventListener para criar um MapEventListener customizado.

```
/**
* Este método é chamado quando a entrada especificada é liberada
* a partir do mapa. A evicção pode ter ocorrido pelo processamento do
* Evictor ou chamando um dos métodos invalidate no ObjectMap.
*
* @param key A chave para a entrada do mapa que foi liberada.
```

```
* @param value O valor que estava na entrada do mapa liberada. O objeto
* de valor não deve ser modificado.
*
*/
void entryEvicted(Object key, Object value);
/**
* Este método é chamado quando o pré-carregamento deste mapa é concluído.
*
* @param t Um objeto Throwable que indica se o pré-carregamento foi
* concluído sem a ocorrência de nenhum Throwable durante o
* pré-carregamento do mapa. Uma referência nula indica que o
* pré-carregamento foi concluído sem a ocorrência de objetos Throwable
* durante o pré-carregamento do mapa.
*/
void preloadCompleted( Throwable t );
```

Incluir e Remover MapEventListeners

Os seguintes métodos do BackingMap permitem que MapEventListeners sejam incluídos e removidos de um mapa:

```
/**
* Inclui um MapEventListener neste BackingMap.
*
* Observe, este método é permitido para ser chamado antes e depois do
* método ObjectGrid.initialize().
* @param eventListener Uma referência não nula a um MapEventListener
* para inclusão na lista.
*
* @throws IllegalArgumentException se eventListener for nulo.
*
* @see MapEventListener
*/
public void addMapEventListener(MapEventListener
eventListener );
/**
* Configura a lista de objetos MapEventListener.
*
* Se este BackingMap já tiver uma Lista de
* MapEventListeners, esta lista será substituída pela
* Lista transmitida como um argumento para a chamada atual
* deste método. Este método pode ser chamado antes e depois
* do método ObjectGrid.initialize().
*
* @param eventListenerList Uma referência não nula a uma Lista de
* objetos MapEventListener.
*
* @throws IllegalArgumentException será emitida se
* o eventListenerList for nulo
* ou o eventListenerList contiver uma referência nula
* ou um objeto que não seja uma instância do
* MapEventListener.
*
* @see MapEventListener
*/
public void setMapEventListeners( List /*MapEventListener*/
eventListenerList );
/**
* Remove um MapEventListener deste BackingMap.
*
* Observe, este método é permitido para ser chamado antes e depois do
* método ObjectGrid.initialize().
*
* @param eventListener Uma referência não nula a um listener de evento
* que foi incluído anteriormente chamando o método
* addMapEventListener(MapEventListener) ou
* setMapEventListeners(List) desta interface.
*
* @throws IllegalArgumentException se eventListener for nulo.
```

```
*
* @see MapEventListener
*/
public void removeMapEventListener(MapEventListener eventListener );
```

Criar um MapEventListener

Para criar um MapEventListener customizado, implemente a interface com.ibm.websphere.objectgrid.plugins.MapEventListener. Para utilizar o MapEventListener, inclua-o em um BackingMap. Um MapEventListener pode ser criado e configurado programaticamente ou com XML:

- **Programaticamente**. O nome da classe do MapEventListener customizado é a classe com.company.org.MyMapEventListener. Esta classe implementa a interface MapEventListener. O trecho de código a seguir cria o MapEventListener customizado e o inclui em um BackingMap:

  ```
  ObjectGridManager objectGridManager =
  ObjectGridManagerFactory.getObjectGridManager();
  ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
  BackingMap myMap = myGrid.defineMap("myMap");
  MyMapEventListener myListener = new MyMapEventListener();
  myMap.addMapEventListener(myListener);
  ```

- **Criação de XML**. Um MapEventListner também pode ser configurado utilizando XML. O XML a seguir obtém uma configuração equivalente à criação programática anterior. 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>
  </backingMapPluginCollection>
  </objectGridConfig>
  ```

  O fornecimento deste arquivo para o ObjectGridManager facilita a criação desta configuração. O trecho de código a seguir mostra como criar um ObjectGrid utilizando este arquivo XML. O ObjectGrid recém-criado 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);
  ```

Evictors

O ObjectGrid fornece um mecanismo de evictor padrão. Também é possível fornecer um mecanismo de evictor conectável.

Um *evictor* controla a associação de entradas em cada BackingMap. O evictor padrão utiliza uma política de evicção *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. Este tópico descreve os dois tipos de evictors.

Evictor Time to Live Padrão

O ObjectGrid fornece um evictor TTL (Time to Live) para cada BackingMap. O evictor TTL mantém um tempo de expiração para cada entrada criada. Quando chegar o tempo de expiração para uma entrada, o evictor removerá a entrada do BackingMap. Para minimizar o impacto no desempenho, o evictor TTL pode esperar para liberar uma entrada após o tempo de expiração, mas nunca antes da expiração da entrada.

O BackingMap possui atributos utilizados para controlar como o evictor time to live calcula o tempo de expiração para cada entrada. Os aplicativos configuram o atributo ttlType para especificar como o evictor TTL deve calcular o tempo de expiração. O atributo ttlType pode ser configurado como um dos seguintes valores:

- **Nenhum** indica que uma entrada no BackingMap nunca expira. O evictor TTL não retira estas entradas.
- **Hora de Criação** indica que a hora de criação de uma entrada é utilizada no cálculo do tempo de expiração.
- **Hora do Último Acesso** indica que a hora do último acesso de uma entrada é utilizada no cálculo do tempo de expiração.

Se o atributo ttlType não estiver configurado em um BackingMap, o tipo padrão de **Nenhum** será utilizado para que o evictor TTL não retire nenhuma das entradas. Se o atributo ttlType for configurado como **hora de criação** ou **hora do último acesso**, o valor do atributo time to live no BackingMap será incluído na hora de criação ou na hora do último acesso para calcular o tempo de expiração. A precisão de tempo do atributo de mapa time to live está em segundos. Um valor de `0` para o atributo time to live é um valor especial utilizado para indicar que a entrada do mapa pode existir sempre, ou seja, a entrada permanece no mapa até que o aplicativo remova ou invalide explicitamente a entrada do mapa.

Especificar Atributos para Evictors TTL

Os evictors TTL estão associados a instâncias do BackingMap. O trecho de código a seguir demonstra como a interface do BackingMap pode ser utilizada para configurar os atributos necessários para que, quando uma entrada for criada, ela tenha um tempo de expiração configurado como dez minutos após sua criação.

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

O argumento do método setTimeToLive é `600` porque indica que o valor 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. Quando o tempo de expiração tiver sido atingido, o evictor TTL limpará a entrada.

Se um aplicativo exigir que o tempo de expiração seja configurado como a hora do último acesso mais dez minutos, uma linha no código anterior deverá ser alterada. O argumento transmitido para o método setTtlEvictorType é alterado 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.

Quando `TTLType.LAST_ACCESS_TIME` é utilizado, as interfaces ObjectMap e JavaMap podem ser utilizadas para substituir o valor time to live de BackingMap. Este mecanismo permite que um aplicativo utilize um valor time to live diferente para cada entrada criada. Suponha que o trecho de código anterior tenha sido utilizado para configurar o atributo ttlType como `LAST_ACCESS_TIME` e o valor time to live tenha sido configurado como dez minutos no BackingMap. Um aplicativo pode então substituir o valor 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 trecho de código anterior, a entrada com a chave `key1` tem um tempo de expiração do tempo de inserção mais 30 minutos como resultado da chamada de método setTimeToLive(1800) no ObjectMap. A variável oldTimeToLive1 está configurada como 600 porque o valor time to live do BackingMap será utilizado como um valor padrão se o método setTimeToLive não tiver sido chamado anteriormente no ObjectMap.

A entrada com a chave `key2` tem um tempo de expiração do tempo de inserção mais 20 minutos como resultado da chamada de método setTimeToLive(1200) no ObjectMap. A variável oldTimeToLive2 está configurada como 1800 porque o valor time to live da chamada de método ObjectMap.setTimeToLive anterior configurou o time to live como 1800.

O exemplo anterior mostra duas entradas do mapa sendo inseridas no mapa myMap para os valores de chave key1 e key2. Em um ponto no tempo posterior, o aplicativo de um novo encadeamento talvez queira atualizar estas entradas do mapa com novos valores do mapa. No entanto, o aplicativo deseja reter os valores time-to-live utilizados no tempo de inserção para cada entrada do mapa. O exemplo a seguir ilustra como reter os valores time-to-live utilizando uma constante definida na interface ObjectMap para esta única finalidade:

```
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 ObjectMap.USE_DEFAULT é utilizado na chamada de método setTimeToLive, key1 retém seu valor time-to-live de 1.800 segundos e key2 retém seu valor time-to-live de 1.200 segundos, porque estes valores foram utilizados quando estas entradas do mapa foram inseridas pela transação anterior.

O exemplo anterior também mostra uma nova entrada do mapa para key3 sendo inserida. 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 de BackingMap” na página 113 para obter informações sobre como o atributo time-to-live está definido no BackingMap.

Consulte a documentação da API para o método setTimeToLive nas interfaces ObjectMap e JavaMap. Ele avisa que será gerada uma exceção `IllegalStateException` se o método BackingMap.getTtlEvictorType() retornar algo diferente do valor `TTLType.LAST_ACCESS_TIME`. ObjectMap e JavaMap podem ser utilizados apenas para substituir o valor time to live quando você estiver utilizando o tipo de evictor TTL `LAST_ACCESS_TIME`. Este método não pode ser utilizado para substituir o valor time to live quando você estiver utilizando o tipo de evictor TTL `CREATION_TIME` ou o tipo de evictor TTL `NONE`.

Utilizar um Arquivo XML para Especificar Atributos para o Evictor TTL

Em vez de utilizar a interface do BackingMap para configurar programaticamente os atributos de BackingMap para serem utilizados pelo evictor TTL, um arquivo XML pode ser utilizado para configurar cada BackingMap. O código a seguir demonstra como configurar estes atributos para três diferentes BackingMaps:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="grid1">
<backingMap name="map1" ttlEvictorType="NONE" />
<backingMap name="map2" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="1800" />
<backingMap name="map3" ttlEvictorType="CREATION_TIME" timeToLive="1200" />
</objectgrid>
</objectGrids>
```

O exemplo anterior mostra que o BackingMap map1 utiliza um tipo de evictor TTL `NONE`. O BackingMap map2 utiliza um tipo de evictor TTL `LAST_ACCESS_TIME` e tem um valor time to live de 1.800 segundos ou 30 minutos. O BackingMap map3 está definido para utilizar um tipo de evictor TTL `CREATION_TIME` e tem um valor time to live de 1.200 segundos ou 20 minutos.

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. Um evictor conectável opcional pode ser utilizado para liberar entradas com base no número de entradas existentes em vez de basear-se 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.

- **LRUEvictor** é um evictor que utiliza um algoritmo *least recently used* para decidir quais entradas liberar quando o BackingMap exceder um número máximo de entradas.

- **LFUEvictor** é um evictor que utiliza um algoritmo *least frequently used* para decidir quais entradas retirar 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 de evicção LRU ou LFU não for adequado para um aplicativo específico, será possível gravar seus próprios evictores para obter a estratégia de evicção desejada.

Especificar um Evictor Conectável

Como os evictores estão associados a BackingMaps, a interface de BackingMap é utilizada para especificar o evictor conectável a ser utilizado. O trecho de código a seguir é um exemplo de especificação de um evictor LRUEvictor para o BackingMap map1 e um evictor LFUEvictor para o BackingMap map2:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor;
import com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
LRUEvictor evictor = new LRUEvictor();
evictor.setMaxSize(1000);
evictor.setSleepTime( 15 );
evictor.setNumberOfLRUQueues( 53 );
bm.setEvictor(evictor);
bm = og.defineMap( "map2" );
LFUEvictor evictor2 = new LFUEvictor();
evictor2.setMaxSize(2000);
evictor2.setSleepTime( 15 );
evictor2.setNumberOfHeaps( 211 );
bm.setEvictor(evictor2);
```

O snippet anterior mostra um evictor LRUEvictor sendo utilizado para BackingMap map1 com um número máximo de entradas de `1000`. O evictor LFUEvictor é utilizado para o BackingMap map2 com um número máximo de entradas de `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 aprimorar 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 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 aprimorar 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.

Utilizar XML para Especificar 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:

```
<?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 LRU evictor">
<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 LFU evictor">
<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>
```

Gravar um Evictor Customizado

O ObjectGrid pode ser estendido para utilizar um algoritmo eviction. É necessário criar um evictor customizado que implementa a interface com.ibm.websphere.objectgrid.plugins.Evictor. A interface é a seguinte:

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

- O método initialize é chamado durante a inicialização do objeto BackingMap. Este método 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 apply é chamado quando as transações que acessam uma ou mais entradas do BackingMap são confirmadas. 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 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 possui 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 utilizados por um plug-in Evictor para retorno de chamada para a estrutura do ObjectGrid da seguinte forma:

- O método setEvictorData é utilizado por um evictor para solicitar a estrutura utilizada para armazenar e associar algum objeto do evictor criado com a entrada indicada pelo argumento de 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 utilizado por um evictor para recuperar os dados transmitidos para o método setEvictorData durante uma chamada de método apply anterior. Se os dados do evictor para o argumento de chave especificado não forem localizados, será retornado um objeto especial `KEY_NOT_FOUND` definido na interface EvictorCallback.
- O método evictMapEntries é utilizado por um evictor para solicitar a evicção de uma ou mais entradas do 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 utilizado por um evictor para solicitar a evicção de uma ou mais entradas do mapa. Este método será utilizado apenas se o objeto transmitido para o método setEvictorData não implementar a interface com.ibm.websphere.objectgrid.plugins.EvictorData.

O ObjectGrid chama o método apply da interface do Evictor *após* a conclusão de uma transação. 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 trecho 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;
// Variáveis da instância
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, é criada uma lista vinculada utilizada como uma fila *primeiro a entrar, primeiro a sair* para implementar 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.getCacheEntry().getKey();
LogElement.Type type = elem.getType();
if ( type == LogElement.INSERT )
{
// efetuar o processamento de inserção aqui incluindo na frente da fila LRU.
EvictorData data = new EvictorData(key);
evictorCallback.setEvictorData(key, data);
queue.addFirst( data );
}
else if ( type == LogElement.UPDATE || type == LogElement.FETCH ||
type == LogElement.TOUCH )
{
// efetuar o processamento de atualização aqui movendo o objeto EvictorData para
// a frente da fila.
EvictorData data = evictorCallback.getEvictorData(key);
queue.remove(data);
queue.addFirst(data);
}
else if ( type == LogElement.DELETE || type == LogElement.EVICT )
{
// efetuar o processamento de remoção aqui removendo o objeto EvictorData
// da fila.
EvictorData data = evictorCallback.getEvictorData(key);
if ( data == EvictionEventCallback.KEY_NOT_FOUND )
{
// A suposição aqui é que seu encadeamento de evictor assíncrono
// liberou a entrada do mapa antes que este encadeamento pudesse
// processar o pedido LogElement. Portanto, você provavelmente não precisa
// fazer nada quando isso ocorrer.
}
```

```
else
{
// A chave foi localizada. Portanto, processe os dados do evictor.
if ( data != null )
{
// Ignorar nulo retornado pelo método remove, pois o encadeamento do
// evictor no qual foi efetuado spawn já pode ter sido removido da fila.
// Mas precisamos deste código caso ele não seja o encadeamento do evictor
// que causou a ocorrência deste LogElement.
queue.remove( data );
}
else
{
// Dependendo de como você grava seu Evictor, esta possibilidade
// pode não existir ou pode indicar um defeito em seu evictor
// devido a uma lógica de sincronização de encadeamento imprópria.
}
}
}
}
}
```

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:

```
// O método destroy apenas interrompe o encadeamento no qual foi efetuado spawn
pelo método initialize.
    public void destroy()
{
evictorThread.interrupt();
}
// A seguir está o método run do encadeamento no qual foi efetuado spawn pelo
método initialize.
public void run()
{
// Efetuar loop até que o método destroy interrompa este encadeamento.
boolean continueToRun = true;
while ( continueToRun )
{
try
{
// Suspender por um momento antes da limpeza da fila.
// O sleepTime é um bom candidato para a configuração de uma
// propriedade do evictor.
Thread.sleep( sleepTime );
int queueSize = queue.size();
```

```
// Liberar entradas se o tamanho da fila crescer além
// do tamanho máximo. Obviamente, o tamanho máximo
// seria outra propriedade do evictor.
int numToEvict = queueSize - maxSize;
if ( numToEvict > 0 )
{
// Remover do final da fila desde que o final seja
// a entrada least recently used.
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 )
{
// A fila está vazia.
queueSize = 0;
}
}
// Solicitar evicção se a lista de chaves não estiver vazia.
if ( ! evictList.isEmpty() )
{
evictorCallback.evictMapEntries( evictList );
}
}
}
catch (InterruptedException e)
{
continueToRun = false;
}
} // encerrar loop while
} // encerrar método run.
```

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.

Loaders

Um loader do ObjectGrid é um componente conectável que permite que um mapa do ObjectGrid se comporte como um cache de memória para dados que geralmente são 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) remota também pode ser utilizada como a origem de dados que permite que caches baseados em hub sejam construídos utilizando o ObjectGrid. Um loader tem a lógica para leitura e gravação de dados de e para um armazenamento persistente.

Um Loader é um plug-in para um mapa de suporte do ObjectGrid. Apenas um Loader pode ser associado a um mapa de suporte especificado. Cada mapa de suporte possui sua própria instância do Loader. O mapa de suporte solicita dados que ele não contém de seu loader. As alterações feitas no mapa são enviadas para o loader. O plug-in do loader permite que o mapa de suporte mova dados entre o mapa e seu armazenamento persistente.

Conectar um Loader

O trecho de código a seguir ilustra como um Loader fornecido pelo aplicativo é conectado ao mapa de suporte para map1 utilizando a 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 );
```

A suposição é que `MyLoader` seja a classe fornecida pelo aplicativo que implementa a interface com.ibm.websphere.objectgrid.plugins.Loader. Como a associação de um Loader com um mapa de suporte não pode ser alterada após a inicialização de um ObjectGrid, o código deve ser executado antes de chamar o método `initialize` da interface do ObjectGrid que está sendo chamada. Ocorrerá uma exceção `IllegalStateException` em um método `setLoader` se ele for chamado após a ocorrência da inicialização.

O Loader fornecido pelo aplicativo pode ter propriedades configuradas. No exemplo, o loader MyLoader é utilizado para ler e gravar dados de uma tabela em um banco de dados relacional. O loader deve ter o nome do banco de dados e o nível de isolamento de SQL a ser utilizado. O loader MyLoader possui os métodos `setDataBaseName` e `setIsolationLevel` que permitem que o aplicativo configure estas duas propriedades do Loader.

Um Loader fornecido pelo aplicativo também pode ser conectado utilizando um arquivo XML. O exemplo a seguir ilustra como o loader MyLoader é conectado ao mapa de suporte map1 com as mesmas propriedades do Loader de nome de banco de dados e nível de isolamento sendo configuradas:

```
<?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>
```

Implementar a Interface do Loader

Um Loader fornecido pelo aplicativo deve implementar a interface *com.ibm.websphere.objectgrid.plugins.Loader*. A interface do Loader 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;
}
```

Cada uma das seções a seguir fornece uma explicação e considerações sobre quando implementar cada um dos métodos na interface do Loader.

método get

O mapa de suporte chama o método `get` do Loader para obter valores associados a uma lista de chaves transmitidas como o argumento **keyList**. O método get é requerido para retornar uma lista de valores java.lang.util.List, um para cada chave que está 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 loader 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 Loader. Como um mapa de suporte pode ser configurado para permitir `null` como um valor válido, é muito importante para o Loader retornar o objeto especial `KEY_NOT_FOUND` quando não puder localizar a chave. Este valor permite que o mapa de suporte faça a distinção entre um valor `null` e um valor que não existe porque a chave não foi localizada. Se um mapa de suporte não suportar valores `null`, um Loader que retorna nulo em vez do objeto `KEY_NOT_FOUND` para uma chave que não existe resultará em uma exceção.

O argumento **forUpdate** informa o Loader se o aplicativo chamou um método `get` no mapa ou um método `getForUpdate` no mapa. Consulte a interface com.ibm.websphere.objectgrid.ObjectMap para obter informações adicionais. 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 Loader 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 Loader utiliza a sintaxe for update apenas quando utiliza

uma política de controle de simultaneidade pessimista. Para um controle de simultaneidade otimista, o Loader nunca utiliza a sintaxe for update na instrução SQL select. O Loader é responsável por decidir utilizar o argumento `forUpdate` com base na política de controle de simultaneidade que está sendo utilizada pelo Loader.

Para obter uma explicação do parâmetro **txid**, consulte o tópico “Plug-in TransactionCallback” na página 217.

Método `batchUpdate`

O método `batchUpdate` é importante na interface do Loader. Este método é chamado sempre que o ObjectGrid precisa aplicar todas as alterações atuais no Loader. O Loader recebe uma lista de alterações para este Mapa. 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 itera sobre o conjunto de alterações e armazena em batch três instruções JDBC (Java Database Connectivity), uma com `insert`, outra com `update` e uma com `delete`.

```
import java.util.Collection;
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
public void batchUpdate(TxID tx, LogSequence sequence)
throws LoaderException
{
// Obter uma conexão SQL para utilizar.
Connection conn = getConnection(tx);
try
{
// Processar a lista de alterações e construir um conjunto de instruções
// preparadas
// para executar uma operação SQL update, insert ou delete
// de batch.
Iterator iter = sequence.getPendingChanges();
while ( iter.hasNext() )
{
LogElement logElement = (LogElement)iter.next();
Object key = logElement.getCacheEntry().getKey();
Object value = logElement.getCurrentValue();
switch ( logElement.getType().getCode() )
{
case LogElement.CODE_INSERT:
buildBatchSQLInsert( tx, key, value, conn );
break;
case LogElement.CODE_UPDATE:
buildBatchSQLUpdate( tx, key, value, conn );
break;
case LogElement.CODE_DELETE:
buildBatchSQLDelete( tx, key, conn );
break;
}
}
// Executar as instruções de batch que foram construídas pelo loop acima.
Collection statements = getPreparedStatementCollection( tx, conn );
iter = statements.iterator();
while ( iter.hasNext() )
{
PreparedStatement pstmt = (PreparedStatement) iter.next();
pstmt.executeBatch();
```

```
}
}
catch (SQLException e)
{
LoaderException ex = new LoaderException(e);
throw ex;
}
}
```

A amostra anterior ilustra a lógica de alto nível de processamento do argumento LogSequence, mas os detalhes de como uma instrução SQL `insert`,`update` ou `delete` é construída não são ilustrados. Alguns dos pontos-chave que estão ilustrados incluem:

- O método `getPendingChanges` é chamado no argumento LogSequence para obter um iterador sobre a lista de LogElements que o Loader 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` que imprime em relatório 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 ObjectGrid, cada mapa de suporte que está definido é inicializado. Se um Loader for conectado a um mapa de suporte, o mapa de suporte chamará o método `preloadMap` na interface do Loader para permitir que o loader 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.

```
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
boolean tranActive = false;
ResultSet results = null;
Statement stmt = null;
      Connection conn  = null;
try
{
session.beginNoWriteThrough();
tranActive = true;
ObjectMap map = session.getMap( backingMap.getName() );
TxID tx = session.getTxID();
// Obter uma conexão de autoconfirmação para utilização que esteja
// configurada para
// um nível de isolamento confirmado por leitura.
conn = getAutoCommitConnection(tx);
// Pré-carregar o Mapa Employee com objetos EmployeeRecord.
// Ler todos os Funcionários a partir da tabela, mas
```

```
// limitar o pré-carregamento às primeiras 100 linhas.
stmt = conn.createStatement();
results = stmt.executeQuery( SELECT_ALL );
int rows = 0;
while ( results.next() && rows < 100 )
{
int key = results.getInt(EMPNO_INDEX);
EmployeeRecord emp = new EmployeeRecord( key );
emp.setLastName( results.getString(LASTNAME_INDEX) );
emp.setFirstName( results.getString(FIRSTNAME_INDEX) );
emp.setDepartmentName( results.getString(DEPTNAME_INDEX) );
emp.updateSequenceNumber( results.getLong(SEQNO_INDEX) );
emp.setManagerNumber( results.getInt(MGRNO_INDEX) );
map.put( new Integer(key), emp );
++rows;
}
// Confirmar a transação.
session.commit();
tranActive = false;
}
catch (Throwable t)
{
throw new LoaderException("preload failure: " + t, t);
}
finally
{
if ( tranActive )
{
try
{
session.rollback();
}
catch ( Throwable t2 )
{
// Tolerar falhas de rollback e
// permitir que o Throwable original seja emitido.
}
}
// Certificar-se de limpar outros recursos do banco de dados aqui,
// bem como instruções de fechamento, conjuntos de resultados, etc.
}
}
```

Esta amostra ilustra os seguintes pontos-chave:

- O mapa de suporte preloadMap utiliza o objeto de Sessão transmitido para ele como o argumento de sessão.
- O método `Session.beginNoWriteThrough()` é utilizado para iniciar a transação em vez do método `begin`. O Loader não pode ser chamado para cada operação `put` que ocorre neste método para carregar o mapa.
- O Loader pode mapear colunas de tabela de funcionários para um campo no objeto Java EmployeeRecord.
- O Loader 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 pode ser emitida que ocorre entre o tempo em que o método `beginNoWriteThrough` é chamado e o tempo em que o `método` commit é chamado faça o bloco `finally` efetuar rollback da 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 desempenhar outras

ações de limpeza que podem ser requeridas, como o fechamento da conexão JDBC e outros objetos JDBC.

A amostra `preloadMap` está utilizando uma instrução SQL select que seleciona todas as linhas da tabela. Em seu Loader fornecido pelo aplicativo, pode ser necessário configurar uma ou mais propriedades do Loader 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 Loader em uma etapa. Mesmo que o Loader 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 Loader mais eficientes. A seguir está um exemplo de armazenamento em cache do objeto TransactionCallback e do objeto OptimisticCallback como variáveis da instância do Loader para que os outros métodos do Loader 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 feito, porque após a inicialização do BackingMap, os objetos TransactionCallback e OptimisticCallback não podem ser alterados ou substituídos. É aceitável armazenar em cache estas referências do objeto como variáveis da instância do Loader.

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.OptimisticCallback;
import com.ibm.websphere.objectgrid.plugins.TransactionCallback;
// Variáveis da instância do Loader.
MyTransactionCallback ivTcb; // MyTransactionCallback
// estende TransactionCallback
MyOptimisticCallback ivOcb; // MyOptimisticCallback
// implementa OptimisticCallback
...
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
// Armazenar em cache os objetos TransactionCallback e OptimisticCallback
// em variáveis da instância deste Loader.
ivTcb = (MyTransactionCallback)
session.getObjectGrid().getTransactionCallback();
ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
// O restante do código preloadMap (conforme mostrado no exemplo anterior).
}
```

Para obter informações sobre pré-carregamento e pré-carregamento recuperável como relacionados ao failover de replicação, consulte “Programação de Replicação” na página 228.

Considerações sobre o Loader

Utilize as seguintes considerações ao implementar um loader.

Considerações sobre Pré-carregamento

Cada mapa de suporte possui um atributo booleano preloadMode que pode ser configurado para indicar se o pré-carregamento de um mapa foi concluído assincronicamente. Por padrão, o atributo preloadMode é configurado como `false`, que indica que a inicialização do mapa de suporte não será concluída até que o pré-carregamento do mapa esteja concluído. Por exemplo, a inicialização do mapa

de suporte não será concluída até que o método preloadMap seja retornado. Se o método preloadMap tiver que ler uma grande quantidade de dados de seu backend e carregá-los para o mapa, sua conclusão pode ser relativamente longa. Neste caso, é possível configurar um mapa de suporte para utilizar o pré-carregamento assíncrono do mapa, configurando o atributo preloadMode como true. Esta configuração faz o código de inicialização do mapa de suporte efetuar spawn de um encadeamento que chama o método preloadMap, permitindo a conclusão da inicialização de um mapa de suporte enquanto o pré-carregamento do mapa ainda está em andamento.

O trecho de código a seguir ilustra como o atributo preloadMode é configurado para ativar o pré-carregamento assíncrono:

```
BackingMap bm = og.defineMap( "map1" );
bm.setPreloadMode( true );
```

O atributo preloadMode também pode ser configurado utilizando um arquivo XML conforme ilustrado no seguinte exemplo:

```
<backingMap name="map1" preloadMode="true"
pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
```

TxID e Utilização da Interface TransactionCallback

O método get e os métodos batchUpdate da interface do Loader são transmitidos para um objeto TxID que representa a transação da Sessão que requer que a operação get ou batchUpdate seja desempenhada. É possível que os métodos get e batchUpdate sejam chamados mais de uma vez por transação. Portanto, os objetos com escopo definido pela transação requeridos pelo Loader geralmente são mantidos em um slot do objeto TxID. Um Loader JDBC (Java Database Connectivity) é utilizado para ilustrar como um Loader utiliza as interfaces TxID e TransactionCallback.

Também é possível que vários mapas do ObjectGrid sejam armazenados no mesmo banco de dados. Cada mapa possui seu próprio Loader e cada Loader pode precisar conectar-se ao mesmo banco de dados. Ao conectar-se ao mesmo banco de dados, cada Loader deseja utilizar a mesma conexão JDBC para que as alterações em cada tabela sejam confirmadas como parte da transação do mesmo banco de dados. Geralmente, a mesma pessoa que grava a implementação do Loader também grava a implementação do TransactionCallback. O melhor método é se a interface TransactionCallback for estendida para incluir métodos requeridos pelo Loader para obter uma conexão com o banco de dados e para armazenar em cache as instruções preparadas. A razão desta metodologia se torna aparente conforme você verifica como as interfaces TransactionCallback e TxID são utilizadas pelo Loader.

Como exemplo, talvez o Loader exija que a interface TransactionCallback seja estendida da seguinte forma:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
public interface MyTransactionCallback extends TransactionCallback
{
Connection getAutoCommitConnection(TxID tx, String databaseName)
throws SQLException;
Connection getConnection(TxID tx, String databaseName,
int isolationLevel ) throws SQLException;
PreparedStatement getPreparedStatement(TxID tx, Connection conn,
```

```
String tableName, String sql) throws SQLException;
Collection getPreparedStatementCollection( TxID tx, Connection conn,
String tableName );
}
```

Utilizando estes novos métodos, os métodos `get` e `batchUpdate` do Loader podem obter uma conexão da seguinte forma:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getConnection(TxID tx, int isolationLevel)
{
Connection conn = ivTcb.getConnection(tx, databaseName, isolationLevel );
return conn;
}
```

No exemplo anterior e nos exemplos seguintes, *ivTcb* e *ivOcb* são variáveis da instância do Loader que foram inicializadas conforme descrito na seção “Considerações sobre Pré-carregamento” na página 207. A variável *ivTcb* é uma referência à instância MyTransactionCallback e *ivOcb* é uma referência à instância MyOptimisticCallback. A variável *databaseName* é uma variável da instância do Loader que foi configurada como uma propriedade do Loader durante a inicialização do mapa de suporte. O argumento `isolationLevel` é uma das constantes da Conexão JDBC que estão definidas para os diversos níveis de isolamento suportados pelo JDBC. Se o Loader estiver utilizando uma implementação otimista, o método `get` geralmente utilizará uma conexão de autoconfirmação JDBC para buscar os dados do banco de dados. Neste caso, o Loader pode ter um método `getAutoCommitConnection` que seja implementado da seguinte forma:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getAutoCommitConnection(TxID tx)
{
Connection conn = ivTcb.getAutoCommitConnection(tx, databaseName);
return conn;
}
```

Lembre-se de que o método `batchUpdate` possui a seguinte instrução switch:

```
switch ( logElement.getType().getCode() )
{
case LogElement.CODE_INSERT:
buildBatchSQLInsert( tx, key, value, conn );
break;
case LogElement.CODE_UPDATE:
buildBatchSQLUpdate( tx, key, value, conn );
break;
case LogElement.CODE_DELETE:
buildBatchSQLDelete( tx, key, conn );
break;
}
```

Cada um dos métodos `buildBatchSQL` utiliza a interface MyTransactionCallback para obter uma instrução preparada. A seguir está um trecho de código que mostra o método `buildBatchSQLUpdate` construindo uma instrução SQL update para atualizar uma entrada EmployeeRecord e incluindo-a na atualização de batch:

```
private void buildBatchSQLUpdate( TxID tx, Object key, Object value, Connection
conn )
throws SQLException, LoaderException
```

```
{
String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?, DEPTNO = ?,
SEQNO = ?, MGRNO = ? where EMPNO = ?";
PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn, "employee",
sql );
EmployeeRecord emp = (EmployeeRecord) value;
sqlUpdate.setString(1, emp.getLastName());
sqlUpdate.setString(2, emp.getFirstName());
sqlUpdate.setString(3, emp.getDepartmentName());
sqlUpdate.setLong(4, emp.getSequenceNumber());
sqlUpdate.setInt(5, emp.getManagerNumber());
sqlUpdate.setInt(6, key);
sqlUpdate.addBatch();
}
```

Quando o loop `batchUpdate` tiver construído todas as instruções preparadas, ele chamará o método `getPreparedStatementCollection`. Este método pode ser implementado da seguinte forma:

```
private Collection getPreparedStatementCollection( TxID tx, Connection conn )
{
return ( ivTcb.getPreparedStatementCollection( tx, conn, "employee" ) );
}
```

Quando o aplicativo chama o método `commit` na Sessão, o código da Sessão chama o método `commit` no método `TransactionCallback` depois de enviar todas as alterações feitas pela transação fora do Loader para cada mapa que foi alterado pela transação. Como todos os Loaders utilizaram o método `MyTransactionCallback` para obter qualquer conexão e instruções preparadas requeridas, o método `TransactionCallback` sabe qual conexão utilizar para solicitar que o backend confirme as alterações. Portanto, estender a interface TransactionCallback com métodos requeridos por cada um dos Loaders tem as seguintes vantagens:

- O objeto TransactionCallback encapsula a utilização de slots TxID para dados com escopo definido pela transação e o Loader não requer informações sobre os slots TxID. O Loader apenas precisa saber sobre os métodos que foram incluídos no TransactionCallback utilizando a interface MyTransactionCallback para as funções de suporte requeridas pelo Loader.
- O objeto TransactionCallback pode assegurar que o compartilhamento da conexão ocorra entre cada Loader que se conecta ao mesmo backend para que um protocolo de confirmação de duas fases seja evitado.
- O objeto TransactionCallback pode assegurar que a conexão com o backend seja orientada para conclusão por meio de uma confirmação ou rollback chamado na conexão quando apropriado.
- O TransactionCallback pode assegurar que a limpeza de recursos do banco de dados ocorra quando uma transação for concluída.
- TransactionCallback pode se ocultar se estiver obtendo uma conexão gerenciada de um ambiente gerenciado como o WebSphere Application Server ou algum outro servidor de aplicativos compatível com J2EE (Java 2 Platform, Enterprise Edition). Esta vantagem permite que o mesmo código do Loader seja utilizado em ambientes gerenciados e não gerenciados. Apenas o plug-in TransactionCallback deve ser alterado.

Para obter informações detalhadas sobre como a implementação de TransactionCallback utiliza os slots TxID para dados com escopo definido pela transação, consulte “Plug-in TransactionCallback” na página 217.

OptimisticCallback

Conforme mencionado anteriormente, o Loader pode decidir utilizar uma abordagem otimista para controle de simultaneidade. Se este for o caso, o exemplo de método `buildBatchSQLUpdate` precisará ser modificado ligeiramente para implementar uma abordagem otimista. Existem várias maneiras possíveis para utilizar uma abordagem otimista. Uma maneira típica é ter uma coluna de time stamp ou uma coluna do contador de números de seqüência para o controle de versões de cada atualização da linha. Suponha que a tabela de funcionários tenha uma coluna de números de seqüência que aumenta sempre que a linha é atualizada.

Em seguida, você modifica a assinatura do método `buildBatchSQLUpdate` para que ela seja transmitida para o objeto LogElement em vez do par de chave e valor. Ele também precisa utilizar o objeto OptimisticCallback que está conectado ao mapa de suporte para obter o objeto da versão inicial e para atualizar o objeto da versão. A seguir está um exemplo de um método `buildBatchSQLUpdate` modificado que utiliza a variável da instância *ivOcb* que foi inicializada conforme descrito na seção preloadMap:

```
private void buildBatchSQLUpdate( TxID tx, LogElement le,
Connection conn )throws SQLException, LoaderException
{
// Obter o objeto da versão inicial quando esta entrada do mapa foi lida pela última
// vez ou atualizada no banco de dados.
Employee emp = (Employee) le.getCurrentValue();
long initialVersion = ((Long) le.getVersionedValue()).longValue();
// Obter o objeto da versão de Employee atualizado para a operação SQL
//update.
Long currentVersion = (Long)ivOcb.getVersionedObjectForValue( emp );
long nextVersion = currentVersion.longValue();
// Agora construa o SQL update que inclui o objeto de versão na cláusula where
// para verificação otimista.
String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?,
DEPTNO = ?,SEQNO = ?, MGRNO = ? where EMPNO = ? and SEQNO = ?";
PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn,
"employee", sql );
sqlUpdate.setString(1, emp.getLastName());
sqlUpdate.setString(2, emp.getFirstName());
sqlUpdate.setString(3, emp.getDepartmentName());
sqlUpdate.setLong(4, nextVersion );
sqlUpdate.setInt(5, emp.getManagerNumber());
sqlUpdate.setInt(6, key);
sqlUpdate.setLong(7, initialVersion);
sqlUpdate.addBatch();
}
```

O exemplo mostra que o LogElement é utilizado para obter o valor de versão inicial. Quando a transação acessa pela primeira vez a entrada do mapa, é criado um LogElement com o objeto Employee inicial obtido do mapa. O objeto Employee inicial também é transmitido para o método getVersionedObjectForValue na interface OptimisticCallback e o resultado é salvo no LogElement. Este processamento ocorre antes de um aplicativo receber uma referência ao objeto Employee inicial e de chamar algum método que altere o estado do objeto Employee inicial.

O exemplo mostra que o Loader utiliza o método getVersiondObjectForValue para obter o objeto de versão para o objeto Employee atual atualizado. Antes de chamar o método batchUpdate na interface do Loader, o ObjectGrid chama o método updateVersionedObjectForValue na interface OptimisticCallback para fazer com que seja gerado um novo objeto de versão para o objeto Employee atualizado. Quando

o método batchUpdate retornar ao ObjectGrid, o LogElement será atualizado com o objeto de versão atual, portanto, ele se torna o novo objeto de versão inicial. Esta etapa é necessária porque o aplicativo pode ter chamado o método flush no mapa em vez do método commit na Sessão. É possível que o Loader seja chamado várias vezes por uma única transação para a mesma chave. Por isso, o ObjectGrid assegura que o LogElement seja atualizado com o novo objeto de versão sempre que a linha for atualizada na tabela de funcionários.

Agora que o Loader tem o objeto de versão inicial e o próximo objeto de versão, ele pode executar uma instrução SQL update que configura a coluna SEQNO para o próximo valor do objeto de versão e utiliza o valor do objeto de versão inicial na cláusula `where`. Esta abordagem às vezes é referida como sendo uma instrução `update` superqualificada. A utilização da instrução `update` superqualificada permite que o banco de dados relacional verifique se a linha não foi alterada por alguma outra transação entre o tempo em que esta transação leu os dados a partir do banco de dados e o tempo em que esta transação atualiza o banco de dados. Se outra transação modificou a linha, a matriz de contagem retornada pela atualização de batch indica que zero linhas foram atualizadas para esta chave. O Loader é responsável por verificar se a operação SQL `update` de fato atualizou a linha. Caso contrário, o Loader exibirá uma exceção `com.ibm.websphere.objectgrid.plugins.OptimisticCollisionException` para informar à Sessão que o método batchUpdate falhou porque mais de uma transação simultânea está tentando atualizar a mesma linha na tabela de banco de dados. Esta exceção faz a Sessão efetuar rollback e o aplicativo deve tentar novamente a transação inteira. O fundamento lógico é que a nova tentativa será bem-sucedida, motivo pelo qual esta abordagem é chamada de otimista. A abordagem otimista, de fato, tem um desempenho melhor se os dados forem alterados com menos freqüência ou se transações simultâneas raramente tentarem atualizar a mesma linha.

É importante que o Loader utilize o parâmetro *key* do construtor OptimisticCollisionException para identificar qual chave ou conjunto de chaves causou a falha do método batchUpdate otimista. O parâmetro de chave pode ser o próprio objeto de chave ou uma matriz de objetos de chave se mais de uma chave resultar em uma falha de atualização otimista. O ObjectGrid utiliza o método getKey do construtor OptimisticCollisionException para determinar quais entradas do mapa contêm dados stale e causaram a exceção. Parte do processamento de rollback é liberar cada entrada do mapa stale do mapa. A liberação de entradas stale é necessária para que qualquer transação subseqüente que acessa a mesma chave ou chaves resulte no método get da interface do Loader que está sendo chamada para atualizar as entradas do mapa com os dados atuais do banco de dados.

Outras maneiras para um Loader implementar uma abordagem otimista incluem:

- Não existe nenhuma coluna de time stamp ou de número de seqüência. Neste caso, o método getVersionObjectForValue na interface OptimisticCallback apenas retorna o próprio objeto de valor como a versão. Com esta abordagem, o Loader precisa construir uma cláusula `where` que inclua cada um dos campos do objeto de versão inicial. Esta abordagem não é muito eficiente e nem todos os tipos de colunas estão qualificados para serem utilizados na cláusula `where` de uma instrução SQL update superqualificada. Esta abordagem geralmente não é utilizada.
- Não existe nenhuma coluna de time stamp ou de número de seqüência. No entanto, diferente da abordagem anterior, a cláusula `where` contém apenas campos de valores que foram modificados pela transação. Uma maneira de

detectar quais campos foram modificados é configurar o modo de cópia no mapa de suporte para ser o modo `CopyMode.COPY_ON_WRITE`. Este modo de cópia requer que uma interface de valor seja transmitida para o método `setCopyMode` na interface BackingMap. O BackingMap cria objetos de proxy dinâmicos que implementam a interface de valor fornecida. Com este modo de cópia, o Loader pode lançar cada valor em um objeto com.ibm.websphere.objectgrid.plugins.ValueProxyInfo. A interface ValueProxyInfo possui um método que permite que o Loader obtenha a Lista de nomes de atributos que foram alterados pela transação. Este método permite que o Loader chame os métodos `get` na interface de valor para os nomes de atributos para obter os dados alterados e para construir uma instrução SQL update que configura apenas os atributos alterados. A cláusula `where` agora pode ser construída para ter a coluna-chave primária mais cada uma das colunas de atributos alteradas. Esta abordagem é mais eficiente do que a abordagem anterior, mas requer que mais código seja gravado no Loader e gera a possibilidade de que o cache de instrução preparado precise ser maior para manipular as diferentes permutações. No entanto, se as transações geralmente modificarem apenas alguns dos atributos, esta limitação pode não ser um problema.

- Alguns bancos de dados relacionais podem ter uma API para ajudar na manutenção automática de dados da coluna que são úteis para o controle de versões otimista. Consulte a documentação do banco de dados para determinar se existe esta possibilidade.

Plug-in ObjectTransformer

Utilize o plug-in ObjectTransformer quando precisar de alto desempenho. Se você tiver problemas de desempenho com o uso da CPU, inclua um plug-in ObjectTransformer em cada mapa. Se não fornecer um plug-in ObjectTransformer, até 60-70% do tempo total de CPU será gasto na serialização e cópia de entradas.

Finalidade

A finalidade do plug-in ObjectTransformer é permitir que 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 esta cópia configurando o modo de cópia do Mapa como `NO_COPY` 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. Tal plug-in pode reduzir a sobrecarga de cópia de 65-70% para 2/3% do tempo total de CPU.

As implementações dos métodos padrão copyKey e copyValue primeiro tentam utilizar o método clone(), se fornecido. Se não for fornecida nenhuma implementação do método clone(), a implementação será padronizada como serialização.

A serialização do objeto também é utilizada diretamente quando o ObjectGrid está 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 pares no ObjectGrid. Tenha atenção ao fornecer um método de serialização customizado em vez de utilizar a serialização de JDK interna. 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 detalha como o ObjectGrid tenta serializar chaves e valores:

- Se um plug-in ObjectTransformer customizado for gravado e conectado, o ObjectGrid chamará métodos nos métodos ObjectTransformer para serializar chaves e valores e obter cópias de chaves e valores de objetos.
- Se um plug-in ObjectTransformer customizado não for utilizado, o ObjectGrid será serializado e desserializado de acordo com o padrão. Se o padrão for utilizado, cada objeto será implementado como externalizável ou implementado como seriável.
  - Se o objeto suportar a interface Externalizable, o método `writeExternal` será chamado. Os objetos que são implementados como externalizáveis geram melhor desempenho.
  - Se o objeto não suportar a interface Externalizable e implementar Serializable, o objeto será salvo utilizando o método `ObjectOutputStream`.

Interface ObjectTransformer

Consulte a documentação da API para obter informações adicionais sobre a interface ObjectTransformer. A interface ObjectTransformer contém os seguintes métodos que serializam e desserializam chaves ou valores e chaves ou valores de cópia:

```
public interface ObjectTransformer
{
void serializeKey(Object key, ObjectOutputStream stream)
throws IOException;
void serializeValue(Object value, ObjectOutputStream stream)
throws IOException;
Object inflateKey(ObjectInputStream stream)
throws IOException, ClassNotFoundException;
Object inflateValue(ObjectInputStream stream)
throws IOException, ClassNotFoundException;
Object copyKey(Object value);
Object copyValue(Object value);
}
```

Uso da Interface ObjectTransformer

É possível utilizar a interface ObjectTransformer nas seguintes situações:

- objeto não seriável
- objeto seriável mas aprimora o desempenho da serialização
- cópia de chave ou de 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)
{
//fazer fluir um
}
}
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyKey(java.lang.Object)
*/
public Object copyKey(Object key) {
String ticket=(String) key;
String ticketCopy= new String (ticket);
return ticketCopy;
}
}
```

Em seguida, conecte esta classe MyStockObjectTransformer customizada ao BackingMap:

```
ObjectGridManager ogf=ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogf.getObjectGrid("NYSE");
BackingMap bm = og.defineMap("NYSEStocks");
MyStockObjectTransformer ot = new MyStockObjectTransformer();
bm.setObjectTransformer(ot);
```

Plug-in TransactionCallback

Um aplicativo geralmente conecta um plug-in TransactionCallback e um Loader como um par. O Loader é responsável por buscar dados do backend, bem como aplicar alterações no backend. Esta busca e limpeza geralmente ocorrem dentro do contexto de uma transação do ObjectGrid.

O plug-in TransactionCallback tem as seguintes responsabilidades:

- Reserva slots para um estado específico da transação, que é necessário para a transação e o Loader
- Converte ou mapeia uma transação do ObjectGrid para uma transação de plataforma
- Configura o estado por transação quando o ObjectGrid inicia uma transação
- Confirma a transação quando a transação do ObjectGrid é confirmada
- Efetua rollback da transação quando é efetuado rollback da transação do ObjectGrid

O ObjectGrid não é coordenador de transaçõesXA. O ObjectGrid depende da plataforma para fornecer esse recurso. Os métodos `begin`, `commit` e `rollback` do

ObjectGrid que são apresentados em uma Sessão são chamadas de ciclo de vida. O plug-in TransactionCallback deve receber estes eventos e fazer a plataforma fornecer o recurso transacional para os recursos utilizados pelos Loaders. Este tópico examina vários cenários e discute como o plug-in TransactionCallback pode ser gravado para funcionar para estes cenários.

Visão Geral do Plug-in TransactionCallback

O plug-in TransactionCallback é um POJO que implementa a interface TransactionCallback. A interface TransactionCallback é semelhante à seguinte amostra:

```
public interface TransactionCallback
{
void initialize(ObjectGrid objectGrid) throws TransactionCallbackException;
void begin(TxID id) throws TransactionCallbackException;
void commit(TxID id) throws TransactionCallbackException;
void rollback(TxID id) throws TransactionCallbackException;
boolean isExternalTransactionActive(Session session);
}
```

Método `initialize`

O método `initialize` é chamado quando o ObjectGrid é inicializado. O retorno de chamada reserva slots para o objeto TxID de que ele precisa. Geralmente, ele reserva um slot para cada parte do estado ou Objeto que deseja criar no método `begin` quando uma transação é iniciada. Por exemplo, você deseja utilizar um Gerenciador de Persistência com o ObjectGrid como um Loader. Supondo que este gerenciador de persistência tenha objetos de estado de sessão e de transação, o TransactionCallback obteria uma sessão e uma transação e manteria referências a esses dois objetos em slots no TxID. Neste caso, o método `initialize` é semelhante à seguinte amostra:

```
/**
* Isto é chamado quando a grade é inicializada pela primeira vez. Apenas
* reservaremos nossos slots no TxID.
*/
public void initialize(ObjectGrid objectGrid) throws
TransactionCallbackException
{
// reservar um slot para a transação do gerenciador de persistência
TXslot = objectGrid.reserveSlot(TxID.SLOT_NAME);
// reservar um slot para a sessão do gerenciador de persistência
SessionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
}
```

Um TxID possui slots. Os slots são entradas em uma matriz ArrayList. Os plug-ins podem reservar uma entrada na matriz ArrayList chamando o método `ObjectGrid.reserveSlot` e indicando que deseja um slot no objeto TxID. O método então retorna o próximo índice de entrada para o aplicativo. O aplicativo pode então armazenar informações neste slot. Os próximos métodos demonstram esta técnica.

Método `begin`

O ObjectGrid chama este método quando inicia uma nova transação. O plug-in mapeia este evento para uma transação real que os Loaders podem então utilizar para as chamadas de métodos `get` e `update` que chegam antes de o método `commit` ser chamado. A seguir está um método `begin` de exemplo que mapeia um begin do ObjectGrid para um begin de transação do gerenciador de persistência:

```
/**
* Isto é chamado quando a grade inicia uma nova transação. Apenas criamos
* uma transação do gerenciador de persistência e chamamos begin nela. Em
* seguida, armazenamos a transação no slot TxID para que possamos obtê-la
* posteriormente sem precisar do ThreadLocal etc.
*/
public void begin(TxID id) throws TransactionCallbackException
{
Session PMsession = getPMcurrentSession();
Transaction tx = PMsession.beginTransaction();
id.putSlot(TXslot, tx);
id.putSlot(SessionSlot, PMsession);
}
```

Esta amostra depende do fato de que o método `initialize` reservou dois slots no objeto TxID. Um slot serve para a sessão do gerenciador de persistência e o outro slot serve para a Transação do gerenciador de persistência. O método `begin` chama o gerenciador de persistência para obter uma sessão, armazena-a no slot SessionSlot indexado e cria uma Transação na sessão e armazena uma referência a esta transação utilizando o slot TXSlot indexado.

Método `commit`

O método `commit` é chamado quando uma transação do ObjectGrid está sendo confirmada. Todos os Loaders já foram limpos. A responsabilidade do plug-in é comunicar este evento de confirmação para a plataforma.

```
/**
* Isto é chamado quando a grade deseja confirmar uma transação.
* Apenas a transmitimos ao gerenciador de persistência.
*/
public void commit(TxID id) throws TransactionCallbackException
{
Transaction tx = (Transaction)id.getSlot(TXslot);
tx.commit();
}
```

O método consulta a transação do gerenciador de persistência armazenada no slot e, em seguida, chama o método `commit`.

Método `rollback`

Este método é chamado quando uma transação do ObjectGrid deseja efetuar rollback de uma transação. O plug-in redireciona-a para o gerenciador de transações da plataforma. A seguir está o trecho de código:

```
/**
* Isto é chamado quando a grade deseja efetuar rollback de uma transação.
* Apenas a transmitimos ao gerenciador de persistência.
*/
public void rollback(TxID id) throws TransactionCallbackException
{
Transaction tx = (Transaction)id.getSlot(TXslot);
tx.rollback();
}
```

Este método é muito semelhante ao método `commit`. Ele obtém uma referência à transação do gerenciador de persistência de um slot e, em seguida, chama o método `rollback`.

Método `isExternalTransactionActive`

Uma sessão do ObjectGrid normalmente funciona no modo de confirmação automática ou no modo de transação. O modo de confirmação automática significa que uma transação implícita é criada em torno de cada chamada de método nas instâncias do ObjectMap para a sessão. Se nenhuma

transação estiver ativa e um aplicativo fizer uma chamada em um método do ObjectMap, a estrutura chamará este método no plug-in TransactionCallback para verificar se existe uma transação ativa apropriada. Se este método retornar true, a estrutura executará um início automático; caso contrário, ela fará uma confirmação automática. Este método permite que o ObjectGrid seja integrado em ambientes nos quais o aplicativo chama os métodos `begin`, `commit` ou `rollback` nas APIs da plataforma em vez das APIs do ObjectGrid.

Cenário: Ambiente J2SE (Java 2 Platform, Standard Edition) Baseado em JDBC (Java Database Connectivity) Simples

Este exemplo utiliza um ambiente J2SE no qual o aplicativo tem um Loader baseado em JDBC. Existem dois Mapas, cada um com um Loader que suporta cada Mapa por uma tabela diferente no banco de dados. O plug-in TransactionCallback obtém uma conexão JDBC e, em seguida, chama os métodos `begin`, `commit` e `rollback` na conexão. A seguir está a implementação de TransactionCallback de amostra:

```
public class JDBCTCB implements TransactionCallback
{
DataSource datasource;
int connectionSlot;
public JDBCTCB(DataSource ds)
{
datasource = ds;
}
public void initialize(ObjectGrid objectGrid)
throws TransactionCallbackException
{
connectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
}
public void begin(TxID id) throws TransactionCallbackException
{
try
{
Connection conn = datasource.getConnection();
conn.setAutoCommit(false);
id.putSlot(connectionSlot, conn);
}
catch(SQLException e)
{
throw new TransactionCallbackException("Cannot start transaction", e);
}
}
public void commit(TxID id) throws TransactionCallbackException
{
      Connection conn  = null;
try
{
conn = (Connection)id.getSlot(connectionSlot);
conn.commit();
conn.close();}
catch(SQLException e)
{
throw new TransactionCallbackException("Cannot commit transaction", e);
}
finally {
if (conn!=null) {
try {
conn.close();}
catch (SQLException closeE) {
}
}
```

```
}
}
public void rollback(TxID id) throws TransactionCallbackException
{
      Connection conn  = null;
try
{
conn = (Connection)id.getSlot(connectionSlot);
conn.rollback();
conn.close();}
catch(SQLException e)
{
throw new TransactionCallbackException("Cannot rollback transaction", e);
}
finally {
if (conn!=null) {
try {
conn.close();}
catch (SQLException closeE) {
}
}
}
}
public boolean isExternalTransactionActive(Session session)
{
return false;
}
public int getConnectionSlot()
{
return connectionSlot;
}
}
```

Este exemplo mostra um plug-in TransactionCallback que converte os eventos de transação do ObjectGrid em uma conexão JDBC. Quando o plug-in é inicializado, ele reserva um único slot para manter uma referência de conexão JDBC. O método `begin` então obtém uma conexão JDBC para a nova transação, desativa a confirmação automática e, em seguida, armazena uma referência à conexão no slot TxID. Os métodos `commit` e `rollback` recuperam a conexão do slot TxID e chamam o método apropriado na conexão. O método `isExternalTransaction` sempre retorna false, indicando que o aplicativo deve utilizar as APIs de transação do ObjectGrid explicitamente para controlar transações. Um Loader emparelhado com este plug-in obtém a conexão JDBC do TxID. Um Loader é semelhante ao seguinte exemplo:

```
public class JDBCLoader implements Loader
{
JDBCTCB tcb;
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
tcb = (JDBCTCB)session.getObjectGrid().getTransactionCallback();
}
public List get(TxID txid, List keyList, boolean forUpdate)
throws LoaderException
{
Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
// implementar get aqui
   return null;
}
public void batchUpdate(TxID txid, LogSequence sequence)
throws LoaderException, OptimisticCollisionException
{
```

```
Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
// TODO implementar atualização de batch aqui
}
}
```

O Loader obtém uma referência à instância JDBCTCB quando o método `initialize` é chamado. Ele então obtém a Conexão obtida por JDBCTCB quando requerido nos métodos `get` e `batchUpdate`. As implementações de TransactionCallback e de Loaders geralmente são gravadas em pares que cooperam entre si. A implementação de TransactionCallback manipula a Transação e armazena objetos requeridos pelos Loaders em slots no TxID. Os Loaders implementam os métodos `get` e `batchUpdate` no contexto de uma transação gerenciada pelo TransactionCallback utilizando recursos geralmente obtidos pelo TCB.

Cenário: Ambiente do Mecanismo de Servlet

Neste cenário, o ObjectGrid está utilizando um Loader baseado em JDBC, mas em um mecanismo de servlet gerenciado. O contêiner espera que seja utilizado o método `UserTransaction` para iniciar e confirmar transações. Isto é um pouco diferente do caso do J2SE, porque não é necessário o armazenamento de uma referência à conexão JDBC em um slot TxID. O contêiner gerencia a conexão JDBC. Quando uma transação do contêiner está ativa, uma conexão consultada utilizando uma origem de dados resulta sempre na mesma conexão, porque o contêiner lembra quais conexões são utilizadas por esta transação e retorna a mesma conexão sempre que o método `DataSource.getConnection` é chamado. Suponha que a referência da origem de dados esteja configurada como `Shareable` no exemplo a seguir:

```
public class ManagedJDBCTCB implements TransactionCallback {
UserTransaction tx;
public void initialize(ObjectGrid objectGrid)
throws TransactionCallbackException
{
try
{
InitialContext ic = new InitialContext();
tx = (UserTransaction)ic.lookup("java:comp/UserTransaction");
}
catch(NamingException e)
{
throw new TransactionCallbackException("Cannot find UserTransaction", e);
}
}
public void begin(TxID id) throws TransactionCallbackException
{
try
{
tx.begin();
}
catch(SystemException e)
{
throw new TransactionCallbackException("Cannot begin tx", e);
}
catch(NotSupportedException e)
{
throw new TransactionCallbackException("Cannot begin tx", e);
}
}
public void commit(TxID id) throws TransactionCallbackException
{
try
{
```

```
tx.commit();
}
catch(SystemException e)
{
throw new TransactionCallbackException("Cannot commit tx", e);
}
catch(HeuristicMixedException e)
{
throw new TransactionCallbackException("Cannot commit tx", e);
}
catch(RollbackException e)
{
throw new TransactionCallbackException("Cannot commit tx", e);
}
catch(HeuristicRollbackException e)
{
throw new TransactionCallbackException("Cannot commit tx", e);
}
}
public void rollback(TxID id) throws TransactionCallbackException
{
try
{
tx.rollback();
}
catch(SystemException e)
{
throw new TransactionCallbackException("Cannot commit tx", e);
}
}
public boolean isExternalTransactionActive(Session session) {
return false;
}
}
```

Este exemplo obtém uma referência ao método `UserTransaction` no método `initialize` e, em seguida, mapeia `begin`, `commit` e `rollback` nos métodos `UserTransaction` apropriados. Não são necessários slots, porque o contêiner verifica se as informações de conexão corretas são recuperadas para esta transação. A seguir está o Loader JDBC que funciona com esta implementação do TransactionCallback:

```
public class ManagedJDBCLoader implements Loader
{
DataSource myDataSource;
ManagedJDBCLoader(DataSource ds)
{
myDataSource = ds;
}
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
}
public List get(TxID txid, List keyList, boolean forUpdate)
throws LoaderException
{
try
{
Connection conn = myDataSource.getConnection();
// TODO implementar get aqui com esta conexão
   return null;
}
catch(SQLException e)
{
throw new LoaderException("Cannot get objects", e);
}
```

```
}
public void batchUpdate(TxID txid, LogSequence sequence)
throws LoaderException, OptimisticCollisionException
{
try
{
Connection conn = myDataSource.getConnection();
// TODO implementar update aqui utilizando esta conexão
}
catch(SQLException e)
{
throw new LoaderException("Cannot update objects", e);
}
}
}
```

Este exemplo pode ser mais simples do que a versão básica do JDBC porque o contêiner gerencia as conexões e verifica se dentro da mesma transação o método `DataSource.getConnection` sempre retorna a mesma conexão quando é chamado sempre com a mesma transação ativa. Não tente armazenar em cache a conexão em um slot como resultado, embora o aplicativo possa armazenar em cache a conexão, se optar por isso.

Interface OptimisticCallback

Você pode fornecer um objeto de retorno de chamada otimista conectável que implementa a interface com.ibm.websphere.objectgrid.plugins.OptimisticCallback.

Finalidade

A interface OptimisticCallback é utilizada para fornecer operações de comparação otimistas para os valores de um mapa. Um OptimisticCallback é requerido quando a estratégia de bloqueio otimista está sendo utilizada, conforme descrito em “Bloqueio Otimista” na página 137. O ObjectGrid fornece uma implementação de OptimisticCallback padrão. No entanto, geralmente o aplicativo deve conectar sua própria implementação da interface OptimisticCallback.

Conectar um Objeto OptimisticCallback Fornecido pelo Aplicativo

O exemplo a seguir demonstra como um aplicativo pode conectar um objeto OptimisticCallback para o mapa de suporte 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 );
```

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>
```

Implementação Padrão

A estrutura do ObjectGrid fornece uma implementação padrão da interface OptimisticCallback que será utilizada se o aplicativo não conectar um objeto OptimisticCallback fornecido pelo aplicativo, conforme 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 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 loader é conectado para o mapa de suporte.
- O loader sabe como desempenhar a comparação otimista sem assistência de um plug-in OptimisticCallback.

Como o Loader sabe como lidar com controle de versões otimista sem assistência de um objeto OptimisticCallback? O Loader tem conhecimento do objeto de classe de valor e sabe qual campo do objeto de valor é utilizado como um valor de controle de versões otimista. Por exemplo, suponha que a seguinte interface seja utilizada para o objeto de valor para o mapa employees:

```
public interface Employee
{
// Número de seqüência seqüencial utilizado para controle de versões otimista.
public long getSequenceNumber();
public void setSequenceNumber(long newSequenceNumber);
// Outros métodos get/set para outros campos do objeto Employee.
}
```

Neste caso, o Loader sabe que pode utilizar o método getSequenceNumber para obter as informações de versão atuais para um objeto de valor Employee. Ele incrementa o valor retornado para gerar um novo número de versão antes de atualizar o armazenamento persistente com o novo valor Employee. Para um Loader JDBC (Java Database Connectivity), o número de seqüência atual na cláusula where de uma instrução SQL update superqualificada é utilizado e utiliza o novo número de seqüência gerado para configurar a coluna de número de seqüência para o novo valor de número de seqüência. Outra possibilidade é que o Loader utiliza alguma função fornecida por backend que atualiza automaticamente uma coluna oculta que pode ser utilizada para controle de versões otimista. 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 Loader estiver utilizando uma destas técnicas para manter informações de controle de versões otimista, o aplicativo não precisará fornecer uma implementação de OptimisticCallback. Neste caso, o OptimisticCallback padrão é utilizável, porque o Loader pode manipular o controle de versões otimista sem assistência de um objeto OptimisticCallback.

Implementar a Interface OptimisticCallback

A interface OptimisticCallback contém os seguintes métodos e valores especiais:

```
public interface OptimisticCallback
{
final static Byte NULL_OPTIMISTIC_VERSION;
Object getVersionedObjectForValue(Object value);
void updateVersionedObjectForValue(Object value);
void serializeVersionedValue(Object versionedValue,
ObjectOutputStream stream) throws IOException;
Object inflateVersionedValue(ObjectInputStream stream) throws
IOException, ClassNotFoundException;
}
```

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 de OptimisticCallback padrão for utilizada em vez de uma implementação de OptimisticCallback fornecida pelo aplicativo.

Método getVersionedObjectForValue
Este método 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 Loader 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 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 Loader estiver conectado, o mapa de suporte não utilizará as informações de controle de versões otimista. Em vez disso, o Loader é responsável por desempenhar a comparação de controle de versões otimista e por atualizar as informações de controle de versões quando necessário. O Loader geralmente obtém o objeto de controle de versões inicial do LogElement transmitido para o método batchUpdate do Loader, 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 Loader, que significa que a mesma pessoa que gravou o Loader gravou a implementação de EmployeeOptimisticCallbackImpl ou trabalhou perto da pessoa que implementou o EmployeeOptimisticCallbackImpl - por exemplo, concordou com o valor retornado pelo método getVersionedObjectForValue.

Conforme descrito anteriormente, o 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 getVersionedObjectForValue retornar um atributo do valor, este método geralmente atualizará o valor de atributo com um novo objeto de versão. Se getVersionedObjectForValue retornar uma cópia do valor, este método geralmente não fará nada. O OptimisticCallback padrão não faz nada, pois a implementação padrão de getVersionedObjectForValue sempre retorna o valor especial NULL_OPTIMISTIC_VERSION como o objeto de versão.

A seguir é mostrada a implementação utilizada pelo objeto EmployeeOptimisticCallbackImpl utilizado 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 a 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 seqüência original mais um, por exemplo, é o próximo valor de versão para esta instância employee. Mais uma vez, este exemplo significa que a pessoa que gravou o Loader gravou o EmployeeOptimisticCallbackImpl ou trabalhou perto da 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 desempenhar 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 executa um readObject.

Programação de Replicação

A replicação é configurada associando um MapSet a um ReplicationGroup e atributos de política de replicação. O ReplicationGroup define os membros do servidor que são utilizados para as réplicas e esperas primárias e associadas. Também define o número mínimo e máximo de réplicas requeridas para esta configuração. Os atributos de política de replicação indicam se será requerida a replicação síncrona ou assíncrona, se será permitido acesso de leitura às réplicas e se será utilizada compactação ao enviar dados de replicação para as réplicas. A replicação tem um impacto mínimo no modelo de programação. O principal impacto está nos aplicativos que pré-carregam dados para seus Mapas.

Pré-carregamento de Mapas

É possível associar um Loader a cada Mapa. É utilizado um Loader para buscar objetos quando eles não podem ser localizados no Mapa e também para gravar alterações em um backend quando uma transação é confirmada. Os Loaders também podem ser utilizados para pré-carregar dados para um mapa. O método preload da interface do Loader é chamado quando a JVM (Java Virtual Machine) se torna um primário para o grupo de replicação. O método preload não é chamado em réplicas ou esperas. O método preload tenta carregar todos os dados pretendidos referidos do backend para o Mapa utilizando a Sessão fornecida. O Mapa a ser utilizado é identificado pelo argumento de BackingMap transmitido para o método preload.

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

Pré-carregamento em um MapSet Particionado

Os Mapas podem ser particionados em *N* partições. Os Mapas podem ser armazenados em vários servidores, com cada entrada identificada por uma chave que é armazenada apenas em um destes servidores. Os Mapas muito grandes podem ficar retidos em um ObjectGrid, porque o aplicativo não está mais limitado pelo tamanho de heap de uma única JVM para reter todas as entradas de um Mapa. Os aplicativos que desejam pré-carregar com o método preload da interface do Loader devem identificar o subconjunto dos dados que devem ser pré-carregados. Sempre existe um número fixo de partições. Ele pode ser determinado utilizando o seguinte trecho de código:

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

Este trecho de código mostra como um aplicativo pode identificar o subconjunto dos dados a serem pré-carregados do banco de dados. Os aplicativos sempre devem utilizar estes métodos mesmo quando o mapa não é inicialmente particionado. Estes métodos permitem flexibilidade: se o Mapa for particionado posteriormente pelos administradores, o Loader continuará funcionando corretamente.

O aplicativo deve emitir consultas para recuperar o subconjunto myPartition do backend. Se um banco de dados estiver sendo utilizado, pode ser mais fácil ter

uma coluna com o identificador de partição para um determinado registro, a menos que exista alguma consulta natural que permita que os dados na tabela sejam particionados facilmente.

Desempenho

A implementação de pré-carregamento deve copiar dados do backend para o Mapa, armazenando vários objetos no Mapa em uma única transação. A próxima pergunta é: ″Quantos registros armazenar por transação?″ e, infelizmente, a resposta é: ″Depende.″ Quando a transação incluir blocos com mais de 100 entradas, o benefício de desempenho será reduzido. O número ideal depende de vários fatores, incluindo a complexidade e o tamanho do objeto. Comece com 100 entradas e, em seguida, aumente o número até que não sejam mais percebidos ganhos de desempenho. Transações maiores resultam em melhor desempenho de replicação. Lembre-se, apenas o primário executa o código de pré-carregamento. Os dados pré-carregados são replicados do primário para quaisquer réplicas que estão on-line.

Pré-carregando MapSets

Se o aplicativo utilizar um MapSet com vários Mapas, o Mapa terá seu próprio Loader. Cada Loader possui um método preload. Cada Mapa é carregado em série pelo ObjectGrid. Pode ser mais eficiente pré-carregar todos os Mapas, projetando um único Mapa como o Mapa de pré-carregamento. Esta é apenas uma convenção do aplicativo. Por exemplo, dois Mapas, department e employee, podem utilizar o Loader do departamento para pré-carregar os Mapas department e employee. Isto assegura que, transacionalmente, se um aplicativo desejar um departamento, os funcionários desse departamento estarão no cache. É claro, isto significa que, quando o Loader do departamento pré-carregar um departamento do backend, ele também buscará os funcionários para esse departamento. O objeto department e seus objetos employee associados devem ser incluídos no Mapa utilizando uma única transação para que isto se aplique.

Pré-carregamento Recuperável

Alguns clientes têm conjuntos de dados muito grandes que precisam ser armazenados em cache. O pré-carregamento de dados pode consumir muito tempo. Às vezes, o pré-carregamento deve ser concluído antes de o aplicativo ficar on-line. Isto pode significar que você deseja tornar o pré-carregamento recuperável. Suponha que exista um milhão de registros a serem pré-carregados. O primário está pré-carregando-os e falha no registro de número 800.000. Normalmente, a réplica escolhida para ser o novo primário limpa qualquer estado replicado e começa do início. O ObjectGrid pode ser executado melhor do que utilizando um ReplicaPreloadController. O Loader para o aplicativo também precisa implementar a interface ReplicaPreloadController. Isto inclui um único método no Loader:

```
Status checkPreloadStatus(Session session, BackingMap bmap);
```

Este método é chamado pelo tempo de execução do ObjectGrid antes de o método preload da interface do Loader ser normalmente chamado. O ObjectGrid testa o resultado deste método (Status) para determinar seu comportamento sempre que uma réplica é promovida a um primário.

Valor do Status Retornado	Comportamento do ObjectGrid em Reação
Status.PRELOADED_ALREADY	O ObjectGrid não chama o método preload porque este valor do status indica que o Mapa está totalmente pré-carregado.
Status.FULL_PRELOAD_NEEDED	O ObjectGrid limpa o Mapa e chama o método preload normalmente.
Status.PARTIAL_PRELOAD_NEEDED	O ObjectGrid deixa o Mapa como está e chama o preload. Esta estratégia permite que o Loader do aplicativo continue o pré-carregamento desse ponto em diante.

Evidentemente, enquanto um primário está pré-carregando o Mapa, ele deve deixar algum estado em um Mapa no MapSet que está sendo replicado para que a réplica possa saber qual status retornar. É possível utilizar um Mapa extra chamado, por exemplo, RecoveryMap. Este RecoveryMap deve fazer parte do mesmo MapSet que está sendo pré-carregado. Isto assegura que ele é replicado de maneira consistente com os dados que estão sendo pré-carregados.

Uma implementação sugerida aparece a seguir. Conforme o pré-carregamento confirma cada bloco de registros, ele também deve atualizar um contador/valor no RecoveryMap como parte dessa transação. Isto significa que os dados pré-carregados e os dados de RecoveryMap são replicados atomicamente para as réplicas. Quando a réplica é promovida para o primário, ela pode verificar o RecoveryMap para saber o que aconteceu. O RecoveryMap pode apenas reter uma única entrada com a chave 'state'. Se não existir nenhum objeto para esta chave, será necessário um pré-carregamento completo (checkPreloadStatus retorna FULL_PRELOAD_NEEDED). Se existir um objeto para esta chave 'state', se o valor for 'COMPLETE', o pré-carregamento será feito e o checkPreloadStatus retornará PRELOADED_ALREADY. Caso contrário, o objeto de valor indicará de onde o pré-carregamento deve ser reiniciado e o método checkPreloadStatus deve retornar PARTIAL_PRELOAD_NEEDED. O Loader pode armazenar o ponto de recuperação em uma variável de instância para o Loader para que, quando o pré-carregamento for chamado, ele saiba o ponto de partida. O RecoveryMap também pode reter uma entrada por Mapa se cada Mapa for pré-carregado de forma independente.

Manipulando a Recuperação no Modo de Replicação Síncrono com um Loader

O tempo de execução do ObjectGrid foi projetado para não perder dados confirmados quando o primário falhar. A seção a seguir mostra os algoritmos utilizados para que isso seja obtido. Estes algoritmos se aplicam apenas quando um grupo de replicação utiliza a replicação síncrona. Um Loader é opcional.

O tempo de execução do ObjectGrid pode ser configurado para replicar todas as alterações de um primário para as réplicas sincronicamente. Quando uma JVM é promovida para ser uma réplica, o primário primeiro envia uma captura instantânea do Mapa para a réplica. Quando a réplica tiver processado esta captura instantânea, o primário iniciará o envio de todas as alterações (transações concluídas) desde a geração da captura instantânea. Portanto, a réplica será capturada com o primário. Este processamento de replicação inicial é assíncrono. Quando uma réplica for capturada com o primário, o par entrará no modo de mesmo nível e, por último, começará a replicação síncrona. Deste ponto em diante,

cada transação confirmada no primário será enviada para todas as réplicas no modo de mesmo nível e o primário esperará uma mensagem de confirmação. Isto deixa o primário lento quando comparado com um cenário de replicação assíncrona, devido ao tempo de espera envolvido no recebimento de mensagens de confirmação. Uma seqüência de confirmação síncrona no primário é semelhante ao seguinte exemplo:

Etapa com o Loader	Etapa sem o Loader
Obter bloqueios para entradas	igual
Limpar alterações no Loader	NOOP
Salvar alterações no cache	igual
Enviar alterações para réplicas e esperar confirmação	igual
Confirmar para o loader por meio do Plug-in TransactionCallback	A confirmação do plug-in TransactionCallBack ainda é chamada mas, geralmente, não faz nada.
Liberar bloqueios para entradas	igual

Observe que as alterações são enviadas para a réplica antes de serem confirmadas para o Loader. Quando as alterações são confirmadas na réplica? Reveja esta seqüência:

No tempo de inicialização, inicialize as listas tx no primário.

- Set CommitedTx = {}, RolledBackTx = {}

Durante o processamento de confirmação síncrona:

Etapa com o Loader	Etapa sem o Loader
Obter bloqueios para entradas	igual
Limpar alterações no Loader	NOOP
Salvar alterações no cache	igual
Enviar alterações com uma transação confirmada e efetuar rollback da transação para a réplica e esperar confirmação	igual
Limpar lista de transações confirmadas e de transações que receberam rollback	igual
Confirmar o Loader por meio do plug-in TransactionCallBack	A confirmação do plug-in TransactionCallBack ainda é chamada mas, geralmente, não faz nada
Se a confirmação for bem-sucedida, inclua a transação nas transações confirmadas; caso contrário, inclua nas transações que receberam rollback	NOOP
Liberar bloqueios para entradas	igual

Processamento de réplicas

- Receber alterações
- Confirmar todas as transações recebidas na lista de transações confirmadas
- Efetuar rollback de todas as transações recebidas na lista de transações que receberam rollback
- Iniciar uma transação ou sessão

- Aplicar alterações à transação ou sessão
- Salvar a transação ou sessão na lista pendente
- Retornar resposta

Observe que, na réplica, não existem interações do Loader enquanto ele está no modo de réplica. O primário deve enviar todas as alterações por meio do Loader. A réplica está inativa.

Um efeito secundário deste algoritmo é que a réplica sempre tem as transações, mas elas não são confirmadas até que a próxima transação primária envie o status de confirmação destas transações. Elas são então confirmadas ou recebem rollback na réplica. Mas, até então, as transações não são confirmadas. Podemos incluir um cronômetro no primário que enviará o resultado da transação após um breve período de tempo (alguns segundos). Isto limitará qualquer staleness nessa janela de tempo, mas não o eliminará totalmente. Este staleness é um problemas apenas ao utilizar o modo de leitura de réplica. Caso contrário, ele fica invisível e não tem nenhum impacto no aplicativo.

Quando o primário falhar, provavelmente haverá algumas transações que foram confirmadas/receberam rollback no primário, mas a mensagem nunca fez isso na réplica com estes resultados. Quando uma réplica for promovida para o novo primário, uma de suas primeiras ações será manipular esta condição. Cada transação pendente é reprocessada no conjunto de Mapas do novo primário. Se houver um Loader, cada transação será fornecida ao Loader. Estas transações são aplicadas na ordem FIFO estrita. Se uma transação falhar, ela será ignorada. Se houver 3 transações pendentes, A, B e C, A pode ser confirmada, B pode receber rollback e C também pode ser confirmada. Nenhuma transação tem impacto sobre as outras. Assumimos que elas são independentes.

Um Loader talvez queira utilizar uma lógica um pouco diferente quando no modo 'recuperação de failover' versus modo 'normal'. O Loader pode saber facilmente quando está em modo de recuperação de failover, implementando a interface ReplicaPreloadController. O método checkPreloadStatus é chamado apenas quando a recuperação de failover é concluída. Portanto, se o método apply da interface do Loader for chamado antes do checkPreloadStatus, ele será uma transação de recuperação. Quando o método checkPreloadStatus for chamado, a recuperação de failover estará concluída.

Singletons com Preservação de Estado Utilizando Replicação

O WebSphere Extended Deployment incluía suporte para singletons em seu primeiro release com o recurso de particionamento. Isto permitia que os aplicativos criassem singletons em um cluster. O tempo de execução do ObjectGrid ativa um recurso semelhante utilizando MapSets replicados. Embora o padrão de singleton do ObjectGrid tenha muitas vantagens, ele também possui algumas desvantagens. O recurso de partição fornece um evento para o aplicativo quando o singleton/partição é ativada localmente, isto é comunicado utilizando o método partitionLoad do recurso de partição. Um MapSet replicado também possui um singleton, o primário. O aplicativo é notificado quando se torna o primário pelo método ReplicaPreloadController#checkPreloadStatus no Loader. Pode ser utilizado de uma maneira semelhante ao recurso de particionamento, mas tem a vantagem de ser portável em diferentes versões do WebSphere Application Server ou servidores de aplicativos competitivos.

O recurso de partição tem um evento de desativação, mas o tempo de execução do ObjectGrid não oferece este recurso. Um primário no ObjectGrid normalmente é

executado até falhar. Ele não pode ser movido. Esta é uma vantagem do recurso de particionamento sobre o ObjectGrid. A seguir está uma tabela de recursos:

Tabela 16.

Recurso	Recurso de Particionamento	Singletons do ObjectGrid
Evento de início do singleton	Sim	Sim
Evento de parada do singleton	Sim	Não
Replicação do estado do singleton	Não	Sim
QoS (Quality of Service) da variável para replicação	Não	Sim
Posicionamento de singleton flexível	Sim	Não
Pode mover o singleton no tempo de execução	Sim	Não
Roteamento de trabalho de IIOP para singleton	Sim	Não
Requer um servidor J2EE (Java 2 Platform, Enterprise Environment)	Sim	Não
Requer uma versão completa do WebSphere Extended Deployment	Sim	Não
Requer EJB (Enterprise JavaBeans)	Sim	Não
O aplicativo é portável para outros servidores de aplicativos	Não	Sim

Estado do Singleton

O recurso de particionamento não oferecia nenhum suporte interno para gerenciamento de estados. Os aplicativos eram deixados em seus próprios dispositivos se o singleton precisasse de estado. Geralmente, isto significava que o estado era enviado para o banco de dados. Se a partição tiver falhado, o servidor que foi qualificado para hospedar e recuperar a partição precisará recuperar este estado desse banco de dados. Em vez disso, se um aplicativo utilizar o ObjectGrid, o singleton poderá manter seu estado no Mapa associado ao ReplicaPreloadController que está gerenciando o singleton. Se o primário ou o singleton falhar, a réplica qualificada para ser o novo primário já terá o estado localmente devido à replicação. A replicação síncrona deve ser utilizada, a menos que a perda de dados seja aceitável para o aplicativo.

Posicionamento de singleton flexível

O recurso de particionamento utiliza o mecanismo de política do gerenciador de alta disponibilidade para determinar onde uma partição será hospedada e estas políticas podem ser alteradas no tempo de execução com efeito imediato. As políticas de grupo de replicação do ObjectGrid não são tão flexíveis quanto as do gerenciador de alta disponibilidade e não podem ser alteradas sem reiniciar todos os servidores. Você perdeu a capacidade de mover singletons no tempo de

execução se estiver utilizando o ObjectGrid.

Replicação de QoS da Variável

O recurso de partição não oferece gerenciamento de estados. O ObjectGrid oferece uma variedade de abordagens de replicação:

- Sem replicação
- Replicação Assíncrona
- Replicação Síncrona

A política de replicação do MapSet associado ao Mapa que está sendo utilizado para o estado determina a política. Replicação síncrona significa nenhuma perda de dados, mas é mais lenta. A replicação assíncrona é rápida, mas significa que uma ou mais transações confirmadas no primário podem ser perdidas se o primário falhar.

Equilíbrio de Carga entre Réplicas

O ObjectGrid, a menos que seja configurado de outra maneira, envia todos os pedidos de leitura e gravação para o servidor principal para um determinado grupo de replicação. Isto significa que só o primário deve atender todos os pedidos de clientes. Você pode permitir que pedidos de leitura sejam enviados para réplicas do primário. Isto permite que o carregamento dos pedidos de leitura seja compartilhado por várias JVMs, no entanto, o envio de pedidos de leitura para as réplicas depende da consistência.

Geralmente, isto é utilizado apenas quando clientes estão armazenando dados em cache que estão sendo alterados sempre ou quando os clientes estão utilizando bloqueio pessimista.

Se os dados estiverem sendo alterados continuamente e sendo invalidados no cliente, near caches e o primário deverão ver uma taxa relativamente alta de pedidos get de clientes como resultado. Da mesma forma, no modo de bloqueio pessimista, não existe cache local, portanto, todos os pedidos são enviados para o primário.

Se os dados forem relativamente estáticos ou o modo pessimista não for utilizado, a leitura de réplicas não terá um grande impacto no desempenho, pois a freqüência de pedidos get dos clientes com caches ativos não será alta.

No entanto, quando um cliente é iniciado pela primeira vez, seu near cache está vazio e os pedidos de cache para esse cache vazio são redirecionados para o primário. O cache cliente obtém dados com o tempo, causando a eliminação deste carregamento de pedido. Se houver um grande número de clientes e muitos deles forem iniciados simultaneamente, este carregamento poderá ser significativo e a leitura de réplicas poderá ser uma opção de desempenho apropriada.

Leituras de Réplicas e Replicação Assíncrona

Se os dados no grupo de replicação não forem alterados com freqüência, isto geralmente será um bom equilíbrio. Isto permite que pedidos get de clientes sejam direcionados para os dados em quaisquer réplicas que estejam on-line. Um pedido get pode ser enviado para uma réplica que não possui uma cópia e a chave/valor pode não ter sido replicado para a réplica nesse ponto. Se os dados não estiverem na réplica, o pedido get será redirecionado para o primário.

Se os dados forem alterados, será muito provável que gets das réplicas retornarão dados stale. Isto pode ou não ser aceitável para o aplicativo. Se não for aceitável, não ative leituras de réplicas.

Leituras de Réplicas em Modo de Replicação Síncrona

A replicação síncrona tenta manter a réplica exatamente igual ao primário. Se o primário falhar, todos os dados confirmados no primário serão assegurados como disponíveis em todas as réplicas que estavam no modo de mesmo nível quando ocorreu a falha. Embora este seja o caso quando ocorrem falhas, permitir leituras de réplicas expõe alguns efeitos secundários dos algoritmos utilizados.

Quando o primário estiver prestes a confirmar uma transação, uma cópia das alterações será enviada para a réplica e a réplica confirmará esta transação nos dois casos a seguir:

- O primário falha
- A próxima transação no primário é enviada

Quando o primário falha, todas as transações pendentes na réplica são confirmadas.

As transações pendentes são confirmadas apenas quando uma transação subseqüente é confirmada no primário. O primário transporta nesta mensagem de réplica o resultado da confirmação. Quando a réplica recebe uma destas mensagens, ela confirma ou efetua rollback das transações pendentes que tinham resultados especificados nessa mensagem.

As transações pendentes se tornam visíveis apenas para leitura em uma réplica quando são confirmadas. Obviamente, se o primário for carregado e receber modificações regulares, estas transações pendentes serão confirmadas muito rapidamente. Se o carregamento de modificação no primário for lento, haverá períodos em que as transações pendentes não serão confirmadas, até que seja feita a próxima modificação do primário.

Evidentemente, a réplica para um primário que está recebendo modificações normalmente está pelo menos uma transação atrás do primário de um ponto de vista de leitura. Não há perda de dados, estas transações estão fisicamente na réplica, elas apenas não serão confirmadas até que o resultado das transações pendentes seja enviado do primário. Esta confirmação ocorre quando a próxima transação de leitura e gravação for executada.

Resumo

Se a leitura da réplica estiver ativada, o aplicativo deverá estar preparado para tolerar alguns gets retornando dados stale. Este problema se aplica se a replicação síncrona ou assíncrona estiver sendo utilizada.

Particionamento

Utilize o particionamento quando os objetos em seu MapSet exigirem mais memória do que a disponível em uma única JVM (Java Virtual Machine) ou se a JVM não puder fornecer o rendimento do processamento requerido para atualizações.

Onde as Entradas Estão Retidas?

Um algoritmo hash determina qual servidor retém cada entrada. O administrador especifica o número de partições a serem utilizadas com a definição de PartitionSet. Esta configuração não pode ser alterada após o início das JVMs. Um valor hash simples é obtido da chave para uma entrada e o resultado deste módulo de valor (%) e o número de partições indicam qual servidor ″possui″ essa entrada.

Normalmente, é utilizado o método Java hashCode no objeto de chave. Substitua este valor substituindo a implementação de hashCode.

Às vezes, um aplicativo pode não querer modificar o valor para um hash normal, mas ainda pode querer utilizar um algoritmo hash diferente para distribuição de entrada. A interface com.ibm.websphere.objectgrid.plugins.PartitionableKey permite esta situação. Esta interface tem um método único:

```
Object ibmGetPartition();
```

Se a chave implementar esta interface, o tempo de execução do ObjectGrid utilizará o hash do objeto retornado por este método em vez do hash no objeto de chave.

Particionamento no Tempo de Execução

A interface com.ibm.websphere.objectgrid.PartitionManager fornece APIs para permitir que um aplicativo determine informações sobre o particionamento no tempo de execução. Um aplicativo pode obter uma referência a uma instância desta interface utilizando o método getPartitionManager da interface BackingMap. Uma referência ao BackingMap para um Mapa pode ser obtida utilizando o método getMap(String) da interface do ObjectGrid em qualquer instância do ObjectGrid. Ou é transmitido como um parâmetro em algum dos retornos de chamada do plug-in, como o preload(Session, BackingMap) da interface do Loader.

Métodos da Interface PartitionManager

A instância PartitionManager permite que um aplicativo determine os seguintes fatos sobre particionamento:

Nome do método	Descrição
int getNumOfPartitions()	Retorna o número de partições nas quais o Mapa está sendo dividido.
int getPartition(Object key)	Retorna o número de partições baseado em 0 utilizado para a entrada com a chave especificada.
List /*Integer*/ getPartitions(List /*Object*/ keys)	Este método é igual ao método getPartition mas opera em uma Lista de chaves. A Lista de Inteiros retornada contém o número de partições para cada chave de entrada correspondente.

Nome do método	Descrição
List /*List Integer*/ getPartitionLists(List /* Object */ keys)	Este método é igual ao método getPartitions mas retorna uma Lista ordenada de Listas de partições. Por exemplo, a primeira entrada na Lista retornada contém uma Lista de chaves de entrada correspondentes à partição 0. A próxima entrada conteria uma Lista de chaves de entrada correspondentes à partição 1 e assim por diante.
List /*LogSequence*/ partitionLogSequence(LogSequence ls)	Este método divide um LogSequence em uma lista de LogSequences para partições especificadas. O LogSequence de entrada é examinado e a partição apropriada é determinada para cada LogElement contido nela. Quando a seqüência tiver sido examinada, para cada partição que possui um LogElement, é retornado um LogSequence dos LogElements.

Limitações do Particionamento

Uma transação pode modificar entradas apenas em uma única partição por transação. Se uma transação modificar várias entradas em um MapSet e essas entradas forem divididas em diferentes partições, será efetuado rollback da transação quando for feita uma tentativa de confirmar a transação. Uma transação pode ler objetos a partir de diferentes partições. No entanto, uma transação pode modificar entradas apenas em uma única partição.

Eventos do Aplicativo quando o Primário para uma Partição É Qualificado

Se um Loader for fornecido para um Mapa e o ReplicaPreloadController também for implementado pelo Loader, o aplicativo poderá utilizar o retorno de chamada de checkPreloadStatus para receber um evento indicando que a JVM que está recebendo esta chamada de método agora é o primário para essa partição. O ID da partição pode ser identificado utilizando o método getPartitionId da interface de BackingMap. Consulte “Loaders” na página 201 para obter informações adicionais sobre pré-carregamento.

Particionamento em um Cliente Versus Execução em um Servidor

O particionamento funciona apenas quando o aplicativo está utilizando um ObjectGrid que é obtido utilizando os métodos connect da interface do ObjectGrid. Se o ObjectGrid for fornecido para o aplicativo por um retorno de chamada em um plug-in, ele será um ObjectGrid local que não faz roteamento. Se você estiver em execução em um servidor e desejar tirar vantagem dos recursos de particionamento de forma transparente, utilize um ObjectGrid que seja obtido utilizando um método connect para todas as transações. No entanto, existe uma perda de desempenho quando comparado com a utilização da referência do ObjectGrid local fornecida pela estrutura. Se você não precisar do recurso de particionamento, utilize a referência local fornecida para o plug-in quando possível.

Indexação

O recurso de indexação pode ser utilizado para construir um índice ou vários índices em um BackingMap. Um índice é construído a partir de um atributo de um objeto no BackingMap. Este recurso fornece uma maneira para os aplicativos localizarem determinados objetos mais rapidamente. Sem um índice, os aplicativos precisam localizar objetos por suas chaves. O recurso de indexação permite que os aplicativos localizem objetos com um valor específico ou dentro de um intervalo de valores. Isto é semelhante a uma Consulta EJB (Enterprise JavaBeans) que pode localizar objetos EJB consultando com um critério especificado. A indexação oferece aos aplicativos a conveniência de localizar objetos mais facilmente e um aprimoramento do desempenho no processo de procura de objetos.

Existem dois tipos de indexação: estática e dinâmica. Com a indexação *Estática*, é necessário configurar o plug-in de índice no BackingMap antes de inicializar a instância do ObjectGrid. É possível fazer esta configuração com a configuração XML ou programática do BackingMap. A indexação estática inicia a construção de um índice durante a inicialização do ObjectGrid. O índice é sempre sincronizado com o BackingMap e está pronto para utilização. Após o início do processo de indexação estática, a manutenção do índice faz parte do processo de gerenciamento de transações do ObjectGrid. Quando as transações confirmam alterações, estas alterações também atualizam o índice estático. Será efetuado rollback das alterações do índice se for efetuado rollback da transação.

A indexação dinâmica permite que um índice seja criado em um BackingMap antes ou depois da inicialização da instância do ObjectGrid de abrangência. Os aplicativos têm controle de ciclo de vida sobre o processo de indexação dinâmica. Um índice dinâmico pode ser removido quando não for mais necessário. Quando um aplicativo cria um índice dinâmico, o índice pode não estar pronto para utilização imediata devido ao tempo gasto na conclusão do processo de construção do índice. Como a quantidade de tempo depende da quantidade de dados indexados, a interface DynamicIndexCallback é fornecida para aplicativos que desejam receber notificações quando ocorrer algum evento de indexação. Estes eventos incluem ready, error e destroy. Os aplicativos podem implementar esta interface de retorno de chamada e registrar-se no processo de indexação dinâmica.

O recurso de indexação é representado pelo plug-in MapIndexPlugin, ou Index para abreviar. O MapIndexPlugin é um plug-in do BackingMap. Um BackingMap pode ter vários plug-ins Index, desde que eles sigam as regras de configuração de Index.

Se um BackingMap tiver um plug-in de índice configurado, o objeto de proxy do índice poderá ser recuperado do ObjectMap correspondente. A chamada do método getIndex no ObjectMap e a transmissão do nome do plug-in de índice retornam o objeto de proxy do índice. O objeto de proxy do índice tem que ser lançado em uma interface de índice do aplicativo apropriada, como MapIndex, MapRangeIndex ou uma interface de índice customizada.

No momento, o recurso de indexação é suportado apenas no cache local, não no cache distribuído. Se for tentada uma operação de indexação em um cache distribuído, isto resultará na exceção `UnsupportedOperationException`.

Implementação de Plug-in do Índice

A classe HashIndex no pacote com.ibm.websphere.objectgrid.plugins.index é a implementação de plug-in de índice interna que pode suportar duas interfaces de índice de aplicativo internas: MapIndex e MapRangeIndex.

Os aplicativos podem fornecer sua própria implementação de plug-in de índice para permitir que índices mais complexos sejam programados. A classe de implementação de índice precisa implementar a interface com.ibm.websphere.objectgrid.plugins.index.MapIndexPlugin. O MapIndexPlugin tem a seguinte definição:

```
/**
* Uma implementação de índice deve implementar esta interface para que as
* modificações no Mapa sejam propagadas de forma seja possível manter o
* índice conforme as transações são confirmadas. Apenas os atributos
* que implementam a interface {@link java.lang.Comparable} são
* qualificados para indexação.
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapIndex
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex
*/
public interface MapIndexPlugin
{
/**
* Este deve ser o nome do atributo a ser indexado. Se o objeto tiver
* um atributo chamado EmployeeName, o índice chamará o método
* "getEmployeeName". O nome do atributo deve ser o nome
* que está no método get e o atributo deve implementar a interface
* {@link java.lang.Comparable}.
*
* @param attributeName
* O nome do atributo a ser configurado.
*/
public void setAttributeName(String attributeName);
/**
* Este nome do índice.
*
* @return O nome do índice.
*
* @see com.ibm.websphere.objectgrid.ObjectMap#getIndex
*/
String getName();
/**
* Obtém um objeto de proxy do índice para desempenhar operações de consulta
* de índice. O responsável pela chamada deve lançar o objeto retornado para
* um objeto MapIndex ou MapRangeIndex para desempenhar as operações de
* consulta.
*
* @param map O objeto MapIndexInfo requerido para manter o índice.
* .
* @return um proxy para um objeto que implementa MapIndex ou MapRangeIndex.
*/
Object getIndexProxy(MapIndexInfo map);
/**
* É chamado pelo núcleo para permitir que o índice seja atualizado como
* resultado de alterações
* aplicadas ao mapa durante o ciclo de confirmação de uma transação.
* Utilize o método {@link LogElement#getType()} para determinar qual operação é
* requerida para atualizar o índice. Utilize {@link LogElement#getBeforeImage()}
* para obter o objeto de valor que existia antes de a transação de confirmação
* aplicar uma alteração no mapa e {@link LogElement#getAfterImage()} para obter
* o objeto de valor após a transação de confirmação ter aplicado a alteração na
* entrada do mapa.
*
* Observe que o método {@link #undoBatchUpdate(TxID, LogSequence)} pode ser chamado
* posteriormente para desfazer estas alterações se ocorrer uma exceção que faça com
* que as transações de confirmação recebam rollback.
*
* @param txid A transação para as alterações.
* @param sequence A seqüência de registro que contém alterações da transação.
*
* @throws ObjectGridRuntimeException é uma ocorrência de falha que requer que
```

```
* seja efetuado rollback da transação.
*/
void doBatchUpdate(TxID txid, LogSequence sequence) throws
ObjectGridRuntimeException;
/**
* É chamado pelo núcleo para desfazer alterações feitas no índice como
* resultado de uma chamada anterior ao método {@link #doBatchUpdate
* (TxID, LogSequence)}. Este método é chamado quando uma exceção ou
* condição de erro que requer que todas as alterações feitas pela
* transação recebam rollback. Por isso, a implementação deste método
* deve capturar tudo o que for Throwable e continuar com o próximo
* LogElement no LogSequence até que todos os LogElements sejam
* processados para que seja desfeito o máximo possível de alterações
* no índice. Um ObjectGridException deve ser emitido apenas após o
* processamento de toda a LogSequence e este método não pôde
* desfazer com êxito 1 ou mais alterações em LogSequence.
*
* Utilize o método {@link LogElement#getUndoType()} para determinar qual
* operação deve desfazer todas as alterações feitas no índice. Utilize
* {@link LogElement#getBeforeImage()} para obter o objeto de valor que existia
* antes de a transação de confirmação aplicar uma alteração no mapa e o {@link
* LogElement#getAfterImage()} para obter o objeto de valor após a transação de
* confirmação ter aplicado a alteração na entrada do mapa.
*
* @param txid A transação para as alterações.
* @param sequence A seqüência de registro que contém alterações da transação.
*
*/
void undoBatchUpdate( TxID txid, LogSequence sequence) throws
ObjectGridException;
}
```

Os métodos setAttributeName e getName são diretos e revelados por seus nomes. Os outros métodos requerem mais atenção.

Método getIndexProxy

O método getIndexProxy deve retornar um objeto de proxy do índice que implementa a interface MapIndex, a interface MapRangeIndex ou uma interface Index customizada. A implementação do objeto de proxy do índice é a parte principal do plug-in de índice.

Um objeto MapIndexInfo é transmitido para este método para fornecer informações de alteração transacionais. Estes são os dados que ficam visíveis apenas para a transação atual que chama o método getIndexProxy. O objeto de proxy do índice pode utilizar este objeto MapIndexInfo para procurar seus dados transacionais.

A seguir está a definição da interface MapIndexInfo:

```
/**
* Esta interface é utilizada para fornecer um índice com informações de
* alteração detalhadas para um Mapa específico em uma transação.
*/
public interface MapIndexInfo
{
/**
* Um índice contém os valores de chave de um conjunto de entradas do mapa
* que possuem um valor de atributo específico. Este método retorna o
* ObjectMap ao qual o índice está se referindo, o ObjectMap ao qual o
* índice está associado.
*
* @return ObjectMap ao qual este índice está associado.
*/
ObjectMap getMap();
```

```
/**
* Retorna o conjunto de todas as alterações feitas pela transação atual no
* ObjectMap retornado pelo método {@link #getMap()}.
*
* @param includeRemoved deve ser configurado como true para incluir os
tipos LogElement.DELETE
* na lista retornada por este método.
*
* @return uma Lista de LogElement criada para cada entrada ObjectMap que
foi inserida, atualizada ou removida pela transação atual.
*
* @throws ObjectGridRuntimeException
*/
List getTransactionChanges(boolean includeRemoved) throws
ObjectGridRuntimeException;
/**
* Retorna o conjunto de alterações conforme elas são aplicadas a um
* determinado conjunto  de chaves
* na transação atual para o ObjectMap retornado pelo
* método {@link #getMap()}. Se uma chave não tiver sido referida
* na transação, será retornado nulo.
*
* @param keys A lista de chaves para as quais os dados são requeridos.
* @return uma Lista de LogElement correspondente às chaves ou nulo se
* as chaves não tiverem sido referidas.
*
* @throws ObjectGridRuntimeException
*
* @see com.ibm.websphere.objectgrid.plugins.LogElement
* @see com.ibm.websphere.objectgrid.ObjectMap
*/
List getTransactionChanges(List keys) throws ObjectGridRuntimeException;
}
```

O método getIndexProxy foi projetado para suportar o método getIndex(String name) da interface ObjectMap. O objeto de proxy de índice será o retornado pelo método getIndex do ObjectMap. Por exemplo, o aplicativo chama o método getIndex do ObjectMap que, em seguida, chama este método getIndexProxy e retorna o Object retornado por este método getIndexProxy. O aplicativo tem que lançar o objeto de proxy do índice retornado para uma interface de índice do aplicativo, como MapIndex, MapRangeIndex ou outra interface de índice customizada.

O exemplo de código a seguir ilustra algumas implementações do objeto de proxy do índice que podem ser retornados pelo método getIndexProxy:

```
/**
* Uma classe utilizada para retornar um proxy para este índice do mapa
* para que os aplicativos possam desempenhar operações de consulta
* utilizando a interface MapIndex.
*/
class Proxy implements MapIndex
{
/**
* O objeto MapIndexInfo associado a este objeto de proxy do índice.
*/
protected MapIndexInfo ivMap;
/**
* Número máximo de novas tentativas quando transações simultâneas
* modificam o índice durante uma operação de consulta.
*/
protected static final int RETRY_LIMIT = 10;
/**
* Comparador EQUAL a ser utilizado.
*/
```

```
final protected ProxyEQComparator ivEQ = new ProxyEQComparator();
final protected ProxyGTComparator ivGT = new ProxyGTComparator();
final protected ProxyRangeComparator ivRange = new
ProxyRangeComparator();
/**
* Construir um objeto de proxy para um ObjectMap especificado.
*
* @param map
* é o objeto MapIndexInfo.
*/
Proxy(MapIndexInfo map)
{
ivMap = map;
}
/**
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapIndex#findAll
*/
public Iterator findAll(Object attributeValue) throws
FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Utilize o maior do que o comparador para verificação de intervalo.
ivEQ.ivAttribute = (Comparable) attributeValue;
ArrayList resultList = null;
int retryCount = 0;
boolean retry;
do
{
// Variáveis que precisam ser reinicializadas sempre por meio de loop.
retry = false;
resultList = new ArrayList();
// Utilize o índice para obter o Conjunto de chaves para entradas do
mapa que contêm o valor de atributo especificado.
Set s = (Set) index.get( attributeValue );
Set keySet = processSet( s, ivEQ );
if ( keySet != null )
{
resultList.addAll( keySet );
}
else
{
// Ops! Outra transação modificou o Conjunto obtido do índice
// enquanto o mencionado acima estava interagindo no Conjunto para
// desempenhar a
// operação addAll. Portanto, é necessário tentar novamente
// recomeçando
// com a obtenção do Conjunto do índice para coletar alterações
// da transação que acabou de modificar o conjunto.
++retryCount;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
retry = true;
}
} while (retry) ;// Retornar iterador para lista de resultados criada
//pelo loop acima.
Iterator result = resultList.iterator();
return result;
}
/**
* Processar um Conjunto obtido do índice para determinar qual das chaves
```

```
* destina-se às entradas do mapa que atendem aos critérios de seleção de
* consulta.
*
* @param s
* é o Conjunto de valores de chave para entradas no BackingMap
* sobre o qual este índice foi construído. Uma referência nula indica
* apenas
* que as alterações da transação atual precisam ser processadas.
*
* @param comparator
* é o comparador a ser utilizado para fazer a verificação do intervalo.
*
* @return Conjunto de chaves que atendem os critérios de seleção ou uma
* referência nula se ocorrer uma Exceção ao iterar sobre o Conjunto.
*
* @throws FinderException
* se uma condição de erro impedir que o processamento do Conjunto
* seja desempenhado.
*/
protected Set processSet(Set s, ProxyComparator comparator)
throws FinderException {
HashSet resultSet = new HashSet();
//...
//processar o Conjunto s, utilizar o comparador e preparar o resultSet.
//...
return resultSet;
}
} // fim da classe Proxy
/**
* Uma classe utilizada para retornar um proxy para este índice do mapa
* para que os aplicativos possam
* desempenhar operações de consulta utilizando a interface
* MapRangeIndex.
*/
class RangeProxy extends Proxy implements MapRangeIndex
{
/**
* Vários comparadores requeridos pelo proxy para desempenhar a
* verificação de intervalo do valor de atributo.
*/
final private ProxyLTComparator ivLT = new ProxyLTComparator();
final private ProxyLEComparator ivLE = new ProxyLEComparator();
final private ProxyGEComparator ivGE = new ProxyGEComparator();
/**
* Índice é um SortedMap sincronizado.
*/
final SortedMap ivIndexSortedMap;
/**
* Construir um proxy MapRangeIndex.
*/
RangeProxy(MapIndexInfo map)
{
super( map );
ivIndexSortedMap = (SortedMap) index;
}
/**
* Executar operação de consulta em um Mapa especificado e no objeto
* ProxyComparator.
*
* @param map
* é um subconjunto do índice no qual será desempenhada a operação do
* localizador.
* @param proxyComparator
* é um comparador utilizado para desempenhar a verificação de intervalo
* no valor de atributo.
*
* @return Conjunto de chaves que precisam ser retornadas pelo método
```

```
* localizador.
*
* @throws FinderException
* é qualquer falha que ocorre durante a execução da consulta.
*/
private Set executeQuery(Map map, ProxyComparator proxyComparator)
throws FinderException {
HashSet resultList = null;
int retryCount = 0;
boolean retry;
do
{
// Variáveis que precisam ser reinicializadas sempre por meio de loop.
retry = false;
resultList = new HashSet();
// Utilize o índice para obter o Conjunto de chaves para entradas do mapa
// que contêm o valor de atributo especificado.
SortedMap treeMap = (SortedMap) index;
Collection values = map.values();
if ( values.isEmpty() )
{
// No momento, não há nada no intervalo do índice, portanto, é necessário
// apenas verificar as alterações da transação atual.
Set keySet = processSet( null, proxyComparator );
if ( keySet != null )
{
resultList.addAll( keySet );
}
}
else
{
// O índice não contém algumas chaves no intervalo, portanto, é necessário
// consultar
// as duas entradas do índice, bem como as alterações na transação atual.
Iterator iter = values.iterator();
while ( iter.hasNext() )
{
Set keySet;
try
{
Set s = (Set) iter.next();
keySet = processSet( s, proxyComparator );
}
catch (ConcurrentModificationException e)
{
// Indica que não é possível obter keySet.
keySet = null;
}
if ( keySet != null )
{
resultList.addAll( keySet );
}
else
{
++retryCount;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
retry = true;
}
}
}
} while (retry) ;return resultList;
}
/*
* (não-Javadoc)
```

```
*
* @see com.ibm.websphere.objectgrid.plugins.index.
MapRangeIndex#findGreater
*/
public Iterator findGreater(Object attributeValue)
   throws FinderException {
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Utilize o maior do que o comparador para verificação de intervalo.
ivGT.ivAttribute = (Comparable) attributeValue;
SortedMap tailMap = ivIndexSortedMap.tailMap( attributeValue );
Set resultSet = executeQuery( tailMap, ivGT );
Iterator result = resultSet.iterator();
return result;
}
/*
* (não-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.
MapRangeIndex#findGreaterEqual
*/
public Iterator findGreaterEqual(Object attributeValue)
   throws FinderException {
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Utilize o maior do que o comparador para verificação de intervalo.
ivGE.ivAttribute = (Comparable) attributeValue;
SortedMap tailMap = ivIndexSortedMap.tailMap( attributeValue );
Set resultSet = executeQuery( tailMap, ivGE );
Iterator result = resultSet.iterator();
return result;
}
/*
* (não-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findLess
*/
public Iterator findLess(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Utilize o maior do que o comparador para verificação de intervalo.
ivLT.ivAttribute = (Comparable) attributeValue;
SortedMap headMap = ivIndexSortedMap.headMap( attributeValue );
Set resultSet = executeQuery( headMap, ivLT );
Iterator result = resultSet.iterator();
return result;
}
/*
* (não-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findLessEqual
*/
public Iterator findLessEqual(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
```

```
"the attributeValue must be a non null reference" );
}
// Utilize o maior do que o comparador para verificação de intervalo.
ivLE.ivAttribute = (Comparable) attributeValue;
Set resultSet;
int retryCount = 0;
boolean retry;
do
{
// reinicializar para cada nova tentativa que ocorre.
retry = false;
SortedMap headMap = ivIndexSortedMap.headMap( attributeValue );
resultSet = executeQuery( headMap, ivLE );
Set s = (Set) ivIndexSortedMap.get( attributeValue );
ivEQ.ivAttribute = (Comparable) attributeValue;
Set equalSet = processSet( s, ivEQ );
if ( equalSet != null )
{
if ( ! equalSet.isEmpty() )
{
resultSet.addAll( equalSet );
}
}
else
{
// Ops! Outra transação modificou o índice durante a execução do processSet.
// Portanto, é necessário tentar novamente a consulta inteira.
++retryCount;
retry = true;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
}
} while (retry) ;// Retornar iterador para lista de resultados criada pelo
//loop acima.
Iterator result = resultSet.iterator();
return result;
}
/*
* (não-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findRange
*/
public Iterator findRange(Object lowAttributeValue,
Object highAttributeValue)
   throws FinderException {
if ( lowAttributeValue == null )
{
throw new IllegalArgumentException(
"the lowAttributeValue must be a non null reference" );
}
if ( highAttributeValue == null )
{
throw new IllegalArgumentException(
"the highAttributeValue must be a non null reference" );
}
// Utilize o maior do que o comparador para verificação de intervalo.
ivRange.ivLowAttribute = (Comparable) lowAttributeValue;
ivRange.ivHighAttribute = (Comparable) highAttributeValue;
SortedMap subMap = ivIndexSortedMap.
subMap( lowAttributeValue, highAttributeValue );
Set resultSet = executeQuery( subMap, ivRange );
Iterator result = resultSet.iterator();
return result;
}
}
```

```
/**
* Classe-base abstrata utilizada para determinar se um valor de atributo
está no intervalo.
*/
abstract class ProxyComparator
{
abstract boolean inRange(Object attribute);
}
/**
* Desempenho menor que a verificação de intervalo.
*/
class ProxyLTComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) < 0 );
}
}
}
/**
* Desempenho menor ou igual à verificação de intervalo.
*/
class ProxyLEComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) <= 0 );
}
}
}
/**
* Desempenho igual à verificação de intervalo.
*/
class ProxyEQComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
return false;
}
else
{
return ( ivAttribute.compareTo( attribute ) == 0 );
}
}
}
/**
* Desempenho maior que a verificação de intervalo.
*/
```

```
class ProxyGTComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) > 0 );
}
}
}
/**
* Desempenho maior ou igual à verificação de intervalo.
*/
class ProxyGEComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) >= 0 );
}
}
}
/**
* Desempenha a verificação de intervalo lowAttribute <= attribute < highAttribute.
*/
class ProxyRangeComparator extends ProxyComparator
{
Comparable ivLowAttribute;
Comparable ivHighAttribute;
boolean inRange(Object o)
{
if ( o == null )
{
return false;
}
Comparable attribute = (Comparable) o;
if ( attribute.compareTo( ivLowAttribute ) < 0 )
{
return false; // attribute < ivLowAttribute
}
else
{
// ivLowAttribute <= attribute
if ( attribute.compareTo( ivHighAttribute ) < 0 )
{
return true; // ivLowAttribute <= attribute < ivHighAttribute
}
else
{
return false; // attribute >= ivHighAttribute
}
}
}
}
```

Métodos doBatchUpdate e undoBatchUpdate

Os métodos doBatchUpdate e undoBatchUpdate são métodos críticos na interface MapIndexPlugin. O método doBatchUpdate é chamado como resultado de alterações aplicadas ao mapa durante o ciclo de confirmação de uma transação. O método undoBatchUpdate é utilizado para desfazer as alterações feitas no índice como resultado de uma chamada anterior para o método doBatchUpdate. Ele é chamado quando ocorre uma exceção ou condição de erro que requer que seja feito rollback de todas as alterações feitas pela transação. Os dois métodos recebem o TxID atual e a lista de alterações para este Mapa. Eles devem iterar sobre as alterações e processá-las.

O exemplo de código a seguir mostra como implementar estes dois métodos e os métodos de suporte.

```
/**
* O mapa sincronizado utilizado como a implementação do índice em que
* o objeto do valor de atributo é a chave e o Conjunto Java é o valor.
* Um membro do Conjunto é a chave de uma entrada de BackingMap que corresponde
ao valor de atributo.
*/
Map index; //<Object attribute, Set keys>
public void doBatchUpdate(TxID txid, LogSequence sequence)
throws ObjectGridRuntimeException
{
Iterator iter = sequence.getAllChanges();
while ( iter.hasNext() )
{
LogElement elem = (LogElement) iter.next();
Object key = elem.getCacheEntry().getKey();
LogElement.Type doType = elem.getType();
if ( doType == LogElement.INSERT )
{
Object newAttribute = getAttribute( elem.getAfterImage() );
insertIntoIndex( key, newAttribute );
}
else if ( doType == LogElement.UPDATE )
{
Object newAttribute = getAttribute( elem.getAfterImage() );
Object oldAttribute = getAttribute( elem.getBeforeImage() );
updateIndex( key, oldAttribute, newAttribute );
}
else if ( doType == LogElement.DELETE )
{
Object oldAttribute = getAttribute( elem.getBeforeImage() );
removeFromIndex( key, oldAttribute );
}
else if ( doType == LogElement.EVICT )
{
Object beforeImage = elem.getBeforeImage();
if ( beforeImage != null )
{
Object oldAttribute = getAttribute( beforeImage );
removeFromIndex( key, oldAttribute );
}
}
}
}
public void undoBatchUpdate(TxID txid, LogSequence sequence)
throws ObjectGridException
{
int errors = 0;
Iterator iter = sequence.getAllChanges();
while ( iter.hasNext() )
{
```

```
try
{
LogElement elem = (LogElement) iter.next();
Object key = elem.getCacheEntry().getKey();
LogElement.Type undoType = elem.getUndoType();
if ( undoType == LogElement.INSERT )
{
Object newAttribute = getAttribute( elem.getBeforeImage() );
insertIntoIndex( key, newAttribute );
}
else if ( undoType == LogElement.UPDATE )
{
Object oldAttribute = getAttribute( elem.getAfterImage() );
Object newAttribute = getAttribute( elem.getBeforeImage() );
updateIndex( key, oldAttribute, newAttribute );
}
else if ( undoType == LogElement.DELETE )
{
Object oldAttribute = getAttribute( elem.getAfterImage() );
removeFromIndex( key, oldAttribute );
}
}
catch ( Throwable t )
{
++errors;
}
}
if ( errors > 0 )
{
throw new ObjectGridException( errors
+ " exceptions occurred during rollback of index changes.");
}
}
/**
* Extrai um atributo de um Objeto de valor especificado.
*
* @param value O Objeto de valor.
*
* @return atributo do Objeto de valor, que pode ser uma referência nula.
*
* @throws ObjectGridRuntimeException será emitida se ocorrer alguma exceção
* ao tentar extrair o valor de atributo do Objeto de valor.
*/
private Object getAttribute(Object value) throws ObjectGridRuntimeException
{
try
{
Object attribute = null;
if ( value != null )
{
Method m = getAttributeMethod( value );
attribute = getAttributeMethod.invoke( value, emptyArray );
}
return attribute;
}
catch ( InvocationTargetException e )
{
Throwable t = e.getTargetException();
throw new ObjectGridRuntimeException( "Caught unexpected Throwable", t );
}
catch ( Throwable t )
{
throw new ObjectGridRuntimeException( "Caught unexpected Throwable", t );
}
}
private void updateIndex(Object key, Object oldAttribute,
Object newAttribute)
```

```
{
// O atributo foi alterado pela atualização?
if ( newAttribute != null && oldAttribute != null &&
oldAttribute.equals( newAttribute ) )
{
// Não, então não é preciso alterar nada no índice.
return;
}
// A menos que coloquemos uma restrição ao Loader para acessar apenas
// tabelas com colunas não-anuláveis,
// teremos que lidar com a possibilidade de que o atributo seja nulo.
Set oldKeys = null;
if ( oldAttribute != null )
{
// Remover oldAttribute da entrada do índice.
oldKeys = (Set) index.get( oldAttribute );
if ( oldKeys != null )
{
oldKeys.remove( key );
if ( oldKeys.isEmpty() )
{
index.remove( oldAttribute );
}
}
}
// A menos que coloquemos uma restrição ao Loader para acessar
apenas tabelas com colunas não-anuláveis,
// teremos que lidar com a possibilidade de que o atributo seja
nulo.
Set keys = null;
if ( newAttribute != null )
{
keys = (Set) index.get( newAttribute );
// Incluir newAttribute no índice.
if ( keys == null )
{
// Como diferentes transações podem atualizar diferentes entradas
// de BackingMap
// e várias entradas do mapa podem ter o mesmo valor de atributo,
// é necessário utilizar um objeto de Conjunto sincronizado para
// assegurar que apenas uma transação por vez faça alterações no
// Conjunto.
keys = Collections.synchronizedSet( new HashSet() );
index.put( newAttribute, keys );
}
// Incluir chave desta entrada do mapa no Conjunto de chaves para
o novo valor de atributo.
keys.add( key );
}
}
private void insertIntoIndex( Object key, Object newAttribute )
{
// A menos que coloquemos uma restrição ao Loader para acessar
apenas tabelas com colunas não-anuláveis,
// teremos que lidar com a possibilidade de que o atributo seja
nulo.
if ( newAttribute != null )
{
Set keys = (Set) index.get( newAttribute );
if ( keys == null )
{
// Como diferentes transações podem atualizar diferentes entradas
do
// Mapa e várias entradas do mapa podem ter o mesmo valor de
atributo, é necessário utilizar um objeto de Conjunto
sincronizado para assegurar que apenas uma transação por
vez faça alterações no Conjunto.
```

```
keys = Collections.synchronizedSet( new HashSet() );
index.put( newAttribute, keys );
}
// Incluir chave desta entrada do mapa no Conjunto de chaves para
o novo valor de  atributo.
keys.add( key );
}
}
private void removeFromIndex(Object key, Object oldAttribute )
{
// Extrair o valor de atributo antigo
Set oldKeys = null;
// A menos que coloquemos uma restrição ao Loader para acessar
apenas tabelas com colunas não-anuláveis,
// teremos que lidar com a possibilidade de que o atributo seja
nulo.
if ( oldAttribute != null )
{
oldKeys = (Set) index.get( oldAttribute );
if ( oldKeys != null )
{
oldKeys.remove( key );
if ( oldKeys.isEmpty() )
{
index.remove( oldAttribute );
}
}
}
}
```

Interfaces de Índice do Aplicativo

As interfaces de índice do aplicativo foram projetadas para suportar métodos de consulta. No momento, existem duas interfaces de índice do aplicativo definidas: MapIndex e MapRangeIndex.

MapIndex

O MapIndex é um índice simples para consultar objetos por um valor de atributo. Ele permite que qualquer valor de atributo em um Mapa seja indexado. Permite que o aplicativo localize rapidamente todos os objetos no Mapa que possuem um valor de atributo específico. A seguir está a definição da interface MapIndex:

```
/**
* Este é um índice abstrato que pode ser criado em um Mapa vazio. O
* índice pode ser utilizado para desempenhar consultas eficientes e,
* possivelmente, outras operações, como operações relacionais em um
* atributo no Mapa.
* O MapIndex é fornecido com todos os eventos de atualização e mantém
* um índice que pode ser utilizado para emitir consultas simples no
* índice posteriormente. O índice pode utilizar um retorno de chamada
definido pelo índice para  gerar um índice em atributo compostos.
*/
public interface MapIndex
{
/**
* Retorna as Chaves para as entradas que possuem o valor de atributo
* especificado.
*
* @param attributeValue
* uma referência não nula ao valor de atributo a ser procurado.
*
* @return Uma lista das chaves para as entradas com esse atributo.
*
* @throws IllegalArgumentException se o argumento attributeValue
* for nulo.
```

```
* @throws FinderException será emitida se o limite de exceções ou
* de novas tentativas for atingido quando transações simultâneas
* que estão atualizando o índice impedirem a conclusão de findAll.
*/
Iterator findAll(Object attributeValue) throws FinderException;
}
```

MapRangeIndex

O MapRangeIndex é um índice simples para consultar objetos com um valor de atributo em um determinado intervalo. Ele permite que qualquer valor de atributo em um Mapa seja indexado. Ele se difere do MapIndex porque permite filas utilizando intervalos de valores e operações de comparação de valores. Permite que consultas localizem todos os objetos com um valor de atributo menor ou maior que um valor específico. A seguir está a definição da interface MapRangeIndex:

```
/**
* Este é um índice que permite procuras de tipos de comparação.
*/
public interface MapRangeIndex extends MapIndex
{
/**
* Localiza todas as chaves com entradas com um atributo maior que o
* valor especificado.
*
* @param attributeValue é o nó de extremidade inferior do intervalo,
* excluindo o valor de atributo inferior.
*
* @return O conjunto de chaves com valores maiores que o atributo.
*
* @throws IllegalArgumentException se o argumento attributeValue
* for nulo.
* @throws FinderException será emitida se o limite de exceções ou
* de novas tentativas for atingido quando transações simultâneas
* que estão atualizando o índice impedirem a conclusão de findAll.
*/
Iterator findGreater(Object attributeValue) throws FinderException;
/**
* Localiza todas as chaves com entradas com um atributo maior ou
* igual ao valor especificado.
*
* @param attributeValue é o nó de extremidade inferior do intervalo,
* incluindo o valor de atributo inferior.
*
* @return O conjunto de chaves com atributos que atendem os critérios
*
* @throws IllegalArgumentException se o argumento attributeValue for
* nulo.
* @throws FinderException será emitida se o limite de exceções ou de
* novas tentativas for atingido
* quando transações simultâneas que estão atualizando o índice
* impedirem a conclusão de
* findAll.
*/
Iterator findGreaterEqual(Object attributeValue) throws
FinderException;
/**
* Localiza todas as chaves com entradas com um atributo menor que o
* valor especificado.
*
* @param attributeValue é o nó de extremidade superior do intervalo,
* excluindo o * valor de nó de extremidade superior.
*
* @return O conjunto de chaves com atributos que atendem os critérios
*
* @throws IllegalArgumentException se o argumento attributeValue for
* nulo.
```

```
* @throws FinderException será emitida se o limite de exceções ou de
* novas tentativas for atingido quando transações simultâneas que
* estão atualizando o índice impedirem a conclusão de findAll.
*/
Iterator findLess(Object attributeValue) throws FinderException;
/**
* Localiza todas as chaves com entradas com um atributo menor ou igual
* ao valor especificado.
*
* @param attributeValue é o nó de extremidade superior do intervalo,
* incluindo o valor de nó de extremidade superior.
*
* @return O conjunto de chaves com atributos que atendem os critérios
*
* @throws IllegalArgumentException se o argumento attributeValue for nulo.
* @throws FinderException será emitida se o limite de exceções ou de novas
* tentativas
* for atingido
* quando transações simultâneas que estão atualizando o índice impedirem
* a conclusão de findAll.
*/
Iterator findLessEqual(Object attributeValue) throws FinderException;
/**
* Retorna todas as chaves para as entradas com o atributo inclusivamente
* no intervalo especificado de forma que lowAttributeValue <= attribute
* < highAattributeValue.
*
* @param lowAttributeValue é o nó de extremidade inferior do intervalo,*
* incluindo o valor de atributo inferior.
* @param highAttributeValue é o nó de extremidade superior do intervalo,
* excluindo o valor de atributo superior.
*
* @return A lista de chaves com entradas nesse intervalo, em ordem
* crescente.
*
* @throws IllegalArgumentException se o argumento lowAttributeValue ou
* highAttributeValue
* for nulo ou lowAttributeValue > highAttributeValue.
* @throws FinderException será emitida se um limite de exceções ou de
* novas tentativas for atingido
* quando transações simultâneas que estão atualizando o índice impedirem
* a conclusão de findAll.
*/
Iterator findRange(Object lowAttributeValue, Object highAttributeValue)
throws FinderException;
}
```

Os aplicativos precisam lançar o objeto de índice obtido do método getIndex da instância ObjectMap para a interface de índice do aplicativo desejada. Se o plug-in de índice tiver sido projetado para suportar a interface MapRangeIndex, o objeto de índice poderá ser lançado para o tipo MapRangeIndex; caso contrário, ele deve ser lançado para o tipo MapIndex.

É possível definir uma interface de índice do aplicativo customizada. Implemente o índice de aplicativo customizado como o objeto de proxy do índice que pode ser retornado pelo método getIndexProxy do MapIndexPlugin. Lance o objeto de índice obtido do método getIndex da instância ObjectMap para esta interface de índice do aplicativo customizada e utilize-o.

Incluindo Plug-ins de Índice Estático

Existem 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:

```
<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>
```

A interface de BackingMap possui dois métodos que podem ser utilizados para incluir plug-ins de índice estático: os métodos addMapIndexPlugin e setMapIndexPlugins. A seguir está a definição destes dois métodos.

```
/**
* Este método inclui um plugin de índice neste Mapa. Assumimos que a
* implementação do índice
* tenha sido construída
* com o nome do atributo a ser indexado. O nome do índice é especificado
* quando o índice é construído.
*
* Observe que, para evitar um {@link IllegalStateException}, este método
* deve ser chamado antes do método {@link ObjectGrid#initialize()}.
* Lembre-se também de que o método
* {@link ObjectGrid#getSession()} chama implicitamente o método
* {@link ObjectGrid#initialize()} se ele ainda tiver que ser chamado pelo
* aplicativo.
*
* @param index A implementação do índice.
*
* @throws IndexAlreadyDefinedException Este índice já existe.
* @throws IllegalStateException se este método for chamado após
* o método {@link ObjectGrid#initialize()}.
*/
public void addMapIndexPlugin(MapIndexPlugin index)
throws IndexAlreadyDefinedException;
/**
* Este método configura a lista de objetos MapIndexPlugin para este
* BackingMap.
* Se o BackingMap já tiver uma Lista de objetos MapIndexPlugin,
* essa lista será substituída pela Lista transmitida como
* um argumento para a chamada atual deste método.
*
* Observe que, para evitar um {@link IllegalStateException}, este método
* deve ser chamado antes do método {@link ObjectGrid#initialize()}.
* Lembre-se também de que o método
* {@link ObjectGrid#getSession()} chama implicitamente o método
* {@link ObjectGrid#initialize()} se ele ainda tiver que ser chamado pelo
* aplicativo.
*
* @param indexList Uma referência não nula para uma Lista de objetos
* {@link MapIndexPlugin}.
*
* @throws IllegalArgumentException será emitida se indexList for nula
* ou indexList contiver um objeto que não seja um
* instanceof {@link MapIndexPlugin}.
*/
public void setMapIndexPlugins(List indexList );
```

O trecho de código a seguir ilustra a abordagem de 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");
//utilize a classe HashIndex interna como a classe de plugin do índice.
HashIndex mapIndexPlugin = new HashIndex();
mapIndexPlugin.setName("CODE");
mapIndexPlugin.setAttributeName("EmployeeCode");
mapIndexPlugin.setRangeIndex(true);
personBackingMap.addMapIndexPlugin(mapIndexPlugin);
```

Utilizando Índices Estáticos

Quando um plug-in de índice estático tiver sido incluído em uma configuração de BackingMap e a instância do ObjectGrid de abrangência tiver sido inicializada, os aplicativos poderão obter o objeto de índice por nome da instância ObjectMap para o BackingMap. Lance o objeto de índice para a interface de índice do aplicativo. As operações de índice suportadas pela interface de índice do aplicativo agora podem ser executadas. A seguir está a definição do método getIndex da interface ObjectMap:

```
/**
* Retorna uma referência ao índice denominado que pode ser utilizado com este Mapa.
* Este índice não pode ser compartilhado entre encadeamentos e funciona de acordo
* com as mesmas regras da Sessão. O valor retornado deve ser lançado para a
* interface de índice correta,
* como MapIndex ou MapRangeIndex ou uma interface de índice customizada, como
* um índice geo-espacial.
*
* @param name O nome do índice
*
* @return Uma referência ao índice, ela ser lançada para a interface
* de índice apropriada.
*
* @throws IndexUndefinedException se o índice não estiver definido no BackingMap
* @throws IndexNotReadyException se o índice não estiver pronto
* @throws UnsupportedOperationException se o mapa for um mapa distribuído
*/
Object getIndex(String name)
throws IndexUndefinedException, IndexNotReadyException,
UnsupportedOperationException;
```

O trecho de código a seguir ilustra como obter e utilizar índices estáticos:

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

Incluindo e Removendo Índices Dinâmicos

Os índices dinâmicos podem ser criados e removidos de uma instância de 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. Diferente da

indexação estática, a indexação dinâmica é um processo assíncrono e precisa estar no estado pronto antes de atender sua finalidade. A maneira de obter e utilizar índices dinâmicos é igual à de índices estáticos. Se um índice dinâmico não for mais requerido, ele poderá ser removido. A interface de BackingMap possui métodos para criar e remover índices dinâmicos. A seguir está a definição destes métodos:

```
/**
* Criar um índice dinâmico no BackingMap
*
* @param name o nome do índice. O nome não pode ser nulo.
* @param isRangeIndex Indicar se deve ser criado um MapRangeIndex ou
* um MapIndex.
* Se configurado como true, o índice será do tipo MapRangeIndex.
* @param attributeName O nome do atributo a ser indexado.
* O attributeName não pode ser nulo.
* @param dynamicIndexCallback O retorno de chamada que chamará durante eventos
* de índice dinâmico.
* O dynamicIndexCallback é opcional e pode ser nulo.
*
* @throws IndexAlreadyDefinedException se já existir um MapIndexPlugin com o
* nome especificado.
* @throws UnsupportedOperationException se o mapa for um mapa distribuído.
*
*/
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb)
throws IndexAlreadyDefinedException, UnsupportedOperationException;
/**
* Criar um índice dinâmico no BackingMap.
*
* @param index A implementação do índice.
O índice não pode ser nulo.
* @param dynamicIndexCallback O retorno de chamada que chamará durante eventos
* de índice dinâmico.
* O dynamicIndexCallback é opcional e pode ser nulo.
*
* @throws IndexAlreadyDefinedException se já existir o MapIndexPlugin com o
* nome especificado.
* @throws UnsupportedOperationException se o mapa for um mapa distribuído.
*/
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback
dynamicIndexCallback)
throws IndexAlreadyDefinedException, UnsupportedOperationException;
/**
* remover um índice dinâmico do BackingMap
*
* @param name o nome do índice. O nome não pode ser nulo.
*
* @throws IndexUndefinedException se não existir um MapIndexPlugin com
* o nome especificado.
* @throws OperationNotSupportedException se o mapa for um mapa distribuído.
*/
public void removeDynamicIndex(String name) throws IndexUndefinedException;
```

O trecho de código a seguir ilustra a abordagem programática de criação, utilização e remoção de um índice dinâmico:

```
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();
```

```
//criar índice após a inicialização do ObjectGrid sem DynamicIndexCallback.
bm.createDynamicIndex("CODE", true, "employeeCode", null);
try {
//Se não estiver utilizando DynamicIndexCallback, será necessário aguardar que o
//Índice esteja  pronto.
//O tempo de espera depende do tamanho atual do mapa
Thread.sleep(3000);
} catch (Throwable t) {
//...
}
//Quando o índice estiver pronto, os aplicativos poderão tentar obter a
//instância da interface de índice do aplicativo.
//Os aplicativos precisam encontrar uma maneira de assegurar que o índice
//esteja pronto para utilização, se não estiver utilizando a interface
//DynamicIndexCallback.
//A seguir é demonstrada a maneira de esperar que o índice esteja pronto
//O tempo de espera total deve considerar o tamanho do mapa
Session session = og.getSession();
ObjectMap m = session.getMap("person");
MapRangeIndex codeIndex = null;
int counter = 0;
int maxCounter = 10;
boolean ready = false;
while(!ready && counter < maxCounter){
try {
counter++;
codeIndex = (MapRangeIndex) m.getIndex("CODE");
ready = true;
} catch (IndexNotReadyException e) {
//implica que o índice não está pronto, ...
System.out.println("Index is not ready. continue to wait.");
try {
Thread.sleep(3000);
} catch (Throwable tt) {
//...
}
} catch (Throwable t) {
//exceção inesperada
   t.printStackTrace();
}
}
if(!ready){
System.out.println("Index is not ready. Need to handle this situation.");
}
//utilizar o índice para desempenhar consultas
//Consulte a interface MapIndex ou MapRangeIndex para operações suportadas.
//O atributo do objeto no qual o índice foi criado é o EmployeeCode.
//Supondo que o atributo EmployeeCode seja do tipo Integer, que deve ser o
//tipo de dados do parâmetro transmitido para operações de índice.
Iterator iter = codeIndex.findLessEqual(new Integer(15));
//remover o índice dinâmico quando não for mais requerido
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. 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 obter a instância da interface de índice do aplicativo e utilizá-la. A seguir é mostrada a definição da Interface DynamicIndexCallback:

```
/**
* Esta é a interface de retorno de chamada para processo de indexação dinâmica.
* Se os aplicativos desejarem obter notificação do evento ready, error ou
* destroy, eles poderão implementar esta interface de retorno de chamada e
* registrar-se no processo de indexação dinâmica ao criar índice dinâmico.
*
*/
public interface DynamicIndexCallback {
/**
* Este método de retorno de chamada será chamado quando o índice dinâmico
* estiver pronto.
*
* @param indexName
* O nome do índice
*
*/
public void ready(String indexName);
/**
* Chamado quando o processo de indexação dinâmica encontra um erro inesperado.
*
* @param indexName o nome do índice
* @param t Um objeto Throwable que causa a situação de erro no processo de
* indexação dinâmica.
*/
public void error(String indexName, Throwable t);
/**
* Este método de retorno de chamada será chamado quando o índice dinâmico for
* removido
*
* @param indexName
* O nome do índice
*/
public void destroy(String indexName);
}
```

O trecho de código a seguir ilustra a utilização da 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);
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);
}
}
```

Considerações sobre Desempenho

Embora um dos principais objetivos do recurso de indexação seja aprimorar o desempenho geral do BackingMap, existem alguns fatores que precisam ser considerados antes de utilizar este recurso. Se a indexação não for utilizada corretamente, o desempenho do aplicativo poderá ficar comprometido.

- O número de transações de gravação simultâneas. O processamento de índice pode ocorrer sempre que uma transação gravar dados em um BackingMap. O desempenho será reduzido se houver muitas transações gravando dados no mapa simultaneamente quando um aplicativo tentar operações de consulta de índice.
- O tamanho do resultset retornado por uma operação de consulta. Conforme aumenta o tamanho do resultset, o desempenho da consulta é reduzido. Um teste mostrou que o desempenho é reduzido quando o tamanho do resultset é 15% ou mais que o BackingMap.
- O número de índices construídos sobre o mesmo BackingMap. Cada índice consome recursos do sistema. Conforme aumenta o número dos índices construídos sobre o BackingMap, o desempenho é reduzido.

A conclusão é que a função de indexação pode aprimorar significativamente o desempenho do BackingMap em alguns ambientes. Um caso ideal para indexação é quando o BackingMap é lido em sua maior parte, o resultset da consulta tem uma porcentagem menor das entradas do BackingMap e apenas alguns índices são construídos sobre o BackingMap.

Configuração do ObjectGrid

O ObjectGrid pode ser configurado para ser executado em um ambiente distribuído ou como um cache local disponível apenas em uma única JVM. Um ObjectGrid local pode ser configurado programaticamente ou com um arquivo XML do ObjectGrid. O arquivo XML do ObjectGrid é o local para definir ObjectGrids, BackingMaps e seus respectivos plug-ins.

Um ObjectGrid local pode ser migrado para um ambiente distribuído. Para configurar um ObjectGrid distribuído, um XML do cluster deve ser fornecido juntamente com o XML do ObjectGrid. O arquivo XML do cluster define os servidores na topologia do ObjectGrid e como os dados do ObjectGrid são particionados e replicados nos servidores. Esta seção detalha como configurar ObjectGrids locais e distribuídos.

Configuração do ObjectGrid Local

Para configurar um ObjectGrid local, consulte “Configuração do ObjectGrid Local”.

ObjectGrid Distribuído

Para configurar um ObjectGrid distribuído, consulte “Configuração do ObjectGrid Distribuído” na página 273.

Configuração do ObjectGrid Local

Um ObjectGrid local pode ser configurado programaticamente ou com XML. O ObjectGridManager é o ponto de entrada para os dois meios de configuração.

Existem vários métodos no ObjectGridManager que podem ser utilizados para criar um ObjectGrid local. Para obter uma descrição completa de cada método, consulte “Interface ObjectGridManager” na página 93.

Utilize os seguintes tópicos para configurar um ObjectGrid local:

- “Configuração do ObjectGrid Básico” discute como criar um arquivo XML muito simples com um ObjectGrid e um BackingMap definidos.
- “Configuração do ObjectGrid Completo” define cada elemento e atributo do arquivo XML e discute como obter o mesmo resultado que o arquivo XML programaticamente.
- “Configuração do ObjectGrid de Modo Misto” na página 272 descreve como utilizar uma combinação de métodos de configuração XML e programático.

Configuração do ObjectGrid Básico

Este tópico demonstra como criar um arquivo XML do ObjectGrid muito simples, o arquivo `bookstore.xml`, com um ObjectGrid e um BackingMap definidos.

As primeiras linhas do arquivo são o cabeçalho requerido para cada arquivo XML do ObjectGrid. O XML a seguir define o ObjectGrid *bookstore* com o BackingMap *book*:

bookstore.xml

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

O arquivo XML é enviado para a interface ObjectGridManager para criar uma instância do ObjectGrid baseada no arquivo. O trecho de código a seguir valida o arquivo `bookstore.xml` no esquema XML e cria o ObjectGrid *bookstore*. A instância do ObjectGrid recém-criada não é armazenada em cache.

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid =
objectGridManager.createObjectGrid("bookstore",
new URL("file:etc/test/bookstore.xml"), true, false);
```

O código a seguir executa a mesma tarefa sem XML. Utilize este código para definir programaticamente um BackingMap em um ObjectGrid. Este código cria o BackingMap book no ObjectGrid bookstoreGrid:

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("
book");
```

Configuração do ObjectGrid Completo

Este tópico é um guia completo para a configuração de um ObjectGrid. Cada elemento e atributo do arquivo XML está definido. São especificados arquivos XML de amostra, junto com o código que executa a mesma tarefa programaticamente.

O seguinte arquivo XML, `bookstore.xml`, é referido em todo este tópico. Os elementos e atributos deste arquivo são descritos detalhadamente após este exemplo.

Arquivo *bookstore.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<bean id="ObjectGridEventListener"
className="com.company.organization.MyObjectGridEventListener" />
<backingMap name="books" pluginCollectionRef="collection1" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
classname="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
<property name="maxSize" type="int" value="321" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Elemento objectGridConfig

Número de ocorrências: uma

Elementos filhos: elemento objectGrids e elemento backingMapPluginCollections

O elemento objectGridConfig é o elemento de nível superior do arquivo XML. Ele deve ser gravado no documento XML conforme mostrado no exemplo anterior. Este elemento configura o espaço de nomes do arquivo e o local do esquema. O esquema está definido no arquivo `objectGrid.xsd`. O ObjectGrid procura este arquivo no diretório raiz dos arquivos JAR (Java Archive) do ObjectGrid.

Elemento objectGrids

Número de ocorrências: uma

Elemento filho: Elemento objectGrid

O elemento objectGrids é um contêiner para todos os elementos objectGrid. No arquivo `sample1.xml`, o elemento objectGrids contém um objectGrid que tem o nome `bookstore`.

Elemento objectGrid

Número de ocorrências: um para muitos

Elementos filhos: Elemento bean e elemento backingMap

Utilize o elemento objectGrid para definir um ObjectGrid em um arquivo XML. Cada um dos atributos no elemento objectGrid corresponde a um método na interface do ObjectGrid.

```
<objectGrid
(1) name="objectGridName"
(2) securityEnabled="true|false"
(3) authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS|
AUTHORIZATION_MECHANISM_CUSTOM"
(4) permissionCheckPeriod="permission check period"
(5) txTimeout="seconds"
/>
```

Atributos:

1. Atributo **name** (requerido): Especifica o nome designado ao ObjectGrid. Se este atributo estiver ausente, a validação de XML falhará.
2. Atributo **securityEnabled** (opcional, o padrão é `false`): A configuração deste atributo como `true` ativa a segurança para o ObjectGrid. A ativação de segurança no nível do ObjectGrid significa a ativação de autorizações de acesso aos dados no mapa. Por padrão, a segurança é desativada.
3. Atributo **authorizationMechanism** (opcional, padronizado como `AUTHORIZATION_MECAHNISM_JAAS`): Configura o mecanismo de autorização para este ObjectGrid. Este atributo pode ser configurado como um de dois valores: `AUTHORIZATION_MECHANISM_JAAS` ou `AUTHORIZATION_MECHANISM_CUSTOM`. Configurado como `AUTHORIZATION_MECHANISM_CUSTOM` ao utilizar um plug-in MapAuthorization customizado. Esta configuração entrará em vigor se o atributo securityEnabled for configurado como `true`.
4. **permissionCheckPeriod** (opcional, padronizado como `0`): Especifica um valor inteiro em segundos que indica a freqüência de verificação da permissão utilizada para permitir acesso do cliente. Se o valor de atributo for `0`, cada chamada de método get, put, update, remove ou evict solicitará o mecanismo de autorização, autorização JAAS ou autorização customizada, para verificar se o subject atual tem permissão. Um valor maior que `0` indica o número de segundos para armazenar em cache um conjunto de permissões antes de retornar ao mecanismo de autorização para atualização. Esta configuração entrará em vigor se o atributo securityEnabled for configurado como `true`. Para obter detalhes adicionais, consulte “Segurança do ObjectGrid” na página 139.
5. **txTimeout** (opcional, padronizado como `0`): A quantidade de tempo, em segundos, que uma transação tem permissão para conclusão. Se uma transação não for concluída dentro desta quantidade de tempo, ela será marcada para rollback e isso resultará em uma exceção `TransactionTimeoutException`. Se o valor estiver configurado como `0`, o tempo limite das transações nunca será excedido.

O arquivo XML de exemplo a seguir, o arquivo `bookstoreObjectGridAttr.xml`, demonstra uma maneira de configurar os atributos de um objectGrid. Neste exemplo, a segurança é ativada, o mecanismo de autorização configurado como JAAS e o período de verificação de permissão configurado como 45 segundos.

Arquivo *bookstoreObjectGridAttr.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS"
permissionCheckPeriod="45">
</objectGrid>
</objectGrids>
</objectGridConfig>
```

O código a seguir demonstra a abordagem programática para obtenção da mesma configuração que o arquivo bookstoreObjectGridAttr.xml no exemplo anterior.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory
.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore", false);
bookstoreGrid.setSecurityEnabled();
bookstoreGrid.setAuthorizationMechanism(
SecurityConstants.AUTHORIZATION_MECHANISM_JAAS);
bookstoreGrid.setPermissionCheckPeriod(45);
```

Elemento backingMap

Número de ocorrências: zero para muitos

Elementos filhos: nenhum

O elemento backingMap é utilizado para definir um BackingMap em um ObjectGrid. Cada um dos atributos no elemento backingMap corresponde a um método na interface BackingMap.

```
<backingMap
(1) name="backingMapName"
(2) readOnly="true|false"
(3) pluginCollectionRef="reference to backingMapPluginCollection"
(4) numberOfBuckets="number of buckets"
(5) preloadMode="true|false"
(6) lockStrategy="OPTIMISTIC|PESSIMISTIC|NONE"
(7) numberOfLockBuckets="number of lock buckets"
(8) lockTimeout="lock timeout"
(9) copyMode="COPY_ON_READ_AND_COMMIT|COPY_ON_READ|
COPY_ON_WRITE|NO_COPY"
(10) valueInterfaceClassName="value interface class name"
(11) copyKey="true|false"
(12) nullValuesSupported="true|false"
(13) ttlEvictorType="CREATION_TIME|LAST_ACCESS_TIME|NONE"
(14) timeToLive="time to live"
/>
```

Atributos:

1. Atributo **name** (requerido): Especifica o nome designado ao BackingMap. Se este atributo estiver ausente, a validação de XML falhará.
2. Atributo **readOnly** (opcional, padronizado como `false`): A configuração deste atributo como `true` gera um BackingMap de leitura. A configuração do atributo como `false` gera um BackingMap de leitura/gravação. Se um valor não for especificado, o resultado será o padrão de leitura/gravação.
3. Atributo **pluginCollectionRef** (opcional): Especifica uma referência a um plug-in backingMapPluginCollection. O valor deste atributo deve corresponder ao atributo id de um plug-in backingMapCollection. A validação falhará se não existir nenhum id correspondente. Esta referência foi projetada para ser uma maneira fácil de reutilizar plug-ins do BackingMap.
4. Atributo **numberOfBuckets** (opcional, padronizado como `503`): O número de depósitos a serem utilizados pelo BackingMap. O BackingMap utiliza um mapa hash para sua implementação. Se existirem várias entradas no BackingMap, mais depósitos resultarão em melhor desempenho porque o risco de colisões é mais baixo, conforme aumenta o número de depósitos. Mais depósitos também resultam em maior simultaneidade.
5. Atributo **preloadMode** (opcional, padronizado como `false`): Configura o modo de pré-carregamento se um plug-in do Loader estiver configurado para este BackingMap. Se o atributo estiver configurado como `true`, o método

Loader.preloadMap(Session, BackingMap) será chamado assincronicamente. Caso contrário, ele bloqueará a execução do método durante o carregamento de dados para que o cache fique indisponível até a conclusão do pré-carregamento. O pré-carregamento ocorre durante a inicialização do ObjectGrid.

6. Atributo **lockStrategy** (opcional, padronizado como `OPTIMISTIC`): Configura o LockStrategy utilizado para o BackingMap. A estratégia de bloqueio determina se o gerenciador de bloqueios do ObjectGrid interno é utilizado sempre que uma entrada do mapa é acessada por uma transação. Este atributo pode ser configurado como um de três valores: `OPTIMISTIC`, `PESSIMISTIC` ou `NONE`.

   `OPTIMISTIC` geralmente é utilizado para um mapa que não possui um plug-in do Loader, o mapa é lido em sua maior parte e o bloqueio não é fornecido pelo gerenciador de persistência utilizando o objectGrid como um cache secundário ou pelo aplicativo. Para a estratégia de bloqueio otimista, é obtido um bloqueio exclusivo em uma entrada do mapa que está sendo inserida, atualizada ou removida no tempo de confirmação. O bloqueio assegura que as informações de versão não poderão ser alteradas por outra transação enquanto a transação que está sendo confirmada estiver desempenhando uma verificação de controle de versões otimista.

   `PESSIMISTIC` geralmente é utilizado para um mapa que não possui um plug-in do Loader e o bloqueio não é fornecido por um gerenciador de persistência utilizando o objectGrid como um cache secundário, por um plug-in do Loader ou pelo aplicativo. A estratégia de bloqueio pessimista é utilizada quando a abordagem otimista falha com muita freqüência, porque as transações de atualização colidem freqüentemente na mesma entrada do mapa. A abordagem otimista pode falhar quando o mapa não for lido na sua maior parte ou um grande número de clientes acessarem um mapa pequeno.

   `NONE` indica que a utilização do LockManager interno não é necessária, porque o controle de simultaneidade é fornecido fora do ObjectGrid, pelo gerenciador de persistência utilizando o ObjectGrid como um cache secundário, aplicativo ou pelo plug-in do Loader que utiliza bloqueios do banco de dados para controlar a simultaneidade.

7. Atributo **numberOfLockBuckets** (opcional, padronizado como `383`): Configura o número de depósitos de bloqueios utilizados pelo gerenciador de bloqueios para este BackingMap. Quando o atributo lockStrategy for configurado como `OPTIMISTIC` ou `PESSIMISTIC`, um gerenciador de bloqueios será criado 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, mais depósitos de bloqueios resultarão em melhor desempenho, porque o risco de colisões é mais baixo conforme aumenta o número de depósitos. Mais depósitos de bloqueios também resultam em maior simultaneidade. Quando o atributo lockStrategy for configurado como `NONE`, nenhum gerenciador de bloqueios será utilizado por este BackingMap. Neste caso, a configuração do atributo numberOfLockBuckets não será necessária.

8. Atributo **lockTimeout** (opcional, padronizado como `15`): Configura o tempo limite de bloqueio utilizado pelo gerenciador de bloqueios para este BackingMap. Quando o atributo lockStrategy for configurado como `OPTIMISTIC` ou `PESSIMISTIC`, um gerenciador de bloqueios será criado para o BackingMap. Para evitar que ocorram conflitos, o gerenciador de bloqueios tem um valor de tempo limite padrão para esperar que um bloqueio seja concedido. Se este tempo limite for excedido, ocorrerá uma exceção `LockTimeoutException`. O valor padrão de `15` segundos é suficiente para a maioria dos aplicativos mas, em um sistema altamente carregado, pode ocorrer um tempo limite quando não existir nenhum conflito. Neste caso, este método pode ser utilizado para

aumentar o valor de tempo limite de bloqueio do padrão para evitar que ocorram falsas exceções de tempo limite. Quando a estratégia de bloqueio for NONE, nenhum gerenciador de bloqueio será utilizado por este BackingMap. Neste caso, a configuração do atributo lockTimeout não é necessária.

9. Atributo **copyMode** (opcional, padronizado como COPY_ON_READ_AND_COMMIT): O atributo copyMode determina se uma operação get de uma entrada no BackingMap retorna o valor real, uma cópia do valor ou um proxy para o valor. O atributo copyMode pode ser configurado como um de quatro valores: COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE ou NO_COPY.

   O modo COPY_ON_READ_AND_COMMIT assegura que um aplicativo nunca tenha uma referência ao objeto de valor que está no BackingMap e, em vez disso, o aplicativo está sempre trabalhando com uma cópia do valor que está no BackingMap.

   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. Para preservar a integridade de dados do BackingMap, o aplicativo garante destruir cada referência existente para uma entrada após a confirmação da transação. Este modo resulta em um método ObjectMap.get retornando uma cópia do valor em vez de uma referência ao valor para assegurar que as alterações feitas pelo aplicativo no valor não afetarão o valor de BackingMap até que a transação seja confirmada.

   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. Em vez disso, 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 do valor.

   O modo NO_COPY permite que um aplicativo assegure que nunca modificará um objeto de valor obtido utilizando um método ObjectMap.get em troca de aprimoramentos de desempenho. Se este modo for utilizado, o valor não será copiado.

10. Atributo **valueInterfaceClassName** (opcional): Quando o atributo copyMode é configurado como COPY_ON_WRITE, um atributo valueInterfaceClassName é requerido. Ele é ignorado para todos os demais modos. A cópia durante a gravação utiliza um proxy quando são feitas chamadas de método ObjectMap.get. O proxy assegura que não seja feita uma cópia do valor, a menos que o aplicativo chame um método set na classe especificada como o atributo valueInterfaceClassName.
11. Atributo **copyKey** (opcional, padronizado como false): Este atributo determina se a chave precisa ser copiada quando é criada uma entrada do mapa. A cópia do objeto de chave permite que o aplicativo utilize o mesmo objeto de chave para cada operação de ObjectMap. A configuração como true copia o objeto de chave quando é criada uma entrada do mapa.
12. Atributo **nullValuesSupported** (opcional, padronizado como true): O suporte a valores nulos significa que um valor nulo pode ser colocado em um mapa. Se configurado como true, os valores nulos serão suportados no ObjectMap; caso contrário, os valores nulos não serão suportados. Se forem suportados valores nulos, uma operação get que retorna null pode significar que o valor é nulo ou que o mapa não contém a chave transmitida.
13. Atributo **ttlEvictorType** (opcional, padronizado como NONE): O atributo ttlEvictorType determina como o tempo de expiração de uma entrada de BackingMap é calculado. Este atributo pode ser configurado como um de três valores: CREATION_TIME, LAST_ACCESS_TIME ou NONE.

NONE indica que uma entrada não possui nenhum tempo de expiração e pode residir no BackingMap até que o aplicativo remova ou invalide explicitamente a entrada.

CREATION_TIME indica que o tempo de expiração de uma entrada é a soma da hora de criação da entrada mais o valor de atributo timeToLive.

LAST_ACCESS_TIME indica que o tempo de expiração de uma entrada é a soma da hora do último acesso da entrada mais o valor de atributo timeToLive.

14. Atributo **timeToLive** (opcional, padronizado como 0): O time to live de cada entrada do mapa, em segundos. O valor padrão de 0 significa que a entrada do mapa dura para sempre ou até que o aplicativo remova ou invalide explicitamente a entrada. Se o atributo não for 0, o evictor TTL será utilizado para retirar a entrada do mapa com base neste valor.

O arquivo XML a seguir, o arquivo bookstoreBackingMapAttr.xml, demonstra uma configuração de backingMap de amostra. Este exemplo utiliza todos os atributos opcionais, exceto o atributo pluginCollectionRef. Para obter um exemplo que mostra como utilizar o pluginCollectionRef, consulte “Elemento backingMapPluginCollection” na página 271.

Arquivo *bookstoreBackingMapAttr.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" readOnly="true" numberOfBuckets="641"
preloadMode="false" lockStrategy="OPTIMISTIC"
numberOfLockBuckets="409" lockTimeout="30" copyMode="COPY_ON_WRITE"
valueInterfaceClassName=
"com.ibm.websphere.samples.objectgrid.CounterValueInterface"
copyKey="true" nullValuesSupported="false"
ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3000" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

O código de amostra a seguir demonstra a abordagem programática para obter a mesma configuração que o arquivo bookstoreBackingMapAttr.xml no exemplo anterior:

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");bookMap.setReadOnly(true);
bookMap.setNumberOfBuckets(641);
bookMap.setPreloadMode(false);
bookMap.setLockStrategy(LockStrategy.OPTIMISTIC);
bookMap.setNumberOfLockBuckets(409);
bookMap.setLockTimeout(30);
// ao configurar o modo de cópia como COPY_ON_WRITE, é requerida uma classe
valueInterface bookMap.setCopyMode(CopyMode.COPY_ON_WRITE,
com.ibm.websphere.samples.objectgrid.CounterValueInterface.class);
bookMap.setCopyKey(true);
bookMap.setNullValuesSupported(false);
bookMap.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);
bookMap.setTimeToLive(3000); // configurar time to live como 50 minutos
```

Elemento bean

Número de ocorrências (no elemento objectGrid): zero para muitos

Número de ocorrências (no elemento backingMapPluginCollection): zero para muitos

Elemento filho: Elemento property

Utilize o elemento bean para definir plug-ins. Os plug-ins podem ser conectados a ObjectGrids e BackingMaps.

Os plug-ins do ObjectGrid:

- Plug-in TransactionCallback
- Plug-in ObjectGridEventListener
- Plug-in SubjectSource
- Plug-in MapAuthorization
- Plug-in SubjectValidation

Os plug-ins de BackingMap:

- Plug-in do Loader
- Plug-in ObjectTransformer
- Plug-in OptimisticCallback
- Plug-in Evictor
- Plug-in MapEventListener
- Plug-in MapIndex

Atributos do Elemento bean

```
<bean
(1) id="TransactionCallback|ObjectGridEventListener|SubjectSource|
MapAuthorization|SubjectValidation|Loader|ObjectTransformer|
OptimisticCallback|Evictor|MapEventListener|MapIndexPlugin"
(2) className="class name"
/>
```

1. Atributo **id** (requerido): Especifica o tipo de plug-in a ser criado. Para um bean que seja um elemento filho do elemento objectGrid, os valores válidos são os plug-ins `TransactionCallback`, `ObjectGridEventListener`, `SubjectSource`, `MapAuthorization` e `SubjectValidation`. Para um bean que é um elemento filho do elemento backingMapPluginCollection, os valores válidos são os plug-ins `Loader`, `ObjectTransformer`, `OptimisticCallback`, `Evictor` e `MapEventListener`. Cada um dos valores válidos para o atributo id representa uma interface.
2. Atributo **className** (requerido): Especifica o nome da classe a ser instanciada para criar o plug-in. A classe deve implementar a interface do tipo de plug-in.

A seguinte amostra do arquivo `bean.xml` demonstra como utilizar o elemento bean para configurar os plug-ins. Neste arquivo XML, um plug-in ObjectGridEventListener é incluído no ObjectGrid bookstore. O atributo className para este bean é a classe com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener. Esta classe implementa a interface com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener conforme necessário.

Um plug-in BackingMap também está definido na seguinte amostra de arquivo `bookstoreBean.xml`. Um plug-in evictor está incluído no BackingMap book. Como o id do bean é *Evictor*, o atributo className deve especificar uma classe que implementa a interface com.ibm.websphere.objectgrid.plugins.Evictor. A classe

com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor implementa esta interface. O backingMap faz referência aos seus plug-ins utilizando o atributo pluginCollectionRef. Consulte “Interface BackingMap” na página 112 para obter informações adicionais sobre como incluir plug-ins em um BackingMap.

Arquivo *bookstoreBean.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<bean id="ObjectGridEventListener"
className="com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener" />
<backingMap name="book" pluginCollectionRef="bookPlugins" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="bookPlugins">
<bean id="Evictor"
classnName="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

O código a seguir demonstra a abordagem programática para obtenção da mesma configuração que o arquivo `bookstoreBean.xml` anterior.

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
TranPropListener tranPropListener = new TranPropListener();
bookstoreGrid.addEventListener(tranPropListener);
BackingMap bookMap = bookstoreGrid.defineMap("book");
Evictor lruEvictor = new
com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
bookMap.setEvictor(lruEvictor);
```

Elemento property

Número de ocorrências: zero para muitos

Elemento filho: nenhum

O elemento property é utilizado para incluir propriedades em plug-ins. O nome da propriedade corresponde a um método set no atributo className do bean que contém a propriedade.

Atributos do Elemento Property

```
<property
(1) name="name"
(2) type="java.lang.String|boolean|java.lang.Boolean|int|
java.lang.Integer|double|java.lang.Double|byte|
java.lang.Byte|short|java.lang.Short|long|
java.lang.Long|float|java.lang.Float|char|
java.lang.Character"
(3) value="value"
(4) description="description"
/>
```

1. Atributo **name** (requerido): Especifica o nome da propriedade. O valor designado a este atributo deve corresponder a um método set na classe

fornecida como o atributo className no elemento bean. Por exemplo, se o atributo className do bean for configurado como com.ibm.MyPlugin e o nome da propriedade fornecida for `size`, a classe com.ibm.MyPlugin deverá ter um método setSize.

2. Atributo **type** (requerido): Especifica o tipo da propriedade. É o tipo do parâmetro transmitido para o método set identificado pelo atributo name. Os valores válidos são os primitivos Java, seus complementos `java.lang` e `java.lang.String`. Os atributos name e type devem corresponder a uma assinatura de método no atributo className do bean. Por exemplo, se o nome for `size` e o tipo for `int`, deverá existir um método `setSize(int)` na classe especificada como o atributo className para o bean.
3. Atributo **value** (requerido): Especifica o valor da propriedade. Este valor é convertido no tipo especificado pelo atributo type e, em seguida, é utilizado como um parâmetro na chamada para o método set identificado pelos atributos name e type. O valor deste atributo não é validado de nenhuma maneira. O implementador do plug-in deve verificar se o valor transmitido é válido. O implementador pode exibir uma exceção `IllegalArgumentException` no método set se o parâmetro não for válido.
4. Atributo **description** (opcional): Utilize este atributo para gravar uma descrição da propriedade.

O seguinte arquivo `bookstoreProperty.xml` demonstra como incluir um elemento property em um bean. Neste exemplo, uma propriedade com o nome `maxSize` e o tipo `int` é incluído em um Evictor. O Evictor com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor tem uma assinatura de método que corresponde ao método setMaxSize(int). Um valor inteiro de `499` é transmitido ao método setMaxSize(int) na classe com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor.

Arquivo bookstoreProperty.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" pluginCollectionRef="bookPlugins" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="bookPlugins">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
<property name="MaxSize" type="int" value="449"
description="The maximum size of the LRU Evictor" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

O código a seguir obtém a mesma configuração que o arquivo `bookstoreProperty.xml`:

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("
book");LRUEvictor lruEvictor =
new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
```

```
// se o arquivo XML tivesse sido utilizado,
// a propriedade incluída causaria a seguinte chamada
lruEvictor.setMaxSize(449);
bookMap.setEvictor(lruEvictor);
```

Elemento backingMapPluginCollections

Número de ocorrências: zero para um

Elemento filho: Elemento backingMapPluginCollection

O elemento backingMapPluginCollections é um contêiner para todos os elementos backingMapPluginCollection. No arquivo `bookstore.xml`, o elemento backingMapPluginCollections contém um elemento backingMapPluginCollection com o id `collection1`.

Elemento backingMapPluginCollection

Número de ocorrências: zero para muitos

Elemento filho: Elemento bean

O elemento backingMapPluginCollection define os plug-ins de BackingMap. Cada elemento backingMapPluginCollection é identificado por seu atributo id. Cada elemento backingMap deve fazer referência aos seus plug-ins utilizando o atributo pluginCollectionRef no elemento backingMap. Se existirem vários BackingMaps que devem ter seus plug-ins configurados de forma semelhante, cada um deles poderá fazer referência ao mesmo elemento backingMapPluginCollection.

Atributos do Elemento backingMapPluginCollection

```
<backingMapPluginCollection
(1) id="id"
/>
```

1. Atributo **id** (requerido): O identificador para o backingMapPluginCollection. Cada id deve ser exclusivo. O id é referido pelo atributo pluginCollectionRef do elemento backingMap. Se o valor de um atributo pluginCollectionRef não corresponder ao id de um elemento backingMapPluginCollection, a validação de XML falhará. Qualquer número de elementos de backingMap pode fazer referência a um único elemento backingMapPluginCollection.

O arquivo `bookstoreCollection.xml` a seguir demonstra como utilizar o elemento backingMapPluginCollection. Neste arquivo, estão definidos três elementos de backingMap. Os BackingMaps `book` e `customer` utilizam o backingMapPluginCollection `collection1`. Cada um destes dois BackingMaps possui seu próprio evictor LRUEvictor. O BackingMap `employee` faz referência ao backingMapPluginCollection `collection2`. Este BackingMap possui um evictor LFUEvictor configurado como um plug-in Evictor e a classe EmployeeOptimisticCallbackImpl configurada como um plug-in OptimisticCallback.

Arquivo bookstoreCollection.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" pluginCollectionRef="collection1" />
```

```
<backingMap name="customer" pluginCollectionRef="collection1" />
<backingMap name="employee" pluginCollectionRef="collection2" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
</backingMapPluginCollection>
<backingMapPluginCollection id="collection2">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor" />
<bean id="OptimisticCallback"
className="com.ibm.websphere.samples.objectgrid.
EmployeeOptimisticCallBackImpl" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

O código a seguir demonstra como obter programaticamente a mesma configuração que o arquivo `bookstoreCollection.xml`.

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("
book");LRUEvictor bookEvictor = new LRUEvictor();
bookMap.setEvictor(bookEvictor);
BackingMap customerMap = bookstoreGrid.defineMap("customer");
LRUEvictor customerEvictor = new LRUEvictor();
customerMap.setEvictor(customerEvictor);
BackingMap employeeMap = bookstoreGrid.defineMap("employee");
LFUEvictor employeeEvictor = new LFUEvictor();
employeeMap.setEvictor(employeeEvictor);
OptimisticCallback employeeOptCallback =
new EmployeeOptimisticCallbackImpl();
employeeMap.setOptimisticCallback(employeeOptCallback);
```

Configuração do ObjectGrid de Modo Misto

O ObjectGrid pode ser configurado utilizando uma combinação de configuração XML e de configuração programática.

Para fazer uma configuração mista, primeiro crie um arquivo XML a ser transmitido para a interface ObjectGridManager. Após a criação de um ObjectGrid com base no arquivo XML, o ObjectGrid pode ser manipulado programaticamente, desde que o método ObjectGrid.initialize() não tenha sido chamado. O método ObjectGrid.getSession() chama implicitamente o método ObjectGrid.initialize() se ele não tiver sido chamado pelo aplicativo.

Exemplo:

A seguir está uma demonstração de como obter uma configuração de modo misto. O arquivo `mixedBookstore.xml` a seguir é utilizado.

Arquivo mixedBookstore.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/ ..objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" readOnly="true" numberOfBuckets="641"
```

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

O trecho de código a seguir, que mostra o XML sendo transmitido para o ObjectGridManager, e o ObjectGrid recém-criado, é ainda mais manipulado.

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore",
new URL("file:etc/test/document/mixedBookstore.xml"), true, false);
// neste ponto, temos o ObjectGrid definido no XML
// agora, modifique o BackingMap que foi criado e configurado
BackingMap bookMap = bookstoreGrid.getMap("
book");
// o readOnly XML configurado como true
// aqui o atributo readOnly foi alterado para false
bookMap.setReadOnly(false);
// o XML não configurou nullValuesSupported, portanto,
// ele foi padronizado como true. Aqui
// o valor é configurado como false
bookMap.setNullValuesSupported(false);
// obter o Evictor que foi configurado no XML
// e configurar seu maxSize
LFUEvictor lfuEvictor = (LFUEvictor) bookMap.getEvictor();
lfuEvictor.setMaxSize(443);
bookstoreGrid.initialize();
// não é permitida configuração adicional
// neste ObjectGrid após a chamada initialize
```

Configuração do ObjectGrid Distribuído

Para criar um ObjectGrid distribuído, um arquivo XML do cluster deve ser criado e emparelhado com um arquivo XML do ObjectGrid.

Com os arquivos XML do cluster e XML do ObjectGrid, é possível iniciar um servidor do ObjectGrid.

Antes de criar um arquivo XML do cluster, crie um arquivo XML do ObjectGrid como faria para um ObjectGrid local. Para obter detalhes sobre como construir um arquivo XML do ObjectGrid, consulte “Configuração do ObjectGrid Local” na página 260. O seguinte arquivo `university.xml` é utilizado como o XML do ObjectGrid para os exemplos em “Configuração de Cluster” na página 274.

Arquivo *university.xml*

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

```
<backingMap name="equipment" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

A seguir está o arquivo XML do cluster `universityCluster.xml` que pode ser utilizado com o arquivo `university.xml` para iniciar um servidor do ObjectGrid. O arquivo `universityCluster.xml` é um arquivo XML do cluster muito básico com todos os atributos XML opcionais removidos.

Arquivo *universityCluster.xml*

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

O uso de amostra de muitos elementos e atributos XML opcionais está na seção “Configuração de Cluster”.

Configuração de Cluster

Cada elemento e atributo do XML do cluster é descrito nesta seção. Também são fornecidos exemplos que mostram como utilizar o XML do cluster com o XML do ObjectGrid para obter uma configuração. O arquivo `university.xml` é utilizado como o XML do ObjectGrid para estes exemplos.

Elemento clusterConfig

Número de ocorrências: uma

Elementos filhos: elementos cluster, objectgridBinding, partitionSet e replicationGroup

O elemento clusterConfig é o elemento de nível superior do arquivo XML do cluster. Ele deve estar na parte superior do arquivo, conforme demonstrado no arquivo `universityCluster.xml`. Este elemento configura o espaço de nomes do arquivo e

o local do esquema. O esquema está definido no arquivo `objectGridCluster.xsd`.

Elemento Cluster

Número de ocorrências: uma

Elementos filhos: elementos serverDefinition, authenticator e adminAuthorization

O elemento cluster é utilizado para definir um cluster do ObjectGrid. Cada um dos servidores no cluster está definido no elemento cluster. O elemento cluster também é utilizado para definir atributos relacionados à segurança e à rede.

```
<cluster
(1) name="clusterName"
(2) securityEnabled="true|false"
(3) statisticsEnabled="true|false"
(4) statisticsSpec="statisticsSpecification"
(5) singleSignOnEnabled="true|false"
(6) loginSessionExpirationTime="seconds"
(7) adminAuthorizationEnabled="true|false"
(8) adminAuthorizationMechanism=""
(9) clientMaxRetries="numberOfRetries"
(10) clientMaxForwards="numberOfForwards"
(11) clientStartupRetries="numberOfRetries"
(12) clientRetryInterval="seconds"
(13) tcpConnectionTimeout="seconds"
(14) tcpMinConnections="numberOfConnections"
(15) tcpMaxConnections="numberOfConnections"
(16) tcpInactivityTimeout="seconds"
(17) tcpMaxWaitTime="seconds"
(18) peerHeartbeatInterval="seconds"
(19) peerTransportBufferSize="sizeInMBs"
(20) threadPoolMinSize="minThreads"
(21) threadPoolMaxSize="maxThreads"
(22) threadPoolInactivityTimeout="seconds"
(23) managementTimeout="seconds"
(24) threadsPerClientConnect="numberOfThreads"
/>
```

Atributos:

1. Atributo **name** (requerido): Este é o nome designado ao cluster. Se este atributo estiver ausente, a validação de XML falhará.
2. Atributo **securityEnabled** (opcional, padronizado como **false**): Ativa a segurança para o cluster quando configurado como true. Se for configurado como false, a segurança de todo o cluster será desativada. Para obter mais informações, consulte “Segurança do ObjectGrid” na página 139.
3. Atributo **statisticsEnabled** (opcional, padronizado como **false**): Ativa estatísticas para o cluster quando configurado como true. Quando as estatísticas são ativadas, o atributo statisticsSpec é utilizado para configurar a especificação de estatísticas.
4. Atributo **statisticsSpec** (opcional): Especifica a cadeia que é utilizada para configurar a especificação de estatísticas. Esta cadeia determina quais estatísticas são reunidas.
5. Atributo **singleSignOnEnabled** (opcional, padronizado como false): Configurar o atributo singleSignOnEnabled como true permite que um cliente conecte-se a qualquer servidor depois de ter sido autenticado em um dos servidores. Quando este atributo é configurado como false, um cliente deve ser autenticado em cada servidor antes de ter permissão para conectar-se.

6. Atributo **loginSessionExpirationTime** (opcional): A quantidade de tempo, em segundos, antes da expiração da sessão de login. Se a sessão de login expirar, o cliente deverá ser reautenticado.
7. Atributo **adminAuthorizationEnabled** (opcional, padronizado como false): Este valor é utilizado para ativar a autorização administrativa. Se o valor for true, todas as tarefas administrativas precisarão de autorização. O mecanismo de autorização utilizado é especificado pelo valor do atributo adminAuthorizationMechanism.
8. Atributo **adminAuthorizationMechanism** (opcional, padronizado como AUTHORIZATION_MECHANISM_JAAS): Este atributo indica o mecanismo de autorização utilizado. O ObjectGrid suporta dois mecanismos de autorização: JAAS (Java Authentication and Authorization Service) e customizado. O mecanismo de autorização JAAS utiliza a abordagem baseada em política JAAS padrão. Para especificar JAAS como o mecanismo de autorização, configure o valor como AUTHORIZATION_MECHANISM_JAAS. O mecanismo de autorização customizado utiliza uma implementação de AdminAuthorization conectada pelo usuário. Para especificar um mecanismo de autorização customizado, configure o valor como AUTHORIZATION_MECHANISM_CUSTOM. Para obter informações adicionais sobre como estes dois mecanismos são utilizados, consulte “Segurança do ObjectGrid” na página 139.
9. Atributo **clientMaxRetries** (opcional, padronizado como 4): O número máximo de vezes que um pedido para um servidor pode ter nova tentativa automaticamente quando o serviço não estiver disponível.
10. Atributo **clientMaxForwards** (opcional, padronizado como 5): O número máximo de vezes que um pedido com falha é redirecionado para outro servidor.
11. Atributo **clientStartupRetries** (opcional, padronizado como 8): O número máximo de vezes que um pedido tem nova tentativa automaticamente enquanto espera a conclusão da inicialização do servidor. Como o gerenciador de alta disponibilidade requer uma quantidade de tempo significativa para ser iniciado, configure este número para ser suficientemente alto. Se o número não for grande o suficiente, os pedidos do cliente enviados antes da inicialização total do servidor falharão.
12. Atributo **clientRetryInterval** (opcional, padronizado como 10): O intervalo de tempo, em segundos, entre novas tentativas de um cliente. É utilizado para os atributos clientMaxRetries e clientStartupRetries.
13. Atributo **tcpConnectionTimeout** (opcional, padronizado como 180): O atributo tcpConnectionTimeout é o tempo limite de conexão de um soquete. O valor está em segundos.
14. Atributo **tcpMinConnections** (opcional, padronizado como 2): O número mínimo de conexões para o conjunto de conexões.
15. Atributo **tcpMaxConnections** (opcional, padronizado como 20): O número máximo de conexões para o conjunto de conexões.
16. Atributo **tcpInactivityTimeout** (opcional, padronizado como infinito): O número de segundos de inatividade em uma conexão que devem decorrer antes da conexão ser removida do conjunto de conexões.
17. Atributo **tcpMaxWaitTime** (opcional, padronizado como 120): O número máximo de segundos que um sistema espera uma conexão disponível quando todas as conexões estão sendo utilizadas e o número de conexões atingiu o valor de atributo tcpMaxConnections.

18. Atributo **peerHeartbeatInterval** (opcional, padronizado como 120): O atributo peerHeartbeatInterval é o intervalo de pulsação utilizado pelo gerenciador de alta disponibilidade. O valor está em segundos.
19. Atributo **peerTransportBufferSize** (opcional, padronizado como 10): Os atributos peerTransportBufferSize representam o tamanho do buffer de mensagem de transporte utilizado pelo gerenciador de alta disponibilidade. Este valor é especificado em megabytes.
20. Atributo **threadPoolMinSize** (opcional, padronizado como 6): Especifica o tamanho mínimo do conjunto de encadeamentos do gerenciador de alta disponibilidade.
21. Atributo **threadPoolMaxSize** (opcional, padronizado como 20): Especifica o tamanho máximo do conjunto de encadeamentos do gerenciador de alta disponibilidade.
22. Atributo **threadPoolInactivityTimeout** (opcional, padronizado como 6000): Especifica o tempo limite de inatividade do encadeamento para o conjunto de encadeamentos do gerenciador de alta disponibilidade. O valor de tempo limite está em segundos.
23. Atributo **managementTimeout** (opcional, padronizado como 30): Várias das funções de MBean do ObjectGrid enviam mensagens para servidores no cluster para reunir informações de ou desempenhar operações nos servidores. O valor de managementTimeout determina por quanto tempo o cliente tenta receber uma mensagem de volta do servidor. Se existir um problema de comunicação entre o cliente e o servidor, ou se o servidor estiver ocupado, o cliente tentará novamente durante a quantidade de tempo especificada pelo valor de managementTimeout. O valor de managementTimeout está especificado em segundos.
24. Atributo **threadsPerClientConnect** (opcional, padronizado como 5): O número de encadeamentos criados por ClientClusterContext. Cada chamada para o método connect na interface ObjectGridManager resulta em um novo ClientClusterContext.

O seguinte arquivo `universityClusterAttr.xml` é uma configuração de amostra que utiliza os diversos atributos opcionais no elemento cluster. Neste exemplo, a segurança está desativada. Os diversos atributos client, tcp, peer e thread relacionados também são alterados. O `universityClusterAttr.xml` não é uma recomendação de quais valores designar a atributos. É um exemplo de como configurar valores de atributos.

Arquivo *universityClusterAttr.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster" securityEnabled="false" statisticsEnabled="true"
statisticsSpec="map.all=enabled" singleSignOnEnabled="false"
loginSessionExpirationTime="1800" adminAuthorizationEnabled="false"
adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_JAAS" clientMaxRetries="2"
clientMaxForwards="2" clientStartupRetries="2" clientRetryInterval="5"
tcpConnectionTimeout="160" tcpMinConnections="2" tcpMaxConnections="15"
tcpInactivityTimeout="3600" tcpMaxWaitTime="160" peerHeartbeatInterval="130"
peerTransportBufferSize="15" threadPoolMinSize="8" threadPoolMaxSize="25"
threadPoolInactivityTimeout="6050" managementTimeout="60">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
```

```
</cluster>
<objectGridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectGridBinding>
<objectGridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectGridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

Elemento serverDefinition

Número de ocorrências: um para muitos

Elementos filhos: nenhum

O elemento serverDefinition é utilizado para definir um servidor do ObjectGrid. Cada servidor é executado em sua própria JVM (Java Virtual Machine) e requer duas portas.

Atributos:

```
<serverDefinition
(1) name="serverName"
(2) host="hostname"
(3) clientAccessPort="portNumber"
(4) peerAccessPort="portNumber"
(5) traceSpec="traceSpecification"
(6) systemStreamToFileEnabled="true|false"
(7) workingDirectory="logsDirectory"
/>
```

1. Atributo **name** (requerido): Este é o nome designado ao servidor. Se este atributo estiver ausente, a validação de XML falhará.
2. Atributo **host** (requerido): O nome do host da máquina na qual a JVM do servidor é executada. Cada máquina pode hospedar vários servidores do ObjectGrid. Se este atributo estiver ausente, a validação de XML falhará.
3. Atributo **clientAccessPort** (requerido): A porta no servidor utilizada para conexões do cliente. Se este atributo estiver ausente, a validação de XML falhará.
4. Atributo **peerAccessPort** (requerido): A porta no servidor utilizada para comunicação entre servidores do ObjectGrid. Se este atributo estiver ausente, a validação de XML falhará.
5. Atributo **traceSpec** (opcional, padronizado como *=all=disabled): A configuração do atributo traceSpec ativa o rastreio para o servidor utilizando a cadeia especificada.
6. Atributo **systemStreamToFileEnabled** (opcional, padronizado como **true**): Se este atributo for configurado como true, System.out, System.err e os fluxos de saída de rastreio irão para um arquivo. Quando este atributo é configurado

como false, System.out vai para o fluxo stdout e System.err vai para o fluxo stderr. Se o rastreio for ativado, a saída de rastreio irá para um arquivo, independentemente do valor do atributo systemStreamToFileEnabled.

7. Atributo **workingDirectory** (opcional): O atributo workingDirectory especifica onde os arquivos de registro serão gravados. Se um atributo workingDirectory não for especificado, os registros serão gravados no diretório atual.

O arquivo `universityClusterServerAttr.xml` demonstra a utilização dos atributos serverDefinition. Neste arquivo XML, o servidor server1 é configurado para execução no host lion.ibm.com. A porta 12501 é utilizada para acesso de cliente ao servidor e a porta 12502 é utilizada para comunicações de servidor para servidor. Como o atributo systemStreamToFileEnabled é configurado como true, System.out, System.err e rastreio irão para um arquivo no diretório especificado com o atributo workingDirectory. Neste exemplo, os arquivos estão no diretório `/objectgrid/`. Como o atributo traceSpec é configurado como ″ObjectGrid=all=enabled″, todo o rastreio relacionado ao ObjectGrid é capturado e enviado para um arquivo.

Arquivo *universityClusterServerAttr.xml*

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

Elemento objectgridBinding

Número de ocorrências: um para muitos

Elemento filho: elemento mapSet

O elemento objectgridBinding é utilizado para ligar os elementos do objectGrid no XML do ObjectGrid à topologia definida no XML do cluster. O valor designado ao atributo ref deve corresponder ao atributo name de um dos elementos do

objectGrid no XML do ObjectGrid. Um elemento objectGrid do XML do ObjectGrid pode ser referido apenas em um objectgridBinding no XML do cluster.

Atributos

```
<objectgridBinding
(1) ref="objectGridReference"
(2) minThreadPoolSize="minSize"
(3) maxThreadPoolSize="maxSize"
/>
```

1. Atributo **ref** (requerido): O atributo ref é utilizado para fazer referência a um elemento do objectGrid definido no arquivo XML do ObjectGrid. Cada elemento objectgridBinding deve fazer referência a um dos elementos do objectGrid a partir do XML do ObjectGrid. O atributo ref deve corresponder ao atributo name de um dos elementos do objectGrid no XML do ObjectGrid.
2. Atributo **minThreadPoolSize** (opcional, padronizado como 3): O atributo minThreadPoolSize é o número mínimo de encadeamentos permitidos no conjunto de encadeamentos para cada membro do grupo de replicação. O número de encadeamentos é controlado por um gerenciador de conjunto de encadeamentos, mas o número não pode ficar abaixo do valor de minThreadPoolSize. Em geral, mais encadeamentos permitem que o cliente receba uma resposta mais rápida do servidor. Mais encadeamentos também resultam em maior contenção. No entanto, máquinas mais rápidas podem manipular encadeamentos simultâneos adicionais de maneira eficiente.
3. Atributo **maxThreadPoolSize** (opcional, padronizado como 10): O atributo maxThreadPoolSize é o número máximo de encadeamentos permitidos no conjunto de encadeamentos para cada membro do grupo de replicação. O número de encadeamentos é controlado por um gerenciador de conjunto de encadeamentos, mas o número não pode ficar acima do valor de maxThreadPoolSize. Em geral, mais encadeamentos permitem que o cliente receba uma resposta mais rápida do servidor. Mais encadeamentos também resultam em maior contenção. No entanto, máquinas mais rápidas podem manipular encadeamentos simultâneos adicionais de maneira eficiente.

O arquivo `universityClusterOGBinding.xml` demonstra como utilizar o elemento objectgridBinding e seus atributos. Neste exemplo, está definido um elemento objectgridBinding. O elemento objectgridBinding está referindo-se ao ″academics″ definido no arquivo `university.xml`. Observe que, mesmo que o objectGrid ″athletics″ esteja no arquivo `university.xml`, nenhum elemento objectgridBinding está referindo-se ao ObjectGrid athletics no arquivo `universityClusterOGBinding.xml`. O ObjectGrid ″athletics″ não é armazenado em cluster porque não está incluído no arquivo `universityClusterOGBinding.xml`. Apenas o ObjectGrid ″academics″ é criado e armazenado em cluster porque está no arquivo `university.xml` e é referido no arquivo `universityClusterOGBinding.xml`.

Os atributos minThreadPoolSize e maxThreadPoolSize também são configurados neste exemplo. O valor de minThreadPoolSize é configurado como 2 e o valor maxThreadPoolSize é configurado como 11. O gerenciador do conjunto de encadeamentos em cada membro do grupo de replicação mantém o número de encadeamentos dentro destes limites para todos os mapas neste ObjectGrid.

Arquivo *universityClusterOGBinding.xml*

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

```
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
</cluster>
<objectgridBinding ref="academics" minThreadPoolSize="2" maxThreadPoolSize="11">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

Elemento mapSet

Número de ocorrências: um para muitos

Elemento filho: elemento map

O elemento mapSet é utilizado para agrupar mapas. Os mapas em um mapSet são particionados de maneira semelhante. Em um ObjectGrid distribuído, cada mapa deve pertencer a apenas um mapSet.

Atributos

```
<mapSet
(1) name="mapSetName"
(2) partitionSetRef="partitionSetReference"
(3) synchronousReplication="true|false"
(4) replicaReadEnabled="true|false"
(5) replicaDeliveryRate="deliveryRate"
(6) compression="true|false"
/>
```

1. Atributo **name** (requerido): Especifica o nome designado ao mapSet.
2. Atributo **partitionSetRef** (requerido): Cada mapSet deve estar associado a um partitionSet por meio do atributo partitionSetRef. O valor de partitionSetRef deve corresponder ao valor do atributo name de um dos elementos partitionSet. Utilizando o atributo partitionSetRef e seu partitionSet correspondente, os mapas no mapSet são particionados.
3. Atributo **synchronousReplication** (opcional, padronizado como **false**): Quando este atributo é configurado como true, a replicação entre membros do grupo de replicação ocorre sincronicamente. Quando configurado como false, a replicação ocorre assincronicamente.
4. Atributo **replicaReadEnabled** (opcional, padronizado como **false**): Se o valor de synchronousReplication for configurado como false e o valor de replicaReadEnabled for true, os clientes poderão ler dados a partir de réplicas. É feito o melhor esforço para distribuir pedidos de leitura entre o primário e suas réplicas. Se o atributo synchronousReplication for configurado como true, o atributo replicaReadEnabled será ignorado.
5. **replicaDeliveryRate** (opcional, padronizado como **1000**): O valor de replicaDeliveryRate representa o número máximo de registros por LogSequence que são entregues a cada réplica.

6. Atributo **compressReplicationEnabled** (opcional, padronizado como **true**): Quando compressReplicationEnabled é configurado como true, as mensagens de replicação são compactadas.

O arquivo `universityClusterMapSet.xml` é um pouco mais complexo do que os exemplos anteriores de arquivo XML. Neste arquivo, o ObjectGrid academics é dividido em dois conjuntos de mapas. O conjunto de mapas academicsMapSet1 contém os mapas faculty e course. Estes dois mapas são particionados de acordo com o partitionSet partitionSet1. As configurações de replicação para estes mapas também são as mesmas, porque eles estão no mesmo mapSet.

O objectgridBinding academics também contém o mapSet academicsMapSet2. Este mapSet contém apenas o mapa student. O mapa student é particionado de maneira diferente dos mapas no mapSet academicsMapSet1. O mapa student é particionado de acordo com o partitionSet studentPSet. Como o academicsMapSet2 não indicou valores explicitamente para os atributos relacionados à replicação, incluindo os atributos synchronousReplication, replicaReadEnabled, replicaDeliveryRate e compressReplicationEnabled, a ele são designados os valores padrão. Esta é outra maneira na qual o comportamento dos dois mapSets no objectgridBinding academics se difere.

O objectgridBinding athletics contém o mapSet athleticsMapSet. Assim como o mapSet academicsMapSet1 no objectgridBinding academics, ele é particionado de acordo com o partitionSet partitionSet1. Os atributos relacionados à replicação para este mapSet são configurados como os valores padrão porque eles não são explicitamente indicados.

Arquivo *universityClusterMapSet.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectgridBinding ref="academics"
<mapSet name="academicsMapSet1" partitionSetRef="partitionSet1"
synchronousReplication="true" replicaReadEnabled="true"
replicaDeliveryRate="1500" compressReplicationEnabled="true">
<map ref="faculty" />
<map ref="course" />
</mapSet>
<mapSet name="academicsMapSet2" partitionSetRef="studentPSet">
<mapRef="student" />
</mapSet>
</objectGridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetREf="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<partitionSet name="studentPSet">
<partition name="studentPartition1" replicationGroupRef="replicationGroup1" />
```

```
<partition name="studentPartition2" replicationGroupRef="replicationGroup2" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
<replicationGroup name="replicationGroup2" minReplicas="1" maxReplicas="2">
<replicationGroupMember serverRef="server1" priority="2" />
<replicationGroupMember serverRef="server2" priority="1" />
</replicationGroup>
</clusterConfig>
```

Elemento map

Número de ocorrências: um para muitos

Elementos filhos: nenhum

Cada mapa em um mapSet faz referência a um dos elementos backingMap que está definido no arquivo XML ObjectGrid. Ao definir um ObjectGrid distribuído, cada backingMap no elemento do objectGrid no XML do ObjectGrid deve ser referido por um mapa no XML do cluster. Cada mapa em um ObjectGrid distribuído deve pertencer a apenas um mapSet.

Atributos

```
<map
(1) ref="backingMapReference"
/>
```

1. Atributo **ref** (requerido): Uma referência a um backingMap no XML do ObjectGrid. Cada mapa em um mapSet deve fazer referência a um backingMap a partir do arquivo XML do ObjectGrid. O valor designado a ref deve corresponder ao atributo name de um dos elementos de backingMap no XML do ObjectGrid.

Consulte o arquivo `universityClusterMapSet.xml` para obter uso de amostra do elemento map. Cada backingMap do objectGrid academics no `university.xml` é referido por um mapa em apenas um mapSet no `universityClusterMapSet.xml`. O mesmo se aplica ao objectGrid athletics. Uma exceção `ObjectGridException` será gerada se um objectgridBinding fizer referência a um objectGrid a partir do XML do ObjectGrid, mas não incluir todos os seus mapas em um mapSet.

Elemento partitionSet

Número de ocorrências: um para muitos

Elemento(s) filho(s): partition

O elemento partitionSet é utilizado para definir partições para um mapSet. Cada mapa no mapSet é particionado pelas partições de um partitionSet. Um mapSet está associado a um partitionSet com o atributo partitionSetRef no elemento mapSet. Quando apenas uma partição estiver definida em um partitionSet, os dados contidos nos mapas de um mapSet associado não serão particionados.

Atributos

```
<partition-set
(1) name="partitionSetName"
/>
```

1. Atributo **name** (requerido): Este atributo é utilizado para designar um nome a um partitionSet. O nome do partitionSet é referido pelo atributo partitionSetRef do mapSet.

Consulte o arquivo `universityClusterMapSet.xml` para obter uso de amostra do partitionSet. No `universityClusterMapSet.xml`, estão definidos dois partitionSets: partitionSet1 e studentPSet. O partitionSet partitionSet1 possui apenas uma partição definida. Como apenas uma partição está definida, qualquer mapSet que utiliza o partitionSet partitionSet1 não tem seus dados particionados. Existem dois mapSets no arquivo `universityClusterMapSet.xml` que são particionados de acordo com o partitionSet partitionSet1. Por meio do partitionSetRef no elemento mapSet, os mapSets academicsMapSet1 e athleticsMapSet estão limitados ao partitionSet partitionSet1.

O partitionSet studentPSet contém duas partições. Qualquer mapSet que utiliza este partitionSet tem seus dados do mapa divididos nas duas partições. No arquivo `universityClusterMapSet.xml`, o mapSet academicsMapSet2 utiliza o partitionSet studentPSet.

Elemento partition

Número de ocorrências: um para muitos

Elementos filhos: nenhum

O elemento partition é utilizado para definir partições em um partitionSet. As partições são utilizadas para dividir os dados nos mapas de um mapSet.

Atributos

```
<partition
(1) name="partitionName"
(2) replicationGroupRef="replicationGroupReference"
/>
```

1. Atributo **name** (requerido): O atributo name é utilizado para designar um nome a uma partição. Um nome de partição deve ser exclusivo em seu partitionSet.
2. Atributo **replicationGroupRef** (requerido): O atributo replicationGroupRef é utilizado para associar um replicationGroup a uma partição. O replicationGroupRef deve corresponder ao atributo name de um dos elementos replicationGroup.

Consulte o arquivo `universityClusterMapSet.xml` para obter uso de amostra do elemento partition. No arquivo `universityClusterMapSet.xml`, estão definidas várias partições. O partitionSet partitionSet1 possui uma partição denominada partition1. O partitionSet studentPSet contém duas partições: studentPartition1 e studentPartition2.

Cada partição está associada a um replicationGroup por meio do atributo replicationGroupRef. No arquivo `universityClusterMapSet.xml`, a partição studentPartition1 é replicada no replicationGroup replicationGroup1. A partição studentPartition2 é replicada no replicationGroup replicationGroup2.

Elemento replicationGroup

Número de ocorrências: um para muitos

Elemento filho: Elemento replicationGroupMember

Um replicationGroup é utilizado para definir como um mapa ou suas partições são replicadas. As partições de um mapa são replicadas entre os membros do grupo de replicação em um replicationGroup.

Atributos

```
<replicationGroup
(1) name="replicationGroupName"
(2) minReplicas="minNumberOfReplicas"
(3) maxReplicas="maxNumberOfReplicas"
```

1. Atributo **name** (requerido): O atributo name é utilizado para designar um nome a um replicationGroup.
2. Atributo **minReplicas** (opcional, padronizado como **0** se apenas um replicationGroupMember estiver no replicationGroup; padronizado como **1** se mais de um replicationGroupMember estiver no replicationGroup): O atributo minReplicas é utilizado para indicar quantos replicationGroupMembers devem estar disponíveis antes da permissão do acesso de gravação a dados do mapa neste replicationGroup. Se o número de réplicas disponíveis ficar abaixo do número de minReplicas especificado, será permitido apenas acesso de leitura aos mapas. Se minReplicas for configurado como 0, o acesso de gravação ainda será permitido no primário, mesmo que todas as réplicas estejam indisponíveis.

   Para ativar a replicação, pelo menos dois membros do grupo de replicação devem estar disponíveis e o atributo minReplicas deve ser pelo menos 1. É importante observar como um replicationGroup se comporta durante o estágio de ″ativação″ de um cluster do ObjectGrid. Se desejar que os dados do mapa fiquem disponíveis após iniciar apenas um servidor, defina um replicationGroup com apenas um replicationGroupMember. Em um replicationGroup com apenas um replicationGroupMember, os dados não são replicados.

   A seguir estão algumas regras para configurar o valor de minReplicas.

   ```
   minReplicas >= 0
   minReplicas <= maxReplicas
   minReplicas <= # of members in the replicationGroup -1
   ```

3. Atributo **maxReplicas** (opcional, padronizado como **0** se apenas um replicationGroupMember estiver no replicationGroup; padronizado como **1** se mais de um replicationGroupMember estiver no replicationGroup): O atributo maxReplicas representa o número máximo de réplicas que são ativadas no replicationGroup. Em um replicationGroup, a replicação ocorre entre o número de maxReplicas especificado se muitos membros estiverem disponíveis. Se maxReplicas for menor que o número de membros do grupo de replicação no grupo, os membros extras serão esperas; ou seja, eles ficarão inativos até que uma das réplicas se torne indisponível.

   A seguir estão algumas regras para configurar o valor de maxReplicas.

   ```
   maxReplicas >= 0
   maxReplicas >= minReplicas
   ```

Consulte o arquivo `universityClusterMapSet.xml` para obter uso de amostra do elemento partition. No `universityClusterMapSet.xml`l, estão definidos dois replicationGroups: replicationGroup1 e replicationGroup2. O replicationGroup replicationGroup1 contém apenas um replicationGroupMember. As partições que estão limitadas a este replicationGroup não serão replicadas porque é requerido mais de um replicationGroupMember para replicação.

O replicationGroup replicationGroup2 contém dois replicationGroupMembers. A partição studentPartition2 do partitionSet studentPSet está utilizando este replicationGroup. Portanto, a partição studentPartition2 é replicada nos dois

replicationGroupMembers. O replicationGroup replicationGroup2 também possui seus atributos minReplicas e maxReplicas configurados. Como minReplicas está configurado como 1, os dados do mapa hospedados neste replicationGroup serão de leitura até que um primário e pelo menos uma réplica se tornem disponíveis. O valor de maxReplicas de 1 indica que o primário deste replicationGroup replica seus dados para, no máximo, uma réplica. No caso do replicationGroup replicationGroup2, não é possível exceder uma réplica, porque o grupo contém apenas dois membros. Um membro é o primário e o outro é uma réplica.

Elemento replicationGroupMember

Número de ocorrências: um para muitos

Elementos filhos: nenhum

Um elemento replicationGroupMember é utilizado para referir-se a uma definição de servidor. Cada elemento replicationGroupMember também possui uma prioridade associada. Esta prioridade é utilizada para determinar qual replicationGroupMember é o servidor principal e quais membros são réplicas.

Atributos

```
<replicationGroupMember
(1) serverRef="serverDefinitionReference"
(2) priority="priority"
/>
```

1. Atributo **serverRef** (requerido): O atributo serverRef é utilizado para associar uma definição de servidor a um elemento replicationGroupMember. O atributo serverRef associada cada replicationGroupMember a um servidor específico.
2. Elemento **priority** (requerido): O atributo priority é utilizado para determinar quais membros do grupo de replicação serão o primário. Os valores de priority variam de `1` ao número de membros do grupo de replicação, com `1` sendo a prioridade mais alta. O ObjectGrid faz um melhor esforço para considerar a prioridade para cada membro do grupo de replicação. O elemento replicationGroupMember com uma prioridade `1` é o primário, a menos que as circunstâncias o impeçam. Se todos os servidores e seus replicationGroupMembers se tornarem disponíveis quase ao mesmo tempo, as configurações de prioridade serão consideradas. No entanto, se um replicationGroupMember com uma prioridade de `2` estiver disponível muito tempo antes de qualquer outro repliationGroupMember, ele se tornará o primário.

   Se o primário for iniciado com êxito e falhar após um período de tempo, um novo primário deverá ser selecionado. O elemento replicationGroupMember com a próxima prioridade mais alta provavelmente se tornará o novo primário. No entanto, uma réplica diferente pode ser selecionada como o novo primário se a réplica com a próxima prioridade mais alta for determinada para ficar atrás em sua replicação.

Consulte o arquivo `universityClusterMapSet.xml` para obter uso de amostra do elemento partition. No `universityClusterMapSet.xml`, o replicationGroup1 replicationGroup possui apenas um replicationGroupMember. Devido a esta definição, este replicationGroup possui apenas um primário. Não existem réplicas neste grupo. O replicationGroupMember definido está ativo no servidor server1 como o valor de serverRef indica.

O replicationGroup replicationGroup2 tem mais de um replicationGroupMember. O primeiro replicationGroupMember listado é ativado no servidor server1. Este

membro tem uma prioridade 2. O segundo replicationGroupMember listado é ativado no servidor server2. Como o segundo membro tem uma prioridade 1, ele será o primário deste replicationGroup se os membros do grupo se tornarem disponíveis quase ao mesmo tempo. O primeiro replicationGroupMember listado serve como uma réplica, porque tem uma prioridade 2.

Também é importante entender como o valor de minReplicas afeta o replicationGroup replicationGroup2. Considere o cenário no qual os servidores server1 e server2 estão em execução. Neste caso, os dois replicationGroupMembers estão disponíveis. Portanto, os valores de minReplicas e maxReplicas são atendidos e os dados são replicados entre o primário e a réplica deste grupo. Se o server1 se tornar indisponível, um dos replicationGroupMembers se tornará indisponível. Neste caso, os dados no replicationGroup replicationGroup2 se tornarão de leitura, porque o valor de minReplicas não será mais atendido.

Elemento authenticator

Número de ocorrências: zero para um

Elemento filho: Elemento property

Um elemento authenticator é utilizado para autenticar clientes nos servidores do ObjectGrid no cluster. A classe especificada pelo atributo className deve implementar a interface com.ibm.websphere.objectgrid.security.plugins.Authenticator. O authenticator pode utilizar propriedades para chamar métodos na classe especificada pelo atributo className. Consulte “Elemento property” na página 289 para obter informações adicionais sobre como utilizar propriedades.

Atributos

```
<authenticator
(1) className="authenticatorClassName"
/>
```

1. Atributo **className** (requerido): O atributo className é utilizado para especificar uma classe que implementa a interface com.ibm.websphere.objectgrid.security.plugins.Authenticator. Esta classe é utilizada para autenticar clientes em servidores no cluster do ObjectGrid.

O arquivo `universityClusterSecurity.xml` a seguir demonstra como utilizar o elemento authenticator. Neste exemplo, a classe com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator é especificada como o authenticator. Esta classe implementa a interface com.ibm.websphere.objectgrid.security.plugins.Authenticator.

Arquivo *universityClusterSecurity.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster" securityEnabled="true"
singleSignOnEnabled="true"
loginSessionExpirationTime="1800" adminAuthorizationEnabled="true"
adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
<serverDefinition name="server1" host="lion.ibm.com"
clientAccessPort="12501" peerAccessPort="12502" />
<authenticator
```

```
className ="com.ibm.websphere.objectgrid.security.plugins.builtins.
KeyStoreLoginAuthenticator" />
<adminAuthorization className= "com.ibm.MyAdminAuthorization">
<property name="interval" type="int" value="60" description="Set the
interval to 60 seconds" />
</adminAuthorization>
</cluster>
<objectgridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

Elemento adminAuthorization

Número de ocorrências: zero para um

Elemento filho: Elemento property

Um elemento adminAuthorization é utilizado para configurar acesso administrativo ao cluster do ObjectGrid. As tarefas administrativas podem ser desempenhadas após o fornecimento do acesso à administração.

Atributos

```
<adminAuthorization
(1) className="adminAuthClassName"
/>
```

1. Atributo **className** (requerido): O atributo className é utilizado para especificar uma classe que implementa a interface com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization.

Consulte o arquivo `universityClusterSecurity.xml` na seção do authenticator para obter uso de amostra do elemento adminAuthorization. No universityClusterSecurity.xml, é utilizado um adminAuthorization customizado. A classe com.ibm.MyAdminAuthorization é utilizada como a classe adminAuthorization. Para utilizar um adminAuthorization customizado, o atributo securityEnabled deve ser `true`, adminAuthorizationMechanism deve ser configurado como `AUTHORIZATION_MECHANISM_CUSTOM` e um elemento adminAuthorization deve ser fornecido. Este elemento adminAuthorization também utiliza uma propriedade. Para obter informações adicionais sobre como utilizar propriedades, consulte “Elemento property” na página 289.

Elemento property

Número de ocorrências: zero para muitos

Elementos filhos: nenhum

O elemento property é utilizado para chamar métodos set no authenticator e no adminAuthorization. O nome da propriedade corresponde a um método set no className do elemento authenticator ou adminAuthorization que contém a propriedade.

Atributos

```
<property
(1) name="propertyName"
(2) type="java.lang.String|boolean|java.lang.Boolean|int|
java.lang.Integer|double|java.lang.Double|byte|
java.lang.Byte|short|java.lang.Short|long|
java.lang.Long|float|java.lang.Float|char|
java.lang.Character"
(3) value="propertyValue"
(4) description="description"
/>
```

1. Atributo **name** (requerido): O nome da propriedade. O valor designado a este atributo deve corresponder a um método set na classe fornecida como o className para o authenticator ou adminAuthorization. Por exemplo, se o className do authenticator for configurado como com.ibm.MyAuthenticator e o nome da propriedade fornecida for `interval`, a classe com.ibm.MyAuthenticator deverá ter um método setInterval.
2. Atributo **type** (requerido): O tipo da propriedade. É o tipo do parâmetro transmitido para o método set identificado pelo atributo name. Os valores válidos são os primitivos Java, seus complementos java.lang e java.lang.String. O nome e tipo devem corresponder a uma assinatura de método do className do bean. Por exemplo, se o nome for interval e o tipo for int, deverá existir um método setInterval(int) na classe que seja especificado como o className para o authenticator ou adminAuthorization.
3. Atributo **value** (requerido): O valor da propriedade. Este valor é convertido no tipo especificado pelo atributo type e, em seguida, utilizado como um parâmetro na chamada para o método set identificado pelos atributos name e type. É importante saber que o valor deste atributo não deve ser validado em hipótese alguma. O implementador do plug-in deve verificar se o valor transmitido é válido. O implementador pode exibir uma exceção `IllegalArgumentException` no método set se o parâmetro não for válido.
4. Atributo **description** (opcional): Utilize este atributo para gravar uma descrição da propriedade.

Consulte o arquivo `universityClusterSecurity.xml` para obter uso de amostra do elemento adminAuthorization. O elemento property pode ser utilizado nos elementos authenticator ou adminAuthorization no XML do cluster. No arquivo `universityClusterSecurity.xml`, a propriedade é utilizada para chamar um método set no adminAuthorization. Neste caso, um método setInterval é chamado na classe com.ibm.MyAdminAuthorization. Ele recebe um valor inteiro de `60`.

Capítulo 10. Integrando o ObjectGrid com o WebSphere Application Server

Utilize o ObjectGrid com os recursos fornecidos com o WebSphere Application Server para aprimorar seus aplicativos com o recurso do ObjectGrid.

Instale o WebSphere Application Server e o WebSphere Extended Deployment. Quando o WebSphere Extended Deployment for instalado, será possível incluir funções do ObjectGrid nos aplicativos J2EE (Java 2 Platform, Enterprise Edition).

A API do ObjectGrid pode ser utilizada em um aplicativo J2EE direcionado para o WebSphere Application Server. O arquivo `wsobjectgrid.jar` está no diretório `\base\lib` após a instalação do WebSphere Extended Deployment. Além de integrar a API do ObjectGrid com o modelo de programação de aplicativo J2EE, é possível alavancar o suporte à propagação de transação distribuída. Com este suporte, é possível configurar instâncias do ObjectGrid para coordenar resultados de confirmação de transação em um cluster do WebSphere Application Server.

1. Desempenhe as etapas de programação básicas para ativar um aplicativo J2EE com o ObjectGrid. Consulte “Integrando o ObjectGrid em um Ambiente Java 2 Platform, Enterprise Edition” para obter maiores informações.
2. Monitore dados de desempenho para seus aplicativos do ObjectGrid. Consulte “Monitorando o Desempenho do ObjectGrid com a PMI (Performance Monitoring Infrastructure) do WebSphere Application Server” na página 295 para obter maiores informações.
3. Quando o ObjectGrid estiver incorporado, as transações poderão ser iniciadas e encerradas por um coordenador de transação externa. Consulte “ObjectGrid e Interação de Transação Externa” na página 302 para obter maiores informações.
4. Utilize o recurso de particionamento com o ObjectGrid. O recurso ObjectGrid fornece a capacidade de armazenamento em cache de pares de chave e valor de forma transacional e o recurso Recurso de Partição fornece a capacidade de roteamento baseado em contexto, de acordo com características do objeto. Consulte “Integrando o ObjectGrid e o Recurso de Particionamento” na página 305 para obter maiores informações.
5. É possível utilizar beans CMP (Container-Managed Persistence) no WebSphere Application Server Versão 6.0.2 e posterior, tirando vantagem do ObjectGrid como um cache externo em vez de um cache interno. Consulte “Configurando o ObjectGrid para Funcionar com Beans Gerenciados por Contêiner” na página 326 para obter maiores informações.

Também é possível utilizar o ObjectGrid com JMS para distribuir alterações entre diferentes camadas ou em ambientes com plataformas mistas. Consulte “Java Message Service para Distribuir Alterações de Transação” na página 344 para obter maiores informações.

Integrando o ObjectGrid em um Ambiente Java 2 Platform, Enterprise Edition

O ObjectGrid suporta modelos de programação de servlet e EJB (Enterprise JavaBeans) no ambiente J2EE (Java 2 Platform, Enterprise Edition).

Este tópico explora as etapas de programação comuns para ativar um aplicativo J2EE com o ObjectGrid.

Cenário do ObjectGrid Local

1. Definir uma configuração do ObjectGrid. Defina uma configuração do ObjectGrid com arquivos XML, por meio de interface programática ou com um uso misto de arquivos XML e configuração programática. Para obter informações adicionais, consulte Configuração do ObjectGrid.
2. Criar um objeto de URL. Se a configuração do ObjectGrid estiver em um arquivo XML, crie um objeto de URL que aponte para esse arquivo XML. É possível utilizar este objeto de URL para criar instâncias do ObjectGrid utilizando a API ObjectGridManager. Se o arquivo XML de configuração do ObjectGrid estiver incluído em um arquivo WAR (Web Archive) ou EJB (Enterprise JavaBeans) JAR (Java Archive), ele ficará acessível como um recurso para o carregador de classes para o módulo da Web e EJB. Por exemplo, se o arquivo XML de configuração do ObjectGrid estiver na pasta WEB-INF do arquivo WAR do módulo da Web, os servlets que estão nesse arquivo WAR poderão criar um objeto de URL com o seguinte padrão:

   ```
   URL url =className.class.getClassLoader().
   getResource("META-INF/objectgrid-definition.xml");
   URL objectgridUrl = ObjectGridCreationServlet.class.getClassLoader().
   getResource("WEB-INF/objectgrid-definition.xml");
   ```
3. Criar ou obter instâncias do ObjectGrid. Utilize a API ObjectGridManager para obter e criar instâncias do ObjectGrid. Com a API ObjectGridManager, é possível criar instâncias do ObjectGrid com XML e utilizar métodos utilitários para criar rapidamente uma instância simples do ObjectGrid. Os aplicativos devem utilizar a API ObjectGridManagerFactory para obter uma referência à API ObjectGridManager. Consulte o seguinte exemplo de codificação:

   ```
   import com.ibm.websphere.objectgrid.ObjectGridManager;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
   ...
   ObjectGridManager objectGridManager = ObjectGridManagerFactory.
   getObjectGridManager();
   ObjectGrid ivObjectGrid = objectGridManager.
   createObjectGrid(objectGridName, objectgridUrl, true, true);
   ```

   Para obter informações adicionais sobre a API ObjectGridManager, consulte o tópico Interface ObjectGridManager.
4. Inicialize as instâncias do ObjectGrid. Utilize o método initialize na interface do ObjectGrid para iniciar a auto-inicialização de instâncias do ObjectGrid e de Sessão. Este método initialize é considerado opcional, porque a primeira chamada para o método getSession desempenha uma inicialização implícita. Quando este método for chamado, a configuração do ObjectGrid será considerada concluída e estará pronta para uso de tempo de execução. As chamadas de método de configuração adicional, com chamar o método defineMap(String mapName), resultam em uma exceção.
5. Obtenha uma instância de Sessão e do ObjectMap. Uma sessão é um contêiner para instâncias do ObjectMap. Um encadeamento deve obter seu próprio objeto de Sessão para interagir com o núcleo do ObjectGrid. Você pode imaginar esta técnica como uma sessão que pode ser utilizada apenas por um único encadeamento por vez. A sessão é compartilhável nos encadeamentos, se utilizar apenas um encadeamento por vez. No entanto, se for utilizada uma infra-estrutura de conexão ou de transação J2EE, o objeto de sessão não será compartilhável nos encadeamentos. Uma boa analogia para este objeto é uma conexão com um banco de dados JDBC (Java Database Connectivity).

   Um mapa do ObjectMap é uma manipulação para um mapa denominado. Os Mapas devem ter chaves e valores homogêneos. Uma instância do ObjectMap pode ser utilizada apenas pelo encadeamento que está associado à sessão

que foi utilizada para obter esta instância do ObjectMap. Vários encadeamentos não podem compartilhar objetos de Sessão e ObjectMap simultaneamente. As palavras-chave são aplicadas em uma transação. Um rollback de transação efetua rollback de qualquer associação de palavra-chave aplicada durante esta transação.A seguir está o exemplo de codificação:

```
Session ivSession = ivObjectGrid.getSession();
ObjectMap ivEmpMap = ivSession.getMap("employees");
ObjectMap ivOfficeMap = ivSession.getMap("offices");
ObjectMap ivSiteMap = ivSession.getMap("sites");
ObjectMap ivCounterMap = ivSession.getMap("counters");
```

6. Inicie uma sessão, leia ou grave objetos e confirme ou efetue rollback da sessão. As operações do Mapa devem estar em um contexto transacional. O método begin do objeto de Sessão é utilizado para iniciar um contexto transacional explícito. Quando a sessão é iniciada, os aplicativos podem começar a desempenhar operações do mapa. As operações mais comuns incluem chamadas de método get, `update`, insert e remove para objetos em mapas. No final das operações do mapa, o método de confirmação ou de rollback do objeto de Sessão é chamado para confirmar um contexto transacional explícito ou efetuar rollback de um contexto transacional explícito. A seguir está um exemplo de programação:

```
ivSession.begin();
Integer key = new Integer(1);
if (ivCounterMap.containsKey(key) == false) {
ivCounterMap.insert(key, new Counter(10));
}
ivSession.commit();
```

Cenário do ObjectGrid Distribuído

Este cenário do ObjectGrid distribuído se difere do cenário do ObjectGrid local apenas na maneira de obter a instância do ObjectGrid. Os exemplos de código a seguir demonstram como obter uma instância do ObjectGrid distribuído:

```
//Utilizar ObjectGridManagerFactory para obter a referência à API ObjectGridManager
ObjectGridManager objectGridManager = ObjectGridManagerFactory.
getObjectGridManager();
//Obter o ClientClusterContext representa o cluster do ObjectGrid do
//ObjectGridManager
//Supondo que o servidor WebSphere também hospeda um servidor do ObjectGrid
//que seja membro do cluster do ObjectGrid.
ClientClusterContext context = objectGridManager.connect(null, null);
//Obter a instância do ObejctGrid do ObjectGridManager de um
//ClientClusterContext específico.
ObjectGrid ivObjectGrid= objectGridManager.getObjectGrid(context,
"objectgridName");
```

Você desempenhou as etapas de programação básicas para ativar um aplicativo J2EE com o ObjectGrid.

Consulte Construindo Aplicativos J2EE (Java 2 Platform, Enterprise Edition) ativados por ObjectGrid e Considerações para a Integração de Aplicativos J2EE (Java 2 Platform, Enterprise Edition) e o ObjectGrid para obter informações adicionais.

Construindo Aplicativos Java 2 Platform, Enterprise Edition Ativados por ObjectGrid

Utilize esta tarefa para configurar o caminho de construção, ou caminho de classe, de aplicativos J2EE (Java 2 Platform, Enterprise Edition) ativados por ObjectGrid. O caminho de classe deve incluir o arquivo `wsobjectgrid.jar` que está localizado no diretório `$install_root/lib`.

Desenvolver um aplicativo J2EE ativado por ObjectGrid. Consulte Integrando o ObjectGrid em um Ambiente J2EE para obter informações adicionais.

Esta tarefa demonstra como configurar o caminho de construção para incluir o arquivo `wsobjectgrid.jar` no IBM Rational Software Development Platform Versão 6.0.

1. Na visualização Project Explorer da perspectiva J2EE, clique com o botão direito no projeto **WEB** ou Enterprise JavaBeans (**EJB**) e selecione **Propriedades**. É exibida a janela Propriedades.
2. Selecione o **Caminho de Construção Java** no painel à esquerda, clique na guia **Bibliotecas** no painel à direita e clique em **Incluir Variável**. É exibida a janela **Nova Entrada de Caminho de Classe da Variável**.
3. Clique em **Configurar Variáveis** para abrir a janela **Preferência**.
4. Inclua uma nova entrada da variável.
   a. Clique em **Novo**.
   b. Digite `OBJECTGRID_JAR` no campo **nome**. Clique em **Arquivo** para abrir a janela **Seleção de JAR**.
   c. Navegue para o diretório `/lib`, clique no arquivo **wsobjectgrid.jar** e clique em **Abrir** para fechar a janela **Seleção de JAR**.
   d. Clique em **OK** para fechar a janela **Nova Entrada da Variável**.

   A variável `OBJECTGRID_JAR` é exibida na lista Variáveis do Caminho de Classe.
5. Clique em **OK** para fechar a janela **Preferência**.
6. Selecione a variável `OBJECTGRID_JAR` da lista de variáveis e clique em **OK** para fechar a janela **Nova Entrada de Caminho de Classe da Variável**. A variável `OBJECTGRID_JAR` é exibida no painel **Bibliotecas**.
7. Clique em **OK** para fechar a janela **Propriedades**.

Você configurou o caminho de construção para incluir o arquivo `wsobjectgrid.jar` no IBM Rational Software Development Platform Versão 6.0.

Considerações para a Integração de Aplicativos Java 2 Platform, Enterprise Edition e o ObjectGrid

Utilize estas considerações ao integrar um aplicativo J2EE (Java 2 Platform, Enterprise Edition) com o ObjectGrid.

Beans de Inicialização e o ObjectGrid

É possível utilizar beans de inicialização para um aplicativo para auto-inicializar uma instância do ObjectGrid quando um aplicativo inicia e destrói a instância do ObjectGrid quando o aplicativo pára. Um bean de inicialização é um bean de sessão sem preservação de estado com uma interface home com.ibm.websphere.startupservice.AppStartUpHome e uma interface remota com.ibm.websphere.startupservice.AppStartUp. Quando o WebSphere Application Server vê um EJB (Enterprise JavaBean), ele reconhece o bean de inicialização. A

interface remota possui dois métodos, o método start e o método stop. Utilize o método start para auto-inicializar a grade e chame o método grid destroy com o método stop. O aplicativo pode manter uma referência à grade utilizando o método ObjectGridManager.getObjectGrid para obter uma referência quando necessário. Para obter informações adicionais, consulte o tópico Interface ObjectGridManager.

Carregadores de Classes e Instâncias do ObjectGrid

É necessário ter atenção ao compartilhar uma única instância do ObjectGrid entre módulos aplicativos que utilizam diferentes carregadores de classes. Os módulos aplicativos que utilizam diferentes carregadores de classes não funcionam e resultam em exceções de lançamento de classes no aplicativo. Um ObjectGrid deve ser compartilhado apenas por módulos aplicativos que utilizam o mesmo carregador de classes ou quando os objetos de aplicativos, por exemplo, plug-ins, chaves e valores estão em um carregador de classes comum.

Gerenciar o Ciclo de Vida de Instâncias do ObjectGrid em um Servlet

É possível gerenciar o ciclo de vida de instâncias do ObjectGrid com o método init e o método destroy de um servlet. Utilize o método init para criar e inicializar instâncias do ObjectGrid que são requeridas pelo aplicativo. Quando as instâncias do ObjectGrid forem criadas e armazenadas em cache, será possível obtê-las por seus nomes com a API do ObjectGridManager. Utilize o método destroy para destruir estas instâncias do ObjectGrid e para liberar recursos do sistema. Para obter informações adicionais, consulte o tópico Interface ObjectGridManager.

Monitorando o Desempenho do ObjectGrid com a PMI (Performance Monitoring Infrastructure) do WebSphere Application Server

O ObjectGrid suporta a PMI (Performance Monitoring Infrastructure) ao executar em um servidor de aplicativos do WebSphere Application Server ou do 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.

Para obter informações adicionais sobre as estatísticas fornecidas pelo ObjectGrid, consulte Estatísticas do ObjectGrid.

O ObjectGrid utiliza o recurso de PMI customizado do WebSphere Application Server para incluir sua instrumentação de PMI. Com esta abordagem, você pode ativar e desativar a PMI do ObjectGrid com o console administrativo ou com interfaces JMX (Java Management Extensions). Além disso, você pode acessar as estatísticas do ObjectGrid com as interfaces PMI e JMX padrão que são utilizadas por ferramentas de monitoramento, incluindo o Tivoli Performance Viewer.

1. Ative a PMI do ObjectGrid. É necessário ativar a PMI para visualizar as estatísticas de PMI. Consulte “Ativando a PMI do ObjectGrid” na página 298 para obter maiores informações.
2. Recupere as estatísticas de PMI do ObjectGrid. Visualize o desempenho de seus aplicativos do ObjectGrid com o Tivoli Performance Viewer. Consulte “Recuperando Estatísticas de PMI do ObjectGrid” na página 301 para obter maiores informações.

Estatísticas do ObjectGrid

O ObjectGrid fornece dois módulos PMI (Performance Monitoring Infrastructure): o módulo objectGridModule e o módulo mapModule.

Módulo objectGridModule

O módulo objectGridModule contém uma estatística de tempo: tempo de resposta da transação. Uma transação do ObjectGrid é definida como a 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 módulo objectGridModule, o elemento ObjectGrids, servem como o ponto de entrada para as estatísticas do ObjectGrid. Este elemento raiz possui instâncias do ObjectGrid como seus filhos que possuem tipos de transação como seus filhos. A estatística de tempo de resposta está associada a cada tipo de transação. A estrutura do módulo objectGridModule é mostrada no diagrama a seguir:

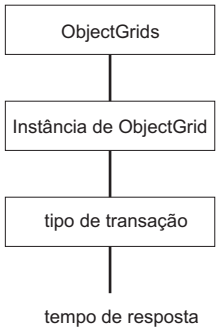


Figura 19. Estrutura do Módulo ObjectGridModule

O diagrama a seguir mostra um exemplo da estrutura do módulo PMI do ObjectGrid. Neste exemplo, existem duas instâncias do ObjectGrid no sistema: o ObjectGrid objectGrid1 e o ObjectGrid objectGrid2. A instância objectGrid1 possui dois tipos de transações: atualização e leitura, e a instância objectGrid2 possui apenas um tipo de transação: atualização.

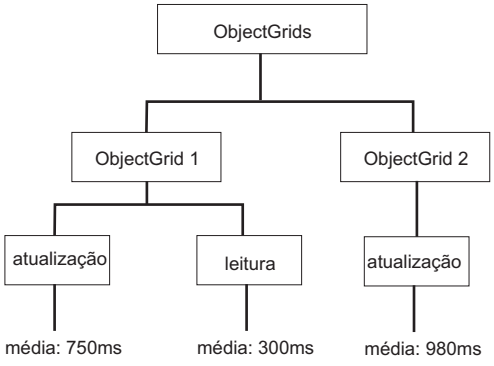


Figura 20. Estrutura do Módulo PMI do ObjectGrid

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 configurado utilizando o seguinte método Session.setTransactionType(String):

```
/**
* 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
* aplicativo.
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`:

```
// Configurar o tipo de transação como updatePrice
// O tempo entre session.begin() e session.commit() será
// rastreado na estatística de tempo para "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`. Existe uma estatística `updatePrice` na instância do ObjectGrid que corresponde à sessão no exemplo. Utilizando interfaces JMX (Java Management Extensions), você pode obter o tempo de resposta de transação para transações updatePrice. Também é possível obter a estatística agregada para todos os tipos de transações no ObjectGrid especificado.

Módulo mapModule

O módulo PMI mapModule contém três estatísticas que estão relacionadas a mapas do ObjectGrid:

- **Taxa de Acesso do Mapa:** Esta estatística BoundedRangeStatistic rastreia a taxa de acesso de um mapa. A taxa de acesso é um valor flutuante entre `0` e `100` inclusivamente, que representa a porcentagem de acessos do mapa em relação a operações get do mapa.
- **Número de entradas**: Esta estatística CountStatistic rastreia o número de entradas no mapa.
- **Tempo de resposta de atualização de batch do Loader**: Esta estatística TimeStatistic rastreia o tempo de resposta utilizado para a operação de atualização de batch do loader.

O elemento raiz do módulo mapModule, o elemento Mapas do ObjectGrid, serve como o ponto de entrada para estatísticas de Mapa do ObjectGrid. Este elemento raiz possui instâncias do ObjectGrid como seus filhos, que possuem instâncias do

mapa como seus filhos. Cada instância do mapa possui três estatísticas listadas. A estrutura mapModule é mostrada no diagrama a seguir:
O diagrama a seguir mostra um exemplo da estrutura mapModule:

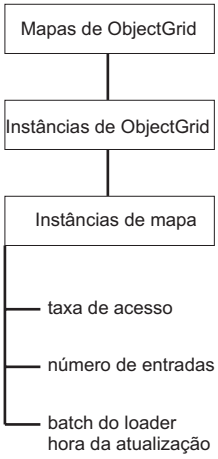


Figura 21. Estrutura do Módulo mapModule

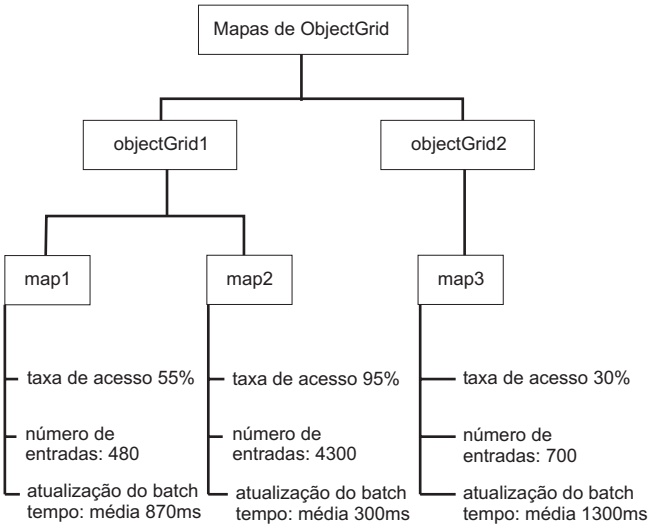


Figura 22. Exemplo de Estrutura do Módulo mapModule

Ativando a PMI do ObjectGrid

É possível utilizar a PMI (Performance Monitoring Infrastructure) do WebSphere Application Server para ativar ou desativar estatísticas em qualquer nível. Por exemplo, você pode optar por apenas 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 batch do loader. Este tópico mostra como utilizar o console administrativo e os scripts wsadmin para ativar a PMI do ObjectGrid.

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, você pode 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 batch do loader. Esta seção mostra como utilizar o console administrativo e os scripts wsadmin para ativar a PMI do ObjectGrid.

1. Abra o console administrativo, por exemplo, http://localhost:9060/ibm/console.
2. Clique em **Monitoramento e Ajuste > Performance Monitoring Infrastructure > *server_name***.
3. Verifique se **Ativar PMI (Performance Monitoring Infrastructure)** está selecionado. Essa definição é ativada por padrão. Se a configuração não estiver ativada, selecione a caixa de opções e, em seguida, reinicie o servidor.
4. Clique em **Customizado**. Na árvore de configuração, selecione o módulo **ObjectGrid** e **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. Você pode ver apenas as subcategorias do ObjectGrid e de estatísticas do Mapa no painel Tempo de Execução.

Por exemplo, você pode desempenhar as seguintes etapas para ativar as estatísticas de 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 em **ObjectGridCreationServlet** e, em seguida, clique nos botões de ação 1, 2, 3, 4 e 5 para gerar algumas ações para o ObjectGrid e mapas. Não feche esta página do servlet neste momento.
3. Retorne ao 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, em seguida, clique no link **clusterObjectGrid**. No grupo **Mapas do ObjectGrid**, existe uma instância do ObjectGrid chamada clusterObjectGrid e, neste grupo clusterObjectGrid, existem quatro mapas: counters, employees, offices e sites. Na instância de **ObjectGrids**, existe uma instância de clusterObjectGrid e nesta instância há um tipo de transação chamado `DEFAULT`.
6. É possível ativar as estatísticas de seu interesse. Para fins de demonstração, você pode ativar o **número de entradas do mapa** para o mapa employees e **tempo de resposta de transação** para o tipo de transação DEFAULT.

É possível automatizar a tarefa de ativação da PMI com script. Consulte Ativando a PMI do ObjectGrid com Script para obter informações adicionais.

Ativando a PMI do ObjectGrid com Script

Automatize a tarefa de ativação da PMI do ObjectGrid com a ferramenta wsadmin.

Seu servidor de aplicativos deve estar iniciado e ter um aplicativo ativado por ObjectGrid instalado. Você também deve poder efetuar login e utilizar a ferramenta wsadmin. Para obter informações adicionais sobre a ferramenta wsadmin, consulte Utilizando Scripts (wsadmin) no centro de informações do WebSphere Extended Deployment Versão 6.0.x.

Utilize esta tarefa para automatizar a ativação da PMI. Para ativar a PMI com o console administrativo, consulte Ativando a PMI do ObjectGrid.

1. Abra um prompt da 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 de tempo de execução de PMI do ObjectGrid. Verifique se a PMI está ativada para o servidor com os 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 variáveis para alterar o conjunto de estatísticas para um conjunto customizado. Execute 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 estatísticas configuradas como customizadas: Execute o seguinte comando:

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
   ```
5. Configure variáveis para ativar a estatística de PMI do objectGridModule. Execute os seguintes comandos:

   ```
   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. Execute o seguinte comando:

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params $sigs
   ```
7. Configure variáveis para ativar a estatística de PMI mapModule. Execute os seguintes comandos:

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

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

Estas etapas ativam a PMI de tempo de execução do ObjectGrid, mas não modificam a configuração da PMI. Se você reiniciar o servidor de aplicativos, as configurações da PMI serão perdidas, diferente da ativação de PMI principal.

Quando ativar a PMI, será possível visualizar estatísticas de PMI com o console administrativo ou por meio de scripts. Consulte “Recuperando Estatísticas de PMI do ObjectGrid” na página 301 e “Recuperando Estatísticas da PMI do ObjectGrid com Scripts” na página 301 para obter informações adicionais.

Recuperando Estatísticas de PMI do ObjectGrid

Consulte as estatísticas de desempenho de seus aplicativos do ObjectGrid.

Quando as estatísticas do ObjectGrid estiverem ativadas, será possível recuperá-las. Para ativar a PMI do ObjectGrid, consulte Ativando a PMI do ObjectGrid.

Utilize esta tarefa para ver as estatísticas de desempenho de seus aplicativos do ObjectGrid.

1. Abra o console administrativo. Por exemplo, http://localhost:9060/ibm/console.
2. Clique em **Monitoramento e Ajuste > Performance Viewer > Atividade Atual**.
3. Clique no servidor que deseja monitorar utilizando o Tivoli Performance Viewer e ative o monitoramento.
4. Clique no servidor para visualizar a página do Performance Viewer.
5. Expanda a árvore de configuração. Clique em **Mapas do ObjectGrid > clusterObjectGrid** e selecione **employees**. Expanda **ObjectGrids > clusterObjectGrid** e selecione **DEFAULT**.
6. No aplicativo de amostra do ObjectGrid, retorne ao servlet ObjectGridCreationServlet, clique no botão 1, **ocupar mapas**. Você pode visualizar as estatísticas no visualizador.

Você pode visualizar Estatísticas do ObjectGrid no Tivoli Performance Viewer.

Você pode automatizar a tarefa de recuperação de estatísticas utilizando JMX (Java Management Extensions) ou a ferramenta wsadmin. Consulte “Recuperando Estatísticas da PMI do ObjectGrid com Scripts”

Recuperando Estatísticas da PMI do ObjectGrid com Scripts

Utilize esta tarefa para recuperar estatísticas de desempenho para aplicativos do ObjectGrid.

Ative a PMI (Performance Monitoring Infrastructure) em seu ambiente de servidor de aplicativos. Consulte Ativando a PMI do ObjectGrid ou Ativando a PMI do ObjectGrid com Scripts para obter informações adicionais. Você também deve poder efetuar login e utilizar a ferramenta wsadmin. Para obter informações adicionais sobre a ferramenta wsadmin, consulte Utilizando Scripts (wsadmin) no WebSphere Extended Deployment Versão 6.0.x Information Center.

Utilize esta tarefa para obter estatísticas de desempenho para seu ambiente de servidor de aplicativos. Para obter informações adicionais sobre as estatísticas do ObjectGrid que podem ser recuperadas, consulte “Estatísticas do ObjectGrid” na página 296.

1. Abra um prompt da linha de comandos. Navegue para o diretório *install_root*/bin. Digite `wsadmin` para iniciar a ferramenta de linha de comandos wsadmin.
2. Configure variáveis para o ambiente. Execute 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 variáveis para obter estatísticas de mapModule. Execute 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 estatísticas de mapModule. Execute o seguinte comando:

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

5. Configure variáveis para obter estatísticas do objectGridModule. Execute 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 estatísticas de objectGridModule. Execute o seguinte comando:

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

Consulte “Estatísticas do ObjectGrid” na página 296 para obter informações adicionais sobre as estatísticas retornadas.

ObjectGrid e Interação de Transação Externa

Geralmente, as transações do ObjectGrid começam com o método session.begin e são encerradas com o método session.commit. No entanto, quando o ObjectGrid é incorporado, as transações podem ser iniciadas e encerradas por um coordenador de transação externa. Neste caso, não é necessário chamar o método session.begin e encerrar com o método session.commit.

Coordenação de Transação Externa

O plug-in TransactionCallback do ObjectGrid é estendido com o método isExternalTransactionActive(Session session) que associa a sessão do ObjectGrid a uma transação externa. O cabeçalho do método é o seguinte:

```
public synchronized boolean isExternalTransactionActive(Session session)
```

Por exemplo, o ObjectGrid pode ser configurado para integrar-se com o WebSphere Application Server e o WebSphere Extended Deployment. A chave para esta integração total é a exploração da API ExtendedJTATransaction no WebSphere Application Server Versão 5.x e Versão 6.x. No entanto, se você estiver utilizando o WebSphere Application Server Versão 6.0.2, deverá aplicar o APAR PK07848 para suportar este método. Utilize o seguinte código de amostra para associar uma sessão do ObjectGrid a um ID de transação do WebSphere Application Server:

```
/**
* Este método é requerido para associar uma sessão do objectGrid a um ID
* de transação do WebSphere.
*/
Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
```

```
// lembre-se de que este localid significa que a sessão é salva posteriormente.
localIdToSession.put(new Integer(jta.getLocalId()), session);
return true;
}
```

Recuperar uma Transação Externa

Às vezes, pode ser necessário recuperar um objeto de serviço de transações externas para ser utilizado pelo plug-in TransactionCallback do ObjectGrid. No servidor WebSphere Application Server, você consulta o objeto ExtendedJTATransaction a partir de seu espaço de nomes, conforme 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 efetuar rollback da sessão do ObjectGrid. Para receber este sinal externo, utilize o retorno de chamada do serviço de transações externas. É necessário implementar a interface de retorno de chamada externa e registrá-la no serviço de transações externas. Por exemplo, no caso do WebSphere Application Server, é necessário implementar a interface SynchronizationCallback, conforme 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);
// localizar a Sessão para o localId
Session session = (Session)localIdToSession.get(lid);
if(session != null)
{
try
{
// se o WebSphere Application Server for confirmado ao
// proteger a transação no backingMap.
// Já fizemos uma limpeza em beforeCompletion
if(didCommit)
{
session.commit();
}
else
{
// caso contrário, efetuar rollback
session.rollback();
}
}
catch(NoActiveTransactionException e)
{
// teoricamente impossível
}
catch(TransactionException e)
{
// sabendo que já fizemos uma limpeza, isto não deve falhar
}
finally
{
// sempre limpar a sessão do mapa de mapeamento.
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)
{
// O WebSphere Application Server não define formalmente
// uma maneira para sinalizar
// que a transação falhou ao fazer isso
throw new RuntimeException("Cache flush failed", e);
}
}
}
}
```

Utilizar APIs do ObjectGrid com o Plug-in TransactionCallback

Este plug-in, quando utilizado como o plug-in TransactionCallback para um ObjectGrid, desativa a confirmação automática. O padrão de uso normal para um ObjectGrid 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 sendo utilizado, o ObjectGrid assume que o aplicativo utiliza o ObjectGrid quando uma transação gerenciada por contêiner está presente. O trecho de código anterior é alterado para o seguinte código neste ambiente:

```
public void myMethod()
{
UserTransaction tx = ...;
tx.begin();
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
tx.commit();
}
```

O método myMethod é semelhante a um caso de aplicativo da Web. O aplicativo utiliza a interface UserTransaction normal para iniciar, confirmar e efetuar rollback de transações. O ObjectGrid é iniciado e confirmado automaticamente na transação do contêiner. Se o método for um método EJB (Enterprise JavaBeans) que utiliza o atributo TX_REQUIRES, remova a referência UserTransaction e as chamadas para iniciar e confirmar transações e o método funcionará da mesma maneira. Neste caso, o contêiner é responsável por iniciar e encerrar a transação.

Integrando o ObjectGrid e o Recurso de Particionamento

Utilize o aplicativo de amostra ObjectGridPartitionCluster para aprender sobre as funções combinadas do ObjectGrid e o recurso de particionamento (WPF).

Consulte “O ObjectGrid e o Recurso de Particionamento” na página 306 para obter um resumo de como o ObjectGrid e o recurso de particionamento funcionam juntos.

Para utilizar o ObjectGrid com o recurso de particionamento, é necessário ter o WebSphere Extended Deployment instalado em seu ambiente.

A amostra ObjectGridPartitionCluster demonstra as funções combinadas do ObjectGrid e do Recurso de Particionamento (WPF). O recurso ObjectGrid fornece a capacidade de armazenar em cache pares de chave e valor por transação e o recurso do recurso de particionamento fornece a capacidade de roteamento baseado em contexto, de acordo com as características do objeto.

- Instale e execute o aplicativo de amostra ObjectGridPartitionCluster. Consulte “Instalando e Executando o Aplicativo de Amostra ObjectGridPartitionCluster” na página 308 para obter maiores informações.

- Se desejar visualizar ou modificar o código fonte do aplicativo de amostra, poderá carregar o arquivo EAR (Enterprise Archive) para sua ferramenta de desenvolvimento. Consulte “Construindo um Aplicativo do ObjectGrid e de um Recurso de Particionamento Integrado” na página 311 para obter maiores informações.
- Aprenda sobre o aplicativo de amostra. Consulte “Exemplo: Programação do ObjectGrid e do Recurso de Particionamento” na página 315 para obter uma explicação sobre o código que está no aplicativo de amostra.

Consulte Capítulo 10, “Integrando o ObjectGrid com o WebSphere Application Server”, na página 291 para obter informações adicionais sobre como integrar os recursos do ObjectGrid e do WebSphere Application Server. Para obter informações adicionais sobre o modelo de programação do ObjectGrid, consulte Capítulo 9, “Visão Geral da Interface de Programação de Aplicativo do ObjectGrid”, na página 93.

O ObjectGrid e o Recurso de Particionamento

Os recursos ObjectGrid e recurso de particionamento (WPF) podem funcionar juntos para fornecer o armazenamento em cache de pares de chave e valor e de roteamento baseado em contexto, com base em características do objeto.

A amostra do ObjectGridPartitionCluster demonstra as funções combinadas do ObjectGrid e o recurso de particionamento (WPF). O ObjectGrid e o recurso de particionamento são dois recursos existentes no produto WebSphere Extended Deployment. O recurso ObjectGrid fornece a capacidade de armazenamento em cache de pares de chave e valor de forma transacional e o recurso recurso de particionamento fornece a capacidade de roteamento baseado em contexto, de acordo com características do objeto.

Além de demonstrar recursos dos plug-ins loader e TransactionCallback, esta amostra também demonstra como utilizar os plug-ins ObjectGridEventListener, ObjectTransformer e OptimisticCallback. Em específico, a amostra demonstra como propagar transações do ObjectGrid local e como invalidar os objetos alterados de um servidor para outros servidores com e sem o verificador de versão otimista.

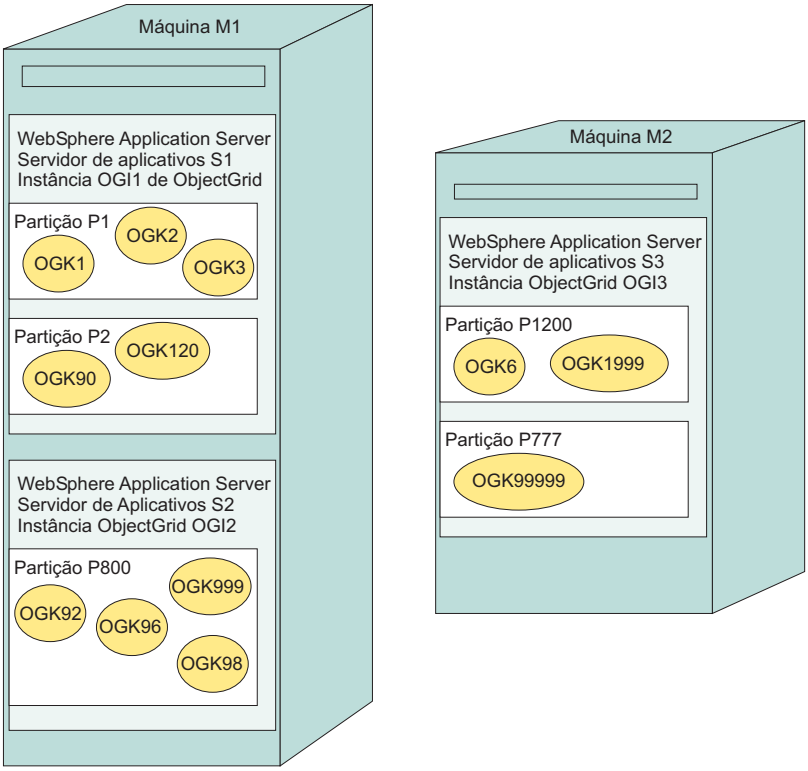
É necessário utilizar o recurso roteamento baseado em contexto do recurso de particionamento para assegurar que os pedidos de atualização, inserção e remoção de objetos para a mesma chave sejam roteados para a mesma JVM (Java Virtual Machine) e que os pedidos de recuperação de objetos possam ser distribuídos em todas as JVMs do ObjectGrid com gerenciamento de carga de trabalho. A utilização do recurso de particionamento mantém a integridade de dados nas diferentes instâncias do ObjectGrid do membro de cluster.

Para manter a consistência e integridade do ObjectGrid, é possível utilizar o recurso de particionamento para dividir um grande ObjectGrid em muitos ObjectGrids particionados e o roteamento baseado em contexto do recurso de particionamento direciona pedidos de acordo com as chaves do ObjectGrid. Por exemplo, é necessário que o ObjectGrid manipule um grande número de objetos que não cabem em um ObjectGrid de JVM. É possível utilizar o recurso de particionamento para carregar dados em diferentes servidores com o método partitionLoadEvent como pré-carregamento e o roteamento baseado em contexto do recurso de particionamento localiza o ObjectGrid correto para você.

A amostra cria um conjunto de partições baseadas em hash e de contextos de roteamento de cluster de partição:

- É possível particionar e mapear chaves do ObjectGrid para as partições do WPF com uma estratégia de muitos para muitos.
- As partições do WPF podem ser hospedadas no cluster do WebSphere Application Server em uma estratégia de muitos para muitos.

O diagrama a seguir mostra as configurações típicas e a configuração para a amostra ObjectGridPartitionCluster:



No diagrama anterior, a máquina M1 e a máquina M2 são utilizadas para implementar a amostra ObjectGridPartitionCluster. Cada máquina física pode hospedar um ou mais WebSphere Application Servers. Por exemplo, a máquina M1 hospeda dois servidores de aplicativos: o servidor de aplicativos S1 e o servidor de aplicativos S2. A máquina M2 hospeda um servidor, que é o servidor de aplicativos S3. Cada servidor possui uma instância do ObjectGrid: A instância OGI1 do ObjectGrid para o servidor de aplicativos S1, a instância OGI2 do ObjectGrid para o servidor de aplicativos S2 e a instância OGI3 do ObjectGrid para o servidor de aplicativos S3.

Cada servidor de aplicativos pode hospedar muitas partições. Por exemplo, o servidor S1 hospeda a partição P1 e a partição P2 e o servidor S3 hospeda a partição P1200 e a partição P777.

Cada partição pode hospedar muitas chaves do ObjectGrid. Por exemplo, a partição P1 hospeda as chaves OGK1, OGK2 e OGK3 do ObjectGrid e a partição P800 hospeda as partições OGK92, OGK96, OGK98 e OGK9999.

Todos os pedidos de atualização, inserção e remoção do ObjectGrid são roteados de acordo com chaves do ObjectGrid. Você tem duas opções para recuperações de objetos: de qualquer servidor em uma estratégia gerenciada por carga de trabalho ou da partição do servidor específica para esta chave.

Instalando e Executando o Aplicativo de Amostra ObjectGridPartitionCluster

Utilize esta tarefa para instalar e executar o aplicativo de amostra ObjectGridPartitionCluster para testar a funcionalidade entre o ObjectGrid e o recurso de particionamento.

Instale o WebSphere Extended Deployment. Consulte a página Biblioteca do WebSphere Extended Deployment para obter instruções.

Um bom ambiente para executar a amostra ObjectGridPartitionCluster inclui a instalação do WebSphere Extended Deployment em duas máquinas físicas ou a criação de dois nós e a federação deles junto com o gerenciador de implementação.

1. Para demonstrar corretamente os recursos desta amostra, configure um cluster que tenha três ou mais membros de cluster.
2. Instale o arquivo `D_ObjectGridPartitionClusterSample.ear`. O arquivo `D_ObjectGridPartitionClusterSample.ear` implementado pelo recurso de particionamento (WPF) está pronto para instalação e execução. Se você modificar o código fonte de amostra, siga as instruções de wpf-deploy e de build para construir e implementar seu arquivo EAR (Enterprise Archive).

   A maneira comum de instalar arquivos EAR do aplicativo é utilizar o console administrativo. Siga o procedimento de instalação do aplicativo corporativo para instalar o arquivo `D_ObjectGridPartitionClusterSample.ear`. Para acessar esta parte do console administrativo, clique em **Aplicativos > Instalar um Novo Aplicativo**. Não implemente o arquivo EAR durante a instalação. Utilize as configurações padrão exceto na etapa em que é solicitado que selecione um local da instalação. Nesta etapa, selecione o cluster definido em vez do servidor server1 padrão.
3. Execute o cliente ObjectGridPartitionClusterSample.
   a. Inicie o cluster. No console administrativo, clique em **Servidores > Clusters**. Selecione o cluster e clique em **Iniciar**.
   b. Execute o comando de script **WAS_INSTALL_ROOT\bin\wpfadmin balance**. Verifique se a partição tem um status ativo utilizando o comando **WAS_INSTALL_ROOT\bin\wpfadmin list**. Para obter informações sobre o script wpfadmin e seus comandos, consulte o Partitioning Facility Guide no WebSphere Extended Deployment Information Center.
   c. Para executar o cliente ObjectGridPartitionClusterSample, execute o seguinte comando:

      ```
      WAS_INSTALL_ROOT/bin/launchClient.bat|sh \
      WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear \
      -CCBootstrapPort=PORT
      ```

      Em que PORT é a porta RMI do servidor que pode ser localizada no arquivo `SystemOut.log` do servidor após o início do servidor. Geralmente, este valor de porta é um dos seguintes valores:`9810`, `9811`, `9812`.

      Por exemplo, você pode executar o seguinte comando:

      ```
      WAS_INSTALL_ROOT/bin/launchClient.bat|sh
      WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear
      -CCBootstrapPort=9811
      ```

      Para uso mais avançado deste script, consulte “Opções do Cliente Aplicativo ObjectGridPartitionClusterSample” na página 309.

4. Altere o número de partições. Altere o número de partições criadas pelo bean corporativo de sessão ObjectGridPartitionCluster: O número de partições criadas pelo bean de sessão PFClusterObjectGridEJB é decidido pela variável de entrada do ambiente NumberOfPartitions que está no arquivo `META-INF\ejb-jar.xml`. O valor padrão é `10`. É possível alterar o valor desta variável de ambiente e reinstalar o aplicativo para criar diferentes números de partições. Configure o número de partições para menos de `999999`.
5. Altere as opções do listener distribuído. É possível alterar as seguintes opções de listener distribuído do ObjectGrid:

Tabela 17. Opções de Listener Distribuído

Nome da Variável	Descrição
enableDistribution,	É possível ativar um listener distribuído do ObjectGrid com a variável de entrada do ambiente enableDistribution que está no descritor de implementação EJB. O padrão é `true`, que é ativado. Configure o valor como `false` para desativar o listener distribuído.
propagationMode	É possível alterar o modo de propagação com a variável de entrada do ambiente propagationMode que está no descritor de implementação EJB. O padrão é `update`. É possível alterar o valor para `invalidate` se não desejar o valor padrão.
propagationVersionOption,	É possível alterar a opção de versão de propagação com a variável de entrada do ambiente propagationVersionOption que está localizada no descritor de implementação EJB. O padrão é `enable`. É possível configurar o valor como `disable`.
compressionMode	É possível alterar o modo de compactação com a variável de entrada do ambiente compressionMode que está localizada no descritor de implementação EJB. O padrão é `enable`. É possível configurar o valor como `disable`.

   O padrão é propagar as atualizações com a verificação de versão. É possível configurar o valor como o modo invalidate sem a verificação de versão.

Você instalou e executou o aplicativo de amostra ObjectGridPartitionCluster.

Opções do Cliente Aplicativo ObjectGridPartitionClusterSample

Utilize estas opções para usuário avançado ao executar o arquivo `D_ObjectGridPartitionClusterSample.ear`.

Uso de Amostra Avançado

Consulte “Instalando e Executando o Aplicativo de Amostra ObjectGridPartitionCluster” na página 308 para obter informações adicionais sobre como instalar e executar o arquivo `D_ObjectGridPartitionClusterSample.ear`.

Para uso avançado da amostra, consulte o seguinte guia de uso completo:

```
WAS_INSTALL_ROOT/bin/launchClient.bat|sh
WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear
-CCproviderURL=corbaloc::HOSTNAME:SERVER_RMI_PORT [-loop LOOP] [-threads
NUMBER_OF_THREADS] [-add NUMBER_OF_STOCKS_PER_PARTITION] [-waitForPropagation
SECONDS_TO_WAIT_FOR_PROPAGATION] [-getIteration
NUMBER_OF_ITERATION_PER_OGKEY]
```

Preencha as seguintes variáveis:

- HOSTNAME : Especifica o nome do host do servidor de aplicativos que está em execução.
- SERVER_RMI_PORT: Especifica a porta de Auto-inicialização do servidor de aplicativos.
- LOOP: Especifica quantos loops o cliente executa. Esse parâmetro é opcional. O valor padrão é 1.
- NUMBER_OF_THREADS: Especifica quantos encadeamentos o cliente executa. Esse parâmetro é opcional. O valor padrão é 1.
- NUMBER_OF_STOCKS_PER_PARTITION: Especifica o número de ações para cada partição a ser incluída. Esse parâmetro é opcional. O padrão é 3.
- SECONDS_TO_WAIT_FOR_PROPAGATION : Especifica os segundos de espera de objetos do ObjectGrid recém-incluídos ou atualizados para serem propagados para outros servidores. O padrão é de 2 segundos.
- NUMBER_OF_ITERATION_PER_OGKEY : Especifica o número de iterações de recuperação de objetos no ObjectGrid de maneira gerenciada por carga de trabalho. O padrão é 6. Com mais iterações especificadas, é visto um padrão limpo para objetos da mesma chave em diferentes servidores do WebSphere Application Server.

Amostra de Saída

A saída deste comando é semelhante ao seguinte exemplo:

```
C:\dev\xd6\bin>launchClient
D_ObjectGridPartitionClusterSample.ear -CCBootstrapPort=9812
IBM WebSphere Application Server, Release 6.0
J2EE Application Client Tool
Copyright IBM Corp., 1997-2004
WSCL0012I: Processando argumentos da linha de comandos.
WSCL0013I: Inicializando o J2EE Application Client Environment.
WSCL0035I: A inicialização do Ambiente do J2EE Application Client foi concluída.
WSCL0014I: Chamando a classe do Cliente
Aplicativo com.ibm.websphere.samples.objectgrid.partitionclust
er.client.PartitionObjectGrid
A Amostra de Partição do ObjectGrid possui 10 partições
PARTITION: ObjectGridHashPartition000007->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000003->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000005->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000010->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000006->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000009->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000008->clusterdevNode01/s1
PARTITION: ObjectGridHashPartition000002->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000001->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000004->clusterdevNode01/s2
************** Partition=ObjectGridHashPartition000004****************
------ObjectGrid Operations: Stock Ticket=Stock000104 --------
get on partition for ticket: Stock000104->clusterdevNode02/s2
update: Stock000104->clusterdevNode02/s2
sleep 2 seconds.....
Iteration 1 : Stock000104->clusterdevNode01/s2
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 2 : Stock000104->clusterdevNode01/s1
```

```
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 3 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 4 : Stock000104->clusterdevNode01/s2
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 5 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 6 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
------ObjectGrid Operations: Stock Ticket=Stock000114 --------
get on partition for ticket: Stock000114->clusterdevNode01/s2
update: Stock000114->clusterdevNode02/s2
sleep 2 seconds.....
Iteration 1 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 2 : Stock000114->clusterdevNode01/s2
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 3 : Stock000114->clusterdevNode01/s1
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 4 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 5 : Stock000114->clusterdevNode01/s2
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 6 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
------ObjectGrid Operations: Stock Ticket=Stock000124 --------
get on partition for ticket: Stock000124->clusterdevNode02/s2
update: Stock000124->clusterdevNode01/s2
sleep 2 seconds.....
Iteration 1 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 2 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 3 : Stock000124->clusterdevNode01/s2
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 4 : Stock000124->clusterdevNode01/s1
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 5 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 6 : Stock000124->clusterdevNode01/s2
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
C:\dev\xd6\bin>
```

Construindo um Aplicativo do ObjectGrid e de um Recurso de Particionamento Integrado

Abrir, modificar e instalar o aplicativo de amostra de particionamento do ObjectGrid.

Utilize estas etapas para modificar, exportar e instalar o arquivo `ObjectGridPartitionSample.ear` em um ambiente do WebSphere Extended Deployment. Se não desejar fazer alterações no arquivo de amostra, poderá utilizar o arquivo `D_ObjectGridPartitionClusterSample.ear` implementado e ativado pelo recurso de particionamento (WPF). Se você utilizar o arquivo `D_ObjectGridPartitionClusterSample.ear`, poderá instalar e executar o arquivo sem desempenhar as etapas a seguir. Os dois arquivos EAR (Enterprise Archive) estão no diretório `WAS_INSTALL_ROOT/installableApps`.

1. Configure o arquivo `ObjectGridPartitionSample.ear` em seu ambiente de construção, como o IBM Rational Application Developer Versão 6.0.x ou o Application Server Toolkit Versão 6.0.x. Consulte “Introdução à Construção de um Aplicativo do ObjectGrid e do Recurso de Particionamento” na página 312 para obter maiores informações.
2. Modifique qualquer código fonte na amostra.

3. Exporte o aplicativo ObjectGridPartitionClusterSample de seu ambiente de construção como um arquivo EAR. “Exportando o Arquivo ObjectGridPartitionClusterSample.ear em um IBM Rational Application Developer” na página 313 para obter informações adicionais.
4. Implemente o aplicativo para que ele possa funcionar com o recurso de particionamento. Consulte “Implementando o Arquivo ObjectGridPartitionClusterSample.ear para Funcionar com o Recurso de Particionamento” na página 314 para obter maiores informações.
5. Instale o arquivo `ObjectGridPartitionClusterSample.ear` no WebSphere Extended Deployment. A maneira comum de instalar arquivos EAR do aplicativo é utilizar o console administrativo do WebSphere Application Server. Siga o procedimento de instalação do aplicativo corporativo do console administrativo para instalar o arquivo `D_ObjectGridPartitionClusterSample.ear`. Não implemente o arquivo durante a instalação; utilize o padrão. Utilize as configurações padrão para cada etapa, exceto quando você é solicitado a selecionar onde instalar. Nesta etapa, selecione o cluster definido em vez do servidor padrão server1.

Você instalou o arquivo `ObjectGridPartitionClusterSample.ear` em um ambiente do WebSphere Extended Deployment.

Para obter informações adicionais sobre programação com o ObjectGrid, recurso de particionamento e aplicativos de amostra, consulte “Exemplo: Programação do ObjectGrid e do Recurso de Particionamento” na página 315.

Introdução à Construção de um Aplicativo do ObjectGrid e do Recurso de Particionamento

Utilize o Application Server Toolkit Versão 6.0.x ou o IBM Rational Application Developer Versão 6.0.x para reconstruir o aplicativo de amostra.

O arquivo `ObjectGridPartitionClusterSample.ear` no diretório `WAS_INSTALL_ROOT/installableApps` contém todo o código fonte. É possível utilizar o Application Server Toolkit Versão 6.0.x ou o IBM Rational Application Developer Versão 6.0.x para reconstruir este aplicativo de amostra. Esta tarefa utiliza o Rational Application Developer como um exemplo para estabelecer o ambiente de construção para o arquivo `ObjectGridPartitionClusterSample.ear`. Também é possível utilizar o Application Server Toolkit; uma ferramenta de montagem livre fornecida com o WebSphere Application Server em um CD separado.

O arquivo EAR (Enterprise Archive) implementado e ativado por WPF, o arquivo `D_ObjectGridPartitionClusterSample.ear` também no diretório `WAS_INSTALL_ROOT/installableApps`, está pronto para instalação e execução.

1. Importe o arquivo `ObjectGridPartitionClusterSample.ear` para o Rational Application Developer.
   a. Inicie o Rational Application Developer.
   b. **Opcional:** Abra a perspectiva J2EE (Java 2 Platform, Enterprise Edition) para trabalhar com projetos J2EE. Clique em **Janela > Abrir Perspectiva > Outro > J2EE**.
   c. **Opcional:** Abra a visualização Project Explorer. Clique em **Janela > Mostrar Visualização > Project Explorer**. Outra visualização útil é a visualização do Navegador: **Janela > Mostrar Visualização > Navegador**.
   d. Importe o arquivo `ObjectGridPartitionClusterSample.ear`. Clique em **Arquivo > Importar > Arquivo EAR** e, em seguida, clique em **Avançar**.

e. Selecione o arquivo `ObjectGridPartitionClusterSample.ear` no diretório `WAS_INSTALL_ROOT/installableApps`.

f. **Opcional:** Clique em **Novo** para abrir o assistente de Tempo de Execução de Novo Servidor e siga as instruções.

g. No campo Servidor de Destino, selecione o tipo de Execução do Servidor do **WebSphere Application Server V6.0**.

h. Clique em **Concluir**.

Os projetos ObjectGridPartitionClusterSample, ObjectGridPartitionClusterSampleEJB e ObjectGridPartitionClusterSampleClient devem ser criados e ficar visíveis na visualização Project Explorer.

2. Configure o projeto ObjectGridPartitionClusterSampleEJB.

a. Na visualização do Project Explorer da perspectiva J2EE, clique com o botão direito no projeto **ObjectGridPartitionClusterSampleEJB** nos projetos EJB e selecione **Propriedades**. É exibida a janela Propriedades.

b. Clique no **Caminho de Construção Java** no painel à esquerda, clique na guia **Bibliotecas** no painel à direita e selecione **Incluir Variável**. A janela Nova Entrada de Caminho de Classe da Variável é exibida.

c. Clique em **Configurar Variáveis** para abrir a janela Preferência.

d. Clique em **Novo** para abrir a janela Nova Entrada de Variável.

e. Digite `ObjectGridPartitionCluster_JAR` para o **Nome** e clique em **Arquivo** para abrir a janela Seleção de JAR.

f. Navegue para o diretório `WAS_INSTALL_ROOT/lib` e selecione **wsobjectgrid.jar**. Clique em **Abrir** para fechar a janela Seleção de JAR.

g. Clique em **OK** para fechar a janela Nova Entrada de Variável. A variável ObjectGridPartitionCluster_JAR é exibida na lista de Variáveis do Caminho de Classe.

h. Clique em **OK** para fechar a janela Preferência.

i. Selecione a variável **ObjectGridPartitionCluster_JAR** da lista de variáveis e clique em **OK** para fechar a janela Nova Entrada de Caminho de Classe da Variável. A variável ObjectGridPartitionCluster_JAR é exibida no painel Bibliotecas.

j. Repita este procedimento para incluir o arquivo `wpf.jar` em seu ambiente.

k. Verifique se o arquivo `wpf.jar` e o arquivo `wsobjectgrid.jar` estão no caminho de classe de construção.

Depois de configurar seu ambiente, é possível modificar o código fonte e aplicar outras alterações. Consulte “Construindo um Aplicativo do ObjectGrid e de um Recurso de Particionamento Integrado” na página 311 para obter maiores informações.

Exportando o Arquivo ObjectGridPartitionClusterSample.ear em um IBM Rational Application Developer

Depois de fazer alterações no arquivo de amostra, você poderá exportar o aplicativo ObjectGridPartitionClusterSample para criar um arquivo EAR (Enterprise Archive) que pode ser instalado nos servidores do WebSphere Extended Deployment.

É necessário ter o arquivo `ObjectGridPartitionSample.ear` importado para suas ferramentas de desenvolvimento para que seja possível fazer alterações na origem. Consulte “Introdução à Construção de um Aplicativo do ObjectGrid e do Recurso de Particionamento” na página 312 para obter maiores informações. Antes de exportar, faça suas alterações no aplicativo de amostra.

Você pode exportar o arquivo ObjectGridPartitionClusterSample.ear do projeto ObjectGridPartitionClusterSample nos aplicativos corporativos no IBM Rational Application Developer. Você pode instalar o arquivo ObjectGridPartitionClusterSample.ear exportado em qualquer servidor do WebSphere Extended Deployment Versão 6.0 depois de implementar o recurso de particionamento.

1. Na visualização Project Explorer da perspectiva J2EE (Java 2 Platform, Enterprise Edition), clique com o botão direito no aplicativo **ObjectGridPartitionClusterSample** que está localizado sob Aplicativos Corporativos. Clique em **Exportar > Arquivo EAR**. É exibida a janela Exportação.
2. Clique em **Procurar** para abrir a janela **Salvar Como**. Localize o diretório de saída de destino, especifique o nome do arquivo como ObjectGridPartitionClusterSample e clique em **Salvar**.
3. Clique em **Procurar** para abrir a janela Salvar Como. Localize o diretório de saída de destino e especifique o nome do arquivo como ObjectGridPartitionClusterSample. Clique em **Salvar**.

O arquivo ObjectGridPartitionClusterSample.ear é criado no diretório de saída de destino especificado.

Depois de implementar o arquivo ObjectGridPartitionClusterSample.ear para o recurso de particionamento (WPF), você pode executar o arquivo no WebSphere Extended Deployment. Consulte “Implementando o Arquivo ObjectGridPartitionClusterSample.ear para Funcionar com o Recurso de Particionamento” para obter maiores informações.

Implementando o Arquivo ObjectGridPartitionClusterSample.ear para Funcionar com o Recurso de Particionamento

Se estiver planejando instalar o arquivo ObjectGridPartitionClusterSample.ear no WebSphere Extended Deployment, será necessário desempenhar uma operação wpf-deploy no arquivo.

É necessário ter um arquivo ObjectGridPartitionClusterSample.ear existente. Para modificar o arquivo existente, consulte “Construindo um Aplicativo do ObjectGrid e de um Recurso de Particionamento Integrado” na página 311.

Desempenhe a operação wpf-deploy para preparar o arquivo EAR (Enterprise Archive) em um ambiente do WebSphere Extended Deployment.

1. Crie um diretório DEST_DIR.
2. Copie o arquivo ObjectGridPartitionClusterSample.ear para o diretório DEST_DIR. Renomeie o arquivo ObjectGridPartitionClusterSample.ear para o arquivo old_ObjectGridPartitionClusterSample.ear.
3. Execute o seguinte comando, em que WORKING_DIR é o diretório de trabalho para a ferramenta ejbdeloy, por exemplo, o diretório c:\temp.

   ```
   WAS_HOME\bin\ejbdeploy.bat|ejbdeploy.sh
   DEST_DIR\old_ObjectGridPartitionClusterSample.ear WORKING_DIR
   DEST_DIR\ObjectGridPartitionClusterSample.ear
   ```

4. Execute o seguinte comando, em que TEMP_DIR é um diretório temporário para a ferramenta. Se o argumento -keep for especificado, os diretórios temporários criados pelo utilitário wpfStubUtil não serão excluídos.

```
WAS_HOME\bin\wpfStubUtil.cmd|wpfStubUtil.sh
DEST_DIR\ObjectGridPartitionClusterSample.ear
ObjectGridPartitionClusterSampleEJB.jar com/ibm/websphere/samples/
objectgrid/partitioncluster/ejb/PFClusterObjectGridEJB.class
TEMP_DIR [-stubDebug|-keep]
```

O arquivo `ObjectGridPartitionClusterSample.ear` está pronto para execução em um ambiente do WebSphere Extended Deployment. O arquivo de amostra `D_ObjectGridPartitionClusterSample.ear` fornecido com o WebSphere Extended Deployment já foi implementado. Não é necessário implementar este arquivo antes de instalar o aplicativo, se você não tiver alterado o código fonte.

Instale o arquivo `ObjectGridPartitionClusterSample.ear` no ambiente do WebSphere Extended Deployment com o console administrativo. Consulte “Construindo um Aplicativo do ObjectGrid e de um Recurso de Particionamento Integrado” na página 311 para obter maiores informações.

Exemplo: Programação do ObjectGrid e do Recurso de Particionamento

Este exemplo demonstra como utilizar as funções combinadas do ObjectGrid e do recurso de particionamento em um ambiente Java 2 Platform, Enterprise Edition no WebSphere Extended Deployment.

Finalidade

Além de demonstrar as funções combinadas do ObjectGrid e do recurso de particionamento (WPF), este exemplo também demonstra a propagação e invalidação do listener distribuído do ObjectGrid.

Os pedidos de atualização, inserção e remoção de objetos são roteados para servidores específicos nos quais as partições são hospedadas para as chaves do ObjectGrid correspondentes. Os pedidos de método get do ObjectGrid são gerenciados por carga de trabalho entre todos os servidores.

Este exemplo também ilustra como particionar um grande ObjectGrid em muitos ObjectGrids menores e utilizar o método partitionLoadEvent para pré-carregar dados para que o ObjectGrid particionado possa hospedar um número ilimitado de objetos.

Visão Geral

O arquivo `ObjectGridPartitionClusterSampler.ear` cria um objeto de ações que ilustra como o ObjectGrid e o recurso de particionamento funcionam juntos. O objeto de ações contém as seguintes propriedades:

- registro
- empresa
- serialNumber
- descrição
- lastTransaction
- preço

Em que a propriedade lastTransaction é a hora de alteração das ações. Utilize a propriedade lastTransaction para indicar a atualização de objetos no ObjectGrid de diferentes JVMs (Java Virtual Machines).

Na amostra, a instância do ObjectGrid é criada no método setContext EJB (Enterprise JavaBeans) com a classe ObjectGridFactory.

Defina um conjunto de partições baseadas em hash. O valor padrão são 10 partições, mas é possível alterar o número de partições. Divida os registros de ações nestas partições utilizando o arquivo `SampleUtility.java`. Cada partição pode hospedar muitos pares de chave e valor do ObjectGrid.

A amostra demonstra como os pedidos de inserção, atualização e remoção do ObjectGrid são roteados para um servidor particionado específico e como os pedidos do método get do ObjectGrid são roteados para um servidor específico para sua chave ou qualquer servidor em um cluster. A amostra compara valores de objetos para uma chave de diferentes servidores após a alteração de um valor devido a uma operação de atualização, inserção ou remoção para esta chave em um servidor específico.

Local

Utilize esta amostra em um ambiente em cluster no qual cada servidor pode hospedar muitas partições e no qual cada partição pode hospedar muitos objetos com diferentes chaves.

Existem dois arquivos de amostra do cluster de partição do ObjectGrid no diretório `<raiz_de_instalação>\installableApps\`:

- O arquivo `ObjectGridPartitionClusterSampler.ear` contém o código fonte. Para ver a origem, expanda o arquivo EAR no sistema de arquivos ou importe a origem para um ambiente de desenvolvimento. Consulte “Construindo um Aplicativo do ObjectGrid e de um Recurso de Particionamento Integrado” na página 311 para obter maiores informações.
- O arquivo `D_ObjectGridPartitionClusterSample.ear` já está implementado para o recurso de particionamento. Siga o arquivo leia-me e instruções para colocar este arquivo em execução rapidamente.

Explicação

As seções a seguir incluem uma explicação sobre o aplicativo de amostra do cluster de partição do ObjectGrid:

- “Interface EJB de Operação do ObjectGrid”
- “Classe PartitionKey” na página 318
- “Classe SampleUtility e Mapeamento de Partições” na página 320
- “Criação do ObjectGrid no Método setContext do Bean Corporativo” na página 322
- “Classe ObjectGridFactory Singleton” na página 324
- “Pré-carregamento de Partição do ObjectGrid” na página 325

Interface EJB de Operação do ObjectGrid

Este artigo demonstra a interface EJB (Enterprise JavaBeans) de operação do ObjectGrid que desempenha operações de obtenção, obtenção do servidor particionado, inserção, atualização e remoção.

Finalidade

A interface EJB de operação do ObjectGrid desempenha operações de obtenção, obtenção do servidor particionado, inserção, atualização e remoção. O método de

obtenção do servidor particionado é roteado para uma partição que corresponde à chave solicitada. O método get é roteado em uma estratégia gerenciada por carga de trabalho para qualquer servidor.

A Interface PFClusterObjectGridEJB

O conteúdo da interface PFClusterObjectGridEJB é o seguinte:

```
/**
* Interface Remota para Bean Corporativo: PFClusterObjectGridEJB
*/
public interface PFClusterObjectGridEJB extends javax.ejb.EJBObject {
public String PARTITION_PREFIX = "ObjectGridHashPartition";
/**
* Obter todas as Partições
*
* @return Matriz de Cadeias
* @throws java.rmi.RemoteException
*/
public String [] getAllPartitions() throws java.rmi.RemoteException;
/**
* Obter onde a partição está hospedada
*
* @param partition
* @return String
* @throws java.rmi.RemoteException
*/
public String getServer(String partition)
throws java.rmi.RemoteException;
/**
* Obter objeto de Ações e suas informações do servidor
* (ServerIDResult) para um registro de ações
* de qualquer servidor em um cluster (que tenha tido gerenciamento de
* carga de trabalho)
*
* @param ticket
* @return
* @throws java.rmi.RemoteException
*/
public ServerIDResult getStock(String ticket)
throws java.rmi.RemoteException;
/**
* Obter objeto de Ações e informações do servidor particionado
* para um registro de ações
* da partição para a qual esta chave de registro foi dividida
*
* @param ticket
* @return ServerIDResult
* @throws java.rmi.RemoteException
*/
public ServerIDResult getStockOnPartitionedServer(String ticket)
throws java.rmi.RemoteException;
/**
* Atualizar ações em um servidor específico no qual a partição
* está ativa para esta chave de registro de ações.
*
* @param stock
* @return ServerIDResult
* @throws java.rmi.RemoteException
*/
public ServerIDResult updateStock(Stock stock)
throws java.rmi.RemoteException;
/**
* Remover ações em um servidor específico no qual a partição
* está ativa para esta chave de registro de ações.
*
```

```
* @param ticket
* @return ServerIDResult
* @throws java.rmi.RemoteException
*/
public ServerIDResult removeStock(String ticket)
throws java.rmi.RemoteException;
/**
* Inserir ações em um servidor específico no qual a partição
* está ativa para esta chave de registro de ações.
*
* @param stock
* @return ServerIDResult
* @throws java.rmi.RemoteException
*/
public ServerIDResult insertStock(Stock stock)
 throws java.rmi.RemoteException;
/**
* Recuperar dados de todos os servidores e comparar valores
*
* @param server
* @return ServerObjectGridVerification
* @throws java.rmi.RemoteException
*/
public ServerObjectGridVerification verifyObjectGrid(String server)
 throws java.rmi.RemoteException;
}
```

Classe PartitionKey

A classe PartitionKey controla o comportamento do roteamento baseado em contexto do recurso de particionamento.

O código a seguir ilustra a classe de chave de partição de amostra. Quando o método retorna `não nulo`, ele é roteado com o roteador do recurso de particionamento (WPF). Quando o método retorna `nulo`, ele é redirecionado para o roteador WLM (Gerenciamento de Carga de Trabalho).

```
/**
* PartitionKey para Partitioned Stateless Session Bean WPFKeyBasedPartition
*
*/
public class PFClusterObjectGridEJB_PartitionKey {
/**
* Número de Partições
*
* O padrão é 10.
*
*/
static int numOfPartitions=10;
/**
* Apenas uma vez para getPartitionNumbers
*/
static boolean getNumOfPartitions=true;
/**
* Obter o número de partições
*
*/
static void getPartitionNumbers(){
//obter apenas uma vez
if (getNumOfPartitions){
try {
InitialContext ic = new InitialContext();
PFClusterObjectGridEJBHome home =
(PFClusterObjectGridEJBHome) PortableRemoteObject.narrow(
ic.lookup("java:comp/env/ejb/PFClusterObjectGridEJB"),
PFClusterObjectGridEJBHome.class);
final PFClusterObjectGridEJB session = home.create();
```

```
String[] PARTITIONS = session.getAllPartitions();
numOfPartitions=PARTITIONS.length;
getNumOfPartitions=false;
}
catch (ClassCastException e) {
e.printStackTrace();
numOfPartitions=10;
}
catch (RemoteException e) {
e.printStackTrace();
numOfPartitions=10;
}
catch (NamingException e) {
e.printStackTrace();
numOfPartitions=10;
}
catch (CreateException e) {
e.printStackTrace();
numOfPartitions=10;
}
}
}
/**
* Return partition key
*
  * @param partition
* @return String
*/
public static String getStock(String key) {
   return null;
}
/**
* Return partition key
*
* @param key
* @return String
*/
public static String getServer(String key) {
return key;
}
/**
* Recuperar dados do ObjectGrid de um servidor particionado no qual
* ocorrem alterações de dados (a qualidade e integridade mais altas).
*
* @param ticket
* @return código hash de registro de ações
*/
public static String getStockOnPartitionedServer(String ticket) {
if (ticket==null){
   return null;
}
getPartitionNumbers();
return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Return partition key
*
* @param stock
* @return código hash de registro de ações
*/
public static String updateStock(Stock stock) {
getPartitionNumbers();
String ticket=null;
if (stock!=null){
ticket=stock.getTicket();
}
return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
```

```
}
/**
* Return partition key
*
* @param stock
* @return código hash de registro de ações
*/
public static String insertStock(Stock stock) {
getPartitionNumbers();
String ticket=null;
if (stock!=null){
ticket=stock.getTicket();
}
return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Return partition key
*
* @param server
* @return String
*/
public static String verifyObjectGrid(String server) {
return server;
}
/**
* Return partition key
*
* @param stock
* @return código hash de registro de ações
*/
public static String removeStock(String ticket) {
if (ticket==null){
   return null;
}
getPartitionNumbers();
return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Return partition key
*
  * @param partition
* @return
*/
public static String getAllPartitions() {
   return null;
}
}
```

Cada método remoto deve ter um método correspondente que retorna uma cadeia válida ou um valor nulo.

Classe SampleUtility e Mapeamento de Partições

Utilize o arquivo `SampleUtility.java` para manipular chaves, registros de ações, hash e partições. Também é possível utilizar este arquivo para mapear chaves do ObjectGrid para partições. É possível desenvolver uma classe do utilitário semelhante para mapear chaves do ObjectGrid para partições que podem atender suas necessidades de negócios. Para utilizar o recurso de particionamento com o ObjectGrid, é necessário mapear diferentes chaves para diferentes partições.

Classe SampleUtility

A classe do utilitário para a amostra ObjectGridPartitionCluster é a seguinte:

```
/**
* Classe do utilitário para a amostra ObjectGridPartitionCluster
*
*
*/
public class SampleUtility {
/**
* Contêiner para registro de partições.
*/
static Map serverPartitions= new HashMap();
/**
* Prefixo de nome de partição
*/
public static String PARTITION_PREFIX = "ObjectGridHashPartition";
/**
* Prefixo de nome de ações
*/
public static String STOCK_PREFIX="Stock";
/**
* Recuperar a parte do número do nome da partição
*
  * @param partition
* @return int
*/
public static int getIntFromPartition(String partition){
int result=-1;
int pre=PARTITION_PREFIX.length();
int p=partition.length();
String num=partition.substring(pre, p);
result=Integer.parseInt(num);
return result;
}
/**
* Recuperar a parte do número do registro de ações
*
* @param ticket
* @return
*/
public static int getIntFromStockTicket(String ticket){
int result=-1;
int pre=STOCK_PREFIX.length();
int p=ticket.length();
String num=ticket.substring(pre, p);
result=Integer.parseInt(num);
return result;
}
/**
* Dividir registro de ações para uma base hash especificada.
*
* @param ticket
* @param base
* @return int
*/
public static int hashTicket(String ticket, int base){
if (base<1){
return 0;
}
int hash=0;
int num=getIntFromStockTicket(ticket);
hash= num % base;
return hash;
}
/**
* Dividir chave de ações para uma partição
*
* @param ticket
* @param base
```

```
* @return String - partition name
*/
public static String hashStockKeyToPartition(String ticket, int base){
String p=null;
int hashcode=hashTicket(ticket, base)+1;
p=PARTITION_PREFIX+ padZeroToString(hashcode+"", 6);
return p;
}
/**
* Registrar servidor/partição
*
* @param server
  * @param partition
*/
public static void addServer(String server, String partition){
serverPartitions.put(server, partition);
}
/**
* Remover servidor/partição
*
* @param server
*/
public static void removeServer(String server){
serverPartitions.remove(server);
}
/**
* Obter todos os servidores nos quais as partições estão ativas.
*
* @return Iterator - String
*/
public static Iterator getAllServer(){
return serverPartitions.values().iterator();
}
}
```

É necessário utilizar a mesma base hash global e analisar a variável a ser dividida para a base hash. Considere o seguinte exemplo:

```
myKey.hashCode % hashBase
```

É necessário analisar myKey como a variável hash e manter a mesma base hash entre diferentes servidores. No exemplo anterior, a mesma variável do ambiente Java é consultada. Não é possível utilizar `key1 % 100`, mas é possível utilizar `key2 % 90`.

Criação do ObjectGrid no Método setContext do Bean Corporativo

Crie a instância do ObjectGrid no método setContext do bean corporativo como no arquivo `PFClusterObjectGridEJBBean.java` e recupere os dados de pré-carregamento.

```
/**
* setSessionContext
*
* com a instância do ObjectGrid
*/
public void setSessionContext(javax.ejb.SessionContext ctx) {
          mySessionCtx = ctx;
try {
InitialContext ic = new InitialContext();
//obter PartitionManager
ivManager = (PartitionManager)
ic.lookup("java:comp/websphere/wpf/PartitionManager");
// obter configuração de enableDistribution
boolean enableDistribution = ((Boolean)
ic.lookup("java:comp/env/enableDistribution")).booleanValue();
```

```
System.out.println("***** enableDistribution="+ enableDistribution);
// obter configuração de propagationMode
String propagationMode = (String) ic.lookup("java:comp/env/propagationMode");
System.out.println("***** pMode="+ propagationMode);
String pMode=null;
if (propagationMode.equals(com.ibm.ws.objectgrid.Constants.
OBJECTGRID_TRAN_PROPAGATION_MODE_DEFAULT_KEY)||
propagationMode.equals(com.ibm.ws.objectgrid.Constants.
OBJECTGRID_TRAN_PROPAGATION_MODE_INVALID_KEY) ){
pMode=propagationMode;
}
// obter configuração de propagationVersionOption
String propagationVersionOption = (String)
ic.lookup("java:comp/env/propagationVersionOption");
System.out.println("***** pVersionOption="+ propagationVersionOption);
String pVersion=null;
if (propagationVersionOption.equals(com.ibm.ws.objectgrid.Constants.
OBJECTGRID_TRAN_PROPAGATION_MODE_VERS_KEY)||
propagationMode.equals(com.ibm.ws.objectgrid.Constants.
OBJECTGRID_TRAN_PROPAGATION_MODE_NOVERS_KEY) ){
pVersion=propagationVersionOption;
}
// obter configuração de compressionMode
String compressionMode = (String) ic.lookup("java:comp/env/compressionMode");
System.out.println("***** compressMode="+ compressionMode);
String compressMode=null;
if (compressionMode.equals(com.ibm.ws.objectgrid.Constants.
OBJECTGRID_TRAN_PROPAGATION_COMPRESS_DISABLED)||
propagationMode.equals(com.ibm.ws.objectgrid.Constants.
OBJECTGRID_TRAN_PROPAGATION_COMPRESS_ENABLED) ){
compressMode=compressionMode;
}
// se o pré-carregamento está ativado
bPreload = ((Boolean)
ic.lookup("java:comp/env/preload")).booleanValue();
System.out.println("***** enablePreload="+ bPreload);
//se a remoção está ativada
bRemove = ((Boolean)
ic.lookup("java:comp/env/remove")).booleanValue();
System.out.println("***** enableRemove="+ bRemove);
// se o Loader está ativado
boolean bLoader = ((Boolean)
ic.lookup("java:comp/env/loader")).booleanValue();
System.out.println("***** enableLoader="+ bLoader);
// obter caminho e nome do arquivo
String filePathandName = (String)
ic.lookup("java:comp/env/filePathandName");
System.out.println("***** fileName="+ filePathandName);
//obter instância do ObjectGrid
og=ObjectGridFactory.getObjectGrid(ogName,
enableDistribution,
pMode, pVersion,
compressMode, bLoader,
filePathandName);
if (og==null){
throw new RuntimeException
("ObjectGrid insance is null in ObjectGridPartitionClusterSample");
}
System.out.println("Bean Context, getObjectGrid="
+ og + " for name="+ ogName);
if (bPreload && !lock){
System.out.println("Preload data");
PersistentStore store=PersistentStore.getStore(filePathandName);
store.preload(10);
store.verify(10);
lock=true;
preloadData=store.getAllRecords();
```

```
}
}
catch (Exception e) {
logger.logp(Level.SEVERE, CLASS_NAME,
"setSessionContext", "Exception: " + e);
  throw new EJBException(e);
}
}
```

Classe ObjectGridFactory Singleton

Uma instância do ObjectGrid é criada com um depósito de informações do provedor customizado que armazena em cache a instância do ObjectGrid com configurações customizadas.

A seguir está um exemplo de como criar uma instância do ObjectGrid programaticamente, configurar o objeto ObjectGridTransformer, configurar o listener de eventos de propagação e configurar este listener como a instância do ObjectGrid. Também é possível utilizar um arquivo XML para desempenhar esta configuração.

```
/**
*
* Crie uma instância do ObjectGrid e configure-a.
*
*
*/
public class ObjectGridFactory {
/**
* Nome do ObjectGrid
*/
static String ogName="WPFObjectGridSample";
/**
* Instância do ObjectGrid
*/
static ObjectGrid og=null;
/**
* Sessão do ObjectGrid
*/
static Session ogSession=null;
/**
* Nome do mapa
*/
static String mapName="SampleStocks";
/**
* Cache do ObjectGrid
*/
static Map ogCache= new HashMap();
/**
* Obter instância do ObjectGrid
*
* @param ogn
* @param enableDist
* @param pMode
* @param pVersion
* @param compressMode
* @return
*/
public static synchronized ObjectGrid getObjectGrid(String ogn,
boolean enableDist,
String pMode,
String pVersion,
String compressMode,
boolean loader,
String fileName){
if (ogn!=null){
ogName=ogn;
```

```
}
else {
throw new IllegalArgumentException ("ObjectGrid name given is null");
}
if (ogCache.containsKey(ogName)){
return (ObjectGrid) ogCache.get(ogName);
}
try {
ObjectGridManager manager= ObjectGridManagerFactory.
getObjectGridManager();
og=manager.createObjectGrid(ogName);
if (enableDist){
TranPropListener tpl=new TranPropListener();
if (pMode!=null){
tpl.setPropagateMode(pMode);
}
if (pVersion!=null){
tpl.setPropagateVersionOption(pVersion);
}
if (compressMode!=null) {
tpl.setCompressionMode(compressMode);
}
og.addEventListener(tpl);
}
// Definir BackingMap e configurar o Loader
BackingMap bm = og.defineMap(mapName);
ObjectTransformer myTransformer=
new MyStockObjectTransformer();
bm.setObjectTransformer(myTransformer);
OptimisticCallback myOptimisticCallback=
new MyStockOptimisticCallback();
if (loader){
TransactionCallback tcb=new MyTransactionCallback();
Loader myLoader= new MyCacheLoader(fileName, mapName);
og.setTransactionCallback(tcb);
bm.setLoader(myLoader);
}
og.initialize();
ogCache.put(ogName, og);
}
catch (Exception e) {
}
return og;
}
}
```

Pré-carregamento de Partição do ObjectGrid

Este tópico discute como pré-carregar uma instância do ObjectGrid.

Utilize o método partitionLoadEvent para carregar objetos que estão relacionados a esta partição apenas quando a partição estiver ativada. Ao carregar objetos quando o particionamento está ativado, você particiona o ObjectGrid para que ele possa manipular grandes números de objetos.

```
/**
* É chamado quando uma partição específica é designada a este processo do servidor.
* @param partitionName
* @return
*/
public boolean partitionLoadEvent(String partitionName) {
//pré-carregar dados
preloadDataForPartition(partitionName);
logger.logp(
Level.FINER,
CLASS_NAME,
"partitionLoadEvent",
```

```
"Loading "+ partitionName );
return true;
}
/**
*
* pré-carregar dados
*
* @param partition
*/
private synchronized void preloadDataForPartition(String partition){
if (bPreload && (preloadData!=null)){
Iterator itr=preloadData.keySet().iterator();
while (itr.hasNext()){
String ticket= (String) itr.next();
String p=SampleUtility.
hashStockKeyToPartition(ticket, numOfPartitions);
if (partition.equals(p)){
Stock stock= (Stock) preloadData.get(ticket);
System.out.println("preload in partition=" +
partition + " with data ticket="+ ticket);
insertStock(stock);
}
}
}
}
```

Pode ser necessário desativar as atualizações distribuídas se você utilizar o pré-carregamento particionado de seu ObjectGrid grande para particionar seu ObjectGrid grande. A versão atual de atualizações distribuídas não pode ser particionada. O roteamento baseado em contexto do recurso de particionamento (WPF) localiza os dados corretos na partição correta.

Configurando o ObjectGrid para Funcionar com Beans Gerenciados por Contêiner

Com o WebSphere Application Server Versão 6.0.2 e posterior, é possível utilizar beans CMP (Container-Managed Persistence) com um produto de cache externo.

Utilize esta tarefa para utilizar beans CMP, tirando vantagem do ObjectGrid como um cache externo em vez de um cache interno. Esta funcionalidade é fornecida pelo mecanismo de persistência no WebSphere Application Server.

1. Defina os argumentos de JVM para definir o adaptador CacheFactoryManager e o local do arquivo de configuração XML do ObjectGrid. O CacheFactoryManager é um adaptador entre o mecanismo de persistência e o ObjectGrid.
   a. Clique em **Servidores > Application Servers >** *server_name* **> Java e Gerenciamento de Processo > Definição de Processo > Java Virtual Machine > Argumentos Genéricos de JVM**.
   b. Inclua as seguintes propriedades:
      - `-Dcom.ibm.ws.pmcache.manager=com.ibm.ws.objectgrid.adapter.pm.CacheFactoryManager`
      - `-Dcom.ibm.ws.pmcache.config=file:/d:/temp/objectGrid.xml`

      A propriedade `-Dcom.ibm.ws.pmcache.config` especifica o arquivo de configuração para o ObjectGrid. O valor é uma URL no arquivo de configuração do ObjectGrid.
2. Configure o arquivo de configuração XML do ObjectGrid. A configuração está no arquivo `objectGrid.xml`. Considere o exemplo a seguir. As informações de aplicativo e de módulo são requeridas para o aplicativo J2EE (Java 2 Platform,

Enterprise Edition). As informações são refletidas no arquivo `objectGrid.xml`. Um aplicativo Accounts possui três Enterprise JavaBeans CMP: Savings, Checkin e MoneyMarket. Estes Enterprise JavaBeans estão contidos no módulo PersonalBankingEJB. O nome de exibição é *Accounts* e *PersonalBankingEJB* é o módulo EJB do descritor de implementação do aplicativo. Savings, Checkin e MoneyMarket são os nomes conforme especificados no elemento ejb-name do descritor de implementação Enterprise Java Bean para os Beans CMP (Container-Managed Entity). A seguir está um snippet de amostra para esta configuração:

```
<ObjectGrids>
<ObjectGrid name="Accounts">
<BackingMap name="PersonalBankingEJB.jar#Savings" readOnly="true"
pluginCollectionRef="default" />
<BackingMap name="PersonalBankingEJB.jar#Checkin" readOnly="true"
pluginCollectionRef="default" />
<BackingMap name="PersonalBankingEJB.jar#MoneyMarket" readOnly="true"
pluginCollectionRef="default" />
</ObjectGrid>
</ObjectGrids>
```

O arquivo `PersonalBankingEJB.jar` está especificado nas tags EJB do descritor de implementação do aplicativo, conforme o seguinte exemplo:

```
<module id="module_1">
<ejb>PersonalBankingEJB.jar</ejb>
</module>
```

3. Ative a configuração do LifeTimeInCache de gerenciador de persistência para cada bean, no aplicativo para utilizar o cache externo. O ObjectGrid requer que você ative esta configuração no descritor de implementação, mas ignora a configuração de LifeTimeInCache. A configuração do ObjectGrid tem precedência.
4. Configure explicitamente um backingMap para liberar objetos do cache. A seguir está um trecho de código XML para o arquivo `objectGrid.xml`:

```
<backingMapPluginCollection id="TotalTimeToLive">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.TTLEvictor">
<property name="pruneSize" type="int" value="2"
description="set max size for TTL Evictor" />
<property name="numberOfHeaps" type="int" value="1"
description="set number of TTL heaps" />
<property name="sleepTime" type="int" value="1"
description="evictor thread sleep time" />
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="LifeTimeInCache">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.TTLEvictor">
<property name="lifeTime" type="int" value="3"
description="lifetime of map entry is 3 seconds" />
<property name="pruneSize" type="int" value="2"
description="set max size for TTL Evictor" />
<property name="numberOfHeaps" type="int" value="1"
description="set number of TTL heaps" />
<property name="sleepTime" type="int" value="1"
description="evictor thread sleep time" />
</bean>
</backingMapPluginCollection>
```

A propriedade lifeTime controla o evictor.

Consulte Capítulo 10, “Integrando o ObjectGrid com o WebSphere Application Server”, na página 291 para obter informações adicionais sobre como utilizar o ObjectGrid com o WebSphere Application Server.

Capítulo 11. Boas Práticas de Desempenho do ObjectGrid

É possível aprimorar o desempenho de um Mapa do ObjectGrid com as boas práticas a seguir. Estas boas práticas são implementadas apenas no contexto do aplicativo e de sua arquitetura.

Cada aplicativo e ambiente utilizam uma solução diferente para desempenho. O ObjectGrid fornece customizações internas para aprimorar o desempenho, mas também é possível aprimorar o desempenho na arquitetura do aplicativo. As áreas a seguir oferecem aprimoramentos de desempenho:

- "Boas Práticas de Desempenho de Bloqueio"

  Escolha entre as diferentes estratégias de bloqueio que podem afetar o desempenho de seus aplicativos.
- "Boas Práticas do Método copyMode" na página 330

  Escolha entre os diferentes modos de cópia que podem ser utilizados para alterar como o ObjectGrid mantém e copia entradas.
- "Boas Práticas da Interface ObjectTransformer" na página 334

  Utilize a interface ObjectTransformer para permitir que retornos de chamada para o aplicativo forneçam implementações customizadas de operações comuns e caras, como serialização de objeto e uma cópia detalhada em um objeto.
- "Boas Práticas de Desempenho do Evictor de Plug-in" na página 336

  Escolha entre estratégias de evicção LFU (Least Frequently Used) e LRU (Least Recently Used).
- "Boas Práticas do Evictor Padrão" na página 337

  As propriedades para o evictor TTL (Time to Live) padrão, o evictor padrão criado com cada backingMap.

Boas Práticas de Desempenho de Bloqueio

As estratégias de bloqueio podem afetar o desempenho de seus aplicativos.

Para obter detalhes adicionais sobre as seguintes estratégias de bloqueio, consulte o tópico "Bloqueio" na página 130.

Estratégia de Bloqueio Pessimista

É possível utilizar a estratégia de bloqueio pessimista para operações de leitura e gravação de mapas nas quais as chaves geralmente colidem. A estratégia de bloqueio pessimista tem o maior impacto no desempenho.

Estratégia de Bloqueio Otimista

O bloqueio otimista é a configuração padrão. Esta estratégia aprimora o desempenho e a escalabilidade sobre a estratégia pessimista. Utilize esta estratégia quando seus aplicativos puderem tolerar algumas falhas de atualização otimistas, enquanto houver um desempenho melhor do que a estratégia pessimista. Esta estratégia funciona bem para aplicativos que são muito lidos e com atualizações não freqüentes.

Estratégia de Bloqueio None

A utilização da estratégia de bloqueio none é recomendável para aplicativos de leitura. A estratégia de bloqueio none não obtém nenhum bloqueio. Portanto, ela oferece a maior simultaneidade, desempenho e escalabilidade.

Boas Práticas do Método copyMode

O ObjectGrid faz uma cópia do valor baseado na configuração de CopyMode. É possível utilizar o método setCopyMode(CopyMode, valueInterfaceClass) da API de BackingMap para configurar o modo de cópia como um dos seguintes campos estáticos finais que estão definidos na classe com.ibm.websphere.objectgrid.CopyMode.

Quando um aplicativo utiliza a interface ObjectMap para obter uma referência a um valor de mapa, é recomendável utilizar essa referência apenas na transação do ObjectGrid que obteve a referência. A utilização da referência em uma transação diferente do ObjectGrid pode conduzir a erros. Por exemplo, se você utilizar a estratégia de bloqueio pessimista para o BackingMap, uma chamada de método get ou getForUpdate adquire um bloqueio S (compartilhado) ou U (atualização), respectivamente. O método get retorna a referência ao valor e o bloqueio 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 ao valor, chamando o método get ou getForUpdate em vez de reutilizar a mesma referência de valor em várias transações.

Utilize as seguintes informações para escolher entre os modos de cópia com as seguintes informações:

Modo 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. Este modo assegura que um aplicativo não contém uma referência ao objeto de valor que está no BackingMap e, em vez disso, o aplicativo está sempre funcionando com uma cópia do valor que está no BackingMap. O modo COPY_ON_READ_AND_COMMIT assegura que o aplicativo nunca pode danificar inadvertidamente os dados que estão armazenados 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, as alterações confirmadas pelo aplicativo serão copiadas para o BackingMap para assegurar que o aplicativo não possui uma referência ao valor confirmado no BackingMap.

Modo 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 este modo, o método ObjectMap.get retorna uma cópia do valor em vez de uma referência ao valor para assegurar que as alterações feitas pelo aplicativo no valor não afetarão 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 fizer isso, ele poderá danificar os dados armazenados em cache no BackingMap. 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.

Modo 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 implementação de *cópia na 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 utilizar os métodos nesta interface ao interagir com o valor em uma Sessão do ObjectGrid. Se este for o caso, o ObjectGrid criará proxies para os valores retornados ao aplicativo. O proxy tem uma referência para ser valor real. Se o aplicativo apenas faz leituras, ele sempre será executado na 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. Isto permite que a operação de cópia seja totalmente evitada para objetos que são lidos apenas pelo aplicativo. Todas as operações de modificação devem começar com o prefixo configurado. Os Enterprise JavaBeans normalmente são codificados para utilizar este estilo de nomenclatura de método para métodos que modificam os atributos do objeto. Esta convenção deve ser seguida. Os objetos modificados são copiados no momento em que são modificados pelo aplicativo. Este é o cenário de leitura e gravação mais eficiente suportado pelo ObjectGrid. É possível configurar um mapa para utilizar COPY_ON_WRITE da seguinte forma. Neste exemplo, o aplicativo deseja armazenar objetos Person chaveados utilizando o nome no Mapa. O objeto pessoal é semelhante ao seguinte trecho 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 a interface IPerson apenas 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");
// utilizar COPY_ON_WRITE para este Mapa com
// IPerson como a Classe valueProxyInfo
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// O aplicativo deve então utilizar o seguinte padrão
// ao utilizar o Mapa PERSON.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// o aplicativo lança o valor retornado em IPerson e não em Person
IPerson p = (IPerson)person.get("Billy");
p.setAge( p.getAge() + 1 );
...
// criar um novo Person e incluir no Mapa
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// o seguinte snippet NÃO FUNCIONARÁ. Isto resultará na ClassCastException
sess.begin();
// o erro aqui é que é utilizado Person em vez de
// 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 e não para o objeto Person 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 Bobby e tenta lançá-lo em 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.

Interface ValueProxyInfo e Suporte Parcial à Atualização

Esta interface permite que um aplicativo recupere o valor de leitura confirmado referido 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 loader baseado em JDBC (Java Database Connectivity) ou 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 loader estiver conectado, o loader poderá lançar os valores dos objetos modificados na interface ValueProxyInfo para obter estas informações.

Manipulando o Método Equals ao Utilizar COPY_ON_WRITE ou Proxies.

Por exemplo, o código a seguir constrói um objeto Person e, em seguida, insere-o 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();
// novo objeto Person
Person p = new Person(...);
personMap.insert(p.getName, p);
// recupere-o novamente, lembre-se de utilizar a interface para o lançamento
IPerson p2 = personMap.get(p.getName());
if(p2 == p)
{
// eles são iguais
}
```

```
else
{
// não são
}
```

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.

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

NO_COPY mode

O modo NO_COPY permite que um aplicativo assegure que ele nunca modificará um objeto de valor obtido utilizando um 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, 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.

Boas Práticas da Interface ObjectTransformer

O ObjectTransformer utiliza retornos de chamada para o aplicativo para fornecer implementações customizadas de operações comuns e caras, como serialização de objetos e uma cópia detalhada em um objeto.

Para obter detalhes específicos sobre a interface ObjectTransformer, consulte o tópico “Plug-in ObjectTransformer” na página 213. Do ponto de vista de desempenho e de informações do método CopyMode no tópico “Boas Práticas do Método copyMode” na página 330, é evidente que o ObjectGrid copia os valores para todos os casos, exceto quando no modo NO_COPY. O mecanismo de cópia padrão empregado no ObjectGrid é a serialização, que é conhecida como uma operação cara. A interface ObjectTransformer pode ser utilizada nesta situação. A interface ObjectTransformer utiliza retornos de chamada para o aplicativo para fornecer uma implementação customizada de operações comuns e caras, como serialização de objetos e cópias detalhadas em objetos.

Um aplicativo pode fornecer uma implementação da interface ObjectTransformer para um mapa. O ObjectGrid então é delegado aos métodos neste objeto e depende do aplicativo para fornecer 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 a um BackingMap utilizando o seguinte código de exemplo:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ajustar a Serialização e Inflação de Objetos

A serialização de objetos geralmente é o maior problema de desempenho com o ObjectGrid. O ObjectGrid utiliza o mecanismo seriável padrão se um plug-in do ObjectTransformer não for fornecido pelo aplicativo. Um aplicativo pode fornecer implementações de Serializable readObject e de writeObject ou pode fazer os objetos implementarem a interface Externalizable, que é aproximadamente 10 vezes mais rápida. Se os objetos no Mapa não puderem ser modificados, um aplicativo poderá associar um 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. Os métodos serialize serializam o objeto e fornecem um fluxo. O método serializa o método para o fluxo fornecido. Os métodos inflate fornecem um fluxo de entrada e esperam que o aplicativo crie o objeto, infle-o utilizando dados do fluxo e, em seguida, retorne o objeto. As implementações dos métodos serialize e inflate devem espelhar uma as outras.

Ajustar Operações de Cópia Detalhada

Após um aplicativo receber um objeto de um ObjectMap, o ObjectGrid desempenha uma cópia detalhada no valor de objeto para assegurar que a cópia no mapa BaseMap permaneça segura. 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 mas, neste caso, o custo do desempenho desta ação foi equilibrado pela instrução ao programador do aplicativo para não utilizar o valor após a confirmação da transação. O mecanismo de cópia de objeto 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, forneça um plug-in ObjectTransformer e implemente os métodos `copyValue` e `copyKey` utilizando uma cópia de objeto mais eficiente.

Boas Práticas de Desempenho do Evictor de Plug-in

Se estiver utilizando evictores de plug-in, eles não estarão ativos até que você os crie e instrua o mapa de suporte a utilizá-los. Utilize estas boas práticas e dicas de desempenho para evictores LFU (Least Frequently Used) e LRU (Least Recently Used).

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 Frequently 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. Por isso, o início da fila contém as entradas do mapa Most Recently Used e o final contém as entradas do mapa Least Recently Used. 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. Em vez disso, ele fica suspenso por um período de tempo definido, sendo ativado apenas a cada *n* número de segundos, em que *n* refere-se 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. No entanto, a utilização não suficiente do evictor pode fazer com que você fique com um mapa de entradas desnecessárias. 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 evicção 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, pode ser mais útil configurar o valor com um tempo inferior. No entanto, se o mapa for acessado com pouca freqüência, será possível configurar o tempo com um valor superior.

Exemplo:

O exemplo a seguir define um mapa, cria um novo evictor LFU, configura as propriedades do evictor e configura o mapa para utilizar o evictor:

```
//Utilizar ObjectGridManager para criar/obter o ObjectGrid. Consulte
// a seção do ObjectGridManger
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");
//Configurar propriedades assumindo 50.000 entradas do mapa
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

A utilização do evictor LRU é muito semelhante à utilização de um evictor LFU. Segue um exemplo:

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
//Configurar propriedades assumindo 50.000 entradas do mapa
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Observe que apenas 2 linhas são diferentes do exemplo LFUEvictor.

Boas Práticas do Evictor Padrão

Boas práticas para o evictor *time to live* padrão.

Além dos evictores de plug-in descritos no tópico “Boas Práticas de Desempenho do Evictor de Plug-in” na página 336, é criado um evictor TTL padrão com cada mapa de suporte. O evictor padrão remove entradas com base em um conceito *time to live*. Este comportamento é definido pelo atributo ttlType. Existem três atributos ttlTypes:

- **Nenhum** : Especifica que as entradas nunca expiram e, portanto, nunca são removidas do mapa.
- **Hora de Criação** : Especifica que as entradas são liberadas dependendo de quando foram criadas.
- **Hora do Último Acesso** : Especifica que as entradas são liberadas dependendo de quando foram acessadas pela última vez.

Propriedades do Evictor Padrão e Boas Práticas para Desempenho

Propriedade TimeToLive

Esta propriedade, junto com a propriedade ttlType, é a mais importante, de uma perspectiva de desempenho. Se estiver utilizando CREATION_TIME ttlType, o evictor liberará uma entrada quando sua hora de criação for igual a seu valor de atributo TimeToLive. Se você configurar o valor de atributo TimeToLive como 10 segundos, tudo o que estiver em todo o mapa será liberado após dez segundos. 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.

A seguir está um exemplo de onde um tipo TTL de CREATION_TIME é útil. Você está utilizando um aplicativo da Web que obtém preços de ações e a obtenção dos preços mais recentes não é crítica. 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. Aproximadamente a cada vinte minutos, o mapa do ObjectGrid utiliza o plug-in do Loader para atualizar os dados do mapa com dados atuais do banco de dados. O banco de dados é atualizado a cada 20 minutos com os preços de ações mais recentes. Portanto, para este aplicativo, é recomendável a utilização de um valor TimeToLive de 20 minutos.

Se estiver utilizando o atributo LAST_ACCESSED_TIME ttlType, configure o TimeToLive como um número inferior do que se estiver utilizando CREATION_TIME ttlType, porque o atributo TimeToLive de entradas é reconfigurado sempre 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.

A seguir está um exemplo de onde um tipo TTL de LAST_ACCESSED_TIME é útil. Um mapa do ObjectGrid é utilizado para reter dados de sessão de um cliente. 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, a utilização de um tipo TTL de

LAST_ACCESSED_TIME com o atributo TimeToLive configurado como 30 minutos é exatamente o que é necessário para este aplicativo.

Exemplo

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_ACCESSED_TIME);
bMap.setTimeToLive(15);
```

A maioria das configurações do evictor devem ser configuradas antes da inicialização do ObjectGrid. Para obter um entendimento mais detalhado dos evictores, consulte “Evictors” na página 191.

Capítulo 12. Distribuindo Alterações entre Java Virtual Machines no Mesmo Nível

Os objetos LogSequence e LogElement comunicam as alterações ocorridas em uma transação do ObjectGrid com um plug-in.

Para obter informações adicionais sobre como o JMS (Java Message Service) pode ser utilizado para distribuir alterações transacionais, consulte “Java Message Service para Distribuir Alterações de Transação” na página 344.

Um pré-requisito é que a instância do ObjectGrid deve ser armazenada em cache pelo ObjectGridManager. Consulte “Métodos createObjectGrid” na página 93 para obter informações adicionais. O valor booleano cacheInstance deve ser configurado como true.

Os objetos fornecem meios de um aplicativo publicar facilmente alterações ocorridas em uma grade de objetos utilizando um transporte de mensagens para ObjectGrids no mesmo nível em JVMs (Java Virtual Machines) remotas e, em seguida, aplicar estas alterações nessa JVM. A classe LogSequenceTransformer é importante para ativar este suporte. Este artigo examina como gravar um listener utilizando um sistema de mensagens JMS (Java Message Service) para propagar as mensagens.

O ObjectGrid suporta a transmissão de LogSequences resultantes de uma confirmação de transação do ObjectGrid nos membros de cluster do WebSphere Application Server com um plug-in fornecido pela IBM. Esta função não é ativada por padrão, mas pode ser configurada para ser operacional. No entanto, quando o consumidor ou produtor não for um WebSphere Application Server, pode ser requerida a utilização de um sistema de mensagens JMS externo.

1. Inicialize o plug-in. O ObjectGrid chama o método `initialize` do plug-in, parte do contrato da interface ObjectGridEventListener quando o ObjectGrid é iniciado. O método `initialize` deve obter seus recursos JMS, incluindo conexões, sessões e publicadores e iniciar o encadeamento no listener JMS. O método initialize é semelhante ao seguinte exemplo:

```
public void initialize(Session session)
{
mySession = session;
myGrid = session.getObjectGrid();
try
{
if(mode == null)
{
throw new ObjectGridRuntimeException("No mode specified");
}
if(userid != null)
{
connection = topicConnectionFactory.createTopicConnection(
userid, password);
}
else
connection = topicConnectionFactory.createTopicConnection();
// é necessário iniciar a conexão para receber mensagens.
connection.start();// a sessão jms não é transacional (false).
jmsSession = connection.createTopicSession(false,
javax.jms.Session.AUTO_ACKNOWLEDGE);
if(topic == null)
if(topicName == null)
{
```

```
throw new ObjectGridRuntimeException("Topic not specified");
}
else
{
topic = jmsSession.createTopic(topicName);
}
publisher = jmsSession.createPublisher(topic);
// iniciar o encadeamento do listener.
listenerRunning = true;
listenerThread = new Thread(this);
listenerThread.start();
}
catch(Throwable e)
{
throw new ObjectGridRuntimeException("Cannot initialize", e);
}
}
```

O código para iniciar o encadeamento utiliza um encadeamento J2SE (Java 2 Platform, Standard Edition). Se estiver executando um servidor do WebSphere Application Server Versão 6.x ou do WebSphere Application Server Versão 5.x Enterprise, utilize a API (Interface de Programação de Aplicativo) do bean assíncrono para iniciar este encadeamento do daemon. Também é possível utilizar as APIs comuns. A seguir está um snippet de substituição de exemplo que mostra a mesma ação utilizando um gerenciador de trabalho:

```
// iniciar o encadeamento do listener.
listenerRunning = true;
workManager.startWork(this, true);
```

O plug-in também deve implementar a interface Work em vez da interface Runnable. Também é necessário incluir um método de liberação para configurar a variável listenerRunning como false. O plug-in deve ser fornecido com uma instância do gerenciador de trabalho em seu construtor ou por injeção, se estiver utilizando um contêiner IoC (Inversion of Control).

2. Transmita as alterações. A seguir está um método transactionEnd de amostra para publicar as alterações locais feitas em um ObjectGrid. Isto utiliza o JMS, embora claramente, seja possível utilizar qualquer transporte de mensagens que seja capaz de publicar e assinar o sistema de mensagens de forma confiável.

```
// Este método é sincronizado para assegurar que
// as mensagens sejam publicadas na ordem de confirmação
// da transação. Se tivermos iniciado a publicação de mensagens
// em paralelo, os receptores poderiam danificar o Mapa,
// pois as exclusões podem chegar antes das inserções, etc.
public synchronized void transactionEnd(String txid,
boolean isWriteThroughEnabled,
boolean committed, Collection changes)
{
try
{
// deve ter gravação direta e confirmada.
if(isWriteThroughEnabled && committed)
{
// gravar as seqüências em um byte []
ByteArrayOutputStream bos = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(bos);
if (publishMaps.isEmpty()) {
// serializar toda a coleta
LogSequenceTransformer.serialize(changes, oos, this, mode);
}
else {
// filtrar LogSequences com base no conteúdo de publishMaps
```

```
Collection publishChanges = new ArrayList();
Iterator iter = changes.iterator();
while (iter.hasNext()) {
LogSequence ls = (LogSequence) iter.next();
if (publishMaps.contains(ls.getMapName())) {
publishChanges.add(ls);
}
}
LogSequenceTransformer.serialize(publishChanges, oos, this,
mode);
}
// gerar uma mensagem de objeto para as alterações
oos.flush();ObjectMessage om = jmsSession.createObjectMessage(
bos.toByteArray());
// configurar propriedades
om.setStringProperty(PROP_TX, txid);
om.setStringProperty(PROP_GRIDNAME, myGrid.getName());
// transmitir.
publisher.publish(om);
}
}
catch(Throwable e)
{
throw new ObjectGridRuntimeException("Cannot push changes", e);
}
}
```

Este método utiliza diversas variáveis de instância:

- Variável **jmsSession**: Uma sessão JMS utilizada para publicar mensagens. É criada durante a inicialização do plug-in
- Variável **mode**: O modo de distribuição.
- Variável **publishMaps**: Um conjunto que contém o nome de cada Mapa com alterações para publicação. Se a variável estiver vazia, todos os Mapas serão publicados.
- Variável **publisher**: Um objeto TopicPublisher criado durante o método initialize do plug-in.

3. Receba e aplique mensagens de atualização. A seguir está o método de execução. Este método é executado em um loop até que o aplicativo pare o loop. Cada iteração do loop tenta receber uma mensagem JMS e aplicá-la ao ObjectGrid.

```
private synchronized boolean isListenerRunning()
{
return listenerRunning;
}
public void run()
{
try
{
System.out.println("Listener starting");
// obter uma sessão jms para receber as mensagens.
// Não transacional.
TopicSession myTopicSession;
myTopicSession = connection.createTopicSession(false,
javax.jms.Session.AUTO_ACKNOWLEDGE);
// obter um assinante para o tópico; true indica não receber
// mensagens transmitidas utilizando publicadores
// nesta conexão. Caso contrário, receberíamos nossas próprias atualizações.
TopicSubscriber subscriber = myTopicSession.createSubscriber(topic,
null, true);
System.out.println("Listener started");
while(isListenerRunning())
{
```

```
ObjectMessage om = (ObjectMessage)subscriber.receive(2000);
if(om != null)
{
// Utilizar Sessão que foi transmitida na inicialização...
// muito importante utilizar a opção sem gravação direta aqui
mySession.beginNoWriteThrough();
byte[] raw = (byte[])om.getObject();
ByteArrayInputStream bis = new ByteArrayInputStream(raw);
ObjectInputStream ois = new ObjectInputStream(bis);
// inflar LogSequences
Collection collection = LogSequenceTransformer.inflate(ois,
myGrid);
Iterator iter = collection.iterator();
while (iter.hasNext()) {
// processar cada uma as alterações dos Mapas de acordo com o modo quando
// LogSequence foi serializado
LogSequence seq = (LogSequence)iter.next();
mySession.processLogSequence(seq);
}
mySession.commit();
} // se houver uma mensagem
} // loop while
// parar a conexão
connection.close();
}
      catch (IOException e)
{
System.out.println("IO Exception: " + e);
}
catch(JMSException e)
{
System.out.println("JMS Exception: " + e);
}
catch(ObjectGridException e)
{
System.out.println("ObjectGrid exception: " + e);
System.out.println("Caused by: " + e.getCause());
}
catch(Throwable e)
{
System.out.println("Exception : " + e);
}
System.out.println("Listener stopped");
}
```

A classe LogSequenceTransformer e as APIs ObjectGridEventListener, LogSequence e LogElement permitem que qualquer publicação e assinatura confiáveis sejam utilizadas para distribuir as alterações e filtrar os Mapas que você deseja distribuir. Os snippets nesta tarefa mostram como utilizar estas APIs com JMS para construir um ObjectGrid no mesmo nível que seja compartilhado por aplicativos hospedados em um conjunto de plataformas diferente que compartilham um transporte de mensagens comum.

Java Message Service para Distribuir Alterações de Transação

Utilize o JMS (Java Message Service) para alterações distribuídas 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 utilizam o ObjectGrid podem ser implementados no Gluecode ou no Tomcat, enquanto outros aplicativos podem ser executados no WebSphere Application Server Versão 6.0. O JMS é ideal para alterações distribuídas entre pares do

ObjectGrid nestes ambientes diferentes. O transporte de mensagens do gerenciador de alta disponibilidade é muito rápido, mas pode distribuir alterações apenas para JVMs que estão em um único grupo principal. O JMS é mais lento, mas permite que um conjunto maior e mais variado de clientes aplicativos compartilhe um ObjectGrid. O JMS é ideal, por exemplo, ao compartilhar dados em um ObjectGrid entre um cliente fat Swing e um aplicativo implementado no WebSphere Extended Deployment.

Visão Geral

O JMS é implementado para distribuir alterações de transação utilizando um objeto Java que se comporta como um listener ObjectGridEventListener. Este objeto pode propagar o estado nas quatro maneiras a seguir:

Invalidar
: Qualquer entrada liberada, atualizada ou excluída é removida em todas as JVMs (Java Virtual Machines) de mesmo nível quando recebem a mensagem.

Invalidar Condicional
: A entrada será liberada apenas se a versão local for igual ou anterior à versão do publicador.

Enviar
: Qualquer entrada liberada, atualizada, excluída ou inserida é incluída ou sobrescrita em todas as JVMs de mesmo nível quando recebem a mensagem JMS.

Enviar Condicional
: A entrada será 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 ObjectGrid chama este método, o plug-in tenta converter a lista LogSequence para cada Mapa acessado pela transação em uma mensagem JMS e, em seguida, publicá-la. 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. Quando todas as LogSequences forem serializadas no fluxo, será criada uma ObjectMessage JMS e publicada 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, é iniciada uma transação sem gravação direta. 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 informações adicionais sobre como utilizar o JMS para distribuir alterações de transação, consulte o Capítulo 12, “Distribuindo Alterações entre Java Virtual Machines no Mesmo Nível”, na página 341.

Capítulo 13. Integração de Contêiner Baseada em Injeção

É possível utilizar a estrutura IoC (Inversion of Control) para configurar totalmente o ObjectGrid. Por isso, não é necessário utilizar a estrutura de configuração XML do ObjectGrid.

Contêineres Baseados em Injeção

Contêineres baseados em injeção, também conhecidos como IoC, são um padrão comum utilizado por aplicativos no lado cliente e no lado do servidor. Existem várias implementações de código aberto destes contêineres. A nova especificação de EJB (Enterprise JavaBeans) Versão 3.0 também empresta alguns destes conceitos. A maioria destas estruturas são contêineres Enterprise JavaBean e são responsáveis por criar uma instância de um bean específico. Estas estruturas também podem inicializar um bean com um conjunto de propriedades e ligam outros Enterprise JavaBeans requeridos utilizando pares getter e setter nos atributos de Enterprise JavaBean. As APIs (Interfaces de Programação de Aplicativo) do ObjectGrid foram projetadas para funcionarem bem com tais contêineres. Começando com os métodos ObjectGridManager.createObjectGrid, é possível configurar estes contêineres para auto-inicializar um ObjectGrid para que o aplicativo tenha uma referência a um ObjectGrid de trabalho ou solicitar que o contêiner forneça uma sessão do ObjectGrid quando necessário.

Padrões Suportados

As seções a seguir discutem o que foi feito para verificar se as APIs do ObjectGrid podem ser utilizadas de forma transparente por um aplicativo que emprega uma estrutura IoC.

Utilizar o ObjectGridManager para criar ObjectGrids

As APIs do ObjectGrid foram projetadas para funcionarem bem com estruturas IoC. O singleton raiz utilizado por tal estrutura é a interface ObjectGridManager, que possui vários métodos do depósito de informações do provedor `createObjectGrid` que retornam uma referência a um ObjectGrid denominado. É possível configurar esta referência do ObjectGrid como um singleton na estrutura IoC, para que os pedidos seguintes para o bean retornem a mesma instância do ObjectGrid.

Plug-ins do ObjectGrid

Os plug-ins no ObjectGrid incluem:

- TransactionCallback
- ObjectGridEventListener
- SubjectSource
- MapAuthorization
- SubjectValidation

Cada um dos plug-ins são JavaBeans simples que implementam uma interface. É possível utilizar a estrutura IoC para criar estes plug-ins e ligá-los aos atributos apropriados na instância do ObjectGrid. Cada um destes plug-ins tem um método set correspondente na interface ObjectGrid para uma integração transparente com a estrutura IoC.

Criar Mapas

O método do depósito de informações do provedor createMap na interface ObjectGrid pode ser utilizado para criar um Mapa recém-denominado. Os plug-ins requeridos pelo BackingMap (o objeto retornado por createMap) são construídos utilizando a estrutura IoC e, em seguida, ligados ao BackingMap utilizando o nome de atributo apropriado. Como o BackingMap não é referido por nenhum outro objeto, ele não é construído automaticamente pelas estruturas IoC. É possível ligar cada BackingMap a um atributo fictício no bean do aplicativo principal como uma solução alternativa. Utilize o método setMaps() para limpar quaisquer BackingMaps que tenham sido definidos anteriormente neste ObjectGrid e substituí-los pela lista de BackingMaps fornecidos.

Plug-ins Backingmap

Os plug-ins no BackingMap se comportam de maneira semelhante aos do ObjectGrid. Cada plug-in tem um método `set` correspondente na interface BackingMap. Os plug-ins BackingMap são:

- Loader
- ObjectTransformer
- OptimisticCallback
- Evictor
- MapEventListener

Padrões de Uso

Quando a estrutura IoC tiver seu arquivo de configuração configurado para produzir um ObjectGrid, crie um bean corporativo que seja chamado de `gridName_Session` ou algo semelhante. Defina-o como um Enterprise JavaBean obtido chamando o método `getSession` no Enterprise JavaBean do singleton do ObjectGrid. O aplicativo então utiliza a estrutura IoC para obter uma referência a um objeto gridName_Session sempre que um objeto requer uma nova sessão.

Resumo

É fácil a utilização do ObjectGrid em ambientes que já utilizam uma estrutura IoC para instanciação e configuração do bean. É possível utilizar a estrutura IoC para configurar totalmente o ObjectGrid e, portanto, não é necessário utilizar a estrutura de configuração XML. O ObjectGrid funciona corretamente com sua estrutura IoC existente.

Capítulo 14. Resolução de Problemas

Esta seção descreve cenários para resolução de problemas que são causados por um erro de aplicativo ou por um problema de design do aplicativo. Se você suspeitar que o ObjectGrid tem um defeito, poderá ser necessário ativar o rastreio do ObjectGrid, conforme descrito na seção de rastreio do ObjectGridManager.

Erros Intermitentes e Inexplicáveis

Um aplicativo tenta aprimorar o desempenho utilizando o modo de cópia COPY_ON_READ, COPY_ON_WRITE ou NO_COPY conforme descrito na seção CopyMode. O aplicativo encontra problemas intermitentes quando o sintoma do problema está sendo alterado e o problema é inexplicável ou inesperado. Os problemas intermitentes não ocorrem quando o modo de cópia é alterado para o modo 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

A solução é para o aplicativo ficar de acordo com o contrato de programação estabelecido para o modo de cópia que está sendo utilizado. Para os modos de cópia COPY_ON_READ e COPY_ON_WRITE, isto significa que o aplicativo utiliza uma referência a um objeto de valor fora do escopo da transação no qual a referência de valor foi obtida. Para utilizar estes modos, o aplicativo deve concordar em destruir a referência ao objeto de valor após a conclusão da transação e obter uma nova referência ao objeto de valor em cada transação que precisa acessar o objeto de valor. Para o modo de cópia NO_COPY, o aplicativo deve concordar em nunca alterar o objeto de valor. Neste caso, o aplicativo deve ser alterado para que não altere o objeto de valor ou o aplicativo deve utilizar um modo de cópia diferente. Consulte a seção CopyMode para obter detalhes adicionais sobre a configuração do modo de cópia.

Técnica Geral de Manipulação de Exceção

Saber a causa raiz de um objeto Throwable é útil no isolamento da origem de um problema. A seguir está um exemplo de um método utilitário que pode ser utilizado por uma rotina de tratamento de exceção para localizar a causa raiz do Throwable ocorrido.

Exemplo:

```
static public Throwable findRootCause( Throwable t )
{
// Inicie com o Throwable que ocorreu como a raiz.
Throwable root = t;
// Siga a cadeia de causas até localizar o último Throwable na cadeia.
Throwable cause = root.getCause();
while ( cause != null )
{
root = cause;
cause = root.getCause();
```

```
}
// Retorne o último Throwable na cadeia como a causa raiz.
return root;
}
```

Técnicas Específicas de Manipulação de Exceção

Inserção Duplicada

Geralmente, este problema deve ocorrer apenas em um ambiente de propagação de transação distribuída. Ele não ocorre com freqüência.

Mensagem

```
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl < processLogSequence Exit
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl > commit for:
TX:08FE0C67-0105-4000-E000-1540090A5759 Entry
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl > rollbackPMapChanges for:
TX:08FE0C67-0105-4000-E000-1540090A5759
as result of Throwable: com.ibm.websphere.objectgrid.plugins.
CacheEntryException:
Duplicate key on an insert!
Entry com.ibm.websphere.objectgrid.plugins.CacheEntryException:
Duplicate key on an insert!
at com.ibm.ws.objectgrid.map.BaseMap.applyPMap(BaseMap.java:528)
at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:405)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.commitPropagatedLogSequence
(TranPropWorkerThread.java:553)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.processCommitRequest
(TranPropWorkerThread.java:449)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.run
(TranPropWorkerThread.java:200)
at java.lang.Thread.run(Thread.java:568)
```

Problema

Quando a seqüência de registro filtrada é propagada de uma JVM para outra, a seqüência de registro externa é processada na segunda JVM. A entrada para esta chave pode existir ou os códigos da operação de seqüência de registro devem ser diferentes. Este problema ocorre ocasionalmente.

Impacto e Solução

Quando este problema ocorre, a entrada não é atualizada em outra JVM, o que pode causar uma inconsistência no ObjectGrid. No entanto, existe uma solução alternativa para evitar este problema. É possível utilizar o recurso de particionamento (WPF) na recuperação de objetos, além da inserção/atualização/remoção de objetos. Consulte a seção Integrando o WPF e o ObjectGrid para obter informações adicionais sobre esta técnica.

Exceção de Colisão Otimista

É possível receber uma exceção OptimisticCollisionException diretamente ou recebê-la ao receber uma exceção 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. A sessão de um cliente é confirmada e atualiza a entrada do mapa. No entanto, o outro cliente já leu os dados antes da confirmação e contém dados antigos ou incorretos. O outro cliente tentará confirmar os dados incorretos, ou seja, quando a exceção foi criada.

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 possui um método getKey interno 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());
}
```

Uma OptimisticCollisionException pode ser a causa de uma ObjectGridException. Se este for o caso, será possível utilizar o seguinte código para determinar o tipo de exceção e imprimir a chave. O código abaixo utiliza o método utilitário findRootCause conforme descrito na seção Técnica Geral de Manipulação de Exceção.

```
try {
...
}
catch (ObjectGridException oe) {
Throwable Root = findRootCause( oe );
if (Root instanceof OptimisticCollisionException) {
OptimisticCollisionException oce = (OptimisticCollisionException)Root;
System.out.println(oce.getKey());
}
}
```

Exceção LockTimeoutException

Mensagem

É possível capturar uma exceção LockTimeoutException diretamente ou ao capturar uma ObjectGridException. O trecho de código a seguir mostra como capturar a exceção e exibir a mensagem.

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

O resultado é:

```
com.ibm.websphere.objectgrid.plugins.LockTimeoutException: %Message
```

%Message representa a cadeia transmitida como um parâmetro quando a exceção é criada e as propriedades e métodos da exceção são utilizados para exibir uma

mensagem de erro precisa. Provavelmente, descreve o tipo de bloqueio solicitado e em qual entrada do mapa a transação agiu.

Causa da Exceção

Quando uma transação ou cliente está solicitando que um bloqueio seja concedido a uma entrada do mapa específica, geralmente ela esperará que o cliente atual libere o bloqueio. Se o pedido de bloqueio permanecer inativo por um longo período de tempo e um bloqueio nunca tiver sido concedido, será criada uma `LockTimeoutException`. Isto serve para evitar um conflito, que é descrito mais detalhadamente na seção seguinte. Provavelmente, você verá esta exceção ao utilizar uma estratégia de bloqueio pessimista, porque o bloqueio nunca será liberado até que a transação seja confirmada.

Obtendo Detalhes Adicionais sobre o Pedido de Bloqueio e Exceção

A `LockTimeoutException` possui um método interno chamado getLockRequestQueueDetails, que retorna uma cadeia que contém uma descrição mais detalhada da situação que acionou a exceção. A seguir está um exemplo de algum código que captura a exceção e exibe uma mensagem de erro.

```
try {
...
}
catch (LockTimeoutException lte) {
System.out.println(lte.getLockRequestQueueDetails());
}
```

O resultado da saída é:

```
fila de pedidos de bloqueio
->[TX:163C269E-0105-4000-E0D7-5B3B090A571D, state =
Granted 5348 milli-seconds ago, mode = U]
->[TX:163C2734-0105-4000-E024-5B3B090A571D, state =
Waiting for 5348 milli-seconds, mode = U]
->[TX:163C328C-0105-4000-E114-5B3B090A571D, state =
Waiting for 1402 milli-seconds, mode = U]
```

Se você obtiver a exceção em um bloco de captura `ObjectGridException`, o seguinte código determinará a exceção e imprimirá os detalhes da fila. Utiliza o método utilitário findRootCause descrito na seção Técnica Geral de Manipulação de Exceção.

```
try {
...
}
catch (ObjectGridException oe) {
Throwable Root = findRootCause( oe );
if (Root instanceof LockTimeoutException) {
LockTimeoutException lte = (LockTimeoutException)Root;
System.out.println(lte.getLockRequestQueueDetails());
}
}
```

Possível Solução

A `LockTimeoutException` serve para evitar possíveis conflitos em seu aplicativo. Uma exceção deste tipo será emitida quando tiver esperado por uma quantidade de tempo definida. No entanto, a quantidade de tempo de espera pode ser configurada utilizando o método `setLockTimeout(int)`, que está disponível para o BackingMap. A utilização do método setLockTimeout elimina a

LockTimeoutException. Se de fato não existir um conflito em seu aplicativo, o ajuste do tempo limite de bloqueio poderá ajudar a evitar a LockTimeoutException.

O código a seguir mostra como criar um ObjectGrid, definir um mapa e configurar seu valor LockTimeout como 30 segundos

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("MapName");
//Isto configurará a quantidade de tempo que um
// pedido de bloqueio esperará antes da emissão de uma exceção
bMap.setLockTimeout(30);
```

O código anterior pode ser utilizado para atribuir hardcode ao ObjectGrid e mapear propriedades. Se você criar o ObjectGrid a partir de um arquivo XML, a propriedade LockTimeout poderá ser configurada na tag backingMap. A seguir está um exemplo de uma tag backingMap que configura um valor LockTimeout do mapa como 30 segundos.

```
<backingMap name="MapName" lockStrategy="PESSIMISTIC" lockTimeout="30">
```

LockDeadlockException

Mensagem

É possível capturar uma LockDeadLockException diretamente ou obtê-la ao capturar uma ObjectGridException. A seguir está um exemplo de código que mostra a captura da exceção e, em seguida, a mensagem exibida.

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

O resultado é:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: %Message
```

%Message representa a cadeia transmitida como um parâmetro quando a exceção é criada e emitida.

Causa da Exceção

O tipo mais comum de conflito ocorre ao utilizar uma estratégia de bloqueio pessimista e cada um de dois clientes separados possui um bloqueio compartilhado em um objeto específico. Em seguida, os dois tentam se promover para um bloqueio exclusivo nesse objeto. A seguir está um diagrama que mostra tal situação com bloqueios de transação que causariam a emissão da exceção.

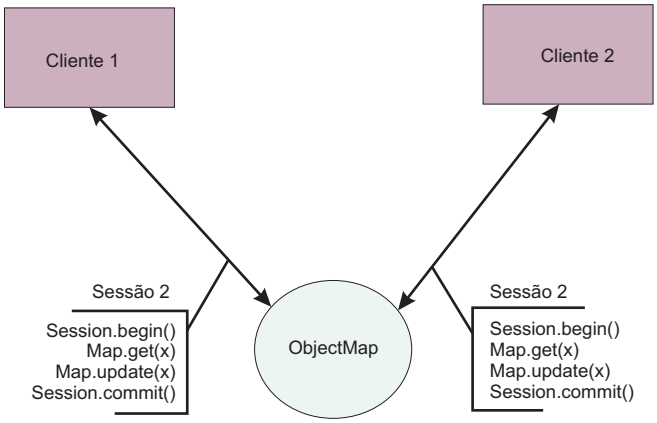


Figura 23. Exemplo de uma Possível Situação de Conflito

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. A seguir está um exemplo passo a passo de quando dois clientes executam os blocos do código de transação, descrito na figura anterior.

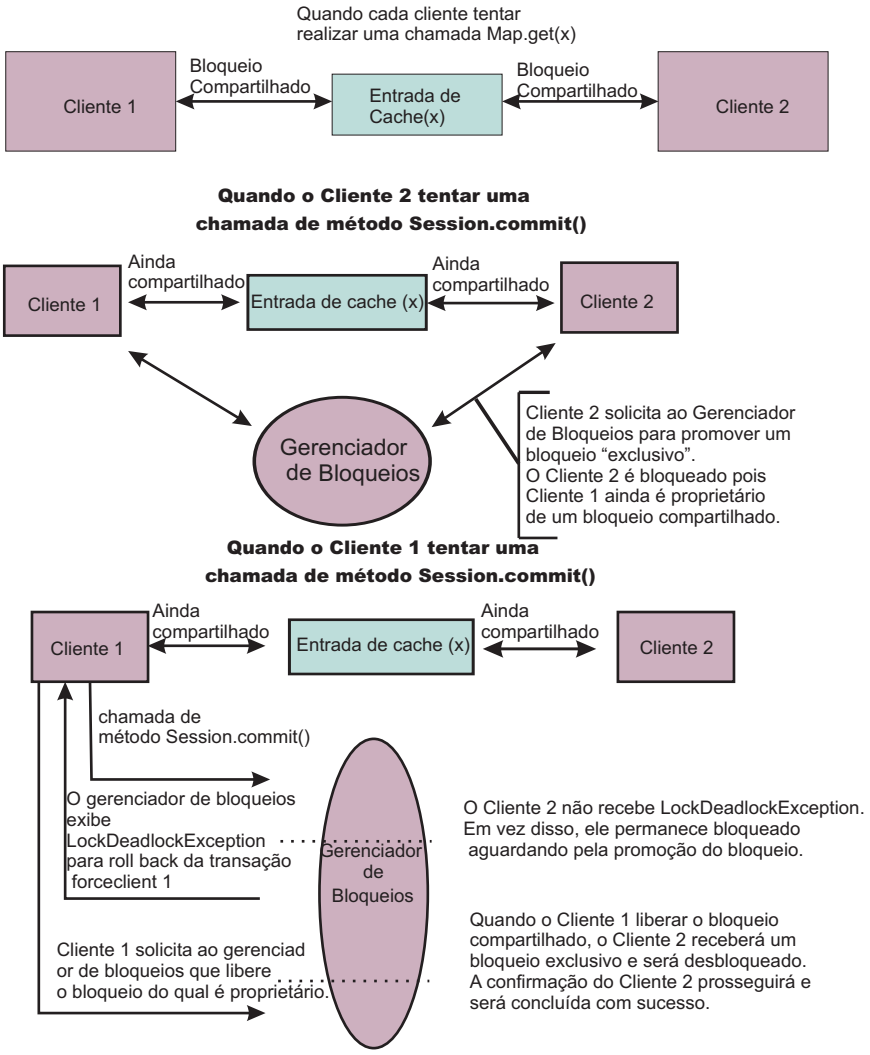


Figura 24. Uma Situação de Bloqueio

Conforme mostrado, quando dois clientes estão tentando promover-se para bloqueios exclusivos e ainda possuem os bloqueios compartilhados, é impossível para qualquer um deles realmente obter um. Eles sempre esperam que o outro cliente libere seu bloqueio compartilhado e, por isso, ocorre uma `LockDeadlockException`.

Soluções Possíveis

Ocasionalmente, é recomendável receber esta exceção. Quando houver muitos encadeamentos, todos os quais executam transações em um mapa específico, é possível encontrar a situação descrita anteriormente (Figura1). Esta exceção é emitida para evitar a interrupção de seu programa. A captura desta exceção permite que você se notifique e, se desejar, inclua código no bloco de captura para que seja possível obter detalhes adicionais da causa. Como você verá esta exceção apenas em uma estratégia de bloqueio pessimista, uma solução simples é apenas utilizar uma estratégia de bloqueio otimista. No entanto, se for requerida a pessimista, será possível utilizar o método getForUpdate em vez do método get. Isto elimina a obtenção de exceções para a situação descrita anteriormente.

Diagnóstico de Problemas de Configuração XML

Fazendo Referência a uma Coleta de Plug-in Inexistente

Ao utilizar o XML para definir plug-ins do BackingMap, o atributo pluginCollectionRef do elemento backingMap deve fazer referência a um backingMapPluginCollection. O atributo pluginCollectionRef deve corresponder ao ID de um dos elementos backingMapPluginCollection.

Mensagem

Se o atributo pluginCollectionRef não corresponder a nenhum dos atributos de id de algum dos elementos backingMapPluginConfiguration, uma mensagem semelhante à seguinte será exibida no registro.

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E:
This is an English only Error message:
Invalid XML file. Line: 14; URI: null;
Message: Key 'pluginCollectionRef' with
value 'bookPlugins' not found for identity
constraint of element 'objectGridConfig'..
```

A mensagem a seguir é um resumo do registro com o rastreio ativado:

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E: This is an
English only Error message:
Invalid XML file. Line: 14; URI: null; Message: Key
'pluginCollectionRef' with
value 'bookPlugins' not found for identity constraint
of element 'objectGridConfig'..
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R com.ibm.websphere.objectgrid.
ObjectGridException:
Invalid XML file: etc/test/document/bookstore.xml
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R at
com.ibm.ws.objectgrid.config.XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R at
com.ibm.websphere.objectgrid.ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R Caused by: org.xml.sax.
SAXParseException: Key 'pluginCollectionRef' with value 'bookPlugins'
not found for identity
constraint of element 'objectGridConfig'.
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.error(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:011 CDT] 686c060e SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:011 CDT] 686c060e SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

Problema

O arquivo XML que foi utilizado para produzir este erro é mostrado abaixo. Observe que o BackingMap book tem seu atributo pluginCollectionRef configurado como bookPlugins e o único backingMapPluginCollection tem um ID de collection1.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config
../objectGrid.xsd" xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" pluginCollectionRef="bookPlugin" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Solução

Para corrigir o problema, certifique-se de que o valor de cada pluginCollectionRef corresponda ao ID de um dos elementos backingMapPluginCollection. Neste exemplo, apenas a alteração do nome do pluginCollectionRef para collection1 evita este erro. Outras maneiras de corrigir o problema incluem a alteração do ID do backingMapPluginCollection existente para corresponder ao pluginCollectionRef ou a inclusão de um backingMapPluginCollection adicional com um ID que corresponda ao pluginCollectionRef.

Atributo Requerido Ausente

Muitos dos elementos no arquivo XML possuem vários atributos opcionais. É possível incluir ou excluir atributos opcionais no arquivo. O XML passará na validação de qualquer maneira. No entanto, existem alguns atributos requeridos. Se estes atributos requeridos não estiverem presentes quando o elemento associado for utilizado, a validação XML falhará.

Mensagem

Quando um atributo requerido estiver ausente, uma mensagem semelhante à seguinte será localizada no registro. Neste exemplo, o atributo type está ausente do elemento property.

```
[7/15/05 13:41:41:267 CDT] 6873dcac XmlErrorHandl E CWOBJ9002E:
This is an English only
Error message: Invalid XML file.
Line: 12; URI: null; Message: cvc-complex-type.4:
Attribute 'type' must appear on element 'property'..
```

A mensagem a seguir é um resumo do registro com o rastreio ativado.

```
[7/15/05 14:08:48:506 CDT] 6873dff9 XmlErrorHandl E CWOBJ9002E: This is an English
only Error message: Invalid XML file.
Line: 12; URI: null; Message: cvc-complex-type.4: Attribute 'type'
must appear on element 'property'..
[7/15/05 14:08:48:526 CDT] 6873dff9 SystemErr R com.ibm.websphere.objectgrid.
ObjectGridException: Invalid XML file: etc/test/document/bookstore.xml
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R at com.ibm.ws.objectgrid.config.
XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R at com.ibm.websphere.objectgrid.
ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R Caused by: org.xml.sax.
SAXParseException: cvc-complex-type.4:
Attribute 'type' must appear on element 'property'.
```

```
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.error(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

Problema

O exemplo a seguir é o arquivo XML que foi utilizado para produzir o erro anterior. Observe que a propriedade no Evictor tem apenas dois dos três atributos requeridos. Os atributos name e value estão presentes, mas o atributo type está ausente. O atributo ausente faz a validação XML falhar.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" pluginCollectionRef="collection1" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
<property name="maxSize" value="89" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Solução

Para resolver este problema, inclua o atributo requerido no arquivo XML. No arquivo XML de exemplo mostrado anteriormente, é necessário incluir o tipo de atributo e designar o valor inteiro.

Elemento Requerido Ausente

Alguns dos elementos XML são requeridos pelo esquema. Se eles não estiverem presentes, a validação XML falhará.

Mensagem

Quando um elemento requerido estiver ausente, uma mensagem semelhante à seguinte será localizada no registro. Neste caso, o elemento objectGrid está ausente.

```
[7/15/05 14:54:23:729 CDT] 6874d511 XmlErrorHandl E CWOBJ9002E:
This is an English only Error message: Invalid XML file.
Line: 5; URI: null; Message: cvc-complex-type.2.4.b: The content of
element 'objectGrids' is not complete.
One of '{"http://ibm.com/ws/objectgrid/config":objectGrid}' is expected..
```

Ative o rastreio para ver informações adicionais sobre este erro. A seção ObjectGridManager cobre informações sobre como ativar o rastreio.

Problema

O exemplo a seguir é o arquivo XML que foi utilizado para produzir este problema. Observe que o elemento objectGrids não possui elementos filho do objectGrid. De acordo com o esquema XML, o elemento objectGrid deve ocorrer nas tags objectGrids pelo menos uma vez. Este elemento ausente faz a validação XML falhar.

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

Solução

Para corrigir este problema, certifique-se de que os elementos requeridos estejam no arquivo XML. No exemplo anterior, pelo menos um elemento objectGrid deve ser colocado na tag objectGrids. Quando os elementos requeridos estiverem presentes, será possível validar o arquivo XML com êxito.

O seguinte arquivo XML válido contém os elementos requeridos presentes.

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

O Valor de Atributo XML não é Válido

Mensagem

A alguns dos atributos no arquivo XML podem ser designados apenas alguns valores. Estes atributos têm seus valores aceitáveis enumerados pelo esquema. Estes atributos incluem:

- Atributo `authorizationMechanism` no elemento objectGrid
- Atributo `copyMode` no elemento `backingMap`
- Atributo `lockStrategy` no elemento `backingMap`
- Atributo `ttlEvictorType` no elemento `backingMap`
- Atributo `type` no elemento `property`

Se a um destes atributos for designado um valor inválido, a validação XML falhará.

Quando um atributo estiver configurado com um valor que não é um de seus valores enumerados, a seguinte mensagem será mostrada no registro:

```
[7/19/05 16:45:40:992 CDT] 6870e51b XmlErrorHandl E CWOBJ9002E: This is an English
only Error message: Invalid XML file. Line: 6; URI: null; Message:
cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid with
respect to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ,
COPY_ON_WRITE, NO_COPY]'. It must be a value from the enumeration..
```

O resumo a seguir é proveniente do registro com rastreio ativado.

```
[7/19/05 16:45:40:992 CDT] 6870e51b XmlErrorHandl E CWOBJ9002E: This is an
English only Error message: Invalid XML file. Line: 6; URI: null; Message:
cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid with
respect to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ,
COPY_ON_WRITE, NO_COPY]'. It must be a value from the enumeration..
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R com.ibm.websphere.objectgrid
.ObjectGridException: Invalid XML file: etc/test/document/backingMapAttrBad.xml
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R at com.ibm.ws.objectgrid.config
.XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R at com.ibm.websphere.objectgrid
.ProcessConfigXML$2.run(ProcessConfigXML.java:99)...
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R Caused by:
org.xml.sax.SAXParseException:
cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid
with respect
to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE,
NO_COPY]'.
It must be a value from the enumeration.
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.util
.ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.util
.ErrorHandlerWrapper.error(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.XMLErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.XMLErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.xs.XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.xs.XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

Problema

Um atributo ao qual é designado um valor fora de um conjunto específico de valores foi configurado incorretamente. Neste caso, o atributo copyMode não está configurado com um de seus valores enumerados. Ele estava configurado como INVALID_COPY_MODE. A seguir está o arquivo XML que foi utilizado para produzir este erro.

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

Solução

Neste exemplo, copyMode tem um valor inválido. Configure o atributo com um destes valores válidos: COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE ou NO_COPY.

Validando XML sem Suporte de uma Implementação

O IBM SDK (Software Development Kit) versão 1.4.2 contém uma implementação de alguma função JAXP a ser utilizada para validação XML em um esquema.

Mensagem

Ao utilizar um SDK que não contém esta implementação, as tentativas de validação poderão falhar. Se desejar validar o XML utilizando um SDK que não contém esta implementação, faça download do Xerces e inclua seus arquivos JAR (Java Archive) no caminho de classe.

Ao tentar validar o XML com um SDK que não possui a implementação necessária, o seguinte erro é localizado no registro.

```
[7/19/05 10:50:45:066 CDT] 15c7850 XmlConfigBuil d XML validation is enabled
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R com.ibm.websphere
.objectgrid[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at
com.ibm.ws.objectgrid
.ObjectGridManagerImpl.getObjectGridConfigurations
(ObjectGridManagerImpl.java:182)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid
.ObjectGridManagerImpl.createObjectGrid(ObjectGridManagerImpl.java:309)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid.test.
config.DocTest.main(DocTest.java:128)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R Caused by: java.lang
.IllegalArgumentException: No attributes are implemented
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at org.apache.crimson.jaxp.
DocumentBuilderFactoryImpl.setAttribute(DocumentBuilderFactoryImpl.java:93)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid.config
.XmlConfigBuilder.<init>(XmlConfigBuilder.java:133)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.websphere.objectgrid
.ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
```

Problema

O SDK utilizado não contém uma implementação da função JAXP necessária para validar arquivos XML em um esquema.

Solução

Depois de fazer download do Apache Xerces e de incluir os JARs no caminho de classe, será possível validar o arquivo XML com êxito.

Mensagens do ObjectGrid

Essas informações de referência fornecem informações adicionais sobre mensagens que você pode encontrar enquanto utiliza o ObjectGrid. As mensagens são identificadas pela chave de mensagem e têm uma explicação e resposta do usuário. Elas podem ser informativas, de aviso ou de erros e são indicadas pela última letra (I, W ou E) da chave de mensagem. A parte da explicação da mensagem explica por que ocorre a mensagem. A parte da resposta do usuário da mensagem descreve qual ação deve ser executada em caso de uma mensagem de aviso ou de erro.

CWOBJ0001E: O método {0} foi chamado após a conclusão da inicialização.
Explicação: Após a conclusão da inicialização, algumas chamadas de método não são mais aceitas.
Resposta do Usuário: Reestruture seu código para que a configuração seja concluída antes que a utilização do tempo de execução seja iniciada.

ICWOBJ0002W: O componente ObjectGrid está ignorando uma
exceção inesperada: {0}.
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ0005W: O encadeamento criou uma InterruptedException: {0}
Explicação: Ocorreu uma InterruptedException.
Resposta do Usuário: Verifique a mensagem de exceção para constatar se
esta interrupção é esperada.

CWOBJ0006W: Ocorreu uma exceção: {0}
Explicação: Ocorreu uma exceção durante o tempo de execução.
Resposta da usuário: Verifique a mensagem de exceção para constatar se esta
é uma exceção esperada.

CWOBJ0007W: O valor nulo foi especificado para {0}, será utilizado
um valor padrão de {1}.
Explicação: Foi especificado um valor nulo para a variável. Será
utilizado um valor padrão.
Resposta do Usuário: Configure o valor apropriado. Consulte a documentação
do ObjectGrid para saber os valores válidos para as variáveis ou propriedades.

CWOBJ0008E: O valor {0} fornecido para
a propriedade {1} é inválido.
Explicação: Foi especificado um valor inválido
para a variável.
Resposta do Usuário: Configure o valor apropriado. Consulte a documentação
do ObjectGrid para saber os valores válidos para as variáveis ou propriedades.

CWOBJ0010E: A chave de mensagem {0} está ausente.
Explicação: A chave de mensagem está ausente no pacote de recursos
da mensagem
Resposta do Usuário: CMSG0002

CWOBJ0011W: Não é possível desserializar o campo {0} na classe {1};
utilizando o valor padrão.
Explicação: Durante a desserialização de um objeto,
um campo esperado não foi localizado. Provavelmente, este campo não
foi localizado
porque o objeto foi desserializado por uma versão diferente da classe
que o serializou.
Resposta do Usuário: Este aviso indica um possível problema.
Nenhuma ação do usuário é requerida, a menos que ocorram erros.

CWOBJ0012E: O código de tipo LogElement, {0} ({1}), não é
reconhecido para esta operação.
Explicação: Ocorreu um erro interno no tempo de execução do ObjectGrid.
Resposta do Usuário: CMSG0002

CWOBJ0013E: Ocorreu uma exceção ao tentar liberar
entradas do cache: {0}
Explicação: Ocorreu um problema ao tentar aplicar
as entradas de evicção ao cache.
Resposta do Usuário: Verifique a mensagem de exceção para constatar se esta
é uma exceção esperada.

CWOBJ0014E: O tempo de execução do ObjectGrid detectou uma tentativa
de aninhar transações.
Explicação: Não é permitido o aninhamento de transações.
Resposta do Usuário: Modifique o código para evitar o aninhamento
das transações.

CWOBJ0015E: Ocorreu uma exceção ao tentar processar uma transação: {0}
Explicação: Ocorreu um problema durante o processamento de transações.
Resposta do Usuário: Verifique a mensagem de exceção para constatar
se esta exceção é esperada.

CWOBJ0016E: Nenhuma transação ativa foi detectada
para a operação atual.
Explicação: É necessária uma transação ativa
para desempenhar esta operação.
Resposta do Usuário: Modifique o código para iniciar uma transação
antes de executar esta operação.

CWOBJ0017E: Foi detectada uma exceção de chave duplicada
durante o processamento da operação do ObjectMap: {0}
Explicação: A chave para a entrada já existe no cache.
Resposta do Usuário: Modifique o código para evitar a inserção da mesma
chave mais de uma vez.

CWOBJ0018E: A chave não foi localizada durante o processamento
da operação do ObjectMap: {0}
Explicação: A chave para a entrada não existe no cache.
Resposta do Usuário: Modifique o código para assegurar que a entrada exista
antes de tentar a operação.

CWOBJ0019W: Não foram localizados dados no slot de entrada de cache reservado
para utilização pelo {0} para o nome do ObjectMap {1}.
Explicação: Ocorreu um erro interno
no tempo de execução do ObjectGrid.
Resposta do Usuário: CMSG0002

CWOBJ0020E: A entrada de cache não está no BackingMap {0}.
Explicação: Erro interno no tempo de execução do ObjectGrid.
Resposta do Usuário: CMSG0002

CWOBJ0021E: Não foi localizada uma instância do ObjectTransformer
utilizável durante a desserialização do objeto LogSequence
para o ObjectGrid {0} e
o ObjectMap {1}.
Explicação: O lado de recebimento de um objeto LogSequence
não possui a configuração apropriada para suportar a
instância ObjectTransformer requerida.
Resposta do Usuário: Verifique a configuração de instâncias
do ObjectGrid
para os lados de envio e recebimento do objeto LogSequence.

CWOBJ0022E: O responsável pela chamada não possui mutex: {0}.
Explicação: Ocorreu um erro interno no tempo de
execução do ObjectGrid.
Resposta do Usuário: CMSG0002

CWOBJ0023E: O CopyMode ({0}) não é reconhecido para esta operação.
Explicação: Ocorreu um erro interno no tempo de execução do ObjectGrid.
Resposta do Usuário: CMSG0002

CWOBJ0024E: Não é possível desserializar o campo {0} na classe {1}.
Falha na desserialização.
Explicação: Durante a desserialização de um objeto, um campo requerido
não foi localizado. Provavelmente, este problema é um erro de
tempo de execução do ObjectGrid.
Resposta do Usuário: CMSG0002

CWOBJ0025E: A serialização do objeto LogSequence falhou.
O número de objetos LogElement serializados ({0}) não corresponde
ao número de objetos LogElement lidos ({1}).
Explicação: Ocorreu um erro interno no tempo de execução do ObjectGrid.
Resposta do Usuário: CMSG0002

CWOBJ0026E: O tipo de credencial JMX não está correto.
Ele deve ser do tipo {0}.
Explicação: O tipo de credencial JMX não está correto.
Se for utilizada a autenticação
básica, o tipo esperado será String[] com o primeiro elemento
sendo o nome do usuário e o segundo a senha. Se for utilizado o certificado cliente,
o tipo esperado será Certificate[].
Resposta do Usuário: Utilize as credenciais corretas.

CWOBJ0027E: Erro de tempo de execução interno. Método clone
não suportado: {0}
Explicação: Ocorreu um erro interno no tempo de execução do ObjectGrid.
CLONE_METHOD_NOT_SUPPORTED_CWOBJ0027.useraction=CMSG0002

CWOBJ0028E: Ocorreu um erro em {0} para o mapa {1}. A chave,
{2}, não foi localizada no mapa. O tipo de LogElement é {3}.
Explicação: Ocorreu um erro interno
ao tentar liberar uma entrada.
Resposta do Usuário: CMSG0002

CWOBJ0029E: Ocorreu um erro em {0} para o mapa {1}. CacheEntry não contém um objeto {2} para a chave {3}. O tipo de LogElement é {4}.
Explicação: Ocorreu um erro interno ao tentar liberar uma entrada.
Resposta do Usuário: CMSG0002

CWOBJ0900I: O componente de tempo de execução do ObjectGrid foi iniciado para o servidor {0}.
Explicação: O componente ObjectGrid foi iniciado.
Resposta do Usuário: Nenhuma. Entrada informativa.

CWOBJ0901E: A propriedade de sistema "{0}" é requerida para iniciar o componente ObjectGrid para o servidor {1}.
Explicação: O componente de tempo de execução do ObjectGrid requer que o "{0}" esteja especificado como uma propriedade de sistema de Java Virtual Machine.
Resposta do Usuário: Consulte o Information Center para utilizar o WebSphere Administrator Console para fornecer propriedades customizadas requeridas
do ObjectGrid.

CWOBJ0902W: Um erro impediu o componente de tempo de execução do ObjectGrid de iniciar
para o servidor {0}.
Explicação: Um erro anterior impediu o componente ObjectGrid de iniciar.
Resposta do Usuário: Consulte as mensagens de erro anteriores para determinar
o que impediu
o componente ObjectGrid de iniciar.

CWOBJ0910I: O componente de tempo de execução do ObjectGrid foi parado para o servidor {0}.
Explicação: O componente ObjectGrid foi parado.
Resposta do Usuário: Nenhuma. Entrada informativa.

CWOBJ0911I: Iniciando o componente de tempo de execução do ObjectGrid para o servidor {0}.
Explicação: O componente ObjectGrid está iniciando.
Resposta do Usuário: Nenhuma. Entrada informativa.

CWOBJ1001I: O Servidor do ObjectGrid {0} está pronto para processar pedidos.
Explicação: O Servidor do ObjectGrid está pronto para processar pedidos.
Resposta do Usuário: Os serviços para este Servidor do ObjectGrid estão disponíveis.

CWOBJ1002E: A porta do servidor {0} já está em utilização.
Explicação: O servidor do ObjectGrid não pode ser iniciado devido a um conflito de porta.
Resposta do Usuário: Os usuários precisam escolher outra porta.

CWOBJ1003I: O serviço Adaptador DCS foi desativado pela configuração, para ativá-lo, altere sua configuração com um nó de extremidade definido.
Explicação: O adaptador DCS foi desativado.
Resposta do Usuário: Os usuários podem ativar o adaptador DCS alterando a configuração.

CWOBJ1004E: O tópico do servidor é nulo
Explicação: O tópico do servidor é nulo
Resposta do Usuário: CMSG0002

CWOBJ1005E: A fila de pedidos que chegam é nula.
Explicação: A rotina de tratamento de pedidos do cliente não pode recuperar pedidos.
Resposta do Usuário: CMSG0002

CWOBJ1006E: A fila de resultados de saída é nula.
Explicação: A rotina de tratamento de pedidos do cliente não pode fornecer resultado ao cliente.
Resposta do Usuário: CMSG0002

CWOBJ1007E: O pedido do cliente do ObjectGrid é nulo.
Explicação:A rotina de tratamento de pedidos do cliente não
pode manipular um pedido que não contém informações sobre o pedido.
Resposta do Usuário: Verifique seu pedido

CWOBJ1008E: O TxID do pedido do cliente do ObjectGrid é nulo.
Explicação: Utilizamos TXID para corresponder conexões e
agrupamento, o TXID não pode ser nulo.
Resposta do Usuário: CMSG0002

CWOBJ1009E: O cliente do ObjectGrid recebeu uma resposta
nula do servidor.
Explicação: Encontrada uma resposta nula do servidor.
Resposta do Usuário: CMSG0002

CWOBJ1010I: O pedido de encerramento está sendo processado.
Explicação: Os servidores de cluster
estão processando o pedido de encerramento.
Resposta do Usuário:nenhuma

CWOBJ1011I: O pedido de encerramento está sendo enviado.
Explicação: Os servidores de cluster
estão processando o pedido de encerramento
Resposta do Usuário:nenhuma

CWOBJ1012I: O pedido de encerramento foi desempenhado.
Explicação: Os servidores de cluster
estão processando o pedido de encerramento.
Resposta do Usuário:nenhuma

CWOBJ1110I: Iniciando o transporte para o cluster do ObjectGrid {0}
utilizando o Endereço IP {1}, porta {2}, tipo de transporte {3}.
Explicação: O transporte do membro de cluster do ObjectGrid
está iniciando.
Resposta do Usuário: Nenhuma. Entrada informativa.

CWOBJ1111W: A resolução de Endereços IP para o nome do host {0}
localizou apenas
o endereço de
auto-retorno. O endereço de auto-retorno será utilizado.
Explicação: Pode haver um problema com o nome do host ou
resolução de DNS.
Para uma implementação relacionada à produção, um endereço sem
auto-retorno normalmente
é esperado.
Resposta do Usuário: Modifique o nome do host ou determine se existe um
problema de DNS.

CWOBJ1112E: Foi encontrado um erro ao consultar o endereço IP
para o nome do host de um membro de cluster do ObjectGrid.
O nome do host é {0}
e o nome do servidor é {1}.
O membro será excluído do cluster.
Explicação: Não é possível resolver o endereço IP para o host indicado.
O membro de cluster do ObjectGrid para o host especificado será excluído.
Resposta do Usuário: Corrija o problema de consulta do nome do host
e tente novamente.

CWOBJ1113E: O serviço de transporte do cluster do ObjectGrid neste processo
não foi iniciado. Este membro de cluster não está definido na configuração.
Explicação: Este membro de cluster do ObjectGrid não
é um membro configurado do cluster. Se este membro de cluster tiver que ser
um membro de um
cluster do ObjectGrid, repare a configuração.
Resposta do Usuário: Reveja a configuração atual.

CWOBJ1114E: O serviço de transporte do cluster do ObjectGrid neste processo
não pôde processar a mensagem que chega. A mensagem é {0} e a
exceção é {1}.
Explicação: Foi detectado um erro interno inesperado.

Resposta do Usuário: Reveja o Web site de suporte na Internet do IBM ObjectGrid
para um problema semelhante ou entre em contato com
a assistência IBM.

CWOBJ1115E: Foi recebido um evento de alteração de visualização não reconhecido
do transporte do cluster do ObjectGrid. O identificador
de visualização é {0} e o evento é {1}.
Explicação: O tipo do evento não é reconhecido. O HA Manager
não sabe como responder ao evento.
Resposta do Usuário: Reveja o Web site de suporte na Internet do IBM
ObjectGrid para um problema semelhante ou entre em contato com a assistência IBM.

CWOBJ1116E: Uma tentativa feita por outro processo para
conectar-se a este processo
por meio do transporte do cluster do ObjectGrid foi rejeitada.
O processo de conexão
forneceu um nome de {0}, um destino de {1},
um nome de membro de {2} e um endereço IP de {3}. A mensagem de erro é {4}.
Explicação: O transporte do cluster do ObjectGrid rejeitou
a tentativa de conexão.
Resposta do Usuário: Esta pode ser uma tentativa de conexão
de
uma parte não autorizada.

CWOBJ1117E: Uma tentativa de autenticar uma conexão falhou. A
exceção é {0}.
Explicação: O transporte do cluster do ObjectGrid rejeitou
a tentativa de conexão.
Resposta do Usuário: Esta pode ser uma tentativa
de conexão de uma parte não autorizada.

CWOBJ1118I: Servidor do ObjectGrid Inicializando [Cluster: {0}
Servidor: {1}].
Explicação: O membro de cluster do ObjectGrid está inicializando.
Resposta do Usuário: Nenhuma. Entrada informativa.

CWOBJ1119I: O cliente do ObjectGrid falhou ao conectar-se ao
host: {0} porta: {1}.
Explicação: O cliente do ObjectGrid falhou ao conectar-se.
Resposta do Usuário: Nenhuma. Entrada informativa.

CWOBJ1120I: Cliente do ObjectGrid conectado com êxito ao
host: {0} porta: {1}.
Explicação: Cliente do ObjectGrid conectado com êxito.
Resposta do Usuário: Nenhuma. Entrada informativa.

CWOBJ1201E: Não há nós de extremidade de acesso do
cliente válidos definidos.
Explicação:=Não há nós de extremidade de acesso do
cliente válidos definidos.
Resposta do Usuário: Defina um nó de extremidade de acesso
do cliente válido.

CWOBJ1202E: O SSL Server Socket falhou ao inicializar. A mensagem
de exceção é {0}
Explicação: O SSL Server Socket falha ao inicializar. As configurações de SSL podem
estar incorretas ou o número da porta já está em utilização.
Resposta do Usuário: Examine a exceção para constatar
o que aconteceu de errado.

CWOBJ1203W: Recebido um evento de tempo limite do servidor
para a transação: {0}
Explicação: O cliente não recebeu a mensagem de resposta esperada
do servidor dentro de um tempo limite configurado.
Resposta do Usuário: Procure mensagens anteriores que possam explicar
o tempo limite.
Se nenhuma for localizada, tente aumentar o tempo limite.

CWOBJ1204W: Recebida uma mensagem de tipo desconhecido.
A mensagem é: {0}
Explicação: Foi detectado um erro interno inesperado.
Resposta do Usuário: Reveja o Web site de suporte na Internet do IBM ObjectGrid para um problema semelhante ou entre em contato com a assistência IBM.

CWOBJ1205E: Falha na Inicialização de SSL. A mensagem de exceção é {0}
Explicação: Falha na Inicialização de SSL. As configurações de SSL podem estar incorretas.
Resposta do Usuário: Examine a exceção para saber o que está errado.

CWOBJ1206W: Falha na Inicialização de SSL. A mensagem de exceção é {0}
Explicação: Falha na Inicialização de SSL. As configurações de SSL podem estar incorretas.
Resposta do Usuário: Examine a exceção para constatar o que aconteceu de errado.

CWOBJ1207W: A propriedade {0} no plug-in {1} está utilizando um tipo de propriedade não suportado.
Explicação: As primitivas Java e seus complementos java.lang são os únicos tipos de propriedade suportados. java.lang.String também é suportado.
Resposta do Usuário: Verifique o tipo de propriedade e altere-o para um dos tipos suportados.

CWOBJ1208W: O tipo de plug-in especificado, {0}, não é um dos tipos de plug-in suportados.
Explicação: Este tipo de plug-in não é suportado.
Resposta do Usuário: Inclua um dos seguintes tipos de plug-in suportados.

CWOBJ1211E: A criação de PMI (Performance Monitoring Infrastructure) de {0} falhou. A exceção é {1}.
Explicação: Uma tentativa de criar a PMI do ObjectGrid falhou.
Resposta do Usuário: Examine a mensagem de exceção e o registro FFDC (First Failure Data Capture).

As mensagens a seguir são utilizadas para reunir o ID do usuário e senha no painel
ou entrada padrão.
LOGIN_PANEL_TITLE=Login no servidor de destino
GENERIC_LOGIN_PROMPT=Digitar informações de login
USER_ID=Identidade do Usuário
PASSWORD=Senha
OK=OK
CANCEL=Cancelar

CWOBJ1215I: O ObjectGrid Transaction Propagation Event Listener está inicializando [ObjectGrid {0}].
Explicação: Esta mensagem informativa indica que o ObjectGrid Transaction
Propagation Event Listener
está inicializando.
Resposta do Usuário: Nenhuma. Entrada informativa.

CWOBJ1216I: O ObjectGrid Transaction Propagation Event Listener foi inicializado [ObjectGrid {0}].
Explicação: ObjectGrid Transaction Propagation Event Listener Inicializado.
Resposta do Usuário:=Nenhuma. Entrada informativa.

CWOBJ1217I: ObjectGrid Transaction Propagation Service Point Inicializado
[ObjectGrid {0}].
Explicação: Esta mensagem informativa indica que o ObjectGrid Transaction Propagation Event Listener foi inicializado.
Resposta do Usuário: Nenhuma. Entrada informativa.

CWOBJ1218E: Ocorreu uma Falha do ObjectGrid Transaction Propagation Event Listener
[ObjectGrid {0} Mensagem de exceção {1}].
Explicação: O tempo de execução do ObjectGrid encontrou uma falha do ObjectGrid Transaction Propagation.
Resposta do Usuário: Examine a exceção para determinar a falha.

CWOBJ1219E: Ocorrida falha do ObjectGrid Transaction Propagation Service End Point
[ObjectGrid {0} Mensagem de exceção {1}].
Explicação: O tempo de execução do ObjectGrid encontrou uma falha de ObjectGrid Transaction Propagation
Service End Point.
Resposta do Usuário: Examine a exceção para determinar a falha.

CWOBJ1220E: O ObjectGrid Transaction Propagation Service não é suportado neste ambiente.
Explicação:=O ObjectGrid Transaction Propagation Service não é suportado no z/OS ou no ambiente do servidor do ObjectGrid independente.
Resposta do Usuário: Não utilize o ObjectGrid Transaction Propagation Service no z/OS ou no ambiente do servidor do ObjectGrid independente

CWOBJ1300I: O Adaptador inicializou com êxito o ObjectGrid.
Explicação: O Adaptador inicializou com êxito o ObjectGrid.
Resposta do Usuário: Nenhuma. Entrada informativa.

CWOBJ1301E: O Adaptador falhou ao inicializar o ObjectGrid. Ocorreu uma exceção [Exception message {0}].
Explicação: A tentativa feita pelos Adaptadores para inicializar o ObjectGrid falhou.
Resposta do Usuário: Examine a exceção para determinar a falha.

CWOBJ1302I: Adaptador parado.
Explicação: Adaptador parado.
Resposta do Usuário: Nenhuma. Apenas Informativa.

CWOBJ1303I: Adaptador iniciado.
Explicação: PMA_CWOBJ1303.explanation=Adaptador iniciado.
Resposta do Usuário: Nenhuma. Apenas Informativa.

CWOBJ1304I: A segurança do ObjectGrid está ativada.
Explicação: A segurança do ObjectGrid está ativada.
Resposta do Usuário:nenhuma

CWOBJ1305I: A segurança do ObjectGrid está desativada.
Explicação: A segurança do ObjectGrid está desativada.
Resposta do Usuário:nenhuma

CWOBJ1306W: Não é possível recuperar os certificados clientes do soquete SSL.
Explicação: Por algum motivo, o tempo de execução não pode recuperar os certificados clientes do soquete SSL.
Resposta do Usuário: Verifique suas configurações de SSL.

CWOBJ1307I: A segurança da instância do ObjectGrid {0} está ativada.
Explicação: A segurança da instância do ObjectGrid {0} está ativada.
Resposta do Usuário:nenhuma

CWOBJ1308I: A segurança da instância do ObjectGrid {0} está desativada.
Explicação: A segurança da instância do ObjectGrid {0} está desativada.
Resposta do Usuário:nenhuma

CWOBJ1309E: Ocorreu um erro inesperado na criação do token de conexão: {0}.
Explicação: Ocorreu um erro inesperado na criação do token de conexão.
Resposta do Usuário: Verifique a configuração de segurança

CWOBJ1310E: Uma tentativa feita por outro processo para conectar-se a este processo por meio do transporte do grupo principal foi rejeitada. O
processo de conexão forneceu um nome de grupo principal de origem de {0}, um destino de {1},
um nome de membro de {2} e um endereço IP de {3}. A mensagem de erro é {4}.

Explicação: O High Availability Manager rejeitou
uma tentativa de conexão.
Resposta do Usuário: Esta pode ser uma tentativa de conexão
de uma parte não autorizada.

CWOBJ1400W: Detectados vários arquivos JARS de tempo de execução do ObjectGrid
na JVM.
A utilização de vários arquivos JAR de tempo de execução do ObjectGrid pode
causar problemas.
Explicação: Geralmente, apenas um JAR de tempo de execução do ObjectGrid deve
ser localizado na JVM.
Resposta do Usuário: Utilize o JAR de tempo de execução do ObjectGrid
apropriado para sua configuração.

CWOBJ1401E: Detectado um arquivo JAR de tempo de execução do ObjectGrid
incorreto para esta configuração.
A configuração detectada é {0}. O arquivo Jar esperado é {1}.
Explicação: Cada arquivo JAR de tempo de execução do ObjectGrid
corresponde a uma
configuração suportada específica.
Resposta do Usuário: Utilize o JAR de tempo de execução do ObjectGrid apropriado
para sua configuração.

CWOBJ1402E: O retorno de chamada de link de conexão do ObjectGrid não
foi localizado para o ID: {0}.
Explicação: Erro interno no tempo de execução do ObjectGrid.
Resposta do Usuário: CMSG0002

CWOBJ1500E: Ocorreu uma exceção ao tentar criar um
GroupName para o Grupo de HA ({0}): {1}.
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ1501E: Ocorreu uma exceção quando o membro ({0}) tentou
juntar-se ao Grupo de HA ({1}): {2}.
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ1503E: Não é possível acessar o ObjectGrid ({0})
para aplicar atualizações no membro
de réplica ({1}).
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ1504E: Ocorreu uma exceção ao tentar
processar LogSequences para a réplica ({0}): {1}.
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ1505E: Mais de um membro do grupo de replicação
foi relatado de volta como o primário.
Apenas um primário pode estar ativo. ({0}).
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ1506E: Existe mais de um membro do grupo de replicação primário
neste grupo ({1}). Apenas um primário pode estar ativo. ({0}).
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ1507W: Ocorreu uma exceção ao tentar encerrar
o processo de replicação para o BackingMap ({0}): {1}.
Explicação: Ao tentar encerrar um membro do grupo de replicação primário,
ocorreu uma exceção durante o processamento de limpeza.
Resposta do Usuário: CMSG0002

CWOBJ1508E: Ocorreu uma exceção ao tentar enviar uma mensagem ({0})
do emissor ({1}) para o receptor ({2}): {3}.
Explicação: Ocorreu um problema ao tentar enviar uma mensagem entre
membros do grupo de replicação.
Resposta do Usuário: CMSG0002

CWOBJ1509E: Ocorreu uma exceção ao tentar serializar
a mensagem ({0}): {1}.
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ1510E: Ocorreu uma exceção ao tentar inflar
a mensagem ({0}): {1}.
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ1511I: {0} ({1}) está aberto para negócios.
Explicação: O membro do grupo de replicação especificado agora
está pronto para aceitar pedidos.
Resposta do Usuário: Nenhuma.

CWOBJ1512W: {0} já existe no grupo de replicação {1}.
Explicação: O membro do grupo de replicação especificado
já está ativo neste grupo de replicação.
Resposta do Usuário: Nenhuma.

CWOBJ1513E: A replicação síncrona falhou no {0} ({1}).
Este membro não está mais ativo.
Explicação: Foi encontrado um problema que impediu
a conclusão bem-sucedida da replicação síncrona.
Resposta do Usuário: Reveja suas mensagens anteriores no registro
para ajudar a diagnosticar o problema.
Pode ser necessário parar e
reiniciar o servidor especificado.

CWOBJ1514I: Está sendo feito downgrade do primário ({0}) para uma
réplica ou espera.
Explicação: Esta não é uma operação normal, mas o processamento
do ObjectGrid pode continuar.
Resposta do Usuário: CMSG0002

CWOBJ1515I: Requisitos de configuração mínimos não atendidos
para o grupo de replicação ({0}).
Explicação: Os requisitos de configuração necessários do primário
e de réplica não foram atendidos com a alteração do membro do
grupo de replicação recente.
Resposta do Usuário: Espere que os recursos adicionais sejam iniciados
e reconhecidos para esta configuração.

CWOBJ1516E: Ocorreu uma exceção ao tentar
ativar o processo de replicação para o ObjectGrid ({0}): {1}.
Explicação: Ao tentar iniciar um membro do grupo
de replicação primário, ocorreu uma exceção durante o processamento
da ativação.
Resposta do Usuário: CMSG0002

CWOBJ1517E: A replicação síncrona falhou para
a transação {2} em {0} ({1}). Este membro não está mais ativo.
Explicação: Foi encontrado um problema que impediu
a conclusão bem-sucedida da replicação síncrona.
Resposta do Usuário: Reveja suas mensagens anteriores no registro para
ajudar a diagnosticar o problema.
Pode ser necessário parar e reiniciar o servidor especificado.

CWOBJ1518E: Ocorreu uma exceção ao tentar
confirmar a transação de réplica ({0}) para a transação primária ({1}) na
Réplica ({2}): {3}.
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ1519E: Ocorreu uma exceção ao tentar
efetuar rollback de LogSequences para a réplica ({0}): {1}.
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ1610W: Tente reconfigurar um cluster nulo para {0}.
Explicação: Os dados do cluster do grupo de replicação não estão disponíveis.
Resposta do Usuário:nenhuma

CWOBJ1611I: O cluster do grupo de replicação {0} está aberto para negócios.
Explicação: Agora o cluster do grupo de replicação pode aceitar pedidos.
Resposta do Usuário:nenhuma

CWOBJ1612I: O cluster do grupo de replicação {0} está fechado para negócios.
Explicação: Agora o cluster do grupo de replicação não pode aceitar pedidos.
Resposta do Usuário:=nenhuma

CWOBJ1620I: Substituindo destino para pedido roteado incorretamente
devido a alterações no servidor. O novo destino é {0}.
Explicação: Destino de roteamento antigo substituído pelo novo destino.
Resposta do Usuário: Se o grupo de replicação
desejado estiver fora de
serviço, será
necessário ativá-lo novamente.

CWOBJ1630I: O grupo de replicação não pode atender este pedido {0}.
Explicação: O roteamento foi rejeitado devido ao serviço indisponível
como o efeito do Domino
Resposta do Usuário: Apenas informações.

CWOBJ1632E: O pedido original não possui um ID válido; não é
possível redirecionar este pedido.
Explicação: Não é possível redirecionar este pedido porque
o pedido original não possui um ID válido.
Resposta do Usuário: Relate ao suporte IBM

CWOBJ1634I: O roteador não pode localizar o destino de
redirecionamento;
utilizando o redirecionamento cego.
Explicação: O roteador não pode localizar o destino de redirecionamento.
Resposta do Usuário: Nenhuma

CWOBJ1660I: O membro do grupo de replicação foi alterado.
Este servidor não hospeda mais o que foi solicitado.
O pedido original é {0}.
Explicação: O membro do grupo de replicação foi alterado.
Resposta do Usuário: Se o grupo de replicação desejado estiver
fora de serviço,
será necessário ativá-lo novamente.

CWOBJ1661I: Os dados de cluster foram atualizados para
o grupo de replicação: {0}
Explicação: Os dados do cluster foram atualizados
Resposta do Usuário:nenhuma

CWOBJ1663E: O roteador do servidor não pode verificar o roteamento
do servidor para {0}, porque os dados do cluster para este grupo de
replicação são nulos no servidor.
Explicação: Os dados do cluster do grupo de replicação
não estão disponíveis para verificação.
Resposta do Usuário: Relate ao suporte IBM

CWOBJ1668W: O pedido está chegando ao servidor que não está
totalmente iniciado.
N**Explicação:** A inicialização do servidor leva de 60 a 120 segundos.
O pedido será automaticamente tentado outra vez se tiver sido configurado
para isso (por padrão, o pedido será
automaticamente tentado outra vez).
Resposta do Usuário: Ajuste sua configuração ou inicie seus clientes
60 a 120
segundos após o início de seus servidores.

CWOBJ1680W: O tempo limite de conexão TCP configurado é menor do que
retryInterval * max(startupRetries, maxRetries),
portanto, é possível que
seja excedido o tempo limite da conexão.
Explicação: O tempo limite de conexão TCP configurado deve ser maior
que retryInterval * max(startupRetries, maxRetries).
Resposta do Usuário: Ajuste sua configuração.

CWOBJ1682W: O tempo limite de transação configurado é menor que
maxForwards retryInterval * max(startupRetries, maxRetries),
portanto, é possível que seja

excedido o tempo limite
da transação.
Explicação: O tempo limite de transação configurado deve ser maior que
maxForwards * retryInterval * max(startupRetries, maxRetries).
Resposta do Usuário: Ajuste sua configuração.

CWOBJ1700I: O HAManager independente
foi inicializado com o grupo principal {0}.
Explicação: O HAManager independente
foi inicializado com êxito.
Resposta do Usuário:nenhuma

CWOBJ1701I: O HAManager independente já foi inicializado.
Explicação: O HAManager independente
já foi inicializado com êxito.
Resposta do Usuário:nenhuma

CWOBJ1702E: O HAManager independente
não foi inicializado, portanto, não pode ser iniciado.
Explicação: O HAManager independente não foi inicializado.
Resposta do Usuário: Inicialize-o
antes de iniciar.

CWOBJ1710I: O HAManager independente
foi iniciado com êxito.
Explicação: O HAManager independente
foi iniciado com êxito.
Resposta do Usuário:nenhuma

CWOBJ1711I: O HAManager independente já foi iniciado com êxito.
Explicação: O HAManager independente
já foi iniciado com êxito.
Resposta do Usuário:nenhuma

CWOBJ1712E: O HAManager independente não foi iniciado.
Explicação: O HAManager independente não foi iniciado.
Resposta do Usuário: Inicialize e inicie-o antes de utilizá-lo.

CWOBJ1713E: O HAManager independente falhou ao iniciar.
Explicação:O
HAManager independente falhou ao iniciar.
Resposta do Usuário: Verifique se as portas
já foram utilizadas.

CWOBJ1720I: O Controlador do HAManager detectou que
o servidor do ObjectGrid está no ambiente do WebSphere, utilizando o
WebSphere HAManager em vez de inicializar e iniciar o
HAManager independente.
Explicação: O servidor do ObjectGrid está em execução no
ambiente do WebSphere.
Resposta do Usuário: Nenhuma

CWOBJ1730I: O Controlador do HAManager detectou
que o HAManager externo do WebSphere é nulo.
Explicação: Não é possível obter o HAManager externo do WebSphere.
Resposta do Usuário: Nenhuma

CWOBJ1790I: É necessário inicializar e
iniciar o HAManager independente.
Explicação: Não é possível obter o HAManager externo do WebSphere.
É necessário inicializar e iniciar o HAManager independente.
Resposta do Usuário: Nenhuma

CWOBJ1792I: O número máximo de encadeamentos é {0} e o número mínimo
de encadeamentos é {1}.
Explicação: Configure o conjunto de encadeamentos.
Resposta do Usuário: Apenas informações.

CWOBJ1800I: É requerido o redirecionamento para o pedido {0} com
resposta de {1}.
Explicação: É requerido
o roteamento de redirecionamento.
Resposta do Usuário: Nenhuma. Manipulado
Automaticamente

CWOBJ1810I: É requerido o redirecionamento para a resposta {0}.
Explicação: É requerido o redirecionamento para a resposta.
Resposta do Usuário: Nenhuma

CWOBJ1811E: É requerido o redirecionamento, mas
o pedido original não pode ser localizado.
Explicação: É requerido o redirecionamento, mas
o pedido original não pode ser localizado.
Resposta do Usuário: Nenhuma

CWOBJ1820E: O pedido de redirecionamento não possui
um identificador do grupo de replicação.
Explicação: Não existe nenhum identificador do grupo de replicação
neste pedido de redirecionamento.
Resposta do Usuário: Entre em contato com o Suporte IBM

CWOBJ1870I: O serviço do servidor não
está disponível para resposta {0}.
Explicação: O serviço do servidor não está disponível devido ao efeito do
Domino ou outros eventos.
Resposta do Usuário: Ative pelo menos o número mínimo de servidores.

CWOBJ1871E: Detectado serviço indisponível, recebida resposta nula,
não é possível tentar novamente.
Explicação: Resposta nula do serviço indisponível
Resposta do Usuário: Entre em contato com o suporte IBM

CWOBJ1872I: O serviço está indisponível com resposta de {0}.
Explicação: O serviço não está disponível
Resposta do Usuário: Ative pelo menos o número mínimo de
servidores ou verifique se a inicialização do servidor foi bem-sucedida.

CWOBJ1890I: Roteando novamente o pedido {0} devido
a um servidor não-responsivo.
Explicação: O pedido para o servidor desejado falhou
ao concluir. O pedido foi roteado novamente para outro servidor.
Resposta do Usuário: Nenhuma.
Manipulado automaticamente. Se o grupo de replicação
desejado estiver fora de serviço, será necessário ativá-lo novamente.

CWOBJ1891E: Todos os servidores não estão disponíveis no grupo de replicação {0}.
Explicação: Todos os servidores não foram iniciados ou
falharam. Eles não estão disponíveis
Resposta do Usuário: Se o grupo de replicação desejado estiver fora de serviço,
será necessário ativá-lo novamente.

CWOBJ1898W: É requerido o redirecionamento, mas o roteador não pode localizar um
novo destino disponível para a resposta {0}
Explicação: O serviço não está disponível.
Resposta do Usuário: Disponibilize o serviço.

CWOBJ1899W: O redirecionamento é requerido, não o roteador não pode localizar o
grupo de replicação correto para a resposta {0}
Explicação: O ID do grupo de replicação foi perdido.
Resposta do Usuário: Entre em contato com o Suporte IBM

CWOBJ1900I: O serviço de chamada de procedimento remoto do servidor cliente
foi inicializado.
Explicação: O serviço de chamada de procedimento remoto do servidor cliente
foi inicializado.
Resposta do Usuário: Nenhuma

CWOBJ1901I: O serviço de chamada de procedimento remoto do servidor cliente
foi iniciado.
Explicação: O serviço de chamada de procedimento remoto do servidor cliente
foi iniciado.
Resposta do Usuário: Nenhuma

CWOBJ1902I: Os encadeamentos da rotina de tratamento de chamada de
procedimento remoto do servidor cliente foram iniciados.
Explicação: Os encadeamentos da rotina de tratamento de chamada de procedimento
remoto do servidor cliente foram iniciados.
Resposta do Usuário: Nenhuma

CWOBJ1903I: O serviço de rede de configuração foi inicializado.
Explicação: O serviço de rede de configuração foi inicializado.
Resposta do Usuário: Nenhuma

CWOBJ1904I: O serviço de rede de configuração foi iniciado.
Explicação: O serviço de rede de configuração foi iniciado.
Resposta do Usuário: Nenhuma

CWOBJ1905I: A rotina de tratamento de configuração foi iniciada.
Explicação: A rotina de tratamento de configuração foi iniciada.
Resposta do Usuário: Nenhuma

CWOBJ1913I: O serviço de rede de administração do sistema foi inicializado.
Explicação: O serviço de rede de administração do sistema foi inicializado.
Resposta do Usuário: Nenhuma

CWOBJ1914I: O serviço de rede de administração do sistema foi iniciado.
Explicação: O serviço de rede de administração do sistema foi iniciado.
Resposta do Usuário: Nenhuma

CWOBJ1915I: A rotina de tratamento de administração do sistema foi iniciada.
Explicação: A rotina de tratamento de administração do sistema foi iniciada.
Resposta do Usuário: Nenhuma

CWOBJ2000E: Não existe nenhum membro neste grupo de replicação {0}.
Explicação: Nenhum membro pode ser localizado neste grupo de replicação.
Resposta do Usuário: Verifique se os servidores foram iniciados ou se os dados estão disponíveis

CWOBJ2001W: Nenhum membro disponível neste grupo de replicação {0}.
Explicação: Nenhum membro disponível pode ser localizado neste grupo de replicação.
Resposta do Usuário: Verifique se o serviço do servidor está disponível

CWOBJ2002W: Nenhuma tabela de roteamento disponível para este grupo de replicação {0}.
Explicação: Nenhuma tabela de roteamento disponível para este grupo de replicação.
Resposta do Usuário: Verifique se os clientes ativaram a tabela de roteamento

CWOBJ2003I: Não é possível localizar o cache de roteamento para a chave do cache {0}, criando novo cache de roteamento.
Explicação: Alterações no primeiro roteamento ou cluster.
R**Resposta do Usuário:** nenhuma

CWOBJ2010E: O destino para este pedido é nulo.
Explicação: O pedido não veio com informações de destino.
Resposta do Usuário: Entre em contato com o suporte IBM.

WOBJ2060I: O cliente recebeu nova versão do cluster do grupo de replicação {0}.
Explicação: O cliente recebeu nova versão do cluster do grupo de replicação
Resposta do Usuário:nenhuma

CWOBJ2068I: O controle de alcance detectou um problema no membro do grupo de replicação {0}.
Explicação: Algum servidor pode não ser alcançado, o mecanismo de alcance o manipulará.
Resposta do Usuário: Nenhuma.

CWOBJ2069I: O cronômetro de controle de alcance libera o membro do grupo de replicação {0}.
Explicação: Este membro está disponível para roteamento.
Resposta do Usuário:nenhuma

CWOBJ2086I: O controle de encadeamento de roteamento foi ativado devido a uma sobrecarga para o grupo de replicação {0}.
Explicação: O controle de encadeamento está em ação.
Resposta do Usuário:nenhuma

CWOBJ2088I: O controle de alcance foi ativado para regular a disponibilidade do servidor para o
grupo de replicação {0}.
Explicação: O alcance está em ação.
Resposta do Usuário:nenhuma

CWOBJ2090W: Não é possível localizar a tabela de roteamento para o grupo de replicação {0}.
Explicação: O cluster do grupo
de replicação é nulo.
Resposta do Usuário:nenhuma

CWOBJ2091W: A tabela de roteamento não é nula, mas não contém servidores para o grupo de replicação {0}.
Explicação: O cluster do grupo
de replicação está vazio.
Resposta do Usuário:nenhuma

CWOBJ2092I: A tabela de roteamento é nula no tempo de execução para o grupo de replicação {0}.
Explicação: Obtendo tabela de roteamento do tempo de execução.
Resposta do Usuário:nenhuma

CWOBJ2093I: A tabela de roteamento não é nula no armazenamento de cluster do grupo de replicação para o grupo de replicação {0}
Explicação: Obtendo tabela de roteamento do armazenamento de cluster.
Resposta do Usuário:nenhuma

CWOBJ2096I: A tabela de roteamento foi obtida do armazenamento de cluster do grupo de replicação para o grupo de replicação {0}.
Explicação: Obtido cluster do grupo de replicação do
armazenamento de cluster do grupo de replicação.
Resposta do Usuário:nenhuma

CWOBJ2097I: O roteamento é baseado no algoritmo de revezamento para o grupo de replicação {0}.
Explicação: O roteamento é baseado no algoritmo de revezamento.
Resposta do Usuário:nenhuma

CWOBJ2098I: O roteamento é baseado na seleção aleatória para o grupo de replicação {0}.
Explicação: O roteamento é baseado em seleção aleatória.
Resposta do Usuário:nenhuma

CWOBJ2100I: A conexão ({0}) é stale, não pode ser utilizada.
Explicação: A conexão é stale.
Resposta do Usuário:nenhuma

CWOBJ2101W: A conexão não pode ser adquirida após o tempo de espera máximo.
Explicação: Não restou nenhuma conexão no conjunto.
Resposta do Usuário: Aumente o número máximo de conexões na configuração.

CWOBJ1600I: O serviço ManagementGateway foi iniciado na porta ({0}).
Explicação: O serviço ManagementGateway está pronto para processar pedidos.
Resposta do Usuário: O serviço ManagementGateway está disponível.

CWOBJ1601E: O serviço ManagementGateway falhou ao iniciar na porta ({0}).
Explicação: O serviço ManagementGateway falhou ao iniciar.
Resposta do Usuário: Certifique-se de que a porta especificada ainda não esteja sendo utilizada.

CWOBJ1602E: O serviço ManagementGateway falhou ao conectar-se ao servidor em ({0}):({1}).
Explicação: O serviço ManagementGateway falhou ao conectar-se ao servidor.
Resposta do Usuário: Certifique-se de que o servidor esteja em execução.

CWOBJ1603E: O serviço de gerenciamento falhou ao responder para o pedido remoto ({0}).
Explicação:CMSG0001
Resposta do Usuário: CMSG0002

CWOBJ2400E: Configuração inválida: o mapa de suporte {0} é membro de mais de um conjunto de mapas.
Explicação: Um backingMap pode pertencer a apenas um conjunto de mapas.
Resposta do Usuário: Edite o arquivo XML do cluster para que cada mapa de suporte pertença a apenas um conjunto de mapas.

CWOBJ2401E: Configuração inválida: o mapa de suporte {0} no ObjectGrid distribuído {1} não está em um conjunto de mapas.
Explicação: Cada mapa de suporte de um ObjectGrid distribuído deve ser colocado em um conjunto de mapas.
Resposta do Usuário: Edite o arquivo XML do cluster para que cada mapa de suporte em um ObjectGrid distribuído pertença a um conjunto de mapas.

CWOBJ2402E: Configuração inválida: o conjunto de mapas tem uma referência a um mapa {0}. Este mapa de suporte não existe no arquivo XML do ObjectGrid.
Explicação: Cada mapa em um conjunto de mapas deve referir-se a um mapa de suporte do arquivo XML do ObjectGrid.
Resposta do Usuário: Edite o(s) arquivo(s) XML para que cada mapa no conjunto de mapas faça referência a um mapa de suporte do arquivo XML do ObjectGrid.

CWOBJ2403E: O arquivo XML é inválido. Foi detectado um problema com o {0} na linha {1}. A mensagem de erro é {2}.
Explicação: O arquivo XML não está de acordo com o esquema.
Resposta do Usuário: Edite o arquivo XML para que ele esteja de acordo com o esquema.

CWOBJ2404W: O valor especificado para {0} é {1}. Este é um valor inválido. {0} não será configurado.
Explicação: O valor para este atributo de configuração não é válido.
Resposta do Usuário: Configure o atributo de configuração como um valor apropriado no arquivo XML.

CWOBJ2405E: A ref de objectgrid-binding {0} no arquivo XML do Cluster não faz referência a um ObjectGrid válido no arquivo XML do ObjectGrid.
Explicação: Cada um dos objectgrid-bindings deve fazer referência a um ObjectGrid no arquivo XML do ObjectGrid.
Resposta do Usuário: Edite os arquivos XML para que o objectgrid-binding no XML do Cluster faça referência a um ObjectGrid válido no XML do ObjectGrid.

CWOBJ2500E: Falha ao iniciar o servidor do ObjectGrid {0}.
Explicação: O servidor do ObjectGrid falhou ao iniciar corretamente.
Resposta do Usuário: Verifique se existem exceções no registro.

CWOBJ2501I: Ativando o servidor do ObjectGrid {0}.
Explicação: Um servidor do ObjectGrid está sendo inicializado.
Resposta do Usuário:nenhuma

CWOBJ2502I: Iniciando o servidor do ObjectGrid utilizando a URL do arquivo XML do ObjectGrid "{0}" e a URL do arquivo XML do Cluster "{1}".
Explicação: Um servidor do ObjectGrid está começando a utilizar um arquivo XML do cluster e um arquivo xml do ObjectGrid.
Resposta do Usuário:nenhuma

CWOBJ2503I: Auto-inicializando em um servidor Objectgrid no mesmo nível no host {0} e porta {1}.
Explicação: Este servidor do ObjectGrid será auto-inicializado para um servidor no mesmo nível para recuperar informações requeridas para iniciar.
Resposta do Usuário:nenhuma

COBJ2504I: Tentando auto-inicializar em um servidor do ObjectGrid no mesmo nível utilizando o(s) seguinte(s) host(s) e porta(s) "{0}".
Explicação: Este servidor do ObjectGrid utilizará a lista de hosts e portas fornecidos em uma tentativa de conectar-se a um servidor do ObjectGrid no mesmo nível.
Resposta do Usuário:nenhuma

CWOBJ2505I: Tentando auto-inicializar em um servidor do ObjectGrid no mesmo nível utilizando a URL do arquivo XML do Cluster "{0}".
Explicação: Este servidor do ObjectGrid utilizará a lista de servidores no arquivo XML do Cluster em uma tentativa de conectar-se a um servidor do ObjectGrid no mesmo nível.
Resposta do Usuário:nenhuma

CWOBJ2506I: O rastreio está sendo registrado no {0}.
Explicação: O arquivo de rastreio foi configurado na linha de comandos.
Resposta do Usuário: Consulte o arquivo de rastreio especificado para o rastreio de inicialização do servidor do ObjectGrid.

CWOBJ2507I: A especificação de rastreio está configurada como {0}.
C**Explicação:** A especificação de rastreio foi configurada na linha de comandos.
Resposta do Usuário:nenhuma

CWOBJ2508I: Um arquivo de propriedades de segurança "{0}" foi especificado e será utilizado para iniciar o servidor.
Explicação: Um arquivo de propriedades de segurança foi fornecido para iniciar um servidor seguro.
Resposta do Usuário:nenhuma

CWOBJ2509E: Tempo limite excedido após esperar {0} segundos o início do servidor.
Explicação: O servidor do ObjectGrid não foi iniciado dentro do intervalo de tempo limite.
Resposta do Usuário: Verifique se existem exceções no registro.

CWOBJ2510I: Parando o servidor do ObjectGrid {0}.
Explicação: Parando o servidor do ObjectGrid.
Resposta do Usuário:

CWOBJ2511I: Esperando a parada do servidor.
Explicação: Esperando a parada do servidor do ObjectGrid.
Resposta do Usuário:nenhuma

CWOBJ2512I: Servidor do ObjectGrid {0} parado.
Explicação: Servidor do ObjectGrid parado.
Resposta do Usuário:nenhuma

CWOBJ2513E: Tempo limite excedido após esperar {0} segundos a parada do servidor.
Explicação: O servidor do ObjectGrid não foi parado dentro do intervalo de tempo limite.
Resposta do Usuário: Verifique se existem exceções no registro.

CWOBJ2514I: Esperando a conclusão da ativação do servidor do ObjectGrid.
Explicação: O servidor do ObjectGrid foi ativado. Esperando a conclusão da ativação do servidor.
Resposta do Usuário: nenhuma.

CWOBJ2515E: Os argumentos fornecidos são inválidos. A seguir estão os argumentos válidos. {0}{1}
Explicação: Os argumentos fornecidos para este script são inválidos.
Resposta do Usuário: Digite argumentos válidos.

CWOBJ2516I: O servidor do ObjectGrid concluiu sua ativação.
Explicação: O servidor do ObjectGrid está ativo e pronto para processar pedidos.
Resposta do Usuário: nenhuma.

CWOBJ2517I: Auto-inicialização bem-sucedida no servidor do Objectgrid no mesmo nível no host {0} e porta {1}.
Explicação: Este servidor do ObjectGrid foi auto-inicializado com êxito para um servidor no mesmo nível para recuperar informações requeridas para início deste servidor.
Resposta do Usuário:nenhuma

CWOBJ2407W: A propriedade {0} na classe de plug-in {1} não pôde ser configurada. A exceção é {2}.
Explicação: A propriedade para este plug-in não pôde ser configurada.
Resposta do Usuário: Consulte a exceção para obter informações adicionais.

Avisos

Referências nesta publicação a produtos, programas ou serviços IBM não significam que a IBM pretende torná-los disponíveis em todos os países onde opera. Qualquer referência a produtos, programas ou serviços IBM não significa que apenas produtos, programas ou serviços IBM possam ser utilizados. Qualquer produto, programa ou serviço funcionalmente equivalente, que não infrinja nenhum direito de propriedade intelectual da IBM, poderá ser utilizado em substituição a este produto, programa ou serviço. A avaliação e verificação da operação em conjunto com outros produtos, exceto aqueles expressamente designados pela IBM, são de inteira responsabilidade do Cliente.

A IBM pode ter patentes ou solicitações de patentes pendentes relativas a assuntos tratados nesta publicação. O fornecimento desta publicação não garante ao Cliente nenhum direito sobre tais patentes. Pedidos de licença devem ser enviados, por escrito, para:

Gerência de Relações Comerciais e Industriais da IBM Brasil
Av. Pasteur, 138-146
Rio de Janeiro, RJ
CEP 22290-240

Licenciados deste programa que desejam obter mais informações sobre este assunto com objetivo de permitir: (i) a troca de informações entre programas criados independentemente e outros programas (incluindo este) e (ii) a utilização mútua das informações trocadas, devem entrar em contato com:

Gerência de Relações Comerciais e Industriais da IBM Brasil
Av. Pasteur 138-146
Botafogo
Rio de Janeiro, RJ
CEP 22290-240

Tais informações podem estar disponíveis, sujeitas a termos e condições apropriados, incluindo em alguns casos, o pagamento de uma taxa.

Marcas Registradas e Marcas de Serviços

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 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 serviço de terceiros.