WebSphere® WebSphere Extended Deployment Versione 6.0.x

Guida al modello di programmazione ObjectGrid

Nota

Prima di utilizzare queste informazioni, assicurarsi di avere letto le informazioni generali in “Informazioni particolari” a pagina 373.

Data di compilazione: 29 marzo 2006

Indice

Come inviare i propri commenti

I vostri commenti risultano di estrema importanza poiché consentono di fornire informazioni della massima accuratezza e qualità.

- Per inviare commenti su articoli nel centro informazioni di WebSphere Extended Deployment Information Center, disponibile all'indirizzo: `http://www.ibm.com/software/webservers/appserv/extend/library/`
  1. Visualizzare l'articolo nel browser Web e andare alla fine dell'articolo.
  2. Compilare il collegamento **Commenti** riportato alla fine dell'articolo e inoltrarlo.
- Per inviare commenti su questo o altri documenti PDF, è possibile inviarli tramite e-mail a: **wasdoc@us.ibm.com**.

  Verificare di avere indicato il numero e il nome del documento e, se possibile, la pagina specifica, la tabella o il numero della figura su cui si desidera esprimere un'opinione.

Inviando informazioni di questo tipo, si riconosce a IBM un diritto non esclusivo a utilizzarle o distribuirle nei modi ritenuti più opportuni senza incorrere in nessun obbligo nei confronti dell'utente.

Capitolo 1. Introduzione a ObjectGrid mediante l'esecuzione dell'applicazione di esempio

Utilizzare questa sezione per cominciare a utilizzare ObjectGrid, un componente di elaborazione distribuito che rende gli oggetti disponibili per una serie di applicazioni.

È necessario che siano installati WebSphere Extended Deployment Versione 6.0 e WebSphere Application Server Versione 6.0.2 o superiori su almeno una macchina dell'ambiente.

Limitazione: Se si utilizza ObjectGrid con WebSphere Extended Deployment Versione 6.0, sono richieste delle modifiche alla licenza per utilizzare ObjectGrid in un ambiente Java 2 Platform, Standard Edition (J2SE) Versione 1.4.2 o successiva o in un ambiente WebSphere Application Server Versione 6.02 o successiva. Contattare il rappresentante commerciale per maggiori dettagli.

Se si desidera sviluppare le applicazioni ObjectGrid senza accedere alle macchine server su cui è installato WebSphere Extended Deployment, è possibile eseguirle sulla macchina locale. La macchina locale richiede l'installazione di IBM Software Developer Kit (SDK) o Eclipse.

Per sviluppare le applicazioni ObjectGrid sulla macchina locale, copiare le seguenti directory dall'installazione sulla macchina:

- Se si utilizza WebSphere Extended Deployment Versione 6.0.1, copiare il file /lib/wsobjectgrid.jar e il file /optionalLibraries/ObjectGrid/objectgridSamples.jar nella directory operativa.
- Se ObjectGrid è stato installato mediante un'installazione di un ambiente server misto, copiare il file /ObjectGrid/lib/objectgrid.jar e il file /ObjectGrid/samples/objectgridSamples.jar nella directory operativa.

Per ulteriori informazioni sui file Java archive (JAR) installati con ObjectGrid, fare riferimento alla sezione Assemblaggio di ObjectGrid.

Utilizzare questa attività per eseguire e utilizzare le applicazioni di esempio ObjectGrid. È possibile eseguire le applicazioni in questa attività in un ambiente della riga comandi Java, Eclipse o J2EE (Java 2 Platform, Enterprise Edition).

- Perché l'applicazione di esempio ObjectGrid venga eseguita dalla riga comandi, fare riferimento alla sezione Esecuzione dell'applicazione di esempio ObjectGrid dalla riga comandi.
- Per eseguire l'applicazione di esempio ObjectGrid in Eclipse, fare riferimento a Importazione e utilizzo dell'applicazione di esempio ObjectGrid in Eclipse.
- Per eseguire l'applicazione di esempio ObjectGrid su WebSphere Extended Deployment, fare riferimento a Caricamento ed esecuzione dell'applicazione di esempio ObjectGrid con WebSphere Extended Deployment

È possibile cominciare a utilizzare ObjectGrid eseguendo l'applicazione di esempio e caricare l'esempio nell'ambiente di sviluppo.

Esecuzione dell'applicazione di esempio ObjectGrid dalla riga comandi

Utilizzare questa sezione per eseguire le applicazioni abilitate per ObjectGrid da una riga comandi Java e per verificare la configurazione ObjectGrid.

Prima di iniziare questa attività, installare l'ambiente di server misto, compreso l'ObjectGrid autonomo.

È necessario che sia installato Software Development Kit (SDK). Inoltre, è necessario poter accedere alle applicazioni di esempio ObjectGrid. Fare riferimento a Introduzione a ObjectGrid per ulteriori informazioni.

Utilizzare questa attività per eseguire rapidamente un'applicazione con ObjectGrid abilitato.

1. Verificare la versione di Software Development Kit (SDK). ObjectGrid richiede IBM JDK 1.42 o successivo. Per verificare l'ambiente Java prima di eseguire l'applicazione di esempio ObjectGrid, effettuare le seguenti operazioni:
   a. Aprire un prompt di comandi.
   b. Immettere il seguente comando:

      ```
      java -version
      ```

      Se il comando viene eseguito correttamente, viene visualizzato un testo simile al seguente:

      ```
      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: È possibile eseguire questi esempi anche utilizzando Java 2 Platform, Standard Edition (J2SE) Versione 1.3.x Software Development Kit (SDK). Per ulteriori informazioni, fare riferimento a Assemblaggio di ObjectGrid.

      Se viene visualizzato un messaggio di errore, verificare che SDK sia installato e inserito nella variabile CLASSPATH.
2. Eseguire l'applicazione di esempio di ObjectGrid. L'applicazione di esempio riporta un caso semplice che coinvolge impiegati uffici e posizioni di lavoro. L'applicazione di esempio crea un'istanza ObjectGrid con associazioni per ogni tipo di oggetto. Ogni associazione ha le voci inserite e modificate in modo da dimostrare il funzionamento della memorizzazione nella cache di ObjectGrid.
   a. Aprire una finestra della riga comandi e passare alla directory operativa. Copiare i file objectgrid.jar, asm.jar e cglib.jar dalla cartella /ObjectGrid/lib alla directory operativa. Copiare il file /ObjectGrid/samples/objectgridSamples.jar nella directory operativa.
   b. Emettere il seguente comando:

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

      Viene visualizzato un output simile al seguente. L'output è stato ridotto a scopo di pubblicazione:

      ```
      Inizializzazione di ObjectGridSample ...
      resourcePath: META-INF/objectgrid-definition.xml
      objectgridUrl:
       jar:file:/C:/temp/objg/objectgridSample.jar!/
      META-INF/objectgrid-definition.xml
      EmployeeOptimisticCallback sta restituendo l'oggetto di versione per
      l'impiegato = Perry Cheng, versione = 0
      EmployeeOptimisticCallback sta restituendo l'oggetto di versione per
      l'impiegato Hao Lee, versione = 0
      EmployeeOptimisticCallback sta restituendo l'oggetto di versione per
      l'impiegato = Ken Huang, version = 0
      ```

```
EmployeeOptimisticCallback sta restituendo l'oggetto di versione per
l'impiegato Jerry Anderson, versione = 0
EmployeeOptimisticCallback sta restituendo l'oggetto di versione per
l'impiegato = Kevin Bockhold, versione = 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 sta restituendo l'oggetto della versione per
nome contatore = Counter1, versione = 0
CounterOptimisticCallback sta restituendo l'oggetto della versione per
nome contatore = Counter2, versione = 0
CounterOptimisticCallback sta restituendo l'oggetto della versione per
nome contatore = Counter3, versione = 0
Commit delle operazioni di ivCounterMap eseguito
Commit delle operazioni di ivOfficeMap eseguito
... terminato con:
CounterOptimisticCallback sta restituendo l'oggetto della versione per
nome contatore = Counter1, versione = 0
EmployeeOptimisticCallback sta restituendo l'oggetto di versione per
l'impiegato = Ken Huang, version = 0
CounterOptimisticCallback sta restituendo l'oggetto della versione per
nome contatore = Counter2, versione = 0
EmployeeOptimisticCallback sta restituendo l'oggetto di versione per
l'impiegato = Perry Cheng, versione = 0
CounterOptimisticCallback sta restituendo l'oggetto della versione per
nome contatore Counter3, versione = 0
EmployeeOptimisticCallback sta restituendo l'oggetto di versione per
l'impiegato Jerry Anderson, versione = 0
CounterOptimisticCallback sta restituendo l'oggetto della versione per
nome contatore = Counter4, versione = 0
EmployeeOptimisticCallback sta restituendo l'oggetto di versione per
l'impiegato Hao Lee, versione = 0
EmployeeOptimisticCallback sta restituendo l'oggetto di versione per
l'impiegato Kevin Bockhold, versione = 1
DONE cleanup
```

3. Eseguire l'applicazione di esempio ObjectGrid distribuita.

   Il programma com.ibm.websphere.samples.objectgrid.basic.ObjectGridSample utilizza un'istanza di ObjectGrid locale come cache di dati. Tutti gli oggetti sono memorizzati nella cache sulla Java virtual machine (JVM) locale. Per utilizzare un ObjectGrid distribuito sviluppato in un cluster ObjectGrid, utilizzare il programma com.ibm.websphere.samples.objectgrid.distributed.DistributedObjectGridSample. Il programma DistributedObjectGridSample è incluso nel fileobjectgridSamples.jar.

   a. Avviare un cluster ObjectGrid. Per ulteriori informazioni sull'avvio di un cluster ObjectGrid autonomo da utilizzare con l'esempio di ObjectGrid distribuito, fare riferimento a Avvio del cluster ObjectGrid di esempio autonomo.

   b. Una volta avviato il server ObjectGrid, è possibile eseguire l'applicazione di esempio ObjectGrid distribuita con il seguente comando:

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

Dopo aver avviato il cluster ObjectGrid richiesto, il programmaDistributedObjectGridSample avrà un output simile a quello del programma ObjectGridSample.

A questo punto è stata eseguita l'applicazione di esempio ObjectGrid su una riga comandi Java per verificare la funzionalità di ObjectGrid.

L'origine per questo esempio si trova nel file objectgridSamples.jar, in particolare nei file com\ibm\websphere\samples\objectgrid\basic\ObjectGridSample.java e com\ibm\websphere\samples\objectgrid\distributed\ DistributedObjectGridSample.java.

Avvio del cluster ObjectGrid di esempio autonomo

Per eseguire l'esempio ObjectGrid distribuito, è necessario avviare un cluster di ObjectGrid in cui è presente l'ObjectGrid richiesto.

Verificare che WebSphere Extended Deployment for Mixed Server Environment, Versione 6.0.x sia installato.

Utilizzare questa attività per avviare un server ObjectGrid basato sui file`cluster-config-1.xml` e`cluster-objectgrid-definition.xml`. Questa attività è richiesta per eseguire l'esempio ObjectGrid distribuito. Fare riferimento alle sezioni Esecuzione dell'applicazione di esempio ObjectGrid dalla riga comandi e Importazione e utilizza dell'applicazione di esempio ObjectGrid in Eclipse per ulteriori informazioni. Il file `cluster-config-1.xml` ha soltanto una definizione server ObjectGrid. Questo server ObjectGrid rappresenta il cluster ObjectGrid di esempio.

1. Individuare il file`objectgridSamples.jar` nella directory*`mse_install_root`*`/ObjectGrid/samples`.
2. Estrarre il file `META-INF/cluster-config-1.xml` e il file `META-INF/cluster-objectgrid-definition.xml` dal file `objectgridSamples.jar` nella directory *`mse_install_root`*`/ObjectGrid/samples`.
3. Verificare che la variabile d'ambiente JAVA_HOME sia impostata e che la versione Java corrisponda ai requisiti. Il server ObjectGrid richiede un ambiente Java 2 Platform, Standard Edition (J2SE) Versione 1.4.2 o successivo. Per controllare l'ambiente Java, effettuare le seguenti operazioni:
   a. Controllare la variabile d'ambiente JAVA_HOME. Dal prompt dei comandi, emettere:

      ```
      echo %JAVA_HOME%
      ```

      Questo comando visualizza il percorso Java. Se è necesario impostare la variabile d'ambiente JAVA_HOME, emettere il seguente comando:

      ```
      set JAVA_HOME=JDK_INSTALL_ROOT
      ```

      Impostare la *JDK_INSTALL_ROOT* sulla directory di installazione Java, ad esempio `c:\java`.
   b. Controllare la versione di Java. Eseguire questo comando:

      ```
      java -version
      ```

      Verificare che la versione sia Java 2 Platform, Standard Edition (J2SE) Versione 1.4.2 o successiva.
4. Avviare il server ObjectGrid. Da un promt dei comandi, emettere i seguenti comandi:

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

Importante: È necessario specificare il file `objectgridSamples.jar` nella variabile classpath mediante l'opzione -jvmArgs. Il file `objectgridSamples.jar` contiene le classi necessarie al server ObjectGrid di esempio per le implementazioni dei plug-in definiti nel file `cluster-objectgrid-definition.xml`. Questo file JAR viene utilizzato anche per la serializzazione e la deserializzazione degli oggetti memorizzati nelle mappe.

Viene visualizzato un output simile al seguente. L'output è stato ridotto a scopo di pubblicazione:

```
************ Inizio della visualizzazione dell'ambiente corrente ************
[1/17/06 14:04:34:144 CST] 7daee176 Launcher
I CWOBJ2501I: Avvio del server ObjectGrid server1.
:
[1/17/06 14:04:37:719 CST] 7daee176 ServerRuntime
I CWOBJ1001I: Server ObjectGrid server1 pronto per elaborare le richieste.
```

Fare riferimento alle sezioni Esecuzione dell'applicazione di esempio ObjectGrid dalla riga comandi e Importazione e utilizza dell'applicazione di esempio ObjectGrid in Eclipse per ulteriori informazioni su come eseguire l'applicazione di esempio ObjectGrid distribuita. Per ulteriori informazioni sull'avvio e l'arresto di server ObjectGrid autonomi dalla riga comandi, fare riferimento a Capitolo 8, "Supporto per la riga comandi", a pagina 81.

Importazione e utilizza dell'applicazione di esempio ObjectGrid in Eclipse

Utilizzare questa attività per importare e utilizzare l'applicazione di esempio di ObjectGrid in Eclipse.

Prima di iniziare questa attività, installare l'ambiente di server misto, compreso l'ObjectGrid autonomo.

Per questa applicazione di esempio, utilizzare Eclipse Versione 3.1 o successivo per importare ed eseguire l'esempio. È possibile ottenere Eclipse da Application Server Toolkit incluso con WebSphere Application Server, dall'installazione di Rational Application Developer o scaricandolo direttamente da Eclipse.org.

Utilizzando Eclipse, è possibile eseguire il debug delle proprie applicazioni. È possibile effettuare una procedura passo-passo mediante l'applicazione di esempio.

1. Importare il progetto in Eclipse:
   a. Eseguire il programma Eclipse. Utilizzare il file `eclipse.exe` dalla directory di installazione Eclipse.
   b. Da Eclipse, creare un nuovo progetto.
      1) Fare clic su **File > Nuovo > Progetto > Java > Progetto Java**. Fare clic su **Avanti**.
      2) Immettere un nome per il progetto. Ad esempio, specificare `ObjectGridSamples`.
      3) Selezionare **Crea nuovo progetto nello spazio di lavoro**.

4) Nella sezione Layout progetto, fare clic su **Configura predefinito**.
5) Per la cartella di origine e di output, selezionare **Progetto** e fare clic su **OK**.
6) Fare clic su **Avanti**.
7) Selezionare la scheda **Librerie**.
8) Fare clic su **Aggiungi JAR esterni**
9) Passare alla cartella `/ObjectGrid/lib` e selezionare i file **objectgrid.jar**, **asm.jar** e **cglib.jar**. Fare clic su **Apri** nella procedura guidata **Selezione JAR**.
10) Fare clic su **Fine**.

2. Importare il file`objectgridSamples.jar` nel progetto Java.
   a. Fare clic con il tasto destro del mouse sul pulsante Java e selezionare **Importa**.
   b. Selezionare **File zip** in **Seleziona un'origine di importazione**.
   c. Fare clic su **Avanti**.
   d. Fare clic su**Sfoglia** per aprire la procedura guidata Importa da file zip.
   e. Aprire il file**objectgridSamples.jar**. Passare alla directory `/ObjectGrid/samples`. Selezionare il file **objectgridSamples.jar** e fare clic su **Apri**.
   f. Verificare che la casella di spunta della struttura file root sia selezionata.
   g. Verificare che la cartella **In** contenga il progetto Java creato nel passo precedente, ad esempio il progetto `ObjectGridSamples`.
   h. Fare clic su **Fine**.
3. Controllare le proprietà del progetto Java.
   a. Aprire la prospettiva Java. Fare clic su **Finestra > Apri prospettiva > Java**.
   b. Passare alla vista della console. Fare clic su **Finestra > Mostra vista > Console**.
   c. Verificare che la vista Explorer pacchetto sia disponibile e selezionata. Fare clic su **Finestra > Mostra vista > Explorer pacchetto**.
   d. Fare clic con il tasto destro del mouse sul progetto Java e selezionare **Proprietà**.
   e. Fare clic su **Percorso build Java** sul pannello di sinistra.
   f. Selezionare la scheda **Origine** nel pannello di destra.
   g. Verificare che la root del progetto sia riportata nelle cartelle di origine nel pannello del percorso del build.
   h. Selezionare la scheda **Librerie** nel pannello di destra.
   i. Verificare che i file`objectgrid.jar`, `asm.jar` e `cglib.jar` e la libreria di sistema JRE siano presenti nelle cartelle JAR e delle classi nel pannello Percorso build.
   j. Fare clic su **OK**.
4. Eseguire l'esempio ObjectGrid.
   a. Dalla vista Explorer pacchetto, espandere il progetto Java.
   b. Espandere il pacchetto com.ibm.websphere.samples.objectgrid.basic.
   c. Fare clic con il tasto destro del mouse sul file **ObjectGridSample.java**. Fare clic su **Esegui > Applicazione Java**.

d. La console restituisce un output simile a quando si esegue l'applicazione dalla riga comandi Java. Per un esempio di output, fare riferimento alla sezione Esecuzione dell'applicazione di esempio ObjectGrid dalla riga comandi.

5. Eseguire l'esempio ObjectGrid distribuito. Per eseguire l'esempio ObjectGrid distribuito, è necessario configurare un cluster ObjectGrid. Per eseguire questo esempio, è possibile utilizzare i file di configurazione XML predefiniti forniti nel file objectgridSamples.jar. Fare riferimento alla sezione Avvio del cluster ObjectGrid di esempio autonomo per ulteriori informazioni.

   Una volta avviato il server ObjectGrid, è possibile eseguire l'applicazione di esempio ObjectGrid distribuita effettuando le seguenti operazioni:

   a. Dalla vista Explorer pacchetto, espandere il progetto Java.
   b. Espandere il pacchetto com.ibm.websphere.samples.objectgrid.distributed.
   c. Fare clic con il tasto destro del mouse sul file **DistributedObjectGridSample.java**. Fare clic su **Esegui > Applicazione Java**.
   d. La console visualizza un output simile a quello dell'esempio ObjectGrid.

Le operazioni per caricare il progetto ed eseguire il programma di debug si trovano anche nel file `SamplesGuide.htm`. Il file `SamplesGuide.htm` si trova nella directory `doc` nel file`objectgridSamples.jar`.

Riferimenti correlati

Assemblaggio di ObjectGrid
È possibile accedere ai pacchetti di ObjectGrid in due modi: installando WebSphere Extended Deployment o installando un ambiente server misto.

Caricamento ed esecuzione dell'applicazione di esempio ObjectGrid con WebSphere Extended Deployment

Utilizzare questa attività per caricare ed eseguire l'applicazione di esempio ObjectGrid J2EE (Java 2 Platform, Enterprise Edition) all'interno di WebSphere Extended Deployment.

WebSphere Application Server e WebSphere Extended Deployment devono essere installati.

Utilizzare questa attività per comprendere e verificare l'integrazione di ObjectGrid con WebSphere Extended Deployment.Per ulteriori informazioni, fare riferimento a Capitolo 10, "Integrazione di ObjectGrid con WebSphere Application Server", a pagina 287.

1. Installare il file `ObjectGridSample.ear`. È possibile installare il file EAR (enterprise archive) su un unico server delle applicazioni o su un cluster. Per installare il `ObjectGridSample.ear` nella console di gestione, effettuare le seguenti operazioni:

   a. Dalla console di gestione, fare clic su **Applicazioni> Installa nuova applicazione**.
   b. Dalla pagina **Preparazione per l'installazione dell'applicazione**, specificare il percorso dell'applicazione di esempio ObjectGrid. Ad esempio, passare a `<install_root>/installableApps/ObjectGridSample.ear`. Fare clic su **Avanti**.
   c. Dalla seconda pagina di **Preparazione per l'installazione dell'applicazione**, lasciare le impostazioni predefinite e fare clic su **Avanti**.

d. Dalla pagina **Seleziona opzioni di installazione**, lasc iare i valori predefiniti e fare clic su **Avanti**.

e. Nella pagina **Mappa moduli sui server**, specificare le destinazioni di distribuzione su cui si desidera installare i moduli contenuti nell'applicazione. Selezionare un server di destinazione o un cluster dall'elenco **Cluster e server**. Selezionare la casella di controllo accanto a **Modulo** per selezionare tutti i moduli dell'applicazione oppure selezionare i singoli moduli.

f. Nelle pagine seguenti, lasciare i valori predefiniti e fare clic su **Fine**.

g. Fare clic su **>Salva sulla configurazione principale** una volta completata l'installazione dell'applicazione.

h. Fare clic sull'opzione **Sincronizza le modifiche con i nodi**. Dalla pagina **Applicazioni enterprise > Salva** fare clic su **Salva**.

i. Fare clic su **OK**.

2. Controllare la porta HTTP dell'host_predefinito dei server e aggiungere un alias host. Per impostazione predefinita, i moduli Web sono collegati al nome host virtuale host_predefinito a meno che non si modifiche il nome host durante l'installazione. Se si installa l'applicazione su un cluster, è necessario configurare almeno un alias host per la porta HTTP dell'host predefinito per ogni membro del cluster. È inoltre necessario controllare la porta HTTP dell'host predefinito per ogni membro del cluster e aggiungere il corrispondente alias host all'elenco Alias host nella console di gestione. Per controllare la porta HTTP dell'host predefinito di un server, effettuare le seguenti operazioni:

   a. Nella console di gestione, fare clic su **Server > Server delle applicazioni > *nome_server***.

   b. Espandere le porte nella sezione Comunicazione. La porta **host_predefinito_WC** è il nome host virtuale host_predefinito.

   Per aggiungere un alias host, effettuare le seguenti operazioni:

   a. Nella console di gestione, fare clic su **Ambiente > Host virtuali > host_predefinito > Alias host > Nuovo**.

   b. Utilizzare il valore predefinito del nome host e specificare la porta.

   c. Fare clic su **OK**.

3. Avviare l'applicazione di esempio ObjectGrid.

   - Per avviare l'applicazione su un server, fare clic su **Server > Server delle applicazioni**. Selezionare il server su cui è installato il file `ObjectGridSample.ear`. Fare clic su **Avvia**.
   - Per avviare l'applicazione su un cluster, fare clic su **Server > Cluster**. Selezionare il cluster su cui è installato il file `ObjectGridSample.ear`. Fare clic su **Avvia**.

   Una volta avviata l'applicazione su un server o su un cluster, è possibile arrestare e avviare l'applicazione indipendentemente dal server host o dal cluster. Per arrestare o avviare l'applicazione di esempio ObjectGrid, effettuare le seguenti operazioni:

   a. Dalla console di gestione, fare clic su **Applicazioni > Applicazioni enterprise**.

   b. Selezionare l'applicazione di esempio ObjectGrid.

   c. Fare clic su **Avvia** o **Arresta**.

4. Accedere all'esempio ObjectGrid. Una volta installato il file `ObjectGridSample.ear` su un unico server o su un cluster e dopo aver avviato l'applicazione, è possibile accedere all'esempio ObjectGrid dal seguente indirizzo Web:

   `http://`*`nomehost`*`:`*`porta`*`/ObjectGridSample`

   Ad esempio, se il nome host è `hostlocale` e il valore della porta è `9080`, utilizzare l'indirizzo Web `http://hostlocale:9080/ObjectGridSample`.
5. Verificare la funzionalità dell'ObjectGrid distribuito nell'ambiente WebSphere Application Server. Il file `ObjectGridSample.ear` contiene anche il servlet DistributedObjectGridServlet che dimostra l'uso di un ObjectGrid distribuito nell'ambiente WebSphere Application Server. Il server delle applicazioni su cui è presente il servlet DistributedObjectGridServlet deve contenere anche il server ObjectGrid, che è un membro del cluster ObjectGrid richiesto.
   - Per ulteriori informazioni sulla configurazione di un cluster ObjectGrid per ottenere il DistributedObjectGridServlet in esecuzione, fare riferimento alla sezione Avvio di un cluster ObjectGrid di esempio nell'ambiente WebSphere.
   - Per ulteriori informazioni sull'avvio di server ObjectGrid sui server delle applicazioni, fare riferimento alla sezione Avvio di un server ObjectGrid su un server delle applicazioni.

   Se il server delle applicazioni con il file `ObjectGridSample.ear` installato contiene anche il server ObjectGrid richiesto, il servlet DistributedObjectGridServlet funziona nello stesso modo degli altri servlet. È possibile accedere al servlet dal seguente indirizzo Web: `http://hostname:port/ObjectGridSample/DistributedObjectGridServlet`. Ad esempio, se il nome host è `localhost` e la porta è 9080, utilizzare l'indirizzo Web `http://localhost:9080/ObjectGridSample/DistributedObjectGridServlet`.

   È possibile abilitare la traccia ObjectGrid utilizzando la seguente stringa di traccia: `ObjectGrid*=all=enabled`.

A questo punto è stata installata e configurata l'applicazione di esempio ObjectGrid e l'applicazione di esempio ObjectGrid distribuita su un server WebSphere Extended Deployment.

Dopo aver installato l'applicazione su un server o su un cluster, è possibile accedere alla documentazione di esempio dal seguente indirizzo Web:

`http://`*`nomehost`*`:`*`porta`*`/ObjectGridSample/docs/introduction.html`

Ad esempio, se il nome host è **hostlocale** e il valore della porta è **9080**, utilizzare l'indirizzo Web `http://hostlocale:9080/ObjectGridSample/docs/introduction.html`.

Avvio di un cluster ObjectGrid di esempio nell'ambiente WebSphere

Utilizzare questa attività per avviare un cluster ObjectGrid per verificare la funzionalità dell'ObjectGrid distribuito nell'ambiente WebSphere Application Server.

È necessario che WebSphere Extended Deployment sia installato. È necessario inoltre che il file ObjectGridSample.ear sia installato sul server delle applicazioni. Per ulteriori informazioni sull'installazione del file ObjectGridSample.ear, fare riferimento a Caricamento ed esecuzione dell'applicazione di esempio ObjectGrid con WebSphere Extended Deployment.

Utilizzare questa attività per impostare un server delle applicazioni per un server ObjectGrid basato sui file cluster-config-1.xml e cluster-objectgrid-definition.xml.

Il file cluster-config-1.xml ha soltanto una definizione server ObjectGrid. Questo server ObjectGrid rappresenta il cluster ObjectGrid di esempio. È possibile utilizzare un server delle applicazioni autonomo o un cluster con un membro del cluster che contiene il server ObjectGrid di esempio.

1. Estrarre i file META-INF/cluster-config-1.xml e META-INF/cluster-objectgrid-definition.xml dal file /optionalLibraries/ObjectGrid/objectgridSamples.jar nella directory /optionalLibraries/ObjectGrid.
2. Definire gli argomenti JVM generici necessari.
   a. Dalla console di gestione, fare clic su **Server > Server delle applicazioni > *nome_server* > Definizione processo > Java Virtual Machine**.
   b. Nel pannello Argomenti JVM generici, immettere il seguente testo:

      ```
      -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
      ```

      `INSTALL_ROOT` è la directory di installazione root di WebSphere Application Server.
   c. Fare clic su **Salva**.
   d. Fare clic su **Salva sulla configurazione principale**.
   e. Selezionare l'opzione **Sincronizza modifiche con i nodi**. Fare clic su **Salva**.
3. Copiare il file /optionalLibraries/ObjectGrid/objectgridSamples.jar nella directory /classes o `lib/ext`. Il file objectgridSamples.jar contiene le classi necessarie al server ObjectGrid di esempio per le implementazioni dei plug-in definite nel file cluster-objectgrid-definition.xml. Questo file JAR è utilizzato anche per la serializzazione e la deserializzazione di oggetti memorizzati in mappe.
4. Riavviare il server per applicare le modifiche.

Per maggiori dettagli sull'avvio e l'arresto dei server ObjectGrid sui server delle applicazioni, fare riferimento alla sezione Avvio di un server ObjectGrid su un server delle applicazioni.

Avvio di un server ObjectGrid su un server delle applicazioni

Un server ObjectGrid può essere configurato in modo da essere avviato all'interno di un server delle applicazioni. WebSphere Application Server rileva il componente ObjectGrid e avvia automaticamente il server ObjectGrid.

È possibile configurare i server ObjectGrid in WebSphere Application Server Versione 6.0.2 e successive, includendo quando vengono installati gli add-on quali WebSphere Extended Deployment o WebSphere Business Integration Server. Le versioni precedenti di WebSphere Application Server, come WebSphere Application Server Versione 5.0.2, possono avere applicazioni che utilizzano l'ObjectGrid come clients ma la funzione server di ObjectGrid non può essere utilizzata con versioni precedenti del server delle applicazioni.

Se si utilizzano configurazioni cluster che consentono la replica, il gestore HA è richiesto. I server ObjectGrid utilizzano il gestore HA in maniera differente dai normali server delle applicazioni. Quando il server ObjectGrid si trova su un server delle applicazioni, il server ObjectGrid non configura, inizializza o crea il servizio del gestore HA, ma utilizza il servizio esistente sul server delle applicazioni. Per la replica tra server ObjectGrid, i server ObjectGrid devono essere in esecuzione su server delle applicazioni che fanno parte dello stesso gruppo principale.

Tutte le altre funzioni del server ObjectGrid sono le stesse se il server viene eseguito su WebSphere Application Server. Se la specifica del cluster ObjectGrid include tre server, uno qualsiasi dei tre server delle applicazioni nel gruppo principale può contenere i server ObjectGrid. I server delle applicazioni possono anche suddividere i cluster, sempre che i cluster facciano parte dello stesso gruppo principale. Il passo più importante consiste nel correlare il nome host TCP/IP del server e le informazioni sulla porta nel file cluster.xml.

Utilizzare questa attività per eseguire i server ObjectGrid su server delle applicazioni nell'ambiente WebSphere Application Server.

1. Aggiunta delle proprietà personalizzate richieste sulla Java Virtual Machine (JVM). Dalla console di gestione, fare clic su **Server > Server delle applicazioni > *nome_server* > Gestione processo e Java > Definizione processo > Java Virtual Machine > Proprietà personalizzate**. Fare clic su **Nuovo**. Creare le seguenti proprietà personalizzate:

Tabella 1. Proprietà personalizzate JVM per i server ObjectGrid

Nome proprietà personalizzata	Descrizione	Valore di esempio
`objectgrid.server.name`	Specifica il nome del server ObjectGrid da utilizzare sul server delle applicazioni. Il nome fornito deve essere uno dei nomi server definiti nel file XML del cluster di ObjectGrid.	server1
`objectgrid.xml.url`	Specifica l'URL (Universal Resource Locator) per il file XML di ObjectGrid. Questa proprietà è obbligatoria.	`file:///d:/was/etc/test/objectGridMatch.xml`
`objectgrid.cluster.xml.url`	Specifica l'URL per il file XML del cluster ObjectGrid. Questa proprietà è obbligatoria.	`file:///d:/was/etc/test/csCluster0.xml`

Tabella 1. Proprietà personalizzate JVM per i server ObjectGrid (Continua)

Nome proprietà personalizzata	Descrizione	Valore di esempio
`objectgrid.security.server.props`	Specifica l'URL per il file delle proprietà di sicurezza del server ObjectGrid. Questa proprietà è richiesta solo se la sicurezza è abilitata nel file XML del cluster ObjectGrid. Per determinare se la sicurezza è abilitata nel file XML del cluster, ricercare il seguente testo: `<cluster name="cluster1" securityEnabled="true" ....` Se l'attributo securityEnabled è impostato su`false`, non è necessario definire questa proprietà. Utilizzare il file `security.ogserver.props` come maschera. Fare riferimento a "Sicurezza di ObjectGrid" a pagina 136 per il significato di queste proprietà in questo file e sul modo in cui queste possono essere utilizzate.	`file:///d:/was/optionalLibraries/ObjectGrid/properties/security.ogserver.props`

È inoltre possibile definire queste proprietà JVM nel campo **Argomenti JVM generici** nel pannello **Java Virtual Machine** della console di gestione. Di seguito è riportato un esempio del campo Argomenti JVM generici:

```
-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. Salvare le modifiche e riavviare il server delle applicazioni. WebSphere Application Server rileva il componente ObjectGrid e avvia automaticamente il server ObjectGrid.

   L'ObjectGrid sul server delle applicazioni utilizza la struttura di canali per interagire con i client ObjectGrid, in particolare con la porta di accesso client. Quando viene avviato il server ObjectGrid, questo rileva la posizione su WebSphere Application Server e utilizzare la struttura di canali già in esecuzione sul server delle applicazioni. Il server ObjectGrid crea e avvia la propria struttura di canali solo se una struttura non è già stata creata o avviata sul server delle applicazioni.
3. Arrestare il server ObjectGrid. Arrestare il server ObjectGrid arrestando il server delle applicazioni associato. Non è possibile arrestare il server ObjectGrid utilizzando i comandi di gestione del sistema ObjectGrid.

I server delle applicazioni nell'ambiente WebSphere Application Server eseguono server ObjectGrid.

Capitolo 2. ObjectGrid

ObjectGrid è una struttura di memorizzazione nella cache degli oggetti estensibile per applicazioni Java 2 Platform, Standard Edition (J2SE) e Java 2 Platform, Enterprise Edition (J2EE).

È possibile utilizzare l'API ObjectGrid quando si sviluppano le proprie applicazioni per richiamare, memorizzare, eliminare ed aggiornare oggetti in ObjectGrid. È inoltre possibile implementare plug-in personalizzati che controllano gli aggiornamenti alla cache, richiamano e memorizzano i dati con le origini dati esterne, gestiscono l'eliminazione delle voci dalla cache e gestiscono le funzioni della cache in background per l'ambiente applicativo ObjectGrid.

API basata sulle mappe

ObjectGrid fornisce un'API basata sull'interfaccia java.util.Map. Questa API viene estesa per supportare il raggruppamento di operazioni in blocchi transazionali. Questa interfaccia fa parte dell'interfaccia java.util.Map e aggiunge il supporto per le operazioni batch, l'invalidazione, l'associazione di parole chiave e gli inserimenti e gli aggiornamenti espliciti. La sintassi di Java Map è stata migliorata con punti di estensione in modo da poter implementare le seguenti funzioni:

- Programmi di esclusione dalla cache per definire la durata delle voci della cache
- Interfacce di richiamata delle transazioni per controllare attentamente la gestione delle transazioni e integrare facoltativamente con il gestore transazioni WebSphere in ambienti J2EE
- Implementazioni del programma di caricamento che richiamano automaticamente e inseriscono i dati in un database quando un programmatore dell'applicazione utilizza le operazioni ObjectGrid Map get e put
- Interfacce del listener che forniscono informazioni su quando vengono eseguite le transazioni di cui è stato eseguito il commit e che vengono applicate all'intera struttura ObjectGrid o a determinate istanze Map.
- Interfacce del programma di trasformazione oggetti che consentono una copia e una serializzazione più efficiente di chiavi e valori.

L'ambiente ObjectGrid

È possibile utilizzare la struttura ObjectGrid installando uno dei seguenti prodotti:

- ObjectGrid è integrato con WebSphere Extended Deployment Versione 6.0.1 ed è parte dell'installazione completa.
- UnObjectGrid autonomo fa parte dell'installazione Mixed Server Environment (MSE).

In entrambi i casi, ObjectGrid supporta le funzioni client/server. Il runtime server supporta il clustering, la replica e il partizionamento delle cache distribuite degli oggetti. Il runtime client supporta il concetto di cache prossima e di logica di indirizzamento della gestione del carico di lavoro a cluster remoti. Il runtime client supporta inoltre la creazione di mappe di oggetti locali.

Il livello di supporto varia a seconda se si esegue il runtime client, il runtime server, l'ObjectGrid integrato o l'ObjectGrid autonomo.

ObjectGrid integrato con WebSphere Extended Deployment

Runtime server: il runtime server è integrato. Per WebSphere Extended Deployment Versione 6.0.1, il runtime integrato non è supportato sulla piattaforma z/OS.

Runtime client: il runtime client è supportato su J2SE e J2EE a livello JDK 1.3.1 e superiore, compreso WebSphere Application Server Versione 5.0.2 e successivo. Il runtime client è supportato sulla piattaforma z/OS.

ObjectGrid autonomo

Runtime server: il runtime server può essere eseguito su una Java Virtual Machines (JVM) autonoma come singolo server o come cluster di server. Il server autonomo è supportato sulla maggior parte di piattaforme J2SE e J2EE a livello JDK 1.4.2 e superiore. Il server autonomo è supportato su WebSphere Application Server Versione 6.0.2 e successive. Il runtime server autonomo non è supportato sulla piattaforma z/OS per WebSphere Extended Deployment Versione 6.0.1.

Runtime client: il runtime client è supportato su J2SE e J2EE a livello JDK 1.3.1 e superiore, compreso WebSphere Application Sever Versione 5.0.2 e successivo.

Gestione delle sessioni

È fornita un'implementazione di gestione delle sessioni HTTP distribuite che memorizza gli oggetti della sessione HTTP nell'ObjectGrid.

Installazione semplice

È possibile installare e configurare ObjectGrid in poche e semplici operazioni. Tali operazioni includono la copia dei file Java archive (JAR) sulla variabile classpath e la definizione di alcune istruzioni di configurazione.

Modifiche transazionali

Tutte le modifiche vengono effettuate nel contesto di una transazione in modo da garantire una interfaccia programmatica più solida. La transazione può essere controllata esplicitamente all'interno dell'applicazione oppure l'applicazione può utilizzare la modalità di programmazione del commit automatico. Tali modifiche transazionali possono essere replicate su un cluster ObjectGrid in modalità asincrona e sincrona per fornire scalabilità e accesso a tolleranza d'errore.

È possibile scalare ObjectGrid da una semplice griglia in esecuzione su un'unica JVM (Java Virtual Machine) su una griglia che interessa uno o più cluster ObjectGrid di JVM. Tali server rendono i dati disponibili mediante le API Map per una serie di client abilitati per ObjectGrid. I client ObjectGrid utilizzano le API Java Map di base. Tuttavia, lo sviluppatore dell'applicazione non deve sviluppare API Java TCP/IP e RMI (remote method invocation) in quanto il client ObjectGrid può raggiungere altri server ObjectGrid che contengono le informazioni sulla rete. Se il volume di dati è troppo elevato per un'unica JVM, è possibile utilizzare l'ObjectGrid per partizionare i dati.

ObjectGrid offre inoltre una soluzione applicativa con funzioni a elevata disponibilità. La condivisione degli oggetti si basa su un modello di replica in cui sono presenti un server primario, uno o più server di replica e uno o più server standby. Questo cluster di server di replica è detto gruppo di replica. Se l'accesso al gruppo di replica è un'operazione di scrittura, allora la richiesta viene indirizzata al server

primario. Se l'accesso è invece un'operazione di lettura o se la mappa è una mappa di sola lettura, allora la richiesta può essere indirizzata ai server primari o di replica. I server standby sono definiti come server di replica potenziali nel caso in cui un server riporta un errore. Se un server primario riporta un errore, allora un server di replica diventa il server primario per ridurre le interruzioni delle operazioni. Questo comportamento può essere configurato in base alle proprie necessità.

Se si desidera utilizzare un approccio di propagazione oggetti più semplice, è disponibile un modello peer to peer con QoS (quality of service) inferiore rispetto a quanto fornito con Extended Deployment Versione 6.0. Con questo supporto transazionale distribuito, ai peer viene inviata una notifica delle modifiche mediante il trasporto dei messaggi. Il trasporto di messaggi è integrato se si esegue WebSphere Application Server Versione 6.0.2 o successivo. Se non si esegue WebSphere Application Server Versione 6.0.2 o successivo, è necessario fornire un altro trasporto di messaggi come un provider Java Message Service (JMS).

API compatibili con contenitori di inserimento

Configurare l'ObjectGrid utilizzando un semplice file XML o in maniera programmatica utilizzando le API Java. Tali API Java sono progettate per funzionare in ambienti in cui si utilizzano strutture basate su introduzioni per configurare le applicazioni. Le API e le interfacce degli oggetti ObjectGrid possono essere richiamate da un contenitore Inversion of Control (IoC) e quindi gli oggetti chiave ObjectGrid possono essere introdotti nell'applicazione.

Architettura estensibile

È possibile estendere la maggior parte di elementi della struttura ObjectGrid sviluppando dei plug-in. È quindi possibile regolare ObjectGrid in modo da consentire a un'applicazione di prendere decisioni per garantire sia coerenza che prestazioni elevate. Il codice personalizzato dai plug-in supporta anche le seguenti funzioni specifiche delle applicazioni:

- Ascolto degli eventi delle istanze ObjectGrid per l'inizializzazione, l'avvio e la fine di una transazione e l'eliminazione.
- Richiamo delle richiamate delle transazioni per abilitare un'elaborazione specifica della transazione.
- Implementazione delle politiche di transazioni comuni con transazioni ObjectGrid generiche.
- Utilizzo dei programmi di caricamento per i punti di entrata e di uscita comuni e trasparenti agli archivi dati esterni e ad altri repository delle informazioni.
- Gestione degli oggetti non serializzabili in un modo specifico con le interfacce ObjectTransformer.

È possibile implementare ognuno di questi comportamenti senza influenzare l'uso delle interfacce delle API della cache diObjectGrid di base. Con questa trasparenza, le applicazioni che utilizzano l'infrastruttura della cache possono avere archivi dati e elaborazione delle transazioni notevolmente modificati senza che queste applicazioni ne vengano influenzate.

Utilizzo di ObjectGrid come cache dell'API primaria o di livello secondario

Le API di ObjectGrid possono essere utilizzate direttamente dall'applicazione come cache separata o come cache di scrittura. In modalità di scrittura, l'applicazione si collega a un oggetto Loader in modo che ObjectGrid possa applicare le modifiche e utilizzare i dati in maniera chiara e diretta sull'applicazione. ObjectGrid può essere

utilizzato anche come cache di livello secondario per i più comuni programmi di associazione relazionali degli oggetti. La cache in questa modalità non è visibile all'applicazione in quanto l'applicazione utilizza le API dal programma di associazione come API primaria per l'accesso ai dati.

Capitolo 3. Panoramica su ObjectGrid

ObjectGrid fornisce un modello di accesso ai dati basato su Java Map e una tecnologia di memorizzazione nella cache distribuita. Con ObjectGrid, è possibile configurare un ambiente cluster HA. I client ObjectGrid possono contattare diversi cluster ObjectGrid simultaneamente per soluzioni di integrazione su vasta scala. ObjectGrid fornisce inoltre una soluzione di partizionamento dei dati distribuiti per grosse quantità di informazioni normalizzate con i dati su più di una Java Virtual Machine. Fondamentalmente, ObjectGrid è una serie di API Java standardizzate e servizi di rete che consentono la memorizzazione nella cache locale e distribuita. La soluzione varia da una sola Java virtual machine (JVM) in cui una soluzione Java Map completa è richiesta per un'array di servizi di dati distribuiti e scalabili da diversi cluster ObjectGrid nell'intera azienda.

ObjectGrid in una sola Java virtual machine (JVM)

L'utilizzo di base di ObjectGrid è su una sola JVM.

È possibile utilizzare ObjectGrid per creare una serie di istanze ObjectGrid. Ogni istanza ObjectGrid può contenere una o più istanze compatibili con Java Map. Le istanze Java Map forniscono le interfacce get e put comuni ai programmatori Java più altre funzioni che l'interfaccia Java Map non forniscono. Il seguente diagramma illustra l'utilizzo di base di ObjectGrid.

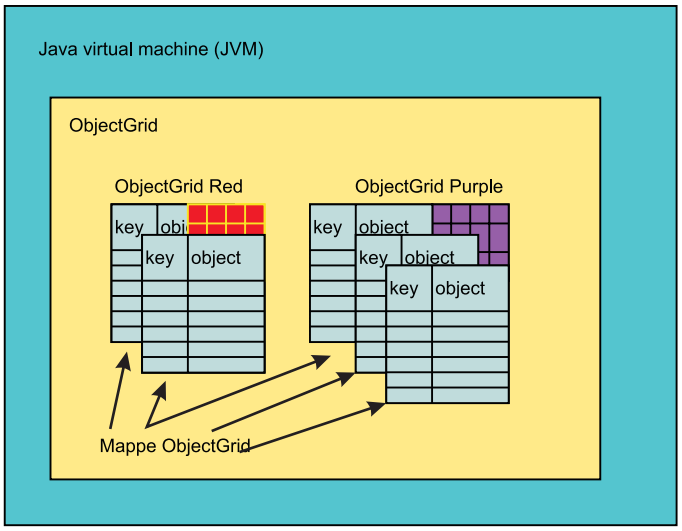


Figura 1. Utilizzo della JVM di ObjectGrid

Un ObjectGrid e una mappa ObjectGrid includono molte funzioni che non sono fornite nell'interfaccia Java Map standard. Queste funzioni comprendono l'accesso alle transazioni, diversi tipi di strategie di blocco (Nessuno, Ottimistico e Pessimistico), Plug and Play Eviction, l'interazione con i database come effetto collaterale di utilizzo delle API get e put e molte altre funzioni. È inoltre possibile sviluppare le proprie estensioni di ObjectGrid. Ad esempio, è possibile sviluppare un listener di mappe che fornisce i risultati per ogni transazione di cui viene eseguito il commit rispetto a una determinata istanza della mappa. Gli utenti possono registrare le modifiche, ad esempio, apportate a un file in un ufficio filiale per essere certi che le transazioni non vadano perse o per propagare le modifiche con Java message service (JMS) o un'altra infrastruttura del genere.

Nel diagramma precedente, la JVM ha due istanze ObjectGrid, una con due oggetti di tipo Java Map da utilizzare e una con tre oggetti Map. Gli oggetti Map sono a due dimensioni, consentendo la modifica di coppie di chiavi e oggetti come fossero un normale Java Map. Una sola istanza di ObjectGrid può supportare molte istanze di mappe.

La seguente configurazione è una configurazione di ObjectGrid di base per le istanze Red e Purple di ObjectGrid:

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

Oltre a utilizzare un file Java archive (JAR) ObjectGrid all'interno di un'unica JVM, è possibile utilizzare ObjectGrid in un ambiente distribuito. In questo ambiente, è possibile creare un cluster ObjectGrid. Un cluster ObjectGrid è costituito da una serie di server ObjectGrid, ognuno con la propria JVM.

La sezione "ObjectGrid in una sola Java virtual machine (JVM)" a pagina 17 descrive il modo in cui ObjectGrid supporta il concetto di una mappa Java. Questo concetto è supportato anche in locale in una singola JVM e in un client Java che si collega a uno o più cluster di cache ObjectGrid remoti. I server ObjectGrid consentono di distribuire le funzioni di base già descritte in precedenza relativamente a una sola JVM. Ad esempio, diversi client possono condividere la stessa mappa di istanza ObjectGrid utilizzando una strategia di blocco Nessuno, Ottimistico o pessimistico. Inoltre, un programma di eliminazione sui server del cluster ObjectGrid può gestire l'eliminazione dei dati delle istanze della mappa sul server. Tutti i client possono utilizzare la semantica get e put e il programma di caricamento configurato sul server del cluster ObjectGrid esegue tutte le interazioni con il database invece che distribuire e gestire i driver JDBC (Java database connectivity) su ogni client.

Nel seguente diagramma, la JVM ha due istanze ObjectGrid: una con due oggetti simili a mappe Java da utilizzare e un altro con tre oggetti dello stesso tipo. Ogni mappa rappresenta un oggetto bidimensionale che consente una chiave e un oggetto. Un'unica istanza ObjectGrid può supportare un numero elevato di mappe, in base ai requisiti dell'applicazione. La differenza in questo caso sta nel fatto che le mappe sono presenti in un server del cluster ObjectGrid. I client possono essere normali server delle applicazioni Java o server delle applicazioni Java 2 Platform, Enterprise Edition (J2EE).

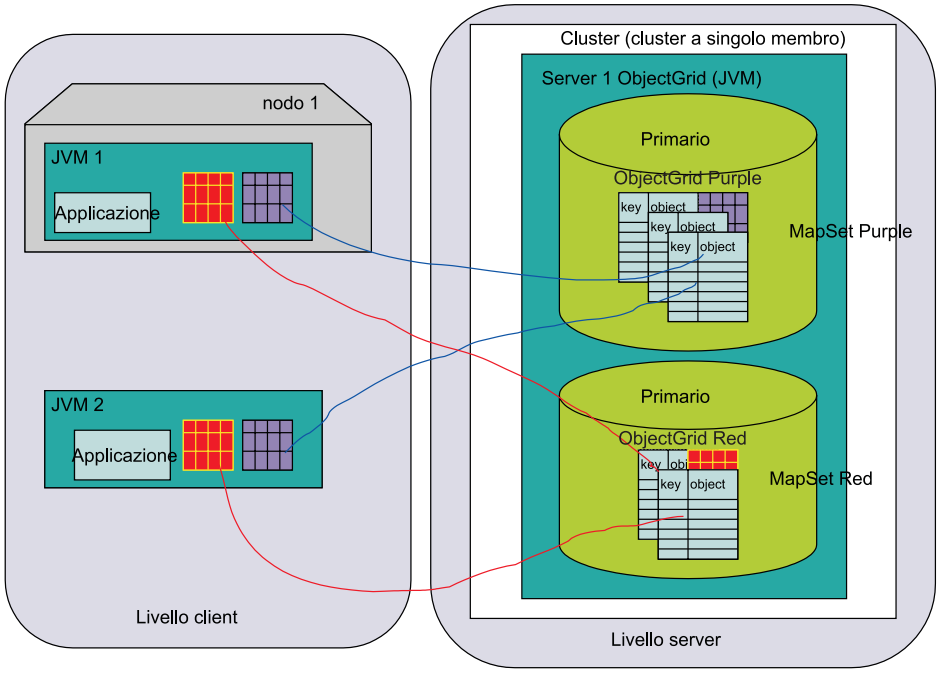


Figura 2. Topologia a singolo server ObjectGrid distribuita (con due MapSets)

Client ObjectGrid

I client ObjectGrid sono costituiti da una serie di API per collegarsi a un cluster ObjectGrid, avviare la configurazione del cluster ObjectGrid ed eseguire quindi le operazioni delle mappe ObjectGrid correntemente distribuite. Un client ObjectGrid è una qualsiasi applicazione Java all'interno della propria istanza JVM che utilizza l'ObjectGrid maniera distribuita. Un client ObjectGrid distribuito può comunque utilizzare le funzioni non distribuite nella stessa Java Virtual Machine. L'utilizzo di un client ObjectGrid può essere complicato quanto un server delle applicazioni con diverse connessioni parallele a ObjectGrid, ognuna con la sicurezza abilitata e funzionante per conto di un utente differente.

Per abilitare il comportamento distribuito, è necessario creare un cluster ObjectGrid (servizi lato server della soluzione ObjectGrid). La configurazione aggiuntiva richiesta è un file XML del cluster ObjectGrid oltre al file di configurazione di ObjectGrid.

Di seguito è riportato il codice XML del cluster ObjectGrid che configura la distribuzione di rete di ObjectGrid nel precedente diagramma:

```
<?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>
```

Questa configurazione describes descrive un unico cluster, ″cluster1″, che contiene il server server1. Il server server1 contiene due ObjectGrids, ″Red″ e ″Purple″. Il file di configurazione riporta le informazioni per il partizionamento e per la replica. Il supporto client-server di ObjectGrid richiede al programmatore di collegarsi a un server definito nel cluster ObjectGrid. Durante questa connessione, la configurazione dell'ObjectGrid e del cluster ObjectGrid viene scaricata dinamicamente sul client, semplificando notevolmente la preparazione del client per l'utilizzo e la gestione del contenuto di configurazione del client. A differenza delclient ObjectGrid che esegue un'operazione ″Connect″, le API di programmazione e i concetti per utilizzare un ObjectGrid esteso alla JVM locale e a una JVM presente nel cluster ObjectGrid sono uguali.

Inizializzazione del cluster ObjectGrid

È possibile avviare ObjectGrid all'interno di un cluster con gli strumenti della riga comandi forniti con ObjectGrid. Un'applicazione ObjectGrid può includere un client ObjectGrid e può essere integrato come verrebbe integrata qualsiasi altra libreria dell'API Java nella struttura di sviluppo dell'applicazione. Tuttavia, in entrambi i casi, l'utilizzo di ObjectGrid deve essere inizializzato.

Per operare nello scenario di utilizzo della Java virtual machine (JVM) locale o in un cluster ObjectGrid distribuito, è necessario ottenere una configurazione valida per l'avvio mediante un approccio facilmente gestibile. I client ObjectGrid e i server del cluster ObjectGrid devono utilizzare una configurazione uniforme. Un programmatore iniziare con una configurazione molto semplice, possibilmente collegata all'interno di un'unica applicazione Java in una JVM.

Quando poi il sistema diventa a più client, con verifica simultanea degli utenti, creare il cluster ObjectGrid a singolo server. Una volta completata la verifica client-server, è possibile lavorare con lo staff di gestione e creare una replica e altri requisiti di soluzione HA. Ognuno degli avanzamenti normali nei requisiti di sviluppo richiede un file di configurazione completo.

Le modifiche al file di configurazione per abilitare ognuna di queste funzioni avanzate per ogni fase di sviluppo descritte sono relativamente modeste, ma ogni fase nello sviluppo della soluzione richiede una versione differente del file di configurazione. Le modifiche si sovrappongono le une alle altre. È possibile verificare una soluzione replicata su una singola macchina se la quantità di dati per sviluppare la soluzione non sovraccarica un singolo sistema o se può essere vincolata a scopo di sviluppo.

Configurazione di ObjectGrid con XML

Una configurazione di ObjectGrid distribuita, con uno o più client e uno o più server ObjectGrid, richiede una configurazione XML. Oltre al file di configurazione XML di base di ObjectGrid, è necessario creare una descrizione XML del cluster ObjectGrid.

Una singola descrizione di configurazione XML di ObjectGrid er una descrizione di configurazione XML del cluster ObjectGrid forniscono ai client e ai server in un unico cluster ObjectGrid le informazioni necessarie per un corretto funzionamento. È possibile avere un qualsiasi numero di cluster ObjectGrid nell'ambiente, tuttavia, un documento XML del cluster ObjectGrid specifico del cluster deve descrivere il determinato cluster.

I file di configurazione richiesti per l'avvio dell'ObjectGrid possono essere acquisiti mediante un normale approccio URL. Ad esempio, i client e i server possono acquisire i file XML con un file fisico o un URL HTTP.

All'interno di un ambiente ObjectGrid distribuito, come illustrato nei seguenti diagrammi, una serie iniziale di server ObjectGrid può essere configurata con la riga comandi per richiamarne le configurazione mediante un URL o, poiché l'URL del file può essere complicato, un file semplice sul file system. Tuttavia, un approccio ancora migliore consiste nell'avviare server successivi all'interno dello stesso ObjectGrid collegandoli da altri server che sono già operativi all'interno del cluster. Questo approccio può essere gestito più facilmente poiché gli amministratori non devono tenere traccia dei file di configurazione su ogni macchina su cui è presente un client o un server ObjectGrid. Inoltre, un server che viene avviato mediante un collegamento avrà sicuramente l'XML elaborato correttamente, riducendo in questo modo il numero di errori di configurazione XML.

Bootstrap

Collegamento al server ObjectGrid

Il seguente diagramma rappresenta il collegamento a un ambiente cluster ObjectGrid tipico su cui è presente la stessa configurazione ObjectGrid, ma che offre una configurazione cluster di replica più completa. In questo caso, il primo server si collega mediante un URL HTTP mentre il secondo e il terzo server sono avviati dal primo. I secondo e il terzo server possono essere avviati dallo stesso URL come il primo server.

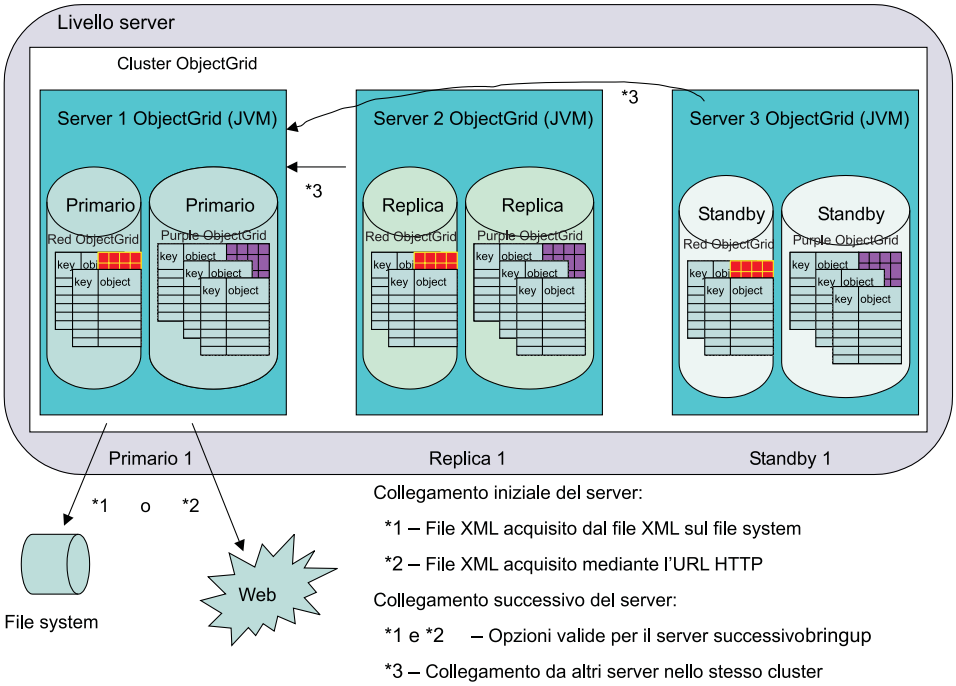


Figura 3. Collegamento iniziale del server mediante la configurazione del file XML o da un server esistente

Come illustrato nel diagramma precedente, il server server1 nel cluster cluster1 si trova nel server iniziale del collegamento. Il server server1 può collegarsi mediante un file XML sul file system mediante un URL a un file locale, un server HTTP remoto o un'altra opzione URL valida. Il server server2 e il server server3 possono essere avviati in questi modi o destinando il server server1 come host di collegamento alla configurazione. In generale, i server successivi al collegamento da altri server garantiscono che la configurazione sia coerente tra vari membri del cluster.

In questo scenario particolare, se il server server1 riporta un errore e i server server2 eserver3 sono operativi, il server1 può essere collegato da server2 o server3, o di nuovo mediante gli approcci file o URL. Fare riferimento a "API di connessione client di ObjectGrid" a pagina 96 per maggiori dettagli sul collegamento e sulle opzioni di configurazione specifiche.

Collegamento al client ObjectGrid

Il client ObjectGrid, per poter utilizzare i servizi dei membri del server del cluster ObjectGrid, deve collegarsi da uno dei server ObjectGrid nel cluster. Ogni client può ″collegarsi″ a un qualsiasi membro attivo del cluster. Gli amministratori possono configurare i server specifici per eseguire tale servizio. Per distribuzioni client elevate, l'unico scopo dei server del cluster ObjectGrid configurato è fornire un supporto di collegamento per i client. Questo approccio è particolarmente utile se il numero di client è elevato e se questi si collegano e si scollegano frequentemente. Quando il client ″si collega″, può ottenere un riferimento distribuito agli ObjectGrids definiti nella configurazione del cluster. Per ulteriori informazioni, consultare "Interfaccia ObjectGridManager" a pagina 89.

I client acquisiscono la configurazione standard dal cluster ObjectGrid, pertanto l'amministratore non deve gestire l'XML per la comunità client. Il client ObjectGrid può utilizzare un URL remoto proprio come i server del cluster ObjectGrid per

sovrascrivere determinate impostazioni che dovrebbero essere specifiche per il client.

Client ObjectGrid in un ambiente ObjectGrid distribuito

I client ObjectGrid possono collegarsi a più di un cluster ObjectGrid simultaneamente. Una singola applicazione Java all'interno di una Java virtual machine (JVM) può collegarsi allo stesso cluster remoto più volte. Questa applicazione può collegarsi anche a cluster remoti differenti allo stesso tempo. Questa funzione è importante poiché consente al client di accedere a molte risorse informative differenti esportate mediante uno o più cluster ObjectGrid.

Il primo caso, in cui lo stesso client ObjectGrid può collegarsi allo stesso server ObjectGrid, è importante per ambienti sicuri, in cui il client può essere un server delle applicazioni e ogni connessione dal server delle applicazioni al cluster ObjectGrid remoto utilizza credenziali di sicurezza differenti. Un altro esempio è un client ObjectGrid che richiede la correlazione di dati da diversi cluster ObjectGrid per un singolo scopo.

Il seguente diagramma riporta uno scenario in cui l'utente del client basato su Web genera, mediante un'applicazione Webm un report da tre diversi settori aziendali. Il motore servlet utilizza la funzione client ObjectGrid del server delle applicazioni per contattare tre diversi cluster ObjectGrid, ognuno gestito da ogni reparto aziendale. In molte aziende, è possibile raccogliere i dati e lo scopo di ObjectGrid sta nel rendere le informazioni disponibili il più possibile. Una volta esternalizzate le informazioni per gli altri utenti, le credenziali di interesse e di sicurezza possono acquisire e utilizzare le informazioni in nuovi modi. I dati possono essere forniti in modalità di sola lettura o per scenari di aggiornamento di sola scrittura.

In questo scenario, i dati possono essere acquisiti in maniera sicura. La memorizzazione nella cache di ObjectGrid in questo scenario non solo consente la condivisione dei dati in maniera programmatica all'interno di ogni reparto aziendale, ma consente l'accesso ai dati dai vari reparti per le informazioni acquisite mediante un modello di programmazione semplice e sicuro utilizzato spesso dagli sviluppatori Java.

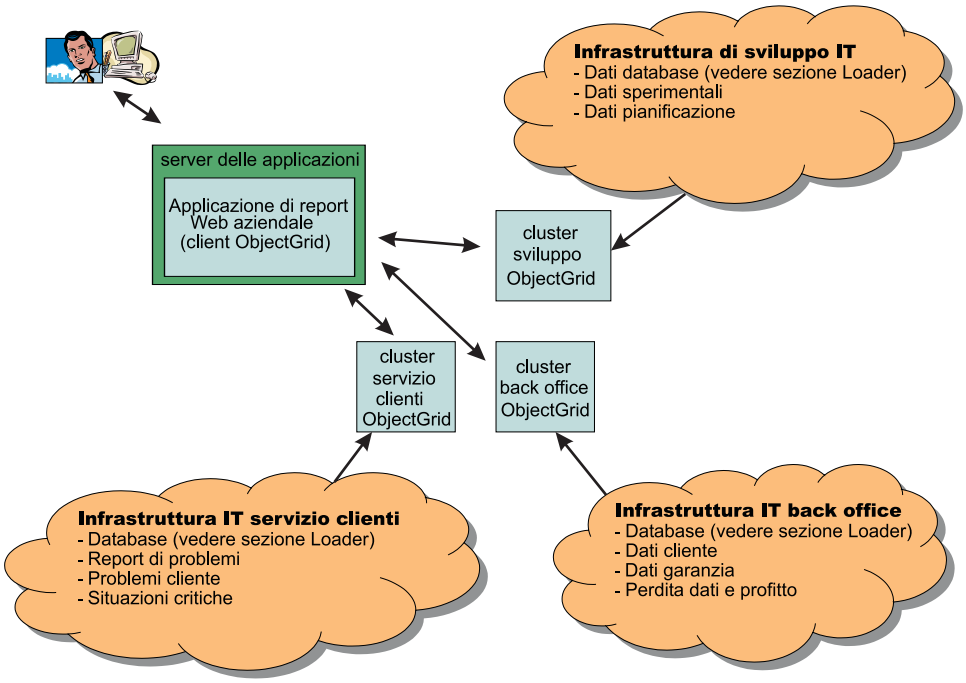


Figura 4. Un utente del client basato su Web genera un report da tre reparti aziendali differenti.

Concetti legati ai cluster ObjectGrid

Il termine *ObjectGrid distribuito* include il concetto che i client possano interagire con uno o più cluster ObjectGrid. Un cluster ObjectGrid è costituito da uno o più server ObjectGrid.

Client ObjectGrid

Un client ObjectGrid può essere concepito in due diversi modi. È possibile pensare a un client come una Java virtual machine (JVM) che utilizza l'API ObjectGrid per collegarsi a un cluster ObjectGrid ed eseguire operazioni di associazione Java rispetto al cluster. Oppure, secondo quello che è il modo più classico di pensare a un client, è possibile considerare a più client all'interno della stessa JVM. Se si utilizza la funzione ObjectGrid fornita, è possibile utilizzare più client all'interno di una stessa JVM.

Ogni volta che un programmatore esegue l'operazione di connessione del client ObjectGrid in una JVM, viene restituito un contesto cluster. Questo contesto rappresenta una sola istanza client. I thread asincroni gestiscono molti aspetti della memorizzazione nella cache per contesto. Per ogni contesto, è possibile utilizzare ObjectGridManager per acquisire ObjectGrids contenuti in istanze del cluster ObjectGrid remoto. Pertanto, in generale, se ci si collega a tre cluster remoti nella stessa JVM, viene implementata una soluzione di tre client all'interno della stessa JVM.

Di seguito sono riportate delle considerazioni importanti per questo scenario. Un'unica sessione di transazioni non può dividere un insieme di mappe all'interno dello stesso cluster. Gli utenti non possono avere un'unica transazione su diversi client collegati allo stesso cluster ObjectGrid o a un cluster differente. Tuttavia, per

gli utenti che provano a integrare grosse quantità di informazioni, è possibile utilizzare una transazione per trasferire le informazioni da ognuno dei cluster ObjectGrid remoti e stampare un report di consolidamento oppure unire le informazioni e inviarle mediante i dati della transazione ObjectGrid a un altro cluster ObjectGrid, o ancora semplicemente aggiornare i singoli cluster ObjectGrid nella maniera definita dal cliente. Ciò è fondamentale in quanto ObjectGrid offre un supporto per le transazioni a singola fase rispetto al supporto a due fasi che di solito offrono i gestori delle transazioni. Per ulteriori informazioni, fare riferimento a “Delimitazione delle transazioni ObjectGrid” a pagina 36.

Replica

È possibile eseguire una replica tra server ObjectGrid che si trovano all'interno dello stesso cluster ObjectGrid. Con la replica, è possibile eseguire più rapidamente un ripristino in seguito a un errore quando il server ObjectGrid primario su cui sono presenti determinate informazioni richieste dall'utente riporta un errore oppure viene arrestato per manutenzione. Nel seguente diagramma, gli ObjectGrid Red ePurple si trovano su due membri del gruppo di replica ObjectGrid differenti. In ObjectGrid, ogni MapSet, un sottoinsieme di ObjectGrid, può essere replicata come una unità. Le PartitionSets rappresentano un'eccezione a questa regola, come verrà discusso nella sezione seguente. La configurazione a singolo server descritta viene modificata nel diagramma per descrivere la replica.

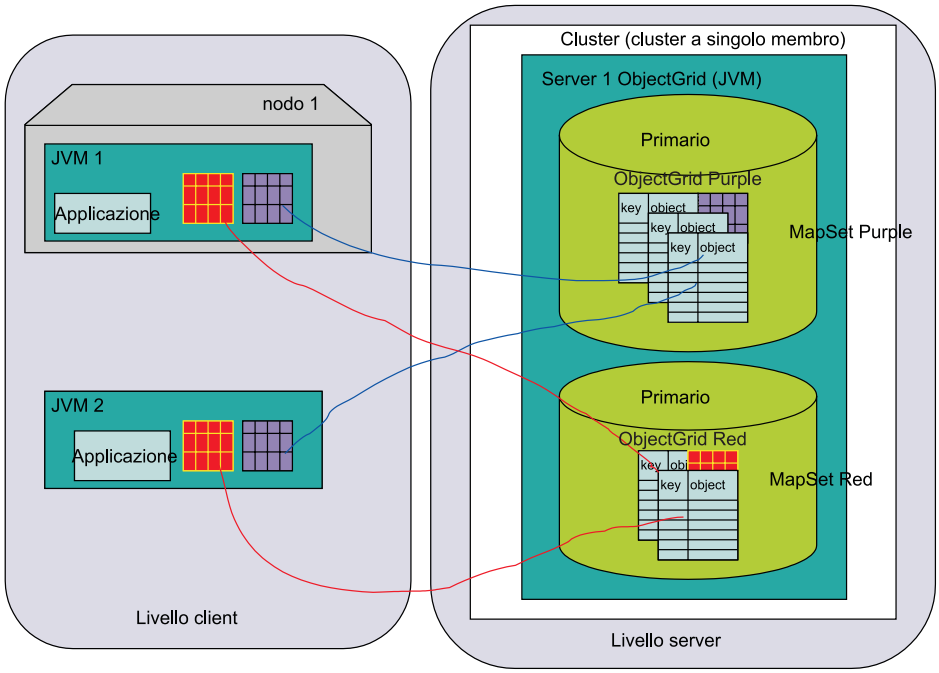


Figura 5. Topologia a singolo server ObjectGrid distribuita con due MapSets

Il diagramma illustra un server delle applicazioni come un'applicazione client e un'applicazione Java autonoma. Entrambi i client richiedono l'accesso a due istanze di ObjectGrid, le istanze Red e Purple nel cluster ObjectGrid a singolo server. Ognuna di queste istanze è contenuta in un membro del gruppo di replica. Il membro del gruppo di replica è un concetto fondamentale ed è la base per le transazioni ObjectGrid. Una transazione può applicare le modifiche soltanto a un unico membro del gruppo di replica.

In un cluster ObjectGrid, il client Java può avviare una transazione ObjectGrid (una sessione) e aggiornare i dati all'interno di un unico membro del gruppo di replica. Ogni membro può quindi essere replicato come unità, in maniera sincrona o asincrona o del tutto non dipendente dai propri requisiti. Ogni richiesta del client ObjectGrid viene quindi indirizzata a un determinato membro del gruppo di replica all'interno dei server del cluster ObjectGrid. L'istanza ObjectGrid nel membro del gruppo di replica che riceve le richieste le elabora e restituisce il risultato al client. Per una replica *sincrona*, ogni richiesta, prima di tornare al client, viene inviata al processo di replica, ovvero al server 2 di ObjectGrid nel seguente diagramma, per confermare che il membro del gruppo di replica ha correttamente applicato l'aggiornamento, quindi il risultato viene restituito al client. In modalità *asincrona*, il client ObjectGrid è in grado di applicare una modifica e il membro del gruppo di replica primario dei server ObjectGrid restituisce il risultato al client, senza attendere la conferma da parte del processo di replica che le modifiche sono state ricevute ed applicate correttamente. In modalità asincrona, l'aggiornamento viene inviato al membro del gruppo di replica del server remoto dopo aver eseguito il commit della transazione sul membro del gruppo di replica primario.

Il seguente diagramma è una versione differente dell'esempio bootstrap. In questo caso, sono presenti tre server, ognuno con un ruolo univoco nella replica delle due istanze ObjectGrid con cui desidera interagire l'utente. Il cluster ObjectGrid è costituito da questi tre server, ognuno dei quali ospita due membri del gruppo di replica. Il server server1 ospita due membri del gruppo primario, il server server2 ospita due membri del gruppo di replica e il server server 3ospita due membri del gruppo standby.

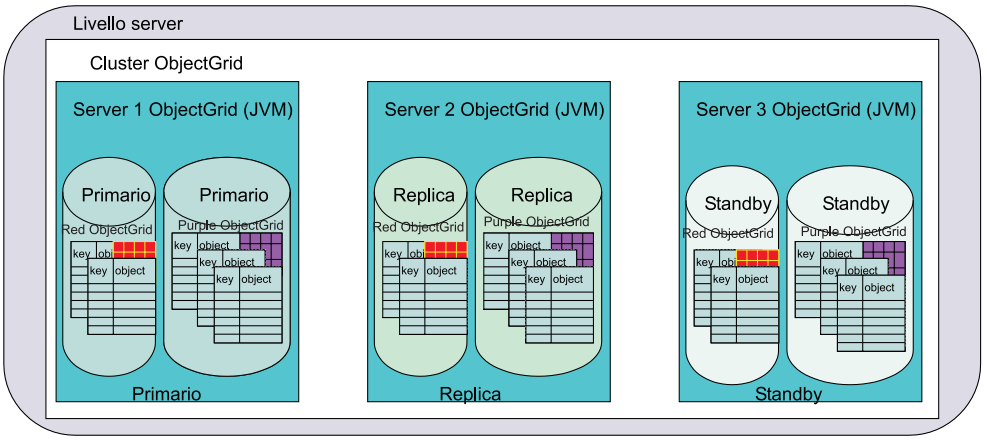


Figura 6. Replica della configurazione di base di esempio

La "Panoramica su HA (high availability)" a pagina 28 descrive questi concetti, tuttavia un concetto fondamentale da comprendere è dato dalle differenze della configurazione necessarie per passare dalla soluzione a singolo server alla soluzione replicata con tre server riportata nel diagramma precedente.

panoramica sulla configurazione di replica a più server

La seguente configurazione è una configurazione ObjectGrid di base per le istanze 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>
```

La conversione di questa configurazione in un cluster ObjectGrid distribuito richiede un ulteriore file di configurazione, il file XML del cluster. Per convertire la configurazione originale per le istanze Red e Purple in un unico server sono necessarie soltanto le aggiunte riportate nel seguente esempio. In particolare, sono aggiunti solo due riferimenti server. Il gruppo di replica era già presente dal file di configurazione iniziale, che faceva riferimento al gruppo di replica ColorMapsReplicationGroup, come illustrato nel seguente esempio:

```
<?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>
```

Nell'esempio precedente, entrambe le MapSets (descritte di seguito) fanno riferimento a ColorMapsReplicationGroup ReplicationGroup, che definisce i server da includere nel gruppo di replica. Il file di configurazione può essere stato ampliato in modo da includere un altro ReplicationGroup, con ogni MapSets con gli stessi server ma in ordine differente o con server differenti in modo da rispondere alle necessità del cliente. La configurazione del cluster ObjectGrid supporta il riutilizzo delle stanze. Per impostazione predefinita, poiché gli attributi di replica di MapSet

non sono impostati e il gruppo di replica ha più di un server, la replica è abilitata e la modalità è asincrona.

Panoramica su HA (high availability)

La replica consente l'utilizzo di HA (high availability) all'interno di un cluster ObjectGrid.

Per comprendere il processo di replica e HA, è necessario conoscere i tipi dei membri del gruppo di replica di ObjectGrid. I tipi di membri del gruppo di replica supportati da ObjectGrid includono i *tipi primari*, i *tipi di replica* e i *tipi standby*. Ognuno di questi tipi ha un determinato ruolo nelle configurazioni HA.

Tipi di membri del gruppo di replica di ObjectGrid

Membro del gruppo di replica primario
Il membro del gruppo di replica primario contiene le ultime viste di dati del client utilizzate. Quando i dati vengono aggiornati, questi vengono propagati sulle repliche. Il gruppo primario è una istanza che comunica con ogni database di connessione mediante l'interfaccia del programma di caricamento di ObjectGrid che propaga i commit in maniera sincrona, in maniera asincrona o in maniera non dipendente dalla configurazione di replica.

Membro del gruppo di replica
Un membro del gruppo di replica contiene una versione dei dati che sono stati propagati dal gruppo primario. Il gruppo primario può essere configurato per inviare le modifiche in diversi modi. Il gruppo di replica deve avere almeno due server con un gruppo di primario e un gruppo di replica, altrimenti la replica non è abilitata.

Membro del gruppo di replica standby
Un membro del gruppo di replica standby non riceve aggiornamento quando vengono apportate le modifiche al gruppo primario come invece accade al gruppo di replica. Esso è semplicemente configurato e pronto per ricevere gli aggiornamento nel caso in cui il grupo primario o il gruppo di replica riporta un errore. Se il gruppo primario riporta un errore, il gruppo di replica diventa il nuovo gruppo primario e il gruppo standby deve essere convertito in un gruppo di replica.

Scenario HA (high availability)

In generale, la replica consente una elevata disponibilità all'interno del cluster ObjectGrid. Le due figure seguenti descrivono gli scenari di errore e di ripristino del gruppo primario. Se il membro di un gruppo di replica primario viene replicato e si verifica un errore, allora uno dei gruppi di replica viene scelto per diventare il nuovo gruppo primario. In questo scenario, esiste un gruppo di replica.

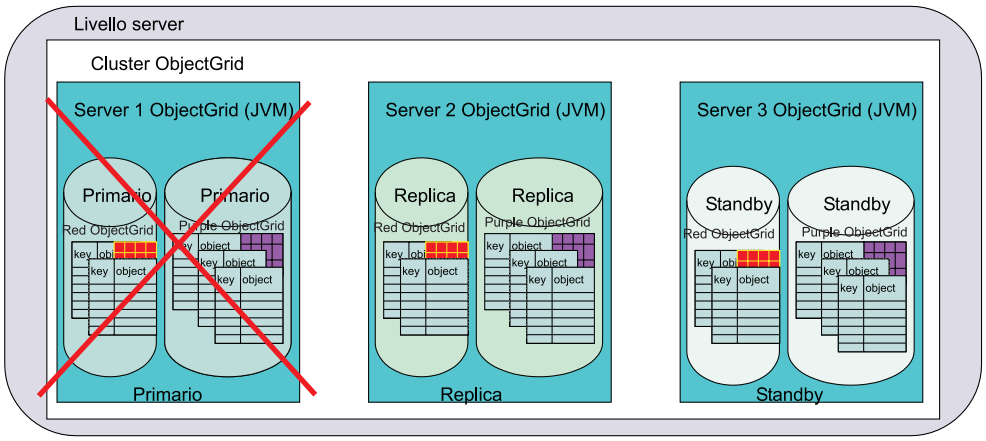


Figura 7. Scenario HA (high availability) di ObjectGrid

Quando viene rilevato un errore, il gruppo primario diventa non disponibile. Il gruppo di replica diventa gruppo primario. Se esiste un gruppo standby, questo diventa un gruppo di replica, come riportato nell'esempio di ripristino del seguente diagramma:

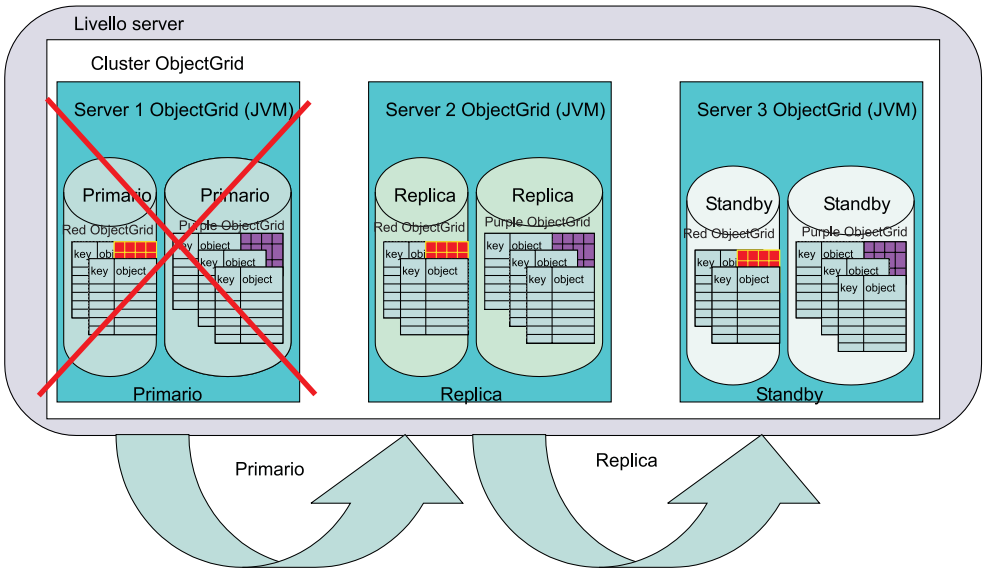


Figura 8. Failover ObjectGrid

I client ObjectGrid sono a conoscenza di questa rettifica durante la connessione a qualsiasi server interessato. I client che contattano il server che ha riportato un errore utilizzano la configurazione runtime e provano a utilizzare gli altri server nel cluster in maniera dinamica. Il client contatta il server successivo nella configurazione. Se il server è attivo ma non operativo, il client attende un periodo di timeout. I client assumono che i membri del gruppo di replica stiano per essere ripristinati da uno stato di errore. Dopo un certo periodo di tempo, i client provano di nuovo e, dopo il gruppo di replica, che ora ha due membri, è diventato operativo, viene fornita una nuova tabella di routing. La tabella di routing descrive dove si trova il gruppo primario, i suoi percorsi di replica e i membri del gruppo di replica che sono al momento in standby.

Serie di configurazioni cluster di ObjectGrid

Quando si configura un cluster ObjectGrid, è necessario separare gli ObjectGrids in MapSets. Questa separazione è importante in quanto unObjectGrid può contenere molte mappe. I MapSets possono essere partizionati con un PartitionSet e replicati con un ReplicationGroup. Ognuna di queste opzioni di configurazione influenza il numero di membri del gruppo di replica che viene creato durante l'avvio del server ObjectGrid. Una rapida panoramica su ogni tipo di serie consente di spiegare il ruolo di ogni tipo.

MapSet di ObjectGrid

Ogni mappa di ObjectGrid può avere requisiti di utilizzo e disponibilità differenti e può essere correlata da un utilizzo tipico dell'applicazione. Ad esempio, una mappa potrebbe essere di sola lettura senza modifiche una volta completato il precaricamento e un'altra potrebbe essere di sola scrittura e partizionata per una maggiore scalabilità. In questo caso, ogni mappa viene inclusa in una MapSet univoca. Nell'esempio precedente, PurpleMapSet e RedMapSet contengono tutte le mappe per ogni ObjectGrid specificato, il che rappresenta l'opzione più semplice.

Una MapSet è una unità che viene replicata sui server ObjectGrid e che viene correlata a un membro del gruppo di replica che non è partizionato. Ogni server del gruppo di replica associato a una MapSet mediante una PartitionSet contiene un membro del gruppo di replica specifico in modo da supportare la configurazione richiesta. Un membro del gruppo di replica è un endpoint univoco all'interno del cluster ObjectGrid e contiene tutte le mappe che una determinata MapSet definisce nella configurazione.

Ad esempio, nel diagramma precedente, il server server1 ha l'unità primaria, il server server2 ha l'unità di replica e il server server3 ha una unità standby che può diventare una unità di replica o primaria a seconda degli scenari di ripristino della replica. Le Mapsets sono correlate alle PartitionSet che descrivono questi tre server. Pertanto, tutte le MapSets, anche se contengono un numero differente di mappe, sono associate allo stesso server in quanto le stanze PartitionSet e ReplicationGroup sono le stesse.

PartitionSet di ObjectGrid

Di solito, la Mapset e i server del gruppo di replica numerati determinano il numero di server che supportano una determinata MapSet in un cluster ObjectGrid. Un membro del gruppo di replica viene creato all'interno di ogni server del gruppo di replica. Tuttavia, il partizionamento può avere un effetto sui membri del gruppo di replica per una MapSet. Il partizionamento è gestito nel file di configurazione mediante la relazione PartitionSet tra MapSet e ReplicationGroup.

Una PartitionSet divide una MapSet in parti in modo che la Java virtual machine (JVM) non debba conservare l'intera MapSet in un unico membro del gruppo di replica primario. Ad esempio, si consideri un database con 1.000.000 di chiavi. Se la dimensione di ogni oggetto a cui fa riferimento ogni chiave è elevata, è probabile che una singola JVM a 32 bit non possa contenere la mappa in memoria in un unico membro del gruppo di replica primario, di replica o standby. Tuttavia, spesso sono richiesti insiemi di dati di dimensioni elevate. Per evitare che l'utente crei manualmente partizioni per i dati, ad esempio per la partizione dell'istanza della prima mappa di ObjectGrid Purple nelle mappe PurpleFirstMapMap1, PurpleFirstMap2, PurpleFirstMapN e l'inserimento di ognuna di esse in una Mapset differente, ObjectGrid può effettuare questa operazione su larga scala.

Successivamente in questo manuale, verrà definito il concetto di PartitionKey. Ciò fa effettivamente riferimento a un'API che può essere richiamata da ObjectGrid per determinare il codice hash della chiave per una determinata voce durante l'inserimento. Se una MapSet ha due partizioni, allora vengono creati due membri del gruppo di replica per questa MapSet. Spesso, a causa del numero elevato di dati, questi membri del gruppo di replica si trovano su server differenti. Per gli sviluppatori, questi membri possono trovarsi sullo stesso server durante la produzione di un modello. Ogni membro del gruppo di replica contiene le chiavi che vengono suddivise sullo stesso valore tra i vari membri del gruppo di replica partizionati che sono disponibili. Come esempio, si consideri che una MapSet sia partizionata in tre modi, 0, 1 e 2. Sono stabiliti tre membri del gruppo di replica primario e uno di essi contiene tutte le chiavi suddivise su un determinato modulo di valore per il numero di membri del gruppo di replica primario. Se un valore hash della chiave ad esempio è 7, 7 modulo3 è 1, pertanto il membro del gruppo di replica primario con l'indice di partizionamento pari a 1 conterrà l'istanza.

Esempio di PartitionSet

Il seguente diagramma illustra la separazione della mappa purple in due partizioni. Ogni chiave nella mappa viene divisa in un intero e assegnata a una determinata serie di partizioni in seguito all'inserimento nel membro del gruppo di replica appropriato. Ogni partizione si trova in un membro del gruppo di replica differente in quanto può essere presente in una JVM differnete e ObjectGrid consente al programmatore di trattare in generale la PurpleFirstMap come singola istanza della mappa logica senza partizioni. Il supporto client e server di ObjectGrid gestisce il corretto instradamento delle richieste tra i membri del gruppo di replica.

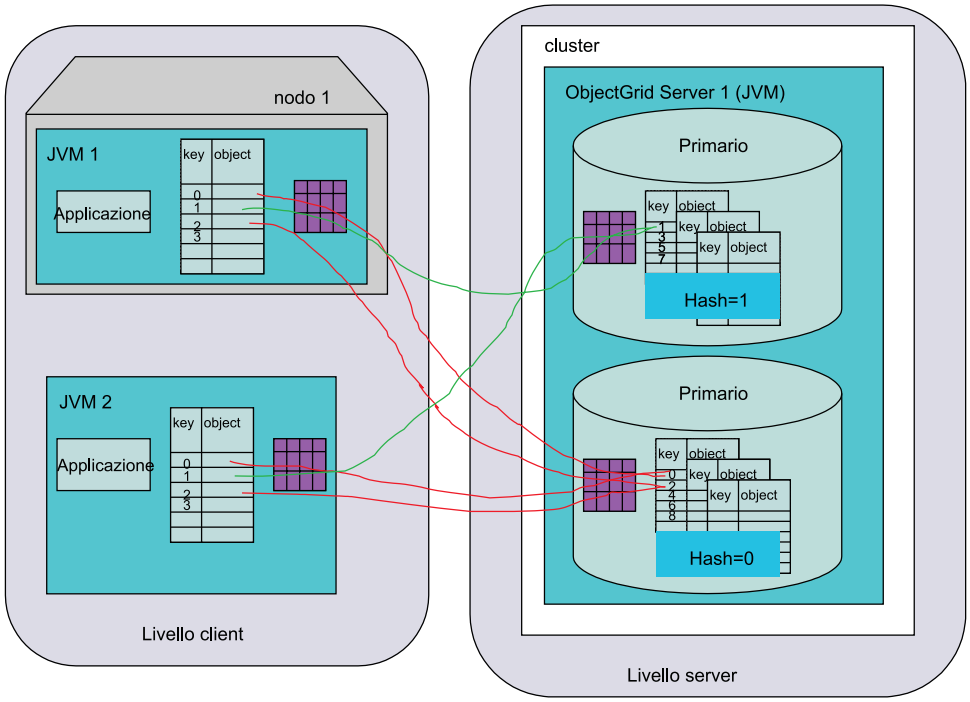


Figura 9. Topologia distribuita di ObjectGrid: singolo server con partizioni

Di seguito è riportata la modifica alla configurazione della serie di partizioni che abilita questa configurazione:

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

La configurazione di esempio riflette come stabilire un ObjectGrid Purple con partizioni replicato. Nel seguente caso, esistono tre server e ognuno dei membri del gruppo di replica primario è associato nello stesso modo all'insieme dei tre server. Essi sarebbero stati associati differentemente se fosse stata utilizzata un'altra stanza 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>
```

Si noti che nell'esempio precedente, una piccola modifica al file di configurazione porta a un nuovo livello di capacità e non richiede la modifica dell'applicazione per

ottenere il risultato desiderato. Il seguente diagramma illustra la vista cluster ObjectGrid e il modo in cui i membri del gruppo di replica vengono sistemati per supportare la configurazione di partizionamento precedente:

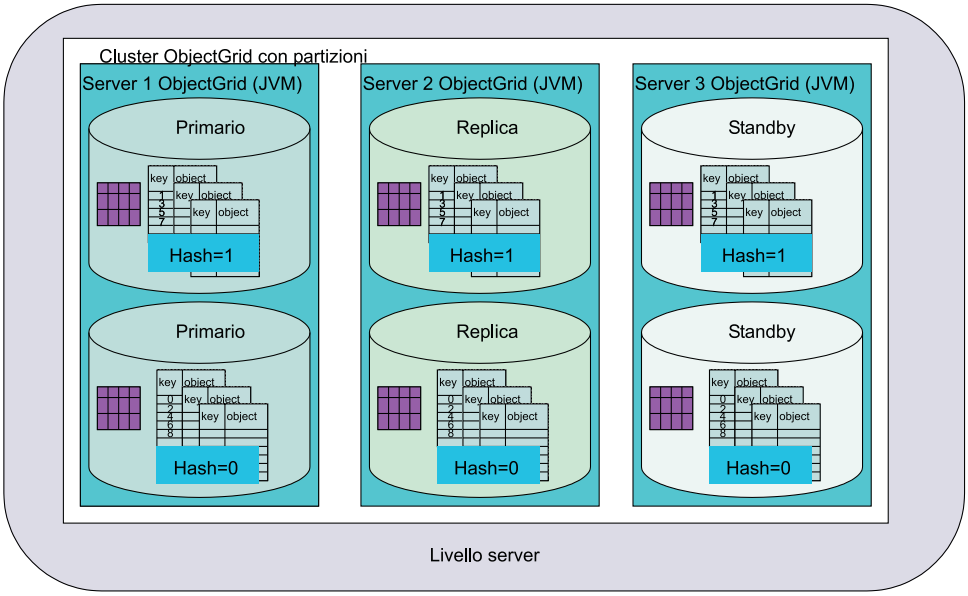


Figura 10. Topologia distribuita di ObjectGrid: più server con partizioni

Per completare la discussione su PartitionSet, di seguito è riportata un'altra variazione dell'esempio precedente. In questo caso, è stato creato un secondo ReplicationGroup. Ogni PartitionSet a questo punto è incluso nel proprio gruppo di replica su server separati.

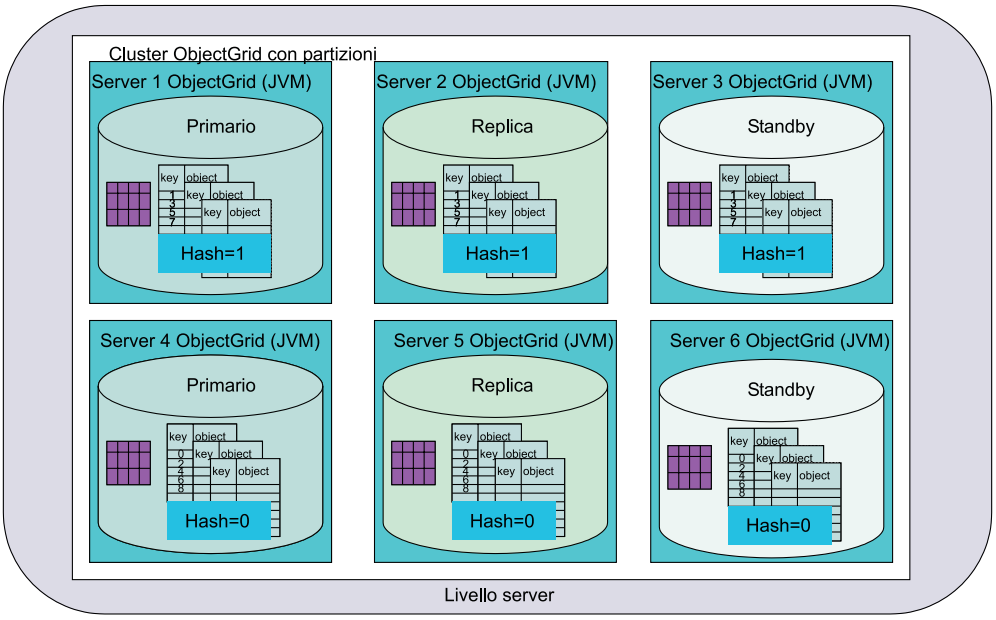


Figura 11. Topologia distribuita di ObjectGrid: più server con partizioni

La configurazione per questa topologia è molto simile agli esempi precedenti. Le modifiche includono tre altre istanze server e un nuovo ReplicationGroup a cui fa riferimento il secondo 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>
```

Questa configurazione dà origine allo stesso numero di membri del gruppo di replica, ma forza il secondo gruppo di replica alla configurazione specifica mediante la propria stanza XML e allo stesso tempo impone gli attributi di ogni istanza su diverse istanze server invece che inserirli come fatto nell'esempio precedente.

Il vantaggio di questa configurazione sta nel fatto che ogni partizione può contenere fino a1.5+ gigabyte di dati, ovvero un totale di 3 gigabyte o più in quanto i due membri del gruppo di replica sono su istanze JVM differenti. Ciò ottimizza quindi l'uso dei 2 gigabyte della JVM a 32 bit di spazio di memoria indirizzabile.

Client ObjectGrid che contattano più cluster ObjectGrid

ObjectGrid è stato progettato non solo per scalare in termini di supporto con partizioni le JVM (Java virtual machines) ma anche per estendere le funzioni di una normale interfaccia Java Map per acquisire le informazioni. È possibile contattare molti cluster ObjectGrid con un unico single client.

Nel seguente scenario, un livello server contiene due clusterObjectGrid. Uno dei client applicativi Java e uno dei server delle applicazioni devono contattare più cluster. Questa è una funzione fondamentale che consente una elevata scalabilità.

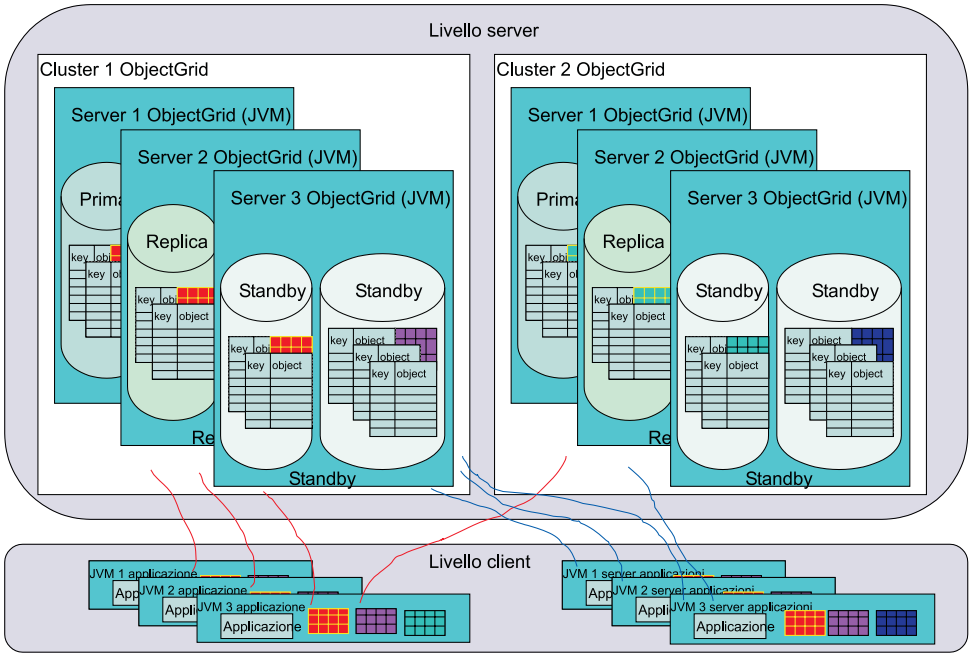


Figura 12. I client ObjectGrid interagiscono con più cluster ObjectGrid

Un cluster ObjectGrid può supportare moltiObjectGrid, ognuno contenente molte configurazioni di MapSet e PartitionSet. Ogni cluster ObjectGrid può essere costituito da una o più JVM. Per un'azienda di grosse dimensioni, la capacità del client ObjectGrid di contattare diversi cluster ObjectGrid allo stesso tempo è una funzione particolarmente importante.

Tuttavia, il client ObjectGrid non può fare riferimento ai dati da più cluster all'interno di una singola transazione. L'applicazione del client ObjectGrid deve eseguire una o più transazioni per memorizzare i dati nella cache nell'istanza della Java virtual machine e correlare quindi le informazioni richiamate come oggetto Java. Gli aggiornamenti basati sulle informazioni devono essere contenuti all'interno di un'unica transazione per ogni cluster. Fare riferimento a "Delimitazione delle transazioni ObjectGrid" a pagina 36 per maggiori dettagli su questo problema.

Supporto per la memorizzazione nella cache prossima del client ObjectGrid

Il client ObjectGrid rappresenta un livello di memorizzazione nella cache. È possibile progettare la propria applicazione in modo da sfruttare la funzione di memorizzazione nella cache se si è a conoscenza del fatto che i dati precedentemente acquisiti dal server remoto non sono inattivi. Ad esempio, se i dati sono stati aggiornati nel cluster ObjectGrid ma non su questo client (il che provoca una nuova richiesta get(...)), allora il client deve aggiornare la cache locale in modo che sia coerente (non tutte le applicazioni richiedono questa operazione) con il cluster ObjectGrid.

Una volta eseguita un'operazione get, una richiesta successiva dell'operazione get per la stessa coppia di chiave e oggetto provoca il rilevamento da parte del client

ObjectGrid dei dati che sono già stati richiamati e l'utilizzo della versione memorizzata nella cache della ″in Java virtual machine (JVM)″ invece che passare sulla rete al cluster ObjectGrid per accedere ai dati. Una volta richiamati i dati dalla rete, questi continueranno a essere forniti dalla cache locale fino a che la voce locale viene eliminata, manualmente o mediante un normale evictor configurato.

Ad esempio, se si è a conoscenza del fatto che i dati sul server vengono aggiornati una volta ogni sei ore, è possibile controllare quando si verifica un aggiornamento alla cache locale o al client della cache prossima. L'utente invalida la voce della cache prossima, quindi emette la richiesta get. La richiesta get contatta il server e acquisisce le informazioni. Si assuma che l'oggetto sia un file di immagini. La prima volta che l'immagine viene scaricata in seguito all'aggiornamento, ogni richiesta successiva non restituisce una chiamata alla procedura remota sul server per ottenere l'immagine.

Il supporto di memorizzazione nella cache prossima non è valido in modalità pessimistica poiché il client potrebbe richiedere il blocco sui dati del cluster ObjectGrid per applicare la strategia di blocco richiesta. Fare riferimento al metodo beginNoWriteThrough() per maggiori dettagli sulla cancellazione o sulla invalidazione di voci della cache prossima che devono essere rimosse senza modificare la vista del cluster dell'ObjectGrid delle informazioni.

Delimitazione delle transazioni ObjectGrid

Un concetto fondamentale che i programmatori devono sempre tenere presente è il concetto relativo alla delimitazione delle transazioni. ObjectGrid non supporta il protocollo di commit a due fasi per l'elaborazione del commit delle transazioni sui membri del gruppo di replica in un cluster ObjectGrid. In una sessione basata su un singolo cluster di ObjectGrid, gli aggiornamenti di lettura-scrittura devono essere applicati a un unico membro del gruppo di replica primario. Se sono interessati più membri, non sarà possibile rendere atomici gli aggiornamenti durante l'elaborazione del commit delle transazioni. Ciò è leggermente differente dal modello di programmazione JAR dell'ObjectGrid locale, che invece consente il commit rispetto a tutte le mappe in un unico ObjectGrid.

Un caso speciale da tenere presente è un PartitionSet in cui è definita più di una partizione. In questi scenari, la chiave 1 si trova sul server 1, la chiave 2 sul server 2 e così via. Se, all'interno di una transazione si verificano degli aggiornamento alla chiave 1 e alla chiave 2, gli aggiornamenti non vengono eseguiti correttamente in quanto una sessione ObjectGrid non può eseguire il commit su due membri del gruppo di replica. Come già detto in precedenza, un PartitionSet che supporta due o più partizioni deve essere utilizzato con attenzione nell'applicazione. Verificare che la sequenza di transazioni non aggiorni i dati da più di una transazione.

Relazione di ObjectGrid con i database

Questa panoramica non è relativa a server ObjectGrid che acquisiscono i dati inizializzati da altrisiti diversi dai client Java. Quando viene avviato un ObjectGrid, viene inizializzato un programma di caricamento per ogni mappa in una MapSet. Questo programma di caricamento consente alle richieste utente in forma di Java Map get o put di essere richiamate o scritte sul database come richiesto. Le operazioni del database non sono visibili per l'utente Java Map e non è richiesta alcuna codifica speciale nell'applicazione. Tuttavia, la funzione del programma di caricamento deve essere sviluppata da un programmatore e configurata per l'utilizzo nell'ObjectGrid prima che l'utente finale possa utilizzare questa funzione. Fare riferimento a “Programmi di caricamento” a pagina 197 per maggiori dettagli.

Inoltre, è fornito un supporto di precaricamento per il precaricamento da un database in seguito all'inizializzazione di un cluster ObjectGrid e un precaricamento con partizioni per garantire che i dati corretti vengano letti per la partizione specifica del membro del gruppo di replica per una determinata MapSet. Gli utenti possono accedere e aggiornare le informazioni di lettura-scrittura con varie strategie di blocco, compreso il tipo nessuno, ottimistico e pessimistico. L'accesso ai dati di sola lettura è supportato e rappresenta il modello più rapido tra tutte le ottimizzazioni che vengono fornite.

Capitolo 4. Supporto didattico per ObjectGrid: modello di programmazione dell'applicazione

Utilizzare questa attività per ottenere maggiori informazioni sul modello di programmazione dell'applicazione ObjectGrid.

Preparare l'ambiente per eseguire le applicazioni ObjectGrid. Fare riferimento alla sezione Capitolo 1, "Introduzione a ObjectGrid mediante l'esecuzione dell'applicazione di esempio", a pagina 1 to per ulteriori informazioni sui percorsi dei file JAR (Java archive), sui requisiti Java e su come eseguire un file semplice per verificare che l'ambiente sia stato impostato correttamente.

Per prima cosa, è necessario decidere qu ale ambiente di programmazione utilizzare per questa attività. È possibile utilizzare un ambiente IDE (integrated development environment) come Eclipse o un ambiente Java della riga comandi. Una volta imparato a utilizzare ObjectGrid, integrarlo con bean enterprise e servlet. Gli esempi riportati in questo supporto didattico non fanno riferimento ad alcun ambiente Java particolare, quindi è possibile utilizzarne uno a scelta.

Nella sua definizione di base, un ObjectGrid è una cache e un repository in memoria per gli oggetti. L'utilizzo di una mappa java.util.Map per memorizzare e accedere agli oggetti è un processo del tutto simile all'utilizzo di ObjectGrid. Allo stesso tempo, ObjectGrid è più che una semplice cache. Esplorando le varie opzioni e i diversi plug-in mediante questa attività, si scoprirà che ObjectGrid è molto estensibile e flessibile. È possibile utilizzare ObjectGrid come una semplice cache *separata* o come una cache più elaborata utilizzata da un gestore risorse.

Gli esempi riportati in questo supporto didattico non sono programmi completi. Le importazioni, l'elaborazione delle eccezioni e alcune delle variabili non sono dichiarate completamente in ogni esempio. È possibile utilizzare questi esempi per scrivere i propri programmi.

Utilizzare questa attività per utilizzare ObjectGrid da un programma Java.

1. Individuare le API ObjectGrid e le eccezioni. Tutte le API pubbliche e le eccezioni di ObjectGrid sono contenute nel pacchetto `com.ibm.websphere.objectgrid`. Per informazioni su una configurazione o un sistema più avanzato, fare riferimento alle API e alle eccezioni aggiuntive nel pacchetto `com.ibm.websphere.objectgrid.plugins`. Quando sono fornite implementazioni di plug-in, posizionare tali classi nel pacchetto `com.ibm.websphere.objectgrid.plugins.builtins`. Per le funzioni di sicurezza di ObjectGrid, ricercare i pacchetti con **security** nel nome, come ad esempio `com.ibm.websphere.objectgrid.security`, `com.ibm.websphere.objectgrid.security.plugins` e `com.ibm.websphere.objectgrid.security.plugins.builtins`.

   Questa attività si basa sulle API presenti nel pacchetto `com.ibm.websphere.objectgrid`. La JavaDoc completa per ObjectGrid si trova nella directory: `<root_install>/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. Richiamare o creare un'istanza di ObjectGrid. Utilizzare ObjectGridManagerFactory per richiamare l'istanza singleton di ObjectGridManager. Quindi, creare un'istanza di ObjectGrid con le seguenti istruzioni:

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

   L'interfaccia ObjectGridManager utilizza diversi metodi per la creazione, il richiamo e la rimozione delle istanze di ObjectGrid. Fare riferimento alla sezione "Interfaccia ObjectGridManager" a pagina 89 per scegliere una variante della propria situazione. È inoltre possibile impostare la traccia mediante l'interfaccia ObjectGridManager. Se il sistema viene eseguito su WebSphere Extended Deployment o WebSphere Application Server, tali metodi non sono necessari in quanto la traccia è gestita dalle funzioni incluse. Se l'esecuzione avviene all'esterno di WebSphere Application Server, questi metodi possono risultare utili. Fare riferimento a "Traccia di ObjectGrid" a pagina 96 per le informazioni complete su questi metodi.

3. Inizializzare l'ObjectGrid.
   a. Impostare un nome per l'ObjectGrid, se non è già stato impostato mediante il metodo `create`.
   b. Definire le BackingMaps, utilizzando la configurazione predefinita per un BackingMap per le applicazioni iniziali.
   c. Una volta definiti i BackingMap, inizializzare l'ObjectGrid. L'inizializzazione di ObjectGrid segnala che la configurazione è completa e si desidera cominciare a utilizzare ObjectGrid.
   d. Una volta inizializzato l'ObjectGrid, richiamare un oggetto Session. Fare riferimento a "Interfaccia ObjectGrid" a pagina 103 e alla JavaDoc per ulteriori informazioni.

   Utilizzare il seguente esempio come guida per questa operazione:

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

4. Utilizzare le sessioni per gestire le operazioni transazionali. L'accesso a una cache ObjectGrid è transazionale: gli accessi, gli inserimenti, gli aggiornamenti e le rimozioni di oggetti dalla cache sono contenuti in un'unica unità di lavoro, detta sessione. Alla fine di una sessione, è possibile eseguire il commit di tutte le modifiche all'interno dell'unità di lavoro oppure eseguire un rollback e annullare tutte le modifiche.

   È inoltre possibile utilizzare la funzione di commit automatico per eseguire singole operazioni atomiche rispetto alla cache. In assenza di un contesto di sessione attivo, i singoli accessi al contenuto della cache sono racchiusi nelle proprie sessioni di cui è stato eseguito il commit automatico.

   Un altro aspetto importante dell'interfaccia Session è il richiamo di un accesso transazionale, o handle, per il BackingMap con l'interfaccia ObjectMap. È possibile utilizzare il metodo getMap per creare un handle ObjectMap su un BackingMap predefinito. Tutte le operazioni eseguite rispetto alla cache, come gli inserimenti, gli aggiornamenti e le eliminazioni, vengono completati con

l'istanza ObjectMap. Per ulteriori informazioni, fare riferimento a "Interfaccia Session" a pagina 112. Utilizzare il seguente esempio per ottenere e gestire una sessione:

```
Session session = objectGrid.getSession();
ObjectMap objectMap = session.getMap("someMap");
session.begin();
objectMap.insert("key1", "value1");
objectMap.insert("key2", "value2");
session.commit();
objectMap.insert("key3", "value3"); // auto-commit
```

5. Utilizzare l'interfaccia ObjectMap per accedere e aggiornare la cache. Nell'interfaccia ObjectMap sono disponibili diversi metodi per l'accesso e l'aggiornamento delle cache. L'interfaccia ObjectMap è modellata come l'interfaccia di una mappa. Tuttavia, sono introdotte le eccezioni controllate per un aiuto durante lo sviluppo delle applicazioni ObjectGrid con un ambiente IDE, come Eclipse. Se si desidera utilizzare un'interfaccia java.util.Map senza le eccezioni controllate. è possibile utilizzare il metodo getJavaMap. Per ulteriori informazioni, consultare "Interfacce ObjectMap e JavaMap" a pagina 117.

   I metodi insert e update espliciti si basano sull'operazione put. È sempre possibile utilizzare il metodo put, ma l'utilizzo dei metodi insert e update è preferibile e risulta più utile. L'utilizzo del metodo put è chiarificato dalla definizione di un metodo put senza una precedente operazione get come metodo insert. Se è stata eseguita un'operazione get prima dell'operazione put, allora l'operazione put viene trattata come un inserimento o un aggiornamento, a seconda della voce esistente nella cache.

   È possibile eseguire le seguenti operazioni di base per ObjectMap: get, put, insert, update, remove, touch, invalidate e containsKey. Per maggiori dettagli e le variazioni sul tema, fare riferimento alla sezione "Panoramica sul modello di programmazione del sistema" a pagina 43 o alla documentazione delle API di ObjectMap. Il seguente esempio riporta l'uso di ObjectMap per modificare la cache:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid objectGrid =
objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
ObjectMap objectMap = session.getMap("someMap");
// Avvia una transazione/sessione...
session.begin();
objectMap.insert("key1", "value1");
objectMap.put("key2", "value2");
session.commit();
// Verifica se è stato eseguito il commit delle modifiche
String value1 = (String)objectMap.get("key1");
String value2 = (String)objectMap.get("key2");
System.out.println("key1 = " + value1 + ", key2 = " + value2);
// Avvia una nuova transazione/sessione...
session.begin();
objectMap.update("key2", "newValue2");
objectMap.remove("key1");
session.rollback();
// Verifica che non sia stato eseguito il commit delle modifiche
String newValue1 = (String)objectMap.get("key1");
String newValue2 = (String)objectMap.get("key2");
System.out.println("key1 = " + newValue1 + ", key2 = " + newValue2);
```

6. Utilizzare l'indice per ricerca gli oggetti memorizzati nella cache. L'indice consente alle applicazioni di trovare gli oggetti in base a un valore o a un intervallo di valori specifico. La mappa BackingMap deve avere il plug-in

dell'indice configurato prima che le applicazioni possano utilizzare la funzione indice. Per utilizzare l'indice, le applicazioni devono ottenerlo dal metodo getIndex() dell'interfaccia ObjectMap e ne devono eseguire il casting sull'interfaccia corretta, come MapIndex o MapRangeIndex o un'interfaccia di indice personalizzata.

Al momento, la funzione di indicizzazione è supportata solo nella cache locale. La funzione di indicizzazione non è supportata nella cache distribuita. Se si prova a eseguire una operazione di indicizzazione su una cache distribuita, viene emessa l'eccezione `UnsupportedOperationException`.

Il seguente esempio illustra come utilizzare l'indice:

```
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));
}
```

Una volta finito di leggere le informazioni riportate in questa sezione e dopo aver apportato le modifiche al codice in base alle proprie necessità, si avrà una conoscenza più approfondita del modello di programmazione di ObjectGrid.

Per informazioni più specifiche, fare riferimento a Capitolo 9, "Panoramica sull'interfaccia di programmazione dell'applicazione ObjectGrid.", a pagina 89.

Introduzione a ObjectGrid remoto

Inserire qui una descrizione da utilizzare per il primo paragrafo e l'oggetto.

La Capitolo 4, "Supporto didattico per ObjectGrid: modello di programmazione dell'applicazione", a pagina 39 ha a che fare di solito con un utilizzo ″locale″ o all'interno di un'applicazione di un ObjectGrid. Un'applicazione ha creato un'istanza di un ObjectGrid e ha utilizzato tale istanza. Una volta terminata la JVM dell'applicazione, viene terminata anche la cache dell'ObjectGrid. Un ObjectGrid remoto, come suggerisce il nome, consente l'acceso a un ObjectGrid che si trova su una JVM differente. Più client possono collegarsi all'ObjectGrid remoto e accedere all'ObjectGrid utilizzando la stessa API.

1. fare riferimento alle seguenti sezioni per iniziare:
   - Capitolo 3, "Panoramica su ObjectGrid", a pagina 17
   - "Configurazione di ObjectGrid" a pagina 256
   - "API di connessione client di ObjectGrid" a pagina 96
   - Capitolo 8, "Supporto per la riga comandi", a pagina 81
2. Per avviare i server, è necessario definire il file XML ObjectGrid e il file XML del cluster. Fare riferimento a "Configurazione di ObjectGrid distribuita" a pagina 269. Questa sezione fa riferimento ai file `university.xml` e `universityCluster.xml`. È possibile utilizzare questi file come esempio, modificando il nome host e la porta per avviare il server. Fare riferimento a Capitolo 8, "Supporto per la riga comandi", a pagina 81 per maggiori dettagli sull'avvio di server ObjectGrid.
3. Quando il server è in esecuzione, un client può collegarsi a questo server. Fare riferimento a "API di connessione client di ObjectGrid" a pagina 96 per maggiori dettagli su come un client può collegarsi e su come eseguire operazioni su ObjectGrid.

Panoramica sul modello di programmazione del sistema

Il modello di programmazione del sistema fornisce diverse funzioni aggiuntive e punti di estensione per ObjectGrid.

Il seguente diagramma illustra il modo in cui il modello di programmazione del sistema fornisce diverse funzioni aggiuntive e punti di estensione.

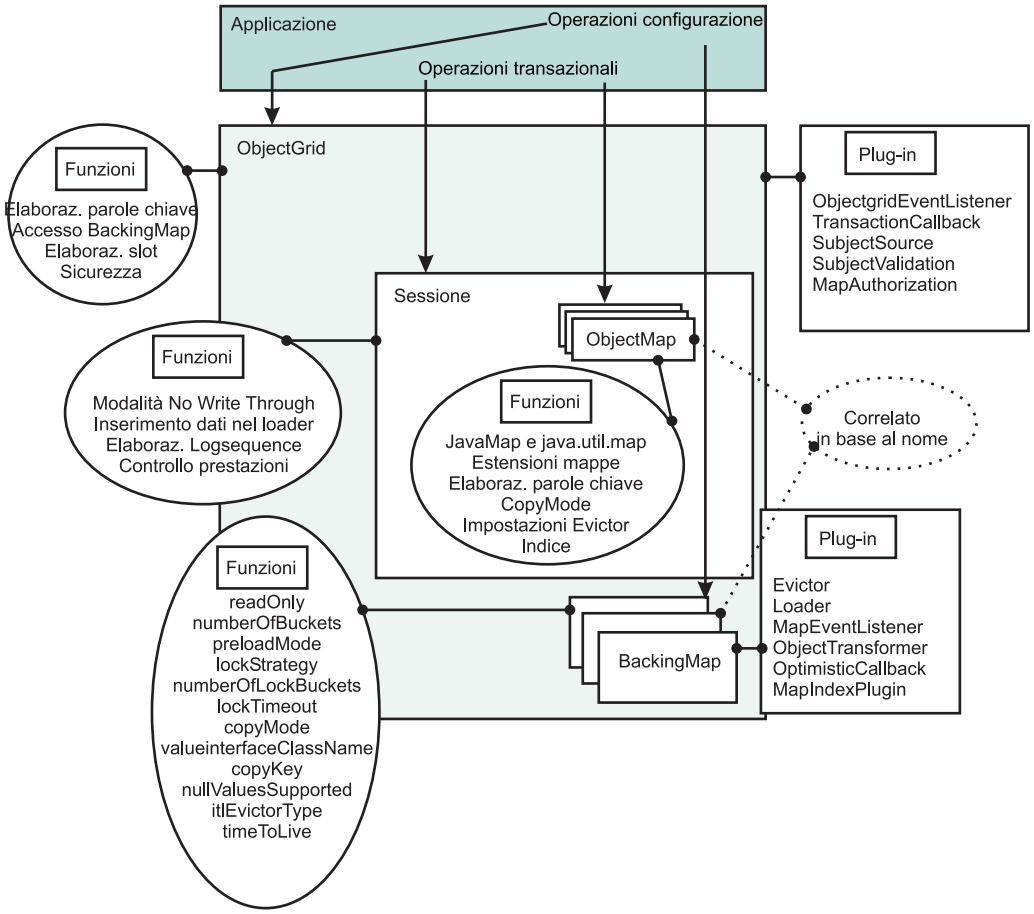


Figura 13. Panoramica su ObjectGrid

Un plug-in in ObjectGrid è un componente che fornisce un determinato tipo di funzione per i componenti ObjectGrid collegabili che includono ObjectGrid e BackingMap. Un'opzione rappresenta una funzione o caratteristica specifica di un componente ObjectGrid, che può essere ObjectGrid, Session, BackingMap, ObjectMap e così via. Se l'opzione rappresenta una funzione, allora può essere utilizzata per raggiungere un determinato obiettivo di elaborazione. Se invece un'opzione è una caratteristica, allora può essere utilizzata per definire il funzionamento dei componenti ObjectGrid.

Nelle seguenti sezioni sono descritte alcune delle opzioni ed estensioni illustrate nel diagramma precedente:

- "Panoramica sul modello di programmazione del sistema: opzioni e punti di collegamento dell'interfaccia ObjectGrid" a pagina 44

  L'interfaccia ObjectGrid ha diversi punti di collegamento e opzioni per un'interazione estensibile con ObjectGrid.

- “Panoramica sul modello di programmazione del sistema: opzioni e plug-in dell’interfaccia BackingMap” a pagina 46

  L’interfaccia BackingMap ha diversi punti di collegamento e opzioni per un’interazione estensibile con ObjectGrid.
- “Panoramica sul modello di programmazione del sistema: opzioni dell’interfaccia Session” a pagina 54

  L’interfaccia Session ha diverse opzioni per un’interazione estensibile con ObjectGrid. Le sezioni riportate in questo capitolo descrivono la la funzione e forniscono frammenti di codice per lo scenario di utilizzo.
- “Panoramica sul modello di programmazione del sistema: opzioni dell’interfaccia ObjectMap” a pagina 56

  L’interfaccia ObjectMap ha diverse opzioni per un’interazione estensibile con ObjectGrid. Le sezioni riportate in questo capitolo descrivono la la funzione e forniscono frammenti di codice per lo scenario di utilizzo.

Per ulteriori informazioni sulle singole funzioni sui plug-in, fare riferimento a Capitolo 9, “Panoramica sull’interfaccia di programmazione dell’applicazione ObjectGrid.”, a pagina 89.

Panoramica sul modello di programmazione del sistema: opzioni e punti di collegamento dell’interfaccia ObjectGrid

L’interfaccia ObjectGrid ha diversi punti di collegamento e opzioni per un’interazione estensibile con ObjectGrid.

Le seguenti sezioni descrivono queste opzioni e forniscono alcuni brevi frammenti di codice per lo scenario di utilizzo. Dove appropriato, è fornito anche un frammento XML per mostrare la configurazione XML alternativa. Per ulteriori informazioni, fare riferimento alle sezioni “Interfaccia ObjectGrid” a pagina 103 e “Configurazione di ObjectGrid” a pagina 256.

Elaborazione delle parole chiave

L’interfaccia ObjectGrid fornisce un meccanismo di invalidazione flessibile basato su parole chiave. Una parola chiave è un’istanza non null di qualsiasi oggetto serializzabile. È possibile associare le parole chiave alle voci BackingMap in più modi. La maggior parte dell’elaborazione delle parole chiave viene eseguita a livello di ObjectMap, ma l’associazione di una parola chiave a un’altra per formare una struttura gerarchica di parole chiave viene eseguita al livello di ObjectGrid.

Il metodo associateKeyword(java.io.Serializable parent, java.io.Serializable child) collega due parole chiave in una relazione direzionale. Se viene invalidata una voce principale, allora verrà invalidata anche la voce secondaria. L’invalidazione dell’elemento secondario non ha invece alcun impatto sulla voce principale. Ad esempio, questo metodo viene utilizzato per aggiungere la voce della mappa New York come elemento secondario della voce USA, e se USA viene invalidata, allora anche tutte le voci New York vengono invalidate. Fare riferimento al seguente codice di esempio:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
// associa diverse città alla parola chiave "USA"
objectGrid.associateKeyword("USA", "New York");
objectGrid.associateKeyword("USA", "Rochester");
objectGrid.associateKeyword("USA", "Raleigh");
:
// inserisce diverse voci con diverse parole chiave
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");
:
// invalida tutte le voci associate alla parola chiave "USA", lasciando
// le voci "key2" e "key6"
objectMap.invalidateUsingKeyword("USA", true);
:
```

Per ulteriori informazioni, vedere “Parole chiave” a pagina 120.

Accesso a BackingMap

ObjectGrid fornisce l’accesso agli oggetti BackingMap. È possibile ottenere l’accesso a BackingMap con i metodi defineMap o getMap. Per ulteriori informazioni, consultare “Interfaccia BackingMap” a pagina 108. Nel seguente esempio vengono creati due riferimenti a BackingMap:

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

Elaborazione degli slot

È possibile riservare uno slot per la memorizzazione di oggetti che vengono utilizzati nel corso di una transazione, come ad esempio l’oggetto ID di transazione (TxID) o una connessione al database (Connection). Questi oggetti memorizzati vengono quindi referenziati con un indice specifico, fornito dal metodo reserveSlot. Per ulteriori informazioni sull’utilizzo degli slot, fare riferimento alle sezioni “Programmi di caricamento” a pagina 197 e “Plug-in TransactionCallback” a pagina 213. Il seguente frammento di codice dimostra l’elaborazione degli slot:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
int index = objectGrid.reserveSlot
(com.ibm.websphere.objectgrid.TxID.SLOT_NAME);
:
// Utilizza l'indice in un secondo momento quando si memorizzano
   o si richiamano gli oggetti
// dall'oggetto TxID ...
TxID tx = session.getTxID();
tx.putSlot(index, someObject);
:
Object theTxObject = tx.getSlot(index);
:
```

Elaborazione della sicurezza

Le mappe possono essere protette utilizzando dei meccanismi di sicurezza. I seguenti metodi sono disponibili su un ObjectGrid per la configurazione e l’utilizzo delle funzioni di sicurezza.

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

Fare riferimento alla sezione “Sicurezza di ObjectGrid” a pagina 136 per ulteriori informazioni sui meccanismi di sicurezza disponibili.

ObjectGridEventListener

Il listener ObjectGridEventListener fornisce un mezzo perché le applicazioni possano ricevere una notifica in caso di un inizio o di un commit di una transazione. Un'istanza di ObjectGridEventListener può essere impostata sull'ObjectGrid. Per ulteriori informazioni, fare riferimento a "Listener" a pagina 182. Di seguito è riportato un esempio di implementazione programmatica dell'interfaccia ObjectGridEventListener:

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

You can also perform the same configuration with XML:

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

Plug-in TransactionCallback

Il richiamo dei metodi sulla sessione invia i corrispondenti eventi al plug-in TransactionCallback. Un ObjectGrid può non aver alcun plug-in TransactionCallback oppure ne può avere uno. I BackingMap definiti su un ObjectGrid con un plug-in TransactionCallback devono avere un programma di caricamento corrispondente. Per ulteriori informazioni, consultare "Plug-in TransactionCallback" a pagina 213. Il seguente frammento di codice dimostra come implementare il plug-in TransactionCallback in maniera programmatica:

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

You can perform the same configuration with XML:

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

Panoramica sul modello di programmazione del sistema: opzioni e plug-in dell'interfaccia BackingMap

L'interfaccia BackingMap ha diversi punti di collegamento per una maggiore interazione con ObjectGrid.

Le seguenti sezioni descrivono queste opzioni e forniscono alcuni brevi frammenti di codice per lo scenario di utilizzo. Dove appropriato, è fornito anche un frammento XML per mostrare la configurazione XML alternativa. Per maggiori informazioni, fare

riferimento alle sezioni “Interfaccia BackingMap” a pagina 108 e “Configurazione di ObjectGrid” a pagina 256 o alla documentazione dell’API.

Attributi di configurazione

Diversi elementi di configurazione sono associati a BackingMaps:

- **ReadOnly** (valore predefinito `false`): l’impostazione di questo attributo su `true` rende la mappa di backup di sola lettura. L’impostazione su `false` la rende invece di lettura e scrittura. Se non si specifica alcun valore, verrà utilizzato il valore predefinito, che è lettura e scrittura.
- **NullValuesSupported** (valore predefinito `true`): il supporto di un valore null significa che un valore null può essere inserito in una mappa. Se questo attributo è impostato su true, i valori null sono supportati nell’ObjectMap, altrimenti non lo sono. Se sono supportati i valori null, un’operazione `get` che restituisce un valore null può significare sia che il valore è realmente null sia che la mappa non contiene la chiave.
- **NumberOfBuckets** (valore predefinito `503`): Specifica il numero di contenitori che sono utilizzati da questo BackingMap. L’implementazione di BackingMap utilizza una mappa hash per l’implementazione. Se in BackingMap esistono molte voci, allora un numero elevato di contenitori porta a migliori prestazioni in quanto il rischio di collisioni diminuisce man mano che il numero di contenitori aumenta. Un numero elevato di contenitori implica anche una maggiore simultaneità.
- **NumberOfLockBuckets** (valore predefinito `383`): specifica il numero di contenitori di blocco utilizzati dal gestore blocco per questo BackingMap. Quando l’attributo lockStrategy è impostato su OPTIMISTIC o PESSIMISTIC, viene creato un gestore blocchi per il BackingMap. Tale gestore utilizza una mappa hash per tenere traccia delle voci che vengono bloccate da una o più transazioni. Se nella mappa hash esistono molte voci, allora un numero elevato di contenitori porta a migliori prestazioni in quanto il rischio di collisioni diminuisce man mano che il numero di contenitori aumenta. Un numero elevato di contenitori di blocco implica anche una maggiore simultaneità. Quando lockStrategy è impostato su NONE, nessun gestore di blocchi è utilizzato da BackingMap. In questo caso, l’impostazione dell’attributo numberOfLockBuckets non ha alcun effetto.

Esempio di configurazione programmatica

Il seguente esempio configura le proprietà su una mappa di backup:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// override default of read/write
backingMap.setReadOnly(true);
// override default of allowing Null values
backingMap.setNullValuesSupported(false);
// override default (prime numbers work best)
backingMap.setNumberOfBuckets(251);
// override default (prime numbers work best)
backingMap.setNumberOfLockBuckets(251);
:
```

Esempio di configurazione XML

Il seguente esempio di configurazione XML configura le stesse proprietà dimostrate nel precedente esempio programmatico.

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

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

Strategia di blocco

Quando la strategia di blocco è impostata su OPTIMISTIC o PESSIMISTIC, viene creato un gestore di blocchi per il BackingMap. Per evitare che si verifichino delle condizioni di stallo, il gestore di blocchi ha un valore di timeout predefinito per attendere che venga concesso un blocco. Se viene superato questo valore di timeout, si verifica una eccezione LockTimeoutException. Il valore predefinito di 15 secondi è sufficiente per la maggior parte delle applicazioni, ma su sistemi con un carico di lavoro elevato, è possibile che il valore di timeout venga raggiunto anche quando non esiste alcuna condizione di stallo reale. In tal caso, il metodo setLockTimeout può essere utilizzato per aumentare il valore di timeout di blocco dal valore predefinito a un valore desiderato per evitare che vengano restituite false eccezioni di timeout. Quando la strategia di blocco è impostata su NONE, nessun gestore di blocchi è utilizzato da BackingMap. In questo caso, l'impostazione dell'attributo lockTimeout non ha alcun effetto. Per ulteriori informazioni, fare riferimento alla sezione "Blocco" a pagina 127.

Esempio di configurazione programmatica

Nell'esempio riportato di seguito viene impostata la strategia di blocco:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// override default value of OPTIMISTIC
backingMap.setLockStrategy(LockStrategy.PESSIMISTIC);
backingMap.setLockTimeout(30); // imposta il valore di timeout su 30 secondi
:
```

Esempio di configurazione XML

Nel seguente esempio viene impostata la stessa strategia di blocco definita nel precedente esempio programmatico.

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

Copia delle chiavi e dei valori

La copia delle chiavi e dei valori può essere un processo costoso, sia dal punto di vista delle risorse che delle prospettive. Senza la possibilità di eseguire queste copie, è possibile che si verifichino problemi strani, di cui è difficile eseguire il debug. ObjectGrid fornisce la possibilità di configurare dove e quando eseguire le copie delle chiavi e dei valori. Di solito, le chiavi sono considerate costanti, quindi non è necessario copiare gli oggetti delle chiavi. Per impostazione predefinita quindi, gli oggetti delle chiavi non vengono copiati. È invece molto più probabile che i valori vengano modificati dall'applicazione. La creazione di un oggetto valore rispetto al riferimento corrente al valore è un'opzione configurabile. Fare riferimento

alla sezione Capitolo 11, “Procedure ottimali delle prestazioni di ObjectGrid”, a pagina 325 e alla JavaDoc per maggiori dettagli sulle impostazioni CopyKey e CopyMode.

Esempio di configurazione programmatica

Di seguito è riportato un esempio di impostazione della modalità di copia e della copia di una chiave:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.setCopyKey(true); // esegue una copia di ogni nuova chiave
backingMap.setCopyMode(NO_COPY); // Più efficiente - considera l'applicazione sicura
:
```

Esempio di configurazione XML

Il seguente esempio porta alla stessa configurazione descritta nel precedente esempio di configurazione programmatica:

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

Programmi di esclusione

I programmi di esclusione sono utilizzati per cancellare periodicamente le voci non necessarie nella mappa. Le voci che vengono rimosse sono definite dal programma di esclusione stesso. I programmi di esclusione integrati si basano sul tempo in cui una voce è rimasta attiva nella mappa. Altre strategie di esclusione si basano sull'utilizzo, sulla dimensione o su una combinazione di altri fattori.

- **Programma di esclusione TTL (Time To Live) integrato**: il programma di esclusione TTL integrato fornisce un paio di elementi di configurazione impostati su BackingMap con i metodi setTtlEvictorType e setTimeToLive. Per impostazione predefinita, questo programma di esclusione non è attivo. È possibile attivarlo richiamando il metodo setTtlEvictorType con uno dei seguenti tre valori: CREATION_TIME, LAST_ACCESS_TIME o NONE (predefinito). Quindi, a seconda del tipo di programma di esclusione TimeToLive selezionato, il valore del metodo setTimeToLive viene utilizzato per impostare la durata per ogni voce della mappa.
- **Plug-in del programma di esclusione:** oltre al programma di esclusione TTL integrato, un'applicazione può fornire il proprio plug--in di implementazione del programma di esclusione. È possibile utilizzare periodicamente un qualsiasi algoritmo per invalidare le voci della mappa.

Configurazione programmatica

La seguente classe crea un programma di esclusione:

```
class MyEvictor implements com.ibm.websphere.objectgrid.plugins.Evictor { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// il timer viene avviato quando viene creata la prima voce
backingMap.setTtlEvictorType(CREATION_TIME);
```

```
// consente a ogni voce della mappa di durare 30 secondi prima dell'invalidazione
backingMap.setTimeToLive(30);
// sia il programma di esclusione integrato che quello personalizzato saranno attivi
backingMap.setEvictor(new MyEvictor()); :
```

Configurazione XML

Il seguente codice XML crea una configurazione identica alla precedente configurazione programmatica:

```
:
<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>
:
```

Per ulteriori informazioni, vedere “Programmi di esclusione” a pagina 187.

Programmi di caricamento

Un programma di caricamento di ObjectGrid è un componente collegabile che consente a una mappa ObjectGrid di funzionare come cache di memoria per i dati che vengono normalmente conservati in un archivio permanente sullo stesso sistema o su un sistema differente. Di solito, come archivio permanente viene utilizzato un database o un file system. Un programma di caricamento utilizza la logica di lettura e scrittura dei dati su un archivio permanente.

Un programma di caricamento è un plug-in opzionale per una mappa di backup di ObjectGrid. Soltanto un programma di caricamento può essere associato a una determinata mappa e ogni mappa ha la propria istanza del programma di caricamento. La mappa di backup richiede tutti i dati che non sono forniti mediante il programma di caricamento. Qualsiasi modifica apportata alla mappa viene inserita mediante il programma di caricamento. Il plug-in del programma di caricamento fornisce un modo per spostare i dati tra la mappa e l’archivio permanente.

Configurazione programmatica

Di seguito è riportato un esempio di implementazione del programma di caricamento:

```
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);
:
```

Configurazione XML

Il seguente esempio XML porta alla stessa configurazione descritta nel precedente esempio di configurazione programmatica:

```
:
<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>
:
```

Per ulteriori informazioni, fare riferimento alla sezione “Programmi di caricamento” a pagina 197.

Interfaccia MapEventListener

L’interfaccia di richiamata MapEventListener è implementata dall’applicazione quando si desidera ricevere eventi relativi a una mappa come ad esempio l’esclusione di una voce o il completamento del precaricamento della mappa. Il seguente esempio di codice dimostra come impostare un’istanza di MapEventListener su un’istanza di BackingMap:

Configurazione programmatica

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

Configurazione XML

Il seguente esempio porta alla stessa configurazione descritta nel precedente esempio di configurazione programmatica:

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

Per ulteriori informazioni, consultare la sezione “Listener” a pagina 182.

Interfaccia ObjectTransformer

L'interfaccia ObjectTransformer può essere utilizzata per serializzare le chiavi e i valori delle voci nella cache che non sono definite come serializzabili in modo che sia possibile definire il proprio schema di serializzazione senza estendere o implementare l'interfaccia serializzabile direttamente. Questa interfaccia fornisce inoltre dei metodi per l'esecuzione della funzione di copia di chiavi e valori. Di seguito è riportata una classe che implementa l'interfaccia ObjectTransformer:

Configurazione programmatica

```
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);
:
```

Configurazione XML

Il seguente esempio XML porta alla stessa configurazione descritta nel precedente esempio di configurazione programmatica:

```
:
<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>
:
```

Per ulteriori informazioni, fare riferimento alla sezione "Plug-in ObjectTransformer" a pagina 209.

Interfaccia OptimisticCallback

L'interfaccia OptimisticCallback può essere utilizzata per creare ed elaborare un campo della versione associato a un determinato valore. In molti casi, l'utilizzo diretto del valore per determinare se un altro client della cache lo ha modificato dall'ultima volta che è stato richiamato, non è un metodo efficienti e presenta degli errori. Un'alternativa consiste nel fornire un altro campo che rappresenti lo stato del valore. Questo è lo scopo di questa interfaccia OptimisticCallback: fornire un valore alternativo che rappresenti l'oggetto valore. Di seguito è riportata una configurazione di esempio di OptimisticCallback:

Configurazione programmatica

```
class MyOptimisticCallback implements
 com.ibm.websphere.objectgrid.plugins.OptimisticCallback { ... }
:
```

```
OptimisticCallback myOptimisticCallback = new MyOptimisticCallback();
myOptimisticCallback.setVersionType("Integer");
backingMap.setOptimisticCallback(myOptimisticCallback);
:
```

Configurazione XML

Il seguente esempio porta alla stessa configurazione descritta nel precedente esempio di configurazione programmatica:

```
:
<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>
:
```

Indici

Un MapIndexPlugin, o Indice in breve, è un'opzione utilizzata da BackingMap per creare un indice in base all'attributo specificato dell'oggetto memorizzato. L'indice consente alle applicazioni di trovare gli oggetti in base a un valore o a un intervallo di valori specifico. Per utilizzare l'indice, le applicazioni devono ottenerlo dal metodo getIndex() dell'interfaccia ObjectMap e ne devono eseguire il casting sull'interfaccia corretta, come MapIndex o MapRangeIndex o un'interfaccia di indice personalizzata.

Al momento, la funzione di indicizzazione è supportata solo nella cache locale e non nella cache distribuita. Se si prova a eseguire una operazione di indicizzazione su una cache distribuita, viene emessa l'eccezione UnsupportedOperationException.

Esistono due tipi di indice: l'indice statico e l'indice dinamico. Gli indici statici possono essere creati mediante una configurazione programmatica e una configurazione XML. Gli indici dinamici invece possono essere creati solo in maniera programmatica.

Configurazione programmatica

Il seguente esempio di codice dimostra come aggiungere un indice statico a una istanza BackingMap:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("indexSampleGrid");
BackingMap personBackingMap= objectGrid.getMap("person");
// utilizzare la classe com.ibm.websphere.objectgrid.plugins.index.HashIndex
// integrata come classe dei plugin dell'indice.
HashIndex mapIndexPlugin = new HashIndex();
mapIndexPlugin.setName("CODE");
mapIndexPlugin.setAttributeName("EmployeeCode");
mapIndexPlugin.setRangeIndex(true);
personBackingMap.addMapIndexPlugin(mapIndexPlugin);
```

```
// Nota: la precedente configurazione dell'indice assume che l'oggetto memorizzato
// abbia un attributo denominato EmployeeCode e un metodo getEmployeeCode()
// che restituiscono il valore dell'attributo EmployeeCode.
:
```

Il seguente esempio di codice dimostra come creare un indice dinamico su un'istanza BackingMap:

```
class DynamicIndexCallbackImpl implements
com.ibm.websphere.objectgrid.plugins.index.DynamicIndexCallback { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("indexSampleGrid");
BackingMap personBackingMap= objectGrid.getMap("person");
objectGrid.initiallize();
:
// inserimento, aggiornamento o rimozione di dati
// L'indice dinamico può essere creato dopo che l'istanza ObjectGrid
// è stata inizializzata
// Se è necessario creare un indice dinamico, crearlo senza
// DynamicIndexCallback
personBackingMap.createDynamicIndex("CODE2", true, "employeeCode", null);
:
// Un'altra opzione consiste nel creare l'indice dinamico con DynamicIndexCallback
// Si assuma che la classe DynamicIndexCallbackImpl implementi
// l'interfaccia DynamicIndexCallback
personBackingMap.createDynamicIndex("CODE3", true, "employeeCode",
new DynamicIndexCallbackImpl());
:
```

Configurazione XML

Il seguente esempio porta alla stessa configurazione descritta nel precedente esempio di configurazione programmatica dell'indice statico:

```
:
<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>
:
```

Fare riferimento alla sezione Indicizzazione per ulteriori informazioni.

Panoramica sul modello di programmazione del sistema: opzioni dell'interfaccia Session

L'interfaccia Session ha diverse opzioni per un'interazione estensibile con ObjectGrid. Le seguenti sezioni descrivono queste opzioni e forniscono alcuni brevi frammenti di codice per lo scenario di utilizzo.

Per ulteriori informazioni sull'interfaccia Session, fare riferimento a "Interfaccia Session" a pagina 112.

Modalità no write through

A volte le applicazioni devono semplicemente applicare le modifiche alla mappa di base ma non al programma di caricamento. Il metodo beginNoWriteThrough dell'interfaccia Session è progettato per far sì che ciò sia possibile. Il metodo isWriteThroughEnabled dell'interfaccia Session può essere utilizzato per verificare se la sessione corrente scrive sul programma di caricamento del backend. Ciò potrebbe essere utile ad altri utenti dell'oggetto Session per conoscere il tipo di sessione correntemente elaborato. Il seguente esempio abilita la modalità no write through:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
boolean isWriteThroughEnabled = session.isWriteThroughEnabled();
// esegue gli aggiornamenti alla mappa ...
session.commit();
:
```

Inserimento di dati solo sul programma di caricamento

Le applicazioni possono applicare le modifiche locali della sessione al programma di caricamento senza eseguire il commit delle modifiche in maniera permanente richiamando il metodo flush, come riportato nel seguente esempio:

```
:
Session session = objectGrid.getSession();
session.begin();
// apporta alcune modifiche
session.flush();
// inserisce le modifiche sul Loader, ma non esegue ancora il commit
// apporta altre modifiche
session.commit();
:
```

Metodo processLogSequence

Il metodo processLogSequence viene utilizzato per elaborare una LogSequence. Ciascun elemento LogElement all'interno di LogSequence viene esaminato e viene eseguita l'operazione appropriata, come un inserimento, un aggiornamento o un'invalidazione, rispetto al BackingMap identificato da LogSequence MapName. Perché questo metodo possa essere richiamato, un oggetto Session di ObjectGrid deve essere attivo. Il chiamante è responsabile del commit o del rollback per completare l'oggetto Session. L'elaborazione di un commit automatico non è disponibile per questo metodo.

Questo metodo viene utilizzato principalmente per elaborare un LogSequence che è stato ricevuto da una JVM remota. Ad esempio, utilizzando il supporto di commit distribuito, i LogSequence che sono associati a una determinata sessione di cui è stata eseguito il commit vengono distribuiti ad altri ObjectGrid e su altre JVM (Java virtual machines). Dopo aver ricevuto i LogSequences sulla JVM remota, il listener può avviare un oggetto Session mediante il metodo beginNoWriteThrough, richiamare il metodo processLogSequence e quindi eseguire il metodo di commit sull'oggetto Session. Di seguito viene riportato un esempio:

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

Controllo delle prestazioni

Le mappe possono facoltativamente essere utilizzate per il controllo delle prestazioni durante l'esecuzione su WebSphere Application Server. Il metodo `setTransactionType` è disponibile su un oggetto Session per la configurazione e l'utilizzo delle funzioni di controllo delle prestazioni. Per ulteriori informazioni, consultare "Controllo delle prestazioni di ObjectGrid con la PMI (performance monitoring infrastructure) di WebSphere Application Server" a pagina 291.

Panoramica sul modello di programmazione del sistema: opzioni dell'interfaccia ObjectMap

L'interfaccia ObjectMap ha diverse opzioni per un'interazione estensibile con ObjectGrid.

Per ulteriori informazioni sull'interfaccia ObjectMap, fare riferimento a "Interfacce ObjectMap e JavaMap" a pagina 117.

Interfaccia JavaMap e java.util.Map

Per le applicazioni che desiderano utilizzare l'interfaccia java.util.Map, ObjectMap utilizza il metodo getJavaMap in modo che le applicazioni possano richiamare l'implementazione dell'interfaccia java.util.Map salvata da ObjectMap. L'istanza della mappa restituita può quindi essere utilizzata con l'interfaccia JavaMap, che estende l'interfaccia Map. L'interfaccia JavaMap utilizza le stesse firme del metodo di ObjectMap, ma con una diversa gestione delle eccezioni. JavaMap estende l'interfaccia java.util.Map, in modo che tutte le eccezioni siano istanze della classe java.lang.RuntimeException. Poiché JavaMap estende l'interfaccia java.util.Map, è possibile utilizzare ObjectGrid con un'applicazione esistente che utilizza un'interfaccia java.util.Map per la memorizzazione nella cache degli oggetti. Di seguito viene riportato un esempio di frammento di codice:

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

Estensioni dell'interfaccia Map

L'interfaccia ObjectMap fornisce opzioni funzionali aggiuntive oltre alle funzioni di eccezione controllate. Ad esempio, un utente può specificare che una determinata voce della mappa viene aggiornata con il metodo getForUpdate che indica al runtime di ObjectGrid e al plug-in del programma di caricamento che la voce può essere bloccata durante l'elaborazione, se possibile. L'elaborazione batch è un'altra funzione aggiuntiva con i metodi getAll, putAll e removeAll. Per ulteriori informazioni su questi metodi, fare riferimento alla documentazione dell'API.

Elaborazione delle parole chiave

La maggior parte delle operazioni delle mappe hanno una versione del parametro delle parole chiave, come insert, get, getForUpdate, put, remove e invalidate. Per una facilità di utilizzo, viene fornito anche il metodo setDefaultKeyword. Questo metodo associa le voci a una parola chiave senza utilizzare la versione della parola chiave dell'operazione della mappa. Di seguito viene riportato un esempio di parola chiave:

```
:
// setDefaultKeyword
session.begin();
objectMap.setDefaultKeyword("New York");
Person p = (Person) objectMap.get("Billy"); // "Billy" entry has "New York" keyword
p = (Person) objectMap.get("Bob", "Los Angeles"); // "Bob" entry
//has "Los Angeles" keyword
objectMap.setDefaultKeyword(null);
p = (Person) objectMap.get("Jimmy"); // "Jimmy" entry has no keyword
session.commit();
:
// Versione del parametro della parola chiave dell'operazione insert
session.begin();
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person, "Rochester"); // "BillyBob" has
//"Rochester" keyword
session.commit();
:
```

Per ulteriori informazioni, consultare “Parole chiave” a pagina 120.

Metodo della modalità di copia

Il metodo setCopyMode consente alla modalità di copia per la mappa di essere sovrascritta sulla mappa solo per questa sessione o transazione. Questo metodo consente a un'applicazione di utilizzare una modalità di copia ottimale per sessione, in base alle necessità. La modalità di copia non può essere modificata durante una sessione attiva. Esiste un metodo clearCopyMode corrispondente che reimposta la modalità di copia su quella definita su BackingMap. È possibile richiamare questo metodo quando non è presente alcuna sessione attiva. Di seguito è riportato un esempio di impostazione della modalità di copia:

```
:
objectMap.setCopyMode(CopyMode.COPY_ON_READ, null);
session.begin();
// modifica objectMap ...
session.commit();
objectMap.clearCopyMode(); // reimposta CopyMode sull'impostazione di BackingMap
session.begin();
// modifica objectMap ...
session.commit();
:
```

Per ulteriori informazioni, fare riferimento alle sezioni “Plug-in ObjectTransformer” a pagina 209 e Capitolo 11, “Procedure ottimali delle prestazioni di ObjectGrid”, a pagina 325.

Impostazioni del programma di esclusione

È possibile sovrascrivere il valore di timeout TimeToLive per il programma di esclusione TTL integrato al livello di ObjectMap. Il metodo setTimeToLive stabilisce il numero di secondi della durata di ogni voce della cache. Se questo metodo viene modificato, viene restituito il valore TimeToLive precedente. Questo valore

TimeToLive è il tempo minimo che una voce rimane nella cache prima di essere considerata per l'esclusione e indica al programma di esclusione TimeToLive integrato per quanto tempo una voce deve essere conservata in seguito all'ultimo accesso. Il nuovo valore TimeToLive si applica soltanto alle voci ObjectMap a cui accede la transazione avviata dall'oggetto Session utilizzato per ottenere l'istanza ObjectMap. Il nuovo valore TimeToLive si applica a qualsiasi transazione in corso per l'oggetto Session e per le transazioni future eseguite dall'oggetto Session. Il nuovo valore TimeToLive non influenza le voci di un'istanza ObjectMap a cui accede una transazione avviata da qualsiasi altro oggetto Session. Richiamando questo metodo sull'ObjectMap, qualsiasi valore precedentemente impostato dal metodo setTimeToLive su BackingMap viene sovrascritto dall'istanza ObjectMap. Di seguito viene riportato un esempio:

```
:
session.begin();
int oldTTL = objectMap.setTimeToLive(60); // imposta il valore TTL a 60 secondi
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person); // "BillyBob" entry will have a TTL
// pari a 60 secondi
session.commit();
:
objectMap.setTimeToLive(oldTTL); // reimposta TTL sul valore originale
Person person2 = new Person("Angelina", "Jolie", "somewhere");
objectMap.insert("Brad", person2); // "Brad" entry will use original TTL value
:
```

Per ulteriori informazioni, fare riferimento alla sezione “Programmi di esclusione” a pagina 187.

Capitolo 5. Esempi di ObjectGrid

In questa sezione sono descritti gli esempi di ObjectGrid che sono forniti quando si installa WebSphere Extended Deployment.

Panoramica

Diversi esempi specifici di ObjectGrid illustrano l'integrazione con le applicazioni Java 2 Platform, Enterprise Edition (J2EE) e l'utilità di partizione (WPF). In questa sezione sono riportati questi esempi, le funzioni dimostrate da ogni esempio, il percorso di ogni esempio e gli ambienti in cui l'esempio viene eseguito.

In questa sezione vengono descritti gli esempi forniti con l'installazione di WebSphere Extended Deployment. Altri esempi relativi all'utilizzo dell'integrazione di Java Message Service (JMS) e di ObjectGrid con altre strutture open source sono forniti al seguente indirizzo Web: http://www-1.ibm.com/support/docview.wss?uid=swg27006432 .

Esempi

- **ObjectGridSamplesSA:** questo esempio è una serie di esempi Java 2 Platform, Standard Edition (J2SE) che sono assemblati nel file `objectgridSamples.jar` per la dimostrazione delle funzioni ObjectGrid. Tali esempi J2SE possono essere eseguiti in un ambiente J2SE. Il file `objectgridSamples.jar` contiene il file `SamplesGuide.htm`, che contiene le istruzioni per l'esecuzione degli esempi.
- **ObjectGridSample:** questo esempio è un esempio J2EE che dimostra il modo in cui i servlet e i bean enterprise Session utilizzano le funzioni ObjectGrid. Questo esempio è fornito nel file EAR (enterprise archive) `ObjectGridSample.ear`. Il file `ObjectGridSample.ear` contiene il file `readme.txt`, che contiene a sua volta le istruzioni per l'impostazione e l'esecuzione di questo esempio.
- **ObjectGridPartitionCluster:** questo esempio è un esempio J2EE che dimostra il modo in cui WPF e ObjectGrid funzionano insieme, come utilizzare ObjectGridEventListener per propagare le modifiche agli oggetti e come abilitare il routing basato sul contesto in modo da garantire integrità e coerenza in ObjectGrid. Questo esempio è fornito nel file EAR (enterprise archive) `D_ObjectGridPartitionClusterSample.ear`. Il file `D_ObjectGridPartitionClusterSample.ear` contiene il file `readme.txt`, che contiene a sua volta le istruzioni per l'impostazione e l'esecuzione di questo esempio.
- **ObjectGridJMSSamples:** questa è una serie di esempi J2EE assemblati nel file ObjectGridJMSSamples.zip che dimostra come utilizzare la funzione JMS per trasmettere le modifiche in una istanza di ObjectGrid a un'altra istanza di ObjectGrid su un'unica JVM o un unico ambiente cluster. Tali esempi J2EE sono disponibili su Web all'indirizzo: http://www-1.ibm.com/support/docview.wss?uid=swg27006432 .

Funzionalità di esempio

Tabella 2. Funzionalità di esempio

Area funzionale	Esempio ObjectGrid SamplesSA	Esempio ObjectGrid Sample	Esempio ObjectGrid Sample	Esempio ObjectGrid JMSSamples
ObjectGrid EventListener			x	x

Tabella 2. Funzionalità di esempio (Continua)

Area funzionale	Esempio ObjectGrid SamplesSA	Esempio ObjectGrid Sample	Esempio ObjectGrid Sample	Esempio ObjectGrid JMSSamples
Richiamata delle transazioni	x	x	x	
Loader	x	x	x	
Listener MapEvent	x			
Object Transformer	x	x	x	x
Richiamata ottimistica	x	x	x	
Modalità di copia di BackingMap	x	x		
Invalidazione distribuita			x	x
Aggiornamento distribuito			x	x
Elaborazione di LogSequence				x
Utilità di partizione (WPF)			x	
Java Message Service (JMS)				x
Indice della mappa	x			
Cluster ObjectGrid	x	x		
Contesto ClusterClient ObjectGrid	x	x		
ObjectGrid distribuito	x	x		
Gestione ObjectGrid	x			
Sicurezza di ObjectGrid	x		x	

Ubicazione

Una volta installato WebSphere Extended Deployment, i seguen ti file .jar si trovano nelle seguenti directory:

Tabella 3. Percorsi degli esempi

Esempio	Ubicazione
ObjectGridSamplesSA	*root_install*\optionalLibraries\ObjectGrid\ objectgridSamples.jar
ObjectGridSample	*root_install*\installableApps\ObjectGridSample.ear

Tabella 3. Percorsi degli esempi (Continua)

Esempio	Ubicazione
ObjectGridPartitionCluster	*root_install*\installableApps\ D_ObjectGridPartitionClusterSample.ear

Le versioni aggiornate degli esempi forniti ed esempi aggiuntivi, come ObjectGridJMSSamples, possono essere trovati al seguente indirizzo Web: http://www-1.ibm.com/support/docview.wss?uid=swg27006432. È inoltre possibile visualizzare articoli di IBM DeveloperWorks che descrivono le sezioni di interesse al seguente indirizzo Web: http://www.ibm.com/developerworks. Ricercare **ObjectGrid**.

Ambienti degli esempi

Alcuni esempi possono essere eseguiti in un ambiente J2SE, mentre altri hanno bisogno di un ambiente J2EE. Alcuni possono essere eseguiti in un'unica istanza del server, mentre altri vengono eseguiti in un cluster. Nella seguente tabella sono riportati gli ambienti di esecuzione degli esempi.

Limitazione: Se si utilizza ObjectGrid in un ambiente WebSphere Extended Deployment Versione 6.0, è possibile utilizzarlo anche in un ambiente Java 2 Platform, Standard Edition (J2SE) Versione 1.4.2 o superiore o WebSphere Application Server Versione 6.02 o superiore con delle modifiche alla licenza. Contattare il rappresentante commerciale per maggiori dettagli.

Tabella 4. Ambienti di esecuzione degli esempi

		ObjectGrid SamplesSA	ObjectGrid Sample	ObjectGrid Partition Cluster	ObjectGrid JMSSamples
J2SE	Eclipse	x			
	riga comandi	x			
WebSphere Application Server Versione 6.0.*x*	singolo server		x		x
	Cluster		x		x
	Rational Application Developer unit test environment (UTE)		x		
WebSphere Application Server Versione 5.0.2.*x* e Versione 5.1.*x*	singolo server		x		
	Cluster		x		
WebSphere Extended Deployment Versione 6.0.*x*	singolo server		x		x
	Cluster		x	x	x

Capitolo 6. Assemblaggio di ObjectGrid

È possibile accedere ai pacchetti di ObjectGrid in due modi: installando WebSphere Extended Deployment o installando un ambiente server misto.

Pacchetto ObjectGrid di WebSphere Extended Deployment Versione 6.0.1

Quando si installa WebSphere Extended Deployment Versione 6.0.1 o successiva, sono installati i seguenti file di runtime:

Tabella 5. File di runtime ObjectGrid di WebSphere Extended Deployment

Nome file	Ambiente runtime	Descrizione
/lib/asm.jar /lib/cglib.jar	Locale, client e server	Questi file jar sono per il programma di utilità cglib quando si utilizza la modalità di copia o copia scrittura.
/lib/wsobjectgrid.jar	Locale, client e server	Questo file Java archive (JAR) contiene il runtime ObjectGrid locale, client e server da utilizzare nell'ambiente WebSphere Extended Deployment Versione 6.0.1 e successive.

Pacchetto ObjectGrid di WebSphere Extended Deployment for Mixed Server Environment Versione 6.0.1

Quando si installa WebSphere Extended Deployment for Mixed Server Environment, vengono installati i seguenti file di runtime:

Tabella 6. File di runtime ObjectGrid di WebSphere Extended Deployment for Mixed Server Environment

Nome file	Ambiente runtime	Descrizione
/ObjectGrid/lib/asm.jar /ObjectGrid/lib/cglib.jar	Locale, client e server	Questi file JAR sono per il programma di utilità cglib quando si utilizza la modalità di copia o copia scrittura. Includere questi file JAR nella variabile CLASSPATH se si utilizza la modalità di copia o di copia scrittura e si desidera utilizzare la funzione proxy cglib. Questi file JAR sono inclusi automaticamente nel runtime del server. Aggiungere questi file al runtime ObjectGrid client o locale.

Tabella 6. File di runtime ObjectGrid di WebSphere Extended Deployment for Mixed Server Environment (Continua)

Nome file	Ambiente runtime	Descrizione
/ObjectGrid/lib/mx4j.jar /ObjectGrid/lib/mx4j-remote.jar /ObjectGrid/lib/mx4j-tools.jar	Client e server del gateway di gestione	Questi file JAR sono per il programma di utilità mx4j utilizzato dai programmi server e client del gateway di gestione di ObjectGrid. Aggiungere questi file JAR alla variabile CLASSPATH del client del gateway di gestione quando ci si collega al server del gateway di gestione.
/ObjectGrid/lib/objectgrid.jar	Locale, client e server	Questo file JAR è utilizzato dal runtime del server autonomo per Java 2 Platform, Standard Edition (J2SE) Versione 1.4.2 e successive. È possibile utilizzare questo file JAR anche per il runtime client e locale per J2SE versione 1.3 e successive.
/ObjectGrid/lib/ogclient.jar	Locale e client	Questo file JAR contiene solo i runtime ObjectGrid locali e client durante l'esecuzione all'esterno di un processo WebSphere. È preferibile utilizzare questo file JAR rispetto a objectgrid.jar in quanto occupa meno spazio inmemoria. Utilizzare questo file JAR con J2SE versione 1.3 e successive.
/ObjectGrid/lib/wsobjectgrid.jar	Locale, client e server	Utilizzare questo file JAR suWebSphere Application Server Versione 6.0.2 o successive. Questo file JAR è lo stesso file JAR installato con WebSphere Extended Deployment.
/ObjectGrid/lib/wsogclient.jar	Locale e client	Utilizzare questo file JAR perWebSphere Application Server Versione 5.0.2 e successive. Questo file JAR contiene solo i runtime ObjectGrid locali e client.

Considerazioni per l'utilizzo di ObjectGrid con J2SE Versione 1.3

Quando si utilizza il file ogclient.jar o objectgrid.jar in un ambiente J2SE Versione 1.3.x, è necessario aggiungere i seguenti requisiti all'ambiente J2SE 1.3.x in modo che questo possa funzionare con ObjectGrid:

- **Implementazione di Java Authentication and Authorization Service (JAAS)** . J2SE Versione 1.3 non includeva l'oggetto javax.security.Subject come parte della specifica JAAS. Le interfacce ObjectGrid e Session richiedono questo oggetto. Inserire l'implementazione JAAS nella directory delle estensioni Java jre/lib/ext.
- **Implementazione di Java API for XML Processing (JAXP)**. Se si inviano file XML al runtime ObjectGrid, è necessaria un'implementazione JAXP perché il runtime ObjectGrid possa analizzare il file XML. ObjectGrid utilizza la convalida della sintassi di definizione dello schema XML pertanto è richiesta un'implementazione che supporti la convalida dello schema. Il prodotto Apache Xerces è un esempio di implementazione che supporta la convalida dello schema.
- **Implementazione di Java Secure Socket Extension (JSSE)**. Quando si utilizza il runtime client, è richeista un'implementazione JSSE. Verificare che l'implementazione JSSE utilizzata sia compatibile con l'implementazione Java Development Kit (JDK) del server ObjectGrid, se in esecuzione con la sicurezza abilitata.

Se il runtime ObjectGrid locale o client è incluso in un ambiente compatibile con J2EE Versione 1.3 che utilizza J2SE Versione 1.3, tutti questi requisiti sono soddisfatti in quanto le implementazioni delle specifiche richieste erano richieste come parte di J2EE Versione 1.3.

Capitolo 7. Panoramica sulla gestione del sistema

Con WebSphere Extended Deployment Versione 6.0.1, ObjectGrid fornisce una infrastruttura di gestione del sistema che consente agli utenti di monitorare e gestire gli ambienti ObjectGrid. L'architettura di gestione del sistema è un approccio a tre livelli: un client utente si collega al server del gateway di gestione, che stabilisce una connessione client ObjectGrid a un cluster ObjectGrid.

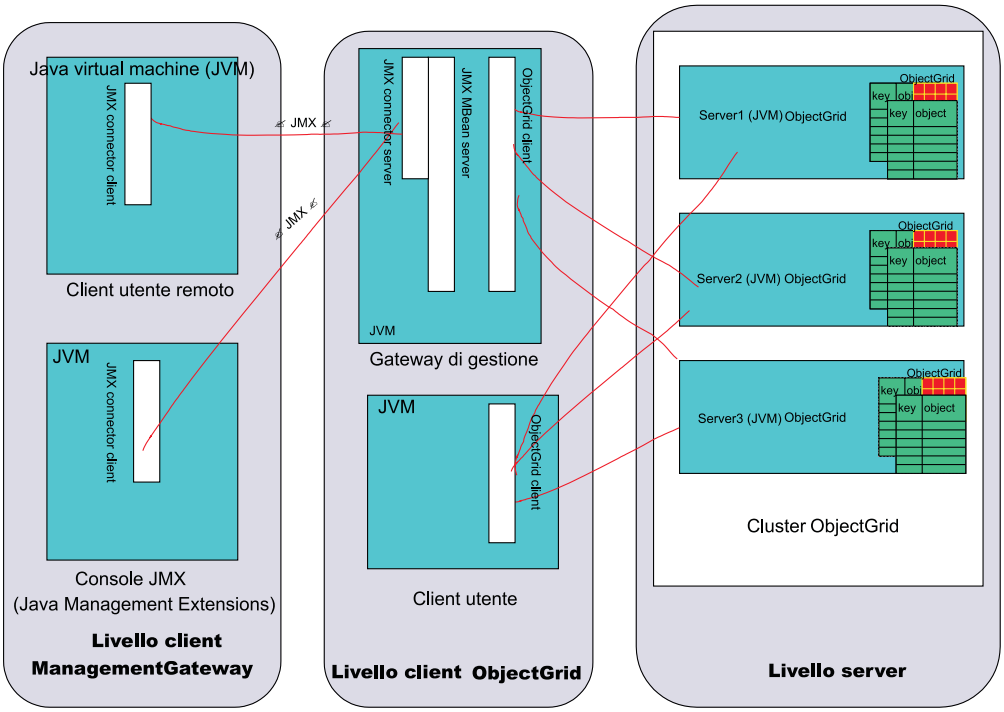


Figura 14. Diagramma di gestione del sistema

Il *livello client del gateway di gestione* contiene i programmi che utilizzano Java Management Extensions (JMX) per collegarsi al server del gateway di gestione. Qualsiasi console JMX di terzi e un programma client che utilizza le API MX4J è incluso. Il *livello client ObjectGrid* è costituito dal server del gateway di gestione. Il gateway di gestione funziona da server per il clientdel gateway di gestione e come client per un cluster ObjectGrid nel *livello server*. Inoltre, un programma client può richiamare le stesse API richiamate dal server del gateway di gestione se l'utente non desidera utilizzare JMX. Infine, il livello server è costituito da un cluster ObjectGrid.

Il gateway di gestione contiene una serie di managed beans (MBeans) e utilizza JMX per gestire e monitorare l'ambiente ObjectGrid implementata dal progetto open source MX4J. MX4J è rilasciato con ObjectGrid.

Il modello di gestione JMX e MBean di ObjectGrid è stato creato per sfruttare le varie console JMX disponibili per la gestione degli ambienti JMX. È possibile utilizzare i pannelli di controllo mediante la console JMX desiderata. Le console possono essere collegate agli MBeans in esecuzione sulla JVM

ManagementGateway e i pannelli di controllo possono essere assemblati mediante tali MBeans. Le console offrono cronologie grafiche o grafici di valori stringa e numerici.

Esistono due opzioni per l'esecuzione dei comandi di gestione del sistema.

- Richiamare i comandi mediante l'infrastruttura client-server mediante l'interfaccia ObjectGridAdministrator.
- Utilizzare JMX per richiamare questi stessi comandi con gli MBeans ObjectGrid funzionanti da wrapper per ObjectGridAdministrator.

Avvio del processo ManagementGateway

una volta avviato un cluster (o un singolo server), è possibile avviare il processo ManagementGateway. ManagementGateway funziona come server per le richieste client e da client ObjectGrid per il cluster a cui è connesso.

Opzioni

Di seguito è riportato un elenco di opzioni che possono essere utilizzate con il processo ManagementGateway:

- **connectorPort** (obbligatoria) - Specifica il numero di porta per il connettore JMX.
- **clusterHost** (obbligatoria) - Specifica il nome host di uno dei server nel cluster ObjectGrid.
- **clusterPort** (obbligatoria) - Specifica la porta di accesso client dei server nel cluster ObjectGrid.
- **clusterName** (obbligatoria) - Specifica il nome del cluster ObjectGrid.
- **traceEnabled** - Specifica se la traccia per il processo ManagementGateway è abilitata.
- **traceSpec** - Indica la specifica di traccia di ManagementGateway.
- **traceFile** - Specifica il file di cui viene stampato l'output di traccia.
- **sslEnabled** - Specifica se SSL è abilitato su ManagementGateway.
- **csConfig** - Specifica l'oggettoClientSecurityConfiguration per proteggere ManagementGateway.
- **refreshInterval** - Specifica l'intervallo di tempo in cui il gateway di gestione aggiorna gli attributi MBean.

Interfaccia di ManagementGateway

Perché gli MBean siano disponibili, è necessario che il processo ManagementGateway sia avviato. L'interfaccia di ManagementGateway mostra le opzioni che possono essere inviate all'avvio diManagementGateway.

```
public interface ManagementGateway {
/**
* Avvia il server del connettore MBean JMX
*/
void startConnector();
/**
* Arresta il server del connettore MBean JMX
*/
void stopConnector();
/**
* @param porta del connettore JMX
*/
void setConnectorPort(int port);
/**
```

```
* @return porta del connettore 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 del server a cui si collega il client del gateway
*/
void setPort(String port);
/**
* @return porta del server a cui si collega il client del gateway
*/
String getPort();
/**
* @param host del server a cui si collega il client del gateway
*/
void setHost(String host);
/**
* @return host del server a cui si collega il client del gateway
*/
String getHost();
/**
* @param boolean true se SSL è abilitato sul gateway
*/
void setSSLEnabled(boolean sslEnabled);
/**
* @return boolean true se SSL è abilitato sul gateway
*/
boolean getSSLEnabled();
/**
* @param cluster a cui si collega il client del gateway
*/
void setClusterName(String clusterName);
/**
* @return cluster a cui si collega il client del gateway
*/
String getClusterName();
/**
* @param true se la traccia è abilitata sul gateway
*/
void setTraceEnabled(boolean traceEnabled);
/**
* @return true se la traccia è abilitata sul gateway
*/
boolean getTraceEnabled();
/**
* @param specifica di traccia sul gateway
*/
void setTraceSpec(String traceSpec);
/**
* @return specifica di traccia sul gateway
*/
String getTraceSpec();
/**
* @param file di output per la traccia del gateway
*/
void setTraceFile(String traceFile);
/**
* @return file di output per la traccia del gateway
```

```
*/
String getTraceFile();
/**
* @param intervallo (in secondi) di aggiornamento degli attributi MBean del cluster
*/
void setRefreshInterval(int refreshInterval);
/**
* @return intervallo (in secondi) di aggiornamento degli attributi MBean del cluster
*/
int getRefreshInterval();
}
```

Opzioni per l'avvio del processo ManagementGateway

In maniera programmatica mediante ManagementGatewayFactory

Di seguito è riportato il codice di esempio per l'utilizzo di questa opzione:

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

Questo codice deve essere presente in un programma utente che viene eseguito in seguito all'avvio del cluster ObjectGrid a cui ci si collega.

Sulla riga comandi con il file batch startManagementGateway

Di seguito viene riportato un esempio:

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

Per ulteriori informazioni sugli script startManagementGateway, fare riferimento a "Avvio del server gateway di gestione" a pagina 86.

ManagementGateway funziona come server per un processo client che desidera effettuare chiamate JMX, ma anche come clientObjectGrid per il cluster a cui l'utente si collega. Una volta avviato ManagementGateway, viene stabilita una connessione al cluster e il servizio del connettore JMX diventa disponibile. È quindi possibile accedere al servizio del connettore JMX mediante le API MX4J o Java 2 Platform, Standard Edition (J2SE) Versione 5.

Esempio

Di seguito è riportato un codice di esempio di come richiamare MapStatsModule da un server denominato Server1 mediante un ManagementGateway con porta di connettore 1.

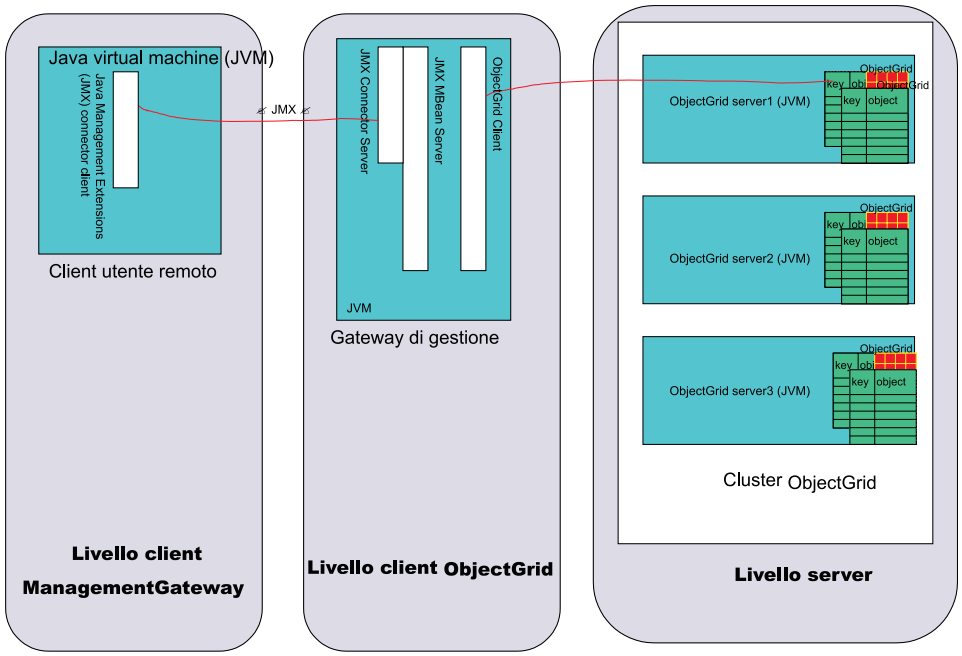


Figura 15. Ottenimento delle statistiche di mappa dal server server1

Eseguire il codice riportato in un programma utente che viene eseguito nella sezione client dell'utente remoto del diagramma precedente:

```
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[] { });
```

Per arrestare il server server1 mediante ManagementGateway:

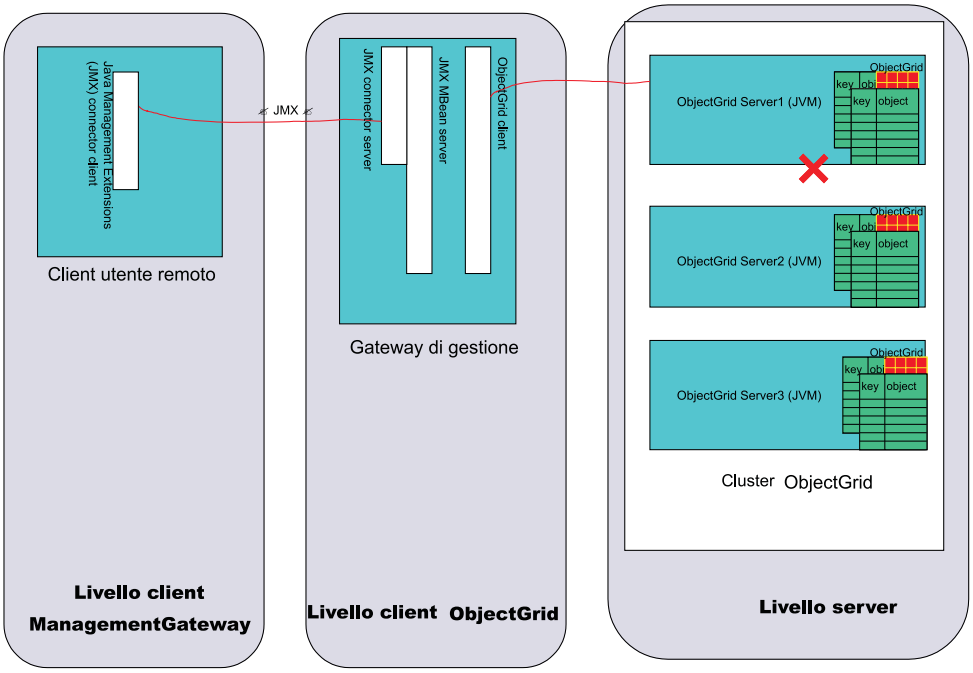


Figura 16. Arrestare il server server1

Eseguire il codice riportato in un programma utente che viene eseguito sul client dell'utente remoto del diagramma precedente:

```
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();
```

Dopo aver eseguito il codice precedente, il server server1 viene arrestato. Una volta arrestato il server server1, questo non potrà essere riavviato con ManagementGateway. Esso potrà essere riavviato mediante la riga comandi. Per ulteriori informazioni, consultare "Arresto di server ObjectGrid" a pagina 85.

MBeans (managed beans) ObjectGrid

Esistono cinque diversi tipi di MBeans nell'ambiente ObjectGrid. Ogni MBean fa riferimento a una determinata entità, come una mappa, un object grid, un server, un gruppo di replica o un membro del gruppo di replica e ha attributi e operazioni associate.

Ogni MBean in ObjectGrid utilizza i metodi getxxx che rappresentano i valori degli attributi. Tali metodi getxxx non possono essere richiamati direttamente da un programma utente. Ciò perché la specifica JMX (Java management extensions) tratta gli attributi in maniera differente dalle operazioni. Gli attributi possono essere

visualizzati mediante una console JMX di terzi e le operazioni possono essere eseguite sia mediante un programma utente che mediante una console JMX.

Mbean MapMbean

Il bean MapMBean consente all'utente di monitorare le statistiche di ogni mappa definita per il cluster. Ogni mappa ha le seguenti statistiche associate.

- Ora di aggiornamento batch (min/max/mean/total)
- Numero
- Frequenza di successi

Inoltre, poiché le mappe possono essere partizionate sui server, è possibile estendere le statistiche di una mappa a un determinato server o membro del gruppo di replica. È inoltre possibile associare le statistiche per l'intero cluster. L'ObjectName per un MapMBean può essere specificato in diversi modi:

- `"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"`

Consideriamo una configurazione di esempio con un OG1 ObjectGrid, una mappa Map1 e due server nel gruppo di replica RG1, server1 eserver2. Inoltre, si assuma che il server server1 sia il server primario e il server server2 sia un server di replica. Per ottenere le statistiche per la mappa Map1 sul server primario, utilizzare uno dei seguenti ObjectNames:

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

In qualsiasi ObjectName per gli MBeans ObjectGrid, quando IDX=0, allora si fa riferimento al server primario del gruppo di replica. IDX=1-10 fa riferimento ai server di replica per il gruppo di replica.

Di seguito è riportato un elenco di interfacce MapMBean:

```
public interface MapMBean {
/**
* Operazione per ottenere l'MapStatsModule associato all'MBean.
*
* @return MapStatsModule
*/
MapStatsModule retrieveStatsModule();
/**
* Le operazioni passeranno solo al server per ottenere StatsModule se
* StatsModule non è memorizzato nella cache di ObjectGridAdministrator.
*
*/
void refreshStatsModule();
/**
* Map.
*
* @return name of map
*/
String getMapName();
/**
* L'ObjectGrid contenente la mappa.
*
* @return name of the object grid
*/
String getObjectGridName();
/**
```

```
* Il nome server del membro del gruppo di replica per la mappa.
*
* @return name of server of the replication group member
*/
String getServerName();
/**
* Il nome del gruppo di replica per la mappa.
*
* @return name of replication group
*/
String getReplicationGroup();
/**
* L'indice del membro del gruppo di replica per la mappa.
*
* @return index of replication group member
*/
int getIndex();
/**
* L'attributo MapStatsModule caricato dalla
* chiamata a retrieveStatsModule.
*
* @return String form of MapStatsModule
*/
String getMapStatsModule();
/**
* L'attributo di numerazione mappe caricato dalla
* chiamata a retrieveStatsModule.
*
* @return number of entries in map
*/
long getMapCountStatistic();
/**
* L'attributo di frequenza successi caricato dalla
* chiamata a retrieveStatsModule.
*
* @return hit rate for map
*/
double getMapHitRateStatistic();
/**
* L'attributo del tempo di aggiornamento batch medio caricato dalla
* chiamata a retrieveStatsModule.
*
* @return mean batch update time for map
*/
double getMapBatchUpdateMeanTime();
/**
* L'attributo del tempo di aggiornamento batch massimo caricato dalla
* chiamata a retrieveStatsModule.
*
* @return maximum batch update time for map
*/
double getMapBatchUpdateMaxTime();
/**
* L'attributo del tempo di aggiornamento batch minimo caricato dalla
* chiamata a retrieveStatsModule.
*
* @return minimum batch update time for map
*/
double getMapBatchUpdateMinTime();
/**
* L'attributo del tempo di aggiornamento batch totale caricato dalla
* retrieveStatsModule call.
*
* @return total batch update time for map
*/
double getMapBatchUpdateTotalTime();
}
```

MBean ObjectGridMBean

L'MBean ObjectGridMBean consente all'utente di monitorare le statistiche per tutte le mappe in ogni ObjectGrid definito per il cluster. Ogni ObjectGrid ha le seguenti statistiche associate:

- Ora transazione (min/max/mean/total)
- Numero

Inoltre, poiché gli ObjectGrids possono essere partizionati sui server, è possibile estendere le statistiche di ObjectGrid a un determinato server o membro del gruppo di replica. È inoltre possibile ottenere le statistiche di ObjectGrid per l'intero cluster. L'ObjectName per un ObjectGridMBean può essere specificato in diversi modi:

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

Di seguito è riportato un elenco dell'interfaccia ObjectGridMbean:

```
public interface ObjectGridMBean {
/**
* Operazione per ottenere l'OGStatsModule associato all'MBean.
*
* @return OGStatsModule
*/
OGStatsModule retrieveStatsModule();
/**
* Le operazioni passeranno solo al server per ottenere StatsModule se
* StatsModule non è memorizzato nella cache di ObjectGridAdministrator.
*
*/
void refreshStatsModule();
/**
* ObjectGrid.
*
* @return name of the object grid
*/
String getObjectGridName();
/**
* Il nome server del membro del gruppo di replica per l'ObjectGrid.
*
* @return name of server of the replication group member
*/
String getServerName();
/**
* Il nome del gruppo di replica per l'ObjectGrid.
*
* @return name of replication group
*/
String getReplicationGroup();
/**
* L'indice del membro del gruppo di replica per l'ObjectGrid.
*
* @return index of replication group member
*/
int getIndex();
/**
* L'attributo OGStatsModule caricato dalla chiamata retrieveStatsModule.
*
* @return String form of OGStatsModule
*/
String getOGStatsModule();
/**
```

```
* L'attributo di numerazione di ObjectGrid caricato dalla chiamata retrieveStatsModule.
*
* @return number of transactions
*/
long getOGCount();
/**
* L'attributo del tempo massimo di transazione caricato dalla chiamata retrieveStatsModule.
*
* @return maximum transaction time for the ObjectGrid
*/
long getOGMaxTranTime();
/**
* L'attributo del tempo minimo di transazione caricato dalla chiamata retrieveStatsModule.
*
* @return minimum transaction time for the ObjectGrid
*/
long getOGMinTranTime();
/**
* L'attributo del tempo medio di transazione caricato dalla chiamata retrieveStatsModule.
*
* @return mean transaction time for the ObjectGrid
*/
double getOGMeanTranTime();
/**
* L'attributo del tempo totale di transazione caricato dalla chiamata retrieveStatsModule.
*
* @return total transaction time for the ObjectGrid
*/
long getOGTotalTranTime();
}
```

MBean ServerMBean

L'Mbean ServerMBean consente agli utenti di eseguire operazioni sui server nel cluster. L'ObjectName per un ServerMBean può essere specificato nel seguente modo:

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

Di seguito è riportato un elenco di interfacce ServerMBean:

```
public interface ServerMBean {
/**
* L'operazione per caricare lo stato di replica per il server.
*
*/
void retrieveReplicationStatus();
/**
* Restituisce il nome del server.
*
* @return il nome del server
*/
String getServerName();
/**
* Operazione per ottenere lo stato del server.
*
* @return lo stato server (true se in esecuzione, altrimenti false)
*/
boolean retrieveServerStatus();
/**
* Operazione per arrestare il server.
*
* @return true se il server era arrestato, altrimenti false
*/
boolean stopServer();
/**
* Operazione per imporre l'arresto del server.
```

```
*
* @return true se il server era arrestato, altrimenti false
*/
boolean forceStopServer();
/**
* Operazione per arrestare il cluster di cui fa parte il server.
*
* @param determina se i server sono arrestati
* @return true se il cluster è stato arrestato, altrimenti false
*/
boolean stopCluster(Boolean force);
/**
* Operazione per modificare la specifica di traccia per tutti i server
* nel cluster di cui fa parte il server.
*
* @param specifica di traccia
*/
void modifyClusterTraceSpec(String spec);
/**
* Operazione per modificare la specifica di traccia per il server
*
* @param specifica di traccia
*/
void modifyServerTraceSpec(String spec);
}
```

Mbean ReplicationGroupMBean

ReplicationGroupMBean consente di monitorare lo stato per tutti i membri del gruppo di replica associati a un determinato gruppo di replica comprendente il server primario e un massimo di dieci server di replica. L'ObjectName per ReplicationGroupMBean può così essere specificato:

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

Di seguito è riportato un elenco di interfacce ReplicationGroupMBean:

```
public interface ReplicationGroupMBean {
/**
* Operazione per caricare lo stato degli attributi del gruppo di replica.
*
*/
String[] retrieveReplicationGroupStatus();
/**
* Attributo ReplicationGroupName.
*
* @return nome di ReplicationGroup
*/
String getReplicationGroupName();
/**
* Attributo primario.
*
* @return il nome del server primario
*/
String getPrimary();
/**
* Attributo di Replica1.
*
* @return nome server di Replica1
*/
String getReplica1();
/**
* Attributo Replica2.
*
* @return nome server di Replica2
*/
```

```
String getReplica2();
/**
* Attributo di Replica3.
*
* @return nome server di Replica3
*/
String getReplica3();
/**
* Attributo Replica4.
*
* @return nome server di Replica4
*/
String getReplica4();
/**
* Attributo di Replica5.
*
* @return nome server di Replica5
*/
String getReplica5();
/**
* Attributo Replica6.
*
* @return nome server di Replica6
*/
String getReplica6();
/**
* Attributo di Replica7.
*
* @return nome server di Replica7
*/
String getReplica7();
/**
* Attributo Replica8.
*
* @return nome server di Replica8
*/
String getReplica8();
/**
* Attributo di Replica9.
*
* @return nome server di Replica9
*/
String getReplica9();
/**
* Attributo Replica10.
*
* @return nome server di Replica10
*/
String getReplica10();
/**
* Tutte le repliche per questo gruppo di replica delimitate da virgola
*
* @return nomi server di tutte le repliche
*/
String getReplicas();
}
```

Mbean ReplicationGroupMemberMBean

ReplicationGroupMemberMBean consente di monitorare le seguenti statistiche per un membro del gruppo di replica:

- Stato di un membro del gruppo di replica. È possibile monitorare i membri del gruppo primario o di replica.
- Proporzione peso replica. Questa statistica è valida solo per i membri del gruppo di replica che sono repliche. Questa proporzione è rappresenta quanto vicino

sono le mappe di un server di replica alla sincronizzazione con le mappe di un server primario. Maggiore è il rapporto, più la replica ha le informazioni aggiornate del server primario.

L'ObjectName per un ReplicationGroupMemberMBean può essere specificato nei seguenti modi:

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

Specificando IDX=0 viene restituito il server primario del gruppo di replica e IDX=1 viene restituito un massimo di 10 di server direplica. Di seguito è riportato un elenco di interfacce ReplicationGroupMBean:

```
public interface ReplicationGroupMemberMBean {
/**
* Operazione per caricare lo stato degli attributi del membro del gruppo di replica.
*
*/
void retrieveReplicationGroupMemberStatus();
/**
* Operazione per caricare lo stato degli attributi del membro del gruppo
* di replica.
* Verrà utilizzata la cache in opposizione al metodo retrieveReplicationGroupMemberStatus
* che passa sul server per ottenere lo stato.
*
*/
void refreshReplicationGroupMemberStatus();
/**
* ReplicationGroupName attribute.
*
* @return nome del ReplicationGroup a cui appartiene questo membro
*/
String getReplicationGroupName();
/**
* Stato di ReplicationGroupMember: primario/replica/standby.
*
* @return stato di ReplicationGroupMember
*/
String getStatus();
/**
* Le statistiche che rappresentano quanto vicino è un server di replica
* a ottenere le informazioni aggiornate con le mappe del server primario.
*
* @return statistiche replica di ReplicationGroupMember
*/
double getReplicaWeightRatio();
/**
* Nome del server su cui si trova ReplicationGroupMember.
*
* @return il nome del server
*/
String getServerName();
/**
* Indice di ReplicationGroupMember.
*
* @return indice della replica
*/
int getIndex();
}
```

Capitolo 8. Supporto per la riga comandi

Utilizzare gli script della riga comandi per gestire i server ObjectGrid.

Una serie di file script è fornita nella directory /ObjectGrid/bin di installazione dell'ambiente di server misto. Tali script possono essere utilizzati per avviare o arrestare un server ObjectGrid, avviare un server del gateway di gestione e codificare le password in un file delle proprietà. Prima di provare a utilizzare tali script, verificare che la variabile d'ambiente JAVA_HOME sia impostata e che il valore sia una versione di Java supportata da ObjectGrid. È possibile aggiornare JAVA_HOME nel filesetupCmdLine.bat|sh in modo che faccia riferimento a una versione appropriata di Java se non si desidera modificare la variabile d'ambiente in generale.

Fare riferimento alle seguenti sezioni per ulteriori informazioni sugli script della riga comandi:

- "Avvio di server ObjectGrid"
- "Arresto di server ObjectGrid" a pagina 85
- "Avvio del server gateway di gestione" a pagina 86
- "Codifica delle password" a pagina 88

Avvio di server ObjectGrid

Lo script startOgServer è fornito per avviare un server ObjectGrid.

Utilizzo

Utilizzare il file startOgServer.bat per avviare un server su una macchina Windows. Utilizzare il file startOgServer.sh per avviare un server ObjectGrid su piattaforme Linux e Unix.

Utilizzo di file XML

Per poter avviare un server ObjectGrid, un file XML di ObjectGrid valido deve sempre essere accoppiato a un file XML di cluster. I file XML possono essere inviati allo script startOgServer mediante un nome file regolare o un URL (Uniform Resource Locator). L'opzione URL fornisce protocolli differenti oltre al protocollo file, come ad esempio http, ftp o jarfile.

Gli argomenti dello script startOgServer per l'avvio di un server mediante i file XML sono:

```
startOgServer.bat <server> -objectgridFile <file XML> | -objectgridUrl
<URL file XML> -clusterFile <file XML> | -clusterUrl <URL file XML> [options]
```

Esempio

Di seguito sono riportati degli esempi di avvio del server ObjectGrid server1. Questi esempi utilizzano il file 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
```

Gli esempi utilizzano il file `universityCluster3Servers.xml`. Poiché il server server1 viene specificato come il server da avviare, il file `universityCluster3Servers.xml` deve avere un valore serverDefinition con il nome `server1`.

Di seguito è riportato il file `universityCluster3Servers.xml`. Il server1 è serverDefinition e l'host è lion.ibm.com. Il server server1 deve essere avviato su lion.ibm.com host. Questo file definisce inoltre i server server2 e server3. Tali server devono essere avviati rispettivamente su tiger.ibm.com e bear.ibm.com hosts.

File *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>
```

Bootstrap

Quando un server ObjectGrid nel cluster è disponibile, gli altri server nel cluster possono essere avviati sul server disponibile. Lo script startOgServer va fornito con l'host e la porta di accesso client di un server già disponibile per l'avvio.

Poiché il primo server in un cluster viene avviato con XML, questo ha già tutte le informazioni di configurazione per tutti i server del cluster. Il bootstrap consente al server inavvio di collegarsi al server disponibile e scaricare la configurazione.

Di seguito sono riportati gli argomenti dello script startOgServer per l'avvio di un server mediante un server disponibile.

```
startOgServer.bat <server> -bootstrap <host:port,host:port> [options]
```

Di seguito sono riportati degli esempi di avvio di un server avviando un altro server. Come primo esempio, si assuma che il server server1 dal file `universityCluster3Servers.xml` sia stato avviato mediante i file XML e che sia disponibile. Questo esempio mostra come avviare il server server1 per avviare il server server2.

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

Per l'esempio successivo, si assuma che il server server2 venga avviato correttamente ma che il server server1 diventi non disponibile. Un elenco separato da virgole di combinazioni host:porta può essere utilizzato quando ci si collega a un altro server. Viene effettuato un tentativo di connessione a ogni host e porta nell'elenco fino a che viene trovato un server disponibile. Nel seguente esempio, server3 prova a contattare l'host e la porta del server server1. Tuttavia, poiché il server server1 non è disponibile in questo scenario, la connessione non riesce. Passando alla voce successiva nell'elenco, il server server3 prova a collegarsi all'host e alla porta per il server server2. Questo collegamento dovrebbe riuscire in quanto il server server2 è disponibile.

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

Argomenti facoltativi

Esistono diversi argomenti facoltativi che possono essere inviati allo script startOgServer. di seguito sono riportati gli argomenti validi per startOgServer.

Opzioni:

- -traceSpec <specifica traccia>
- -traceFile <file traccia>
- -serverSecurityFile <file proprietà sicurezza server>
- -timeout <secondi>
- -script <nome file script>
- -jvmArgs <argomenti JVM>

-traceSpec

L'argomento -traceSpec può essere utilizzato per impostare una specifica di traccia che ha effetto quasi immediatamente durante l'avvio del server. Durante un normale avvio del server, la specifica di traccia non viene impostata fino a che non viene letta dal file XML del cluster o dalla configurazione bootstrap. Se durante l'avvio del server si verificano dei problemi, potrebbe risultare utile l'aver impostato la specifica di traccia.

Di seguito è riportato un esempio di come impostare l'opzione -traceSpec:

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

-traceFile

L'argomento -traceFile può essere utilizzato per specificare un percorso per la traccia che viene restituita durante l'avvio del server. Una volta letta la configurazione per questo server, le relative impostazioni di traccia così come specificate dal file XML del cluster avranno effetto.

Di seguito è riportato un esempio di come impostare l'opzione -traceFile:

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

-serverSecurityFile

L'argomento -serverSecurityFile può essere utilizzato per inviare a un server il file delle proprietà di sicurezza. Questa opzione è richiesta quando sul server è abilitata la sicurezza. Di seguito è riportato un esempio di come impostare l'opzione -serverSecurityFile:

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

-timeout

L'argomento -timeout può essere utilizzato per specificare la quantità di tempo, espressa in secondi, che deve trascorrere prima che l'avvio del server venga interrotto. Per impostazione predefinita, il server può attendere 90 secondi per diventare disponibile dal suo avvio. Se questo tempo è troppo breve per un determinato scenario, utilizzare l'argomento -timeout per impostarlo su un valore più appropriato. Di seguito è riportato un esempio di come utilizzare l'argomento timeout:

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

-script

L'argomento -script può essere utilizzato per creare uno script che avvia un processo server di ObjectGrid e ne restituisce l'output dal prompt dei comandi. In casi normali, quando viene avviato un server ObjectGrid, lo script startOgServer visualizza l'output dal processo del server sul prompt dei comandi fino a che il server diventa disponibile. Quando il server è disponibile, startOgServer interrompe la visualizzazione dell'output dal processo server ed esce. In alcuni casi, è possibile avviare un processo del server che restituisce l'output sul prompt dei comandi corrente.

Quando si specifica un nome file per lo script, non aggiungervi un percorso. Il file viene inserito automaticamente nella directory `bin` del percorso OBJECTGRID_HOME. Fornire il nome del file. Il file script che viene creato include gli argomenti che sono inoltrati allo script startOgServer in modo che non sia necessario fornire gli stessi argomenti quando lo script viene eseguito.

di seguito è riportato un esempio di come utilizzare l'opzione -script:

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

Questo esempio crea uno script `universityClusterServer1.bat` nella directory `OBJECTGRID_HOME/bin`. per eseguire lo script appena creato, passare alla directory appropriata dal prompt dei comandi, immettere il nome dello script e premere **Invio**.

-jvmArgs

L'argomento -jvmArgs può essere utilizzato per inviare argomenti alla JVM del server ObjectGrid che viene avviato. Qualsiasi argomento che può essere inviato alla JVM di solito può essere inviato al server mediante l'argomento -jvmArgs.

L'argomento -jvmArgs deve trovarsi nell'ultimo argomento facoltativo di ObjectGrid e deve essere specificato come argomento per lo script startOgServer. Ogni voce presente dopo l'argomento -jvmArgs viene inviato

alla JVM del server come argomento della JVM. Di seguito è riportato un esempio di come impostare l'argomento -jvmArgs:

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

Se l'argomento -jvmArgs include un argomento -classpath o -cp, la variabile classpath specificata verrà aggiunta alla variabile classpath di ObjectGrid. Di seguito è riportato un esempio di utilizzo dell'argomento -jvmArgs per includere i file Java archive (JAR) Xerces nella variabile classpath utilizzati per avviare un server ObjectGrid.

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

Arresto di server ObjectGrid

Utilizzare lo script stopOgServer per arrestare i server ObjectGrid.

Utilizzo

Utilizzare il file `stopOgServer.bat` per arrestare un server su una macchina Windows. Utilizzare il file `stopOgServer.sh` per arrestare un server ObjectGrid su piattaforme Linux e Unix. Lo script stopOgServer crea un client che può arrestare un server connettendosi a qualsiasi server disponibile nel cluster. Il funzionamento di questo script è simile al collegamento a un server disponibile per avviare un altro server. Di seguito sono riportati gli argomenti dello script stopOgServer per l'arresto di un server.

```
stopOgServer.bat <server> -bootstrap <host:port,host:port> [opzioni]
```

Esempi

Di seguito sono riportati degli esempi di arresto di diversi server ObjectGrid. Tali esempi utilizzano il file `stopOgServer.bat`. Per questi esempi, si assume che i tre server siano attivi e in esecuzione: i serverserver1, server2 e server3 sono definiti dal file`universityCluster3Servers.xml` in "Avvio di server ObjectGrid" a pagina 81.

Questo primo esempio arresta il serverserver1 collegandosi all'host e alla porta di accesso client.

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

Si assuma che il server server1 sia stato arrestato correttamente. Nel successivo esempio, il server server2 viene arrestato provando prima a collegarsi al server server1. Poiché il server server1 è già stato arrestato, il collegamento non riesce. L'host e la porta successivi nell'elenco appartengono al server server3. Poiché il server server3 è disponibile, il collegamento al server server3 riesce correttamente e pertanto il server server2 viene arrestato.

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

Argomenti facoltativi

Esistono diversi argomenti facoltativi che possono essere inviati allo script stopOgServer. In questa sezione viene mostrato come utilizzare questi argomenti. Di seguito sono riportati gli argomenti validi per stopOgServer seguiti dagli argomenti facoltativi.

```
stopOgServer.bat <server> -bootstrap <host:port,host:port> [opzioni]
```

Opzioni:

- -traceSpec <specifica traccia>
- -traceFile <file traccia>
- -clientSecurityFile <file delle proprietà di sicurezza client>

-traceSpec

L'argomento -traceSpec può essere utilizzato per impostare una specifica di traccia sul client che prova ad arrestare un server ObjectGrid. Di seguito è riportato un esempio di come impostare l'argomento -traceSpec:

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

-traceFile

L'argomento -traceFile può essere utilizzato per specificare un percorso per la traccia del client che viene restituita durante l'arresto del server. Di seguito è riportato un esempio di come impostare l'argomento -traceFile:

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

-clientSecurityFile

L'argomento -clientSecurityFile può essere utilizzato per inviare the un client il file delle proprietà di sicurezza. Questo argomento è richiesto quando si prova a collegarsi a un server con la sicurezza abilitata.

Di seguito è riportato un esempio di come impostare l'argomento -clientSecurityFile:

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

Avvio del server gateway di gestione

Per monitorare e gestire un clusterObjectGrid mediante Java management extensions (JMX), il gateway di gestione deve essere avviato mediante uno script della riga comandi oppure in maniera programmatica.

Funzione

Per avviare il gateway di gestione mediante la riga comandi, utilizzare lo script startManagementGateway. Utilizzare il file `startManagementGateway.bat` per avviare un server ManagementGateway su una macchina Windows. Utilizzare il file `startManagementGateway.sh` per avviare un server ManagementGateway su piattaforme Linux e Unix. Per ulteriori informazioni sulla funzione del gateway di gestione, sui MBeans ObjectGrid e su JMX, fare riferimento aCapitolo 7, "Panoramica sulla gestione del sistema", a pagina 67.

Lo script startManagementGateway crea un serve del connettoreJMX e un client ObjectGrid che si collega a un cluster ObjectGrid per arrestare i server, raccogliere le informazioni sullo stato ed eseguire diverse altre funzioni.

Di seguito sono riportati gli argomenti dello script startManagementGateway per l'avvio di un server del gateway di gestione.

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

Argomenti facoltativi

Esistono diversi argomenti facoltativi che possono essere inviati allo script startManagementGateway. Di seguito sono riportati gli argomenti validi per startManagementGateway seguiti dagli argomenti facoltativi.

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

Opzioni

- -traceEnabled <true/false traccia abilitata>
- -traceSpec <specifica traccia>
- -traceFile <file traccia>
- -refreshInterval <intervallo aggiornamento attributi MBean>
- -sslEnabled <true/false SSL abilitato per il gateway di gestione>
- -clientSecurityFile <percorso file di sicurezza client>

-traceEnabled

L'argomento -traceEnabled può essere utilizzato se la traccia è attivata per il server del gateway di gestione. Il valore predefinito è false, pertanto l'unico modo per visualizzare la traccia di ObjectGrid è abilitarla impostando -traceEnabled su ″true″ e fornendo valori validi per -traceSpec e -traceFile.

-traceSpec

L'argomento -traceSpec può essere utilizzato per impostare una specifica di traccia per il server del gateway di gestione.

-traceFile

L'argomento -traceFile può essere utilizzato per specificare un percorso per l'output di traccia del gateway di gestione. Di seguito è riportato un esempio di come impostare gli argomenti 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

L'argomento -refreshInterval può essere utilizzato per inviare il tempo (in secondi) che il gateway di gestione attende tra gli aggiornamenti dei valori degli attributi dei MBean. Il valore predefinito è 120 secondi. Di seguito è riportato un esempio di come impostare l'argomento -refreshInterval:

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

-sslEnabled

L'argomento -sslEnabled può essere utilizzato per impostare se SSL è abilitato per il gateway di gestione. Se il valore per questo argomento è `true`, qualsiasi client utente che si collega al server del gateway di gestione deve fornire le seguenti proprietà SSL:

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

Il valore predefinito se l'argomento -sslEnabled non è fornito è ″false″.

-clientSecurityFile

L'argomento -clientSecurityFile può essere utilizzato per inviare il nome file che contiene le proprietà di sicurezza client per l'accesso al client sicuro tra il server del gateway di gestione e il cluster ObjectGrid. Questo argomento è richiesto quando si prova a collegarsi a un cluster con la sicurezza

abilitata. ObjectGrid fornisce la seguente maschera del file delle proprietà di sicurezza del client: `security.ogclient.props`.

Di seguito è riportato un esempio di come impostare le proprietà sslEnabled e clientSecurityFile:

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

Codifica delle password

La codifica delle password impedisce la visualizzazione casuale delle password nei file delle proprietà di sicurezza ObjectGrid.

Utilizzo

ObjectGrid contiene diverse password codificate che non sono crittografate. ObjectGrid fornisce il programma di utilità FilePasswordEncoder che può essere utilizzato per codificare tali password. Utilizzare il file `FilePasswordEncoder.bat` per codificare le password su una macchina Windows. Utilizzare il file`FilePasswordEncoder.sh` per codificare le password su piattaforme Linux e Unix.

La sintassi del comando è la seguente:

```
FilePasswordEncoder.bat nome_file elenco_prop_password [tipo_file]
```

Opzioni

Le seguenti opzioni sono disponibili per il comando FilePasswordEncoder:

nome_file
Il nome_file è utilizzato per specificare il nome del file che contiene le password da codificare. Ad esempio, `security.ogserver.props`.

elenco_prop_password
L'elenco_prop_password è un elenco di nomi di proprietà delle password separati da virgole, ad esempio ″trustStorePassword,keyStorePassword″.

tipo_file
Questo argomento è facoltativo. Il tipo_file può essere un valore xml o una proprietà che indica se il file fornito è un file delle proprietà o un file XML. Il valore predefinito è `proprietà`. Al momento, ObjectGrid non memorizza le password in un file XML, pertanto questa opzione non è richiesta. Il seguente esempio riporta la sintassi corretta:

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

Questo programma di utilità FilePasswordEncoder non è rilasciato con WebSphere Extended Deployment. È possibile utilizzare invece il programma di utilità PropFilePasswordEncoder fornito da WebSphere Application Server per codificare le password. Fare riferimento a PropFilePasswordEncoder command reference per maggiori dettagli.

Capitolo 9. Panoramica sull'interfaccia di programmazione dell'applicazione ObjectGrid.

In questa sezione viene descritto come configurare l'ObjectGrid con XML o mediante le interfacce programmatiche. Inoltre, sono riportate le informazioni relative all'implementazione delle interfacce esterne fornite da ObjectGrid. In tutti i casi, sono riportati una panoramica, le interfacce dell'API e gli esempi.

Documentazione API

La JavaDoc per ObjectGrid è la fonte principale delle informazioni relative alle API. La JavaDoc si trova nella seguente directory del percorso di installazione di WebSphere Extended Deployment: *root_install*\web\xd\apidocs

Interfaccia ObjectGridManager

La classe ObjectGridManagerFactory e l'interfaccia ObjectGridManager forniscono un meccanismo per creare, accedere e memorizzare nella cache le istanze di ObjectGrid. La classe ObjectGridManagerFactory è una classe helper statica per accedere all'interfaccia ObjectGridManager, un singleton. L'interfaccia ObjectGridManager include diversi metodi convenzionali per creare le istanze di un oggetto ObjectGrid. L'interfaccia ObjectGridManager facilita inoltre la creazione e la memorizzazione nella cache di istanze ObjectGrid a cui possono accedere diversi utenti.

Metodi createObjectGrid

In questa sezione sono riportati i sette metodi createObjectGrid che sono presenti nell'interfaccia ObjectGridManager.

Metodi createObjectGrid

L'interfaccia ObjectGridManager utilizza sette metodi createObjectGrid. Di seguito è riportato uno scenario di esempio:

Caso semplice con una configurazione predefinita

Di seguito è riportato un caso semplice di creazione di un ObjectGrid da condividere tra vari utenti.

```
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..*/
```

Il frammento di codice Java precedente crea e memorizza nella cache Employees ObjectGrid. Employees ObjectGrid viene inizializzato con la configurazione predefinita ed è quindi pronto per essere utilizzato. Il secondo parametro nel metodo createObjectGrid è impostato su `true`, che indica all'ObjectGridManager di memorizzare nella cache l'istanza di ObjectGrid creata. Se questo parametro è

impostato su `false`, l'istanza non viene memorizzata nella cache. Ogni istanza ObjectGrid ha un `nome` e può essere condivisa tra più client o utenti in base a tale nome.

Se l'istanza objectGrid viene utilizzata in una condivisione peer-to-peer, la memorizzazione nella cache deve essere impostata su `true`. Per ulteriori informazioni sulla condivisione peer-to-peer, fare riferimento a Capitolo 12, "Distribuzione delle modifiche tra JVM del peer", a pagina 337.

Configurazione XML

ObjectGrid può essere configurato. Nell'esempio precedente è riportato come creare un ObjectGrid semplice senza alcuna configurazione. Con questo esempio, viene invece creata un'istanza di ObjectGrid preconfigurata basata su un file di configurazione XML. Esistono due modi per configurare un'istanza di ObjectGrid: in maniera programmatica o utilizzando un file di configurazione basato su XML. È inoltre possibile configurare ObjectGrid utilizzando una combinazione dei due approcci.

L'interfaccia ObjectGridManager consente la creazione di un'istanza ObjectGrid basata sulla configurazione XML. L'interfaccia ObjectGridManager contiene diversi metodi che utilizzano un URL come argomento. Ogni file XML inviato a ObjectGridManager deve essere convalidato rispetto allo schema. La convalida XML può essere disabilitata solo se il file è stato precedentemente convalidato e non è stata apportata alcuna modifica al file dalla convalida in poi. La disabilitazione della convalida consente di ridurre il carico ma introduce la possibilità di utilizzo di un file XML non valido. IBM Java Developer Kit (JDK) 1.4.2 è dotato di un supporto per la convalida XML. Quando si utilizza un JDK che non ha questo supporto, per convalidare l'XML potrebbe essere necessario Apache Xerces.

Il seguente frammento di codice Java dimostra come inoltrare un file di configurazione XML per creare un ObjectGrid.

```
import java.net.MalformedURLException;
import java.net.URL;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
boolean validateXML = true; // attiva la convalida XML
boolean cacheInstance = true; // memorizza l'istanza nella cache
String objectGridName="Employees"; // Name of Object Grid URL
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees = oGridManager.createObjectGrid(objectGridName,
allObjectGrids,
validateXML,
                        cacheInstance);
```

Il file XML può contenere informazioni sulla configurazione per diversi ObjectGrid. Il codice precedente restituisce l'ObjectGrid ″Employees″, assumendo che la configurazione di ″Employees″ sia definita nel file. Per informazioni dettagliate sulla sintassi XML, consultare "Configurazione di ObjectGrid" a pagina 256.

Sono disponibili sette metodi createObjectGrid. I metodi sono documentati nel seguente blocco di codice.

```
/**
* Un metodo predefinito semplice per restituire un'istanza di un
* Object Grid. Viene assegnato un nome univoco.
* L'istanza di ObjectGrid non è memorizzata nella cache.
* Gli utenti possono quindi utilizzare
* {@link ObjectGrid#setName(String)} per modificare
* il nome di ObjectGrid.
*
* @return ObjectGrid Un'istanza di ObjectGrid con un nome univoco assegnato
* @throws ObjectGridException Qualsiasi errore rilevato durante la creazione dell'ObjectGrid
* @ibm-api
*/
public ObjectGrid createObjectGrid() throws ObjectGridException;
/**
* Un metodo predefinito semplice per restituire un'istanza di un ObjectGrid con
* il nome specificato. Le istanze di ObjectGrid possono essere memorizzate nella cache.
* Se un ObjectGrid con questo nome è già stato memorizzato nella cache, allora
* verrà emessa.
*
* @param objectGridName Il nome dell'ObjectGrid da creare.
* @param cacheInstance true, se l'istanza di ObjectGrid deve essere memorizzata
* nella cache
* @return Un'istanza di ObjectGrid
* @this con questo nome è già stato memorizzato nella cache o
* qualsiasi errore durante la creazione di ObjectGrid.
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, boolean cacheInstance)
throws ObjectGridException;
/**
* Crea un'istanza ObjectGrid con il nome ObjectGrid specificato. L'istanza
* ObjectGrid cretat verrà memorizzata nella cache.
* @param objectGridName Il nome dell'istanza ObjectGrid da creare.
* @return Un'istanza di ObjectGrid
* @throws ObjectGridException se un ObjectGrid con questo nome è già
* stato memorizzato nella cache o se si è verificato un errore durante
* la creazione di ObjectGrid
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName)
throws ObjectGridException;
/**
* Crea un'istanza ObjectGrid in base al nome ObjectGrid specificato e
* al file XML. L'istanza ObjectGrid definita nel file XML con il nome
* ObjectGrid specificato verrà creata e restituita. Se un tale ObjectGrid
* non viene trovato nel file xml, verrà emessa un'eccezione.
*
* Questa istanza di ObjecGrid può essere memorizzata nella cache.
*
* Se l'URL è null, verrà semplicemente ignorato. In questo caso, questo metodo funziona
* come {@link #createObjectGrid(String, boolean)}.
*
* @param objectGridName Il nome dell'istanza ObjectGrid da restituire. Inoltre,
* non deve essere null.
* @param xmlFile Un URL di un file xml basato sullo schema ObjectGrid.
* @param enableXmlValidation Se true, l'XML viene convalidato
* @param cacheInstance Un valore booleano che indica se le istanze
* di ObjectGrid
* definite nell'XML verranno memorizzate nella cache o meno. Se true, le istanze
* memorizzata nella cache.
*
* @throws ObjectGridException Se un ObjectGrid con lo stesso nome
* è stato precedentemente memorizzato nella cache, nessun nome di ObjectGrid
* può essere trovato nel file xml,
* o qualsiasi altro errore durante la creazione di ObjectGrid.
* @return Un'istanza di ObjectGrid
* @see ObjectGrid
```

```
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, final URL xmlFile,
final boolean enableXmlValidation, boolean cacheInstance) throws
ObjectGridException;
/**
* Elabora un file XML e crea un elenco di oggetti ObjectGrid basati
* sul file.
* Queste istanze di ObjecGrid possono essere memorizzate nella cache.
* Un'eccezione ObjectGridException viene emessa quando si prova a memorizzare nella cache
* un ObjectGrid appena creato
* che ha lo stesso nome di un ObjectGrid che è già stato memorizzato nella cache.
*
* @param xmlFile Il file che definisce un ObjectGrid o più
* ObjectGrids
* @param enableXmlValidation L'impostazione true convaliderà il file XML
* rispetto allo schema
* @param cacheInstances Impostato su true per memorizzare nella cache tutte
* le istanze di ObjectGrid create in base al file
* @return Un'istanza di ObjectGrid
* @throws ObjectGridException Se prova a creare e memorizzare nella cache
* un ObjectGrid con lo stesso nome di
* un ObjectGrid che è già stato memorizzato nella cache, o qualsiasi altro errore
* che si è verificato durante la
* creazione di ObjectGrid
* @ibm-api
*/
public List createObjectGrids(final URL xmlFile,
final boolean enableXmlValidation,
boolean cacheInstances)
throws ObjectGridException;
/*** Crea tutti gli ObjectGrid trovati nel file XML. Il file XML verrà
* convalidato rispetto allo schema. Ogni istanza ObjectGrid creata verrà
* memorizzata nella cache. Un ObjectGridException verrà emesso quando
* si prova a memorizzare nella cache
* l'ObjectGrid appena creato che ha lo stesso nome di un ObjectGrid che è
* già stato memorizzato nella cache.
* @param xmlFile Il file XML da elaborare. ObjectGrids verrà creato in base
* a quanto riportato nel file.
* @return Un elenco delle istanze ObjectGrid che sono state create.
* @throws ObjectGridException Se un ObjectGrid con lo stesso nome di un altro
* trovato nell'XML è già stato memorizzato nella cache, o
* qualsiasi altro errore rilevato durante la creazione di ObjectGrid.
* @ibm-api
*/
public List createObjectGrids(final URL xmlFile) throws ObjectGridException;
/**
* Elabora il file XML e crea un'unica istanza di ObjectGrid con
* objectGridName specificato solo se un ObjectGrid con lo stesso nome viene trovato
* nel file. Se non è presente alcun ObjectGrid con questo nome definito nel file XML,
* l'eccezione ObjectGridException
* verrà emessa. L'istanza ObjectGrid creata verrà memorizzata nella cache.
* @param objectGridName Il nome dell'ObjectGrid da creare. Questo ObjectGrid
* deve essere definito nel file XML.
* @param xmlFile Il file XML da elaborare
* @return Un ObjectGrid appena creato
* @throws ObjectGridException Se un ObjectGrid con lo stesso nome è stato
* precedentemente memorizzato nella cache, nessun nome ObjectGrid viene trovato nel file xml
* o qualsiasi altro errore durante la creazione di ObjectGrid.
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
throws ObjectGridException;
```

Metodi getObjectGrid

Utilizzare i metodi getObjectGrid per richiamare le istanze memorizzate nella cache.

Richiamo di un'istanza memorizzata nella cache

Poiché l'istanza Employees ObjectGrid è stata memorizzata nella cache dall'interfaccia ObjectGridManager, qualsiasi altro utente può accedere ad essa mediante il seguente frammento di codice:

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

Di seguito sono riportati due metodi getObjectGrid che restituiscono le istanze di ObjectGrid memorizzate nella cache.

```
/**
* Richiama un elenco delle istanze di ObjectGrid che sono state precedentemente
* memorizzate nella cache.
* Restituisce null se nessuna istanza di ObjectGrid è stata memorizzata nella cache.
* @return Un elenco di istanze di ObjectGrid che sono state precedentemente
* memorizzate nella cache
* @ibm-api
*/
public List getObjectGrids();
/**
* Da utilizzare se un ObjectGrid esiste già. Esso restituisce una
* istanza di ObjectGrid memorizzata nella cache in base al nome. Questo metodo
* restituisce null se
* nessun ObjectGrid con questo objectGridName è stato memorizzato nella cache.
*
* @param objectGridName Il nome di objectgrid memorizzato nella cache.
* @return Un ObjectGrid esistente memorizzato nella cache.
*
* @since WAS XD 6.0
* @ibm-api
*
*/
public ObjectGrid getObjectGrid(String objectGridName);
```

Metodi removeObjectGrid

In questa sezione viene descritto come utilizzare i due metodi removeObjectGrid.

Rimozione di un'istanza ObjectGrid

Per rimuovere le istanze di ObjectGrid dalla cache, utilizzare uno dei metodi removeObjectGrid. ObjectGridManager elimina il riferimento delle istanze che vengono rimosse. Esistono due metodi di rimozione. Un metodo utilizza un parametro booleano. Se il parametro booleano è impostato su **true**, il metodo di eliminazione viene richiamato sull'ObjectGrid. la chiamata al metodo sull'ObjectGrid arresta l'ObjectGrid e libera le risorse che lo utilizzavano. Di seguito è riportata una descrizione di come utilizzare i due metodi removeObjectGrid:

```
/**
* Rimuove un ObjectGrid dalla cache delle istanze di ObjectGrid
* @param objectGridName il nome dell'istanza ObjectGrid da rimuovere
* dalla cache
* @throws ObjectGridException Se un ObjectGrid con objectGridName
* non è stato trovato nella cache
* @ibm-api
*/
public void removeObjectGrid(String objectGridName) throws ObjectGridException;
/**
* Rimuove un ObjectGrid dalla cache delle istanze di ObjectGrid e
* elimina le risorse associate
* @param objectGridName il nome dell'istanza ObjectGrid da rimuovere
* dalla cache
* @param destroy Elimina l'istanza di objectgrid e le risorse
* associate
```

```
* @throws ObjectGridException Se un ObjectGrid con objectGridName
* non è stato trovato nella cache
* @ibm-api
*/
public void removeObjectGrid(String objectGridName, boolean destroy)
throws ObjectGridException;
```

Metodo getObjectGridAdministrator

Restituisce un'istanza ObjectGridAdministrator per il cluster

```
public ObjectGridAdministrator getObjectGridAdministrator(ClientClusterContext ctx)
/**
* Restituisce un'istanza ObjectGridAdministrator per il cluster. Ogni
* cluster richiede l'uso di un ObjectGridAdministrator differente.
*
* @param clientClusterContext. Un contesto di cluster univoco, con cui il client
* deve interagire.
* @param objectGridName Il nome di objectgrid memorizzato nella cache.
* @return un ObjectGrid
*
* @since WAS XD 6.0.1
* @ibm-api
*
*/
public ObjectGridAdministrator getObjectGridAdministrator(ClientClusterContext
ontext);
```

Fare riferimento a Capitolo 7, "Panoramica sulla gestione del sistema", a pagina 67 per ulteriori informazioni su questo metodo.

Utilizzare l'interfaccia ObjectGridManager per controllare il ciclo di vita di un'istanza di ObjectGrid.

In questa sezione viene dimostrato il modo in cui l'interfaccia ObjectGridManager può essere utilizzata per controllare il ciclo di un'istanza di ObjectGrid utilizzando un bean di avvio e un servlet.

Gestione del ciclo di un'istanza di ObjectGrid in un bean di avvio

Un bean di avvio può essere utilizzato per controllare il ciclo di un'istanza di ObjectGrid. Un bean di avvio viene caricato all'avvio di un'applicazione. Con un bean di questo genere, il codice può essere eseguito all'avvio o all'arresto dell'applicazione come previsto. Per creare un bean di avvio, utilizzare l'interfaccia `com.ibm.websphere.startupservice.AppStartUpHome` principalee l'interfaccia `com.ibm.websphere.startupservice.AppStartUp` remota. Implementare il metodi `start` e `stop` sul bean. Il metodo `start` viene richiamato quando l'applicazione viene avviata. Il metodo `stop` viene invece richiamato quando l'applicazione viene arrestata. Il metodo `start` può essere utilizzato per creare le istanze di ObjectGrid. Il metodo `stop` può essere invece utilizzato per eliminare le istanze di ObjectGrid. Di seguito è riportato un frammento di codice che dimostra la gestione del ciclo ObjectGrid in un bean di avvio.

```
public class MyStartupBean implements javax.ejb.SessionBean {
private ObjectGridManager objectGridManager;
/*
* I metodi sull'interfaccia SessionBean sono stati lasciati
* al di fuori di questo esempio per motivi di brevità
*/
public boolean start(){
 // Avvia il bean di avvio
 // Questo metodo viene richiamato quando viene avviata l'applicazione
 objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
```

```
try {
  // crea 2 ObjectGrid e memorizza nella cache tali istanze
  ObjectGrid bookstoreGrid =
objectGridManager.createObjectGrid("bookstore", true);
bookstoreGrid.defineMap("book");
  ObjectGrid videostoreGrid =
objectGridManager.createObjectGrid("videostore", true);
  // all'interno della JVM,
  // tali ObjectGrids possono essere richiamati da
  //ObjectGridManager mediante il metodo getObjectGrid(String)
  } catch (ObjectGridException e) {
e.printStackTrace();
return false;
}
return true;
}
public void stop()  {
 // Arresta il bean di avvio
 // Questo metodo viene richiamato quando l'applicazione viene arrestata
try {
  // rimuove gli ObjectGrid memorizzati nella cache e li elimina
objectGridManager.removeObjectGrid("bookstore", true);
objectGridManager.removeObjectGrid("videostore", true);
  } catch (ObjectGridException e) {
e.printStackTrace();
}
}
}
```

Una volta richiamato il metodo `start`, le istanze di ObjectGrid appena create possono essere richiamate da ObjectGridManager. Ad esempio, se nell'applicazione è incluso un servlet, questo può accedere a tali ObjectGrid utilizzando il seguente frammento di codice:

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

Gestione del ciclo di ObjectGrid in un servlet

Un metodo per gestire il ciclo di un ObjectGrid in un servlet consiste nel creare l'istanza di ObjectGrid nel metodo `init` ed eliminare l'ObjectGrid nel metodo `destroy`. Se l'istanza di ObjectGrid viene memorizzata nella cache, questa può essere richiamata e modificata nel codice del servlet. Di seguito è riportato un codice di esempio che dimostra la creazione dell'ObjectGrid, la modifica e l'eliminazion e all'interno di un servlet.

```
public class MyObjectGridServlet extends HttpServlet implements Servlet {
private ObjectGridManager objectGridManager;
 public MyObjectGridServlet() {
 super();
}
public void init(ServletConfig arg0) throws ServletException {
 super.init();
 objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
try {
  // crea e memorizza nella cache un ObjectGrid denominato 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");
 // esegue le operazioni sull'ObjectGrid memorizzato nella cache
// ...
}
public void destroy() {
 super.destroy();
try {
  // rimuove ed elimina il bookstore ObjectGrid memorizzato nella cache
objectGridManager.removeObjectGrid("bookstore", true);
  } catch (ObjectGridException e) {
e.printStackTrace();
}
}
}
```

Traccia di ObjectGrid

In questa sezione viene riportato come impostare la traccia per ObjectGrid.

Ambiente Java 2 Platform, Standard Edition (J2SE)

Quando è necessario inviare le informazioni sul debug a IBM, utilizzare il meccanismo di traccia. Di seguito è riportato un esempio di come richiamare la traccia di debug in un ambiente J2SE:

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

Il precedente esempio non include la traccia dei plug-in dei programmi di esclusione integrati per ObjectGrid. Se si utilizza uno o più plug-in del programma di esclusione forniti dall'ObjectGrid e dsi verificano dei problemi che potrebbero essere relativi all'eliminazione, abilitare la traccia per ObjectGrid e per i relativi programmi di esclusione come riportato nel seguente esempio:

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

Ambiente WebSphere Application Server

Non è necessario utilizzare ObjectGridManager per impostare la traccia all'interno di un ambiente WebSphere Application Server. È possibile utilizzare la console di gestione per impostare la specifica della traccia.

API di connessione client di ObjectGrid

Informazioni di base

Un client di collega a un processo server attivo o in esecuzione all'interno di un cluster. Per collegarsi, il client ha bisogno almeno del nome host e del numero di porta del server. Il nome host e le informazioni sulla porta sono disponibili dal file XML di definizione del cluster che è stato inizialmente utilizzato per avviare il server. Fare riferimento a "Configurazione di ObjectGrid" a pagina 256 per i dettagli di configurazione XML. Di seguito è riportato un frammento di codice della definizione XML del cluster utilizzato come esempio per questa sezione. Utilizzando le API descritte in questa sezione, il client si collega a un ObjectGrid remoto configurato per ricevere ed elaborare le richieste client. Il client ″scarica″ le definizioni XML di ObjectGrid e del cluster per collegarsi. La configurazione del client si basa sulla configurazione del server a cui si collega.

```
<?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
.
.
.
```

Questa definizione cluster non è completa, ma è sufficiente per questo esempio. All'interno del cluster cluster1 sono presenti tre server, server1, server2 e server3. L'attributo clientAccessPort specifica la porta listener utilizzata dal server e la porta su cui il client stabilisce una connessione. dal precedente codice XML, le porte per i server server1, server2 e server3 sono `12503`, `12504` e `12504` rispettivamente.

API connect

L'interfaccia ObjectGridManager utilizza i metodi connect descritti nel seguente esempio. Le seguenti API connect sono disponibili dall'interfaccia ObjectGridManager. Fare riferimento alla documentazione dell'API per una descrizione di questi metodi.

```
/**
* Consente a un client di collegarsi a un ObjectGrid remoto
* L'ObjectGrid remoto è specificato dai seguenti parametri:
* @param clusterName: il nome del cluster a cui il client
* si collega
* @param host: l'host a cui si collega
* @param port: la porta clientAceess utilizzata
* @param ClientSecurityConfiguration: Security la configurazione di sicurezza, può essere
* null se la sicurezza non è configurata
* @param overRideObjectGrid xml. Questo parametro può essere null. Se non è null,
* la configurazione client del plug-in dell'ObjectGrid viene sovrascritta.
* Non tutti i plug-in possono essere sovrascritti. Per maggiori dettagli,
* fare riferimento alla documentazione di ObjectGrid
* @throws ConnectException
*
*/
public ClientClusterContext connect(String clusterName,
String host,
String port,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
/**
*
* @param clusterName
* @param attributes gli attributi della coppia host-porta che sono provati in
```

```
* ordine sequenziale per collegarsi. Se un tentativo di connessione non riesce
* su un server, viene utilizzata la coppia successiva
* di attributi host e porta per provare a collegarsi di nuovo.
* @param ClientSecurityConfiguration: Configurazione di sicurezza. Questa può essere null se
* la sicurezza non è configurata.
* @param overRideObjectGrid xml. Questo parametro può essere null. Se non è null,
* la configurazione client del plug-in dell'ObjectGrid viene sovrascritta.
* Non tutti i plug-in possono essere sovrascritti. Per maggiori dettagli fare
* riferimento alla documentazione ObjectGrid
* @return ClientClusterContext
* @throws ConnectException
*
*
*/
public ClientClusterContext connect(String clusterName,
HostPortConnectionAttributes[] attributes,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
/**
* Questo metodo può essere utilizzato solo se un client ha un
* server ObjectGrid, in particolare in un ambiente Java 2 Platform,
* Enterprise Edition (J2EE) con IBM WebSphere
* Application Server, che supporta
* il server ObjectGrid integrato.
* Questo metodo consente il collegamento del client al server in esecuzione
* sulla stessa Java virtual machine (JVM).
* @param securityProps. Questo parametro può essere null se non eseguito in modalità sicura.
* @param overRideObjectGrid xml. Questo parametro può essere null. Se è
* null, la configurazione client del plug-in ObjectGrid viene sovrascritta.
* Non tutti i plug-in possono essere sovrascritti. Per maggiori dettagli,
* fare riferimento alla documentazione di ObjectGrid
* @return ClientClusterContext
* @throws ConnectException
*
*/
public ClientClusterContext connect(ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException;
/**
* Consente al client di collegarsi a un ObjectGrid remoto
* @param clusterConfigFile Un URL per il file clusterConfig. Questo è lo stesso
* file utilizzato per avviare i server.
* Esso è utilizzato per richiamare le informazioni sulla porta host. Non può essere null. Se è
* null viene restituita una eccezione IllegalArgumentException.
* @param serverName Una stringa, il nome del server specifico a cui ci si collega.
* Se il nome del server non è presente nella configurazione, allora
* viene restituita l'eccezione IllegalArgumentException.
* Questo parametro può essere null e in questo caso viene
* effettuato un tentativo di connessione
* a uno dei server specificati nel file XML
* del cluster. Se si verifica un erroe durante un tentativo di connessione,
* viene selezionato un altro server.
* Questa operazione continua fino a che viene completato l'elenco.
* @param securityProps
* @param overRideObjectGrid xml. Questo parametro può essere null. Se non è
* null, la configurazione client del plug-in ObjectGrid viene sovrascritta.
* Non tutti i plug-in possono essere sovrascritti. Per maggiori dettagli fare
* riferimento alla documentazione
* ObjectGrid
* @return ClientClusterContext
* @throws ConnectException
*
* @ibm-api
*/
public ClientClusterContext connect(URL clusterConfigFile,
String serverName,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
```

Esempio mediante i parametri di host e porta

Il seguente codice utilizza l'XML del cluster come riportato in "Informazioni di base " a pagina 96. Questo client si collega all'host `s1.myco.com` sulla 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(); //step 1
ClientClusterContext ctx = null;
try {
ctx=oGridManager.connect("cluster1","s1.myco.com","12503",null,null);
// passo 2
ObjectGrid employees = oGridManager.getObjectGrid(ctx,"employees");
// passo 3
// Esegue operazioni objectGrid
// get
// update
// commit...ecc..
} catch (ConnectException e) {
// connessione non riuscita
e.printStackTrace();
// fine
}finally {
if(ctx !=null) {
oGridManager.disconnect(ctx); // step 4
}
}
}
}
```

1. Richiamare l'oggetto singleton ObjectGridManager da ObjectGridManagerFactory.
2. Richiamare l'API connect.
3. Se si assume che gli impiegati objectGrid esistano sull'ObjectGrid remoto, richiamare il metodo getObjectGrid passando il parametro ClientClusterContext.
4. Richiamare il metodo disconnect. Come ultimo passo, disconnettere tutti i client e l'operazione sarà completa. Questo è un passo molto importante.

Specifica di più host per provare una nuova connessione nel caso di una eccezione ConnectException

In questo esempio, gli attributi HostPortConnectionAttributes sono utilizzati per fornire un'array di attributi host-porta mediante i quali un clien t può collegarsi. L'API utilizza questi attributi in ordine sequenziale per collegarsi. Se un tentativo di connessione non riesce su un server, viene utilizzata la coppia successiva di attributi.

```
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");
// Esegue operazioni objectGrid come
// get
// update
// commit.... ecc...
} catch (ConnectException e) {
e.printStackTrace();
}finally {
if(ctx !=null) {
oGridManager.disconnect(ctx);
}
}
}
}
```

Client eserver nello stesso processo

Se il client si trova sulla stessa JVM del server, è possibile utilizzare il seguente metodo connect.

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

Specifica dell'XML del cluster

Se il client ha accesso al file XML del cluster, non è necessario specificare il nome host o il numero di porta. Questa API richiama il nome del server e il numero di porta e li utilizza per collegarsi. Il nome del server è facoltativo e può essere null, nel qual caso, l'API prova a collegarsi a uno dei server definiti nel file XML del cluster.

```
ctx=oGridManager.connect(urlToClusterxml,"server1",null,null);
// connessione a server1
// o
ctx=oGridManager.connect(urlToClusterxml,null,null,null);
// connessione a qualsiasi server nel cluster
```

Sicurezza client con l'API connect

In tutti gli esempi, il parametro as ClientSecurityConfiguration era null. L'invio del valorenull implica che la sicurezza è disabilitata. Se la sicurezza è invece abilitata, inviare l'oggetto ClientSecurityConfiguration come argomento. Per ulteriori informazioni, consultare "Sicurezza di ObjectGrid" a pagina 136.

Sovrascrittura della configurazione XML di ObjectGrid

Il client ″scarica″ le definizioni di ObjectGrid dal server per la propria configurazione. Tutti i plug-in definiti nell'ObjectGrid sono resi disponibili sul client. In breve, un ObjectGrid locale è presente sul client che comunica con l'ObjectGrid del server. Fornendo un file XML di sovrascrittura sull'API connect, è possibile ″sovrascrivere″ la configurazione del plug-in, che è specifica solo per l'uso con un client. Questi plug-in sono:

ObjectGrid plug-ins:

- Plug-in TransactionCallback
- Plug-in ObjectGridEventListener

Plug-in **BackingMap**:

- Plug-in Evictor
- Plug-in MapEventListener

Qualsiasi altro plug-in definito nell'XML di sovrascrittura verrà ignorato.

Esempio

Si assuma che il client debba sovrascrivere la configurazione di Evictor per un determinato BackingMap. In altre parole, sul client l'Evictor deve essere differente da quello configurato sul server.

Sul server l'Evictor viene utilizzato come riportato di seguito. Esso utilizza l'evictor LFUEvictor integrato:

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

I requisiti del client sono differenti. È necessario utilizzare un Evictor definito dall'utente, myco.og.MyEvictor. L'XML di sovrascrittura può includere il frammento di codice riportato di seguito. Tutti i backingMaps configurati per l'utilizzo di LFUEvictor, utilizzano la definizione dell'utente:

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

Completamento dell'XML

Il seguente codice XML visualizza due file XML, uno utilizzato per il server e uno utilizzato per il client. Questa configurazione consente a un client di sovrascrivere la configurazione dell'Evictor per il dow BackingMap.

File XML dell'ObjectGrid del server

```
<?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>
```

File XML dell'ObjectGrid del client da sovrascrivere durante la connessione

```
<?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>
```

Considerazioni sull'applicazione

L'API client connect() API è un'operazione costosa. A seconda del carico di lavoro, un client stabilisce una o più connessioni fisiche a un unico server. Il numero di connessioni client varia in base ai valori specificati dagli attributi tcpMinConnections e tcpMaxConnections definiti all'interno dell'elemento cluster della definizione XML di configurazione del cluster. Questa è la stessa definizione XML del cluster utilizzato per avviare il server. Il gestore di connessioni raccoglie queste connessioni fisiche e l'ObjectGrid le riutilizza in base alle necessità. Gli attributi tcpMinConnections e tcpMaxConnections specificano il numero di connessioni client per un singolo server. Se un client si collega a più di un server, il numero massimo di connessioni client è minore o uguale al numero di volte dell'attributo tcpMaxConnections dei server a cui si collegano i client. Ad esempio, se il client si collega a tre server e il valore di tcpMaxConnections è cinque, allora il client ha un massimo di `(5*3)=15` connessioni e un minimo di tre connessioni, sempre che l'impostazione di tcpMinConnection sia `1`. Le connessioni sono condivise tra più client.

L'attributo**threadsPerClientConnect** specifica il numero di thread operativi. Tali thread distribuiscono il lavoro tra le varie connessioni fisiche. Essi elaborano i dati di configurazione, le richieste client, le risposte server e le richieste di gestione del sistema. Il valore predefinito è `5`. Questo attributo è disponibile nel primo iFix. Se l'iFix non è disponibile, utilizzare la proprietà di sistema della JVM -Dthreads per specificare il numero di thread operativi. A seconda dell'applicazione, l'aumento di questo numero migliora le prestazioni. Questo numero dovrebbe essere sempre inferiore al numero di connessioni fisiche utilizzate dal client.

Se l'applicazione lo permette, un client può creare più thread per completare il lavoro in una chiamata al metodo connect riutilizzando il metodo *ClientClusterContext*.

Interfaccia ObjectGrid

Utilizzare questa sezione come riferimento ai moduli necessari per modificare un ObjectGrid.

Introduzione

ObjectGrid è una struttura di memorizzazione nella cache degli oggetti estensibile e transazionale basata sull'interfaccia Java Map. Le operazioni dell'API ObjectGrid sono raggruppate in un'unità di lavoro transazionale e consentono l'estensibilità grazie al supporto dei plug-in personalizzati. ObjectGrid è un contenitore logico che comprende un determinato numero di BackingMaps. Per ulteriori informazioni sulle mappe di backup, fare riferimento a "Interfaccia BackingMap" a pagina 108.

Creazione e inizializzazione

Fare riferimento alla sezione relativa all'interfaccia ObjectGridManager per le operazioni necessarie per la creazione di un'istanza di ObjectGrid. Esistono due modi distinti per creare un ObjectGrid: in maniera programmatica o mediante i file di configurazione XML. Per ulteriori informazioni, consultare "Interfaccia ObjectGridManager" a pagina 89.

Richiamo o impostazione e metodi predefiniti

Attenzione: Qualsiasi metodo *set* deve essere richiamato prima dell'inizializzazione dell'istanza ObjectGrid. Se si richiama un metodo set dopo aver richiamato il metodo di inizializzazione, viene restituita un'eccezione `java.lang.IllegalStateException`. Ogni metodi getSession dell'interfaccia ObjectGrid richiama implicitamente il metodo initialize. Pertanto, è necessario richiamare i metodi set prima di richiamare qualsiasi metodo getSession. L'unica eccezione a questa regola è data dall'impostazione, l'aggiunta e la rimozione di oggetti EventListener. Tali oggetti possono essere elaborati una volta completata l'elaborazione di ″inizializzazione″.

L'interfaccia ObjectGrid contiene i seguenti metodi:

Tabella 7. Metodi dell'interfaccia ObjectGrid

Metodo	Descrizione
BackingMap defineMap(String name);	*defineMap*: è un metodo predefinito per definire un BackingMap denominato in maniera univoca. Per ulteriori informazioni sulle mappe di backup, fare riferimento a "Interfaccia BackingMap" a pagina 108.
BackingMap getMap(String name);	*getMap*: restituisce un BackingMap precedentemente definito richiamando *defineMap*. Utilizzando questo metodo, è possibile configurare il BackingMap, se non è già stato configurato mediante la configurazione XML.

Tabella 7. Metodi dell'interfaccia ObjectGrid (Continua)

Metodo	Descrizione
BackingMap createMap(String name);	*createMap*: crea un BackingMap, ma non lo memorizza nella cache per utilizzarlo con questo ObjectGrid. Utilizzare questo metodo insieme al metodo setMaps(List) dell'interfaccia ObjectGrid, che memorizza nella cache i BackingMaps da utilizzare con questo ObjectGrid. Utilizzare questi metodi quando si configura un ObjectGrid con Spring Framework.
void setMaps(List mapList);	*setMaps*: cancella i BackingMaps che sono stati precedentemente definiti su questo ObjectGrid e li sostituisce con l'elenco di BackingMaps fornito.
public Session getSession() throws ObjectGridException, TransactionCallbackException;	*getSession*: restituisce un oggetto Session, che fornisce le funzionalità di inizio, commit e rollback per una unità di lavoro. Per ulteriori informazioni sugli oggetti Session, fare riferimento a "Interfaccia Session" a pagina 112.
Session getSession(CredentialGenerator cg);	*getSession(CredentialGenerator cg)*: richiama una sessione con un oggetto CredentialGenerator. Questo metodo può essere richiamato solo dal client ObjectGrid in un ambiente server client.
Session getSession(Subject subject);	*getSession(Subject subject)*: consente l'utilizzo di un determinato oggetto Subject invece che di quello configurato sull'ObjectGrid per richiamare una sessione.
void initialize() throws ObjectGridException;	*initialize*: l'ObjectGrid viene inizializzato ed è disponibile per un uso generale. Questo metodo viene richiamato implicitamente quando viene richiamato il metodo `getSession`, se l'ObjectGrid non si trova in uno stato inizializzato.
void destroy();	*destroy*: la struttura viene disassemblata e non può essere più utilizzata dopo aver richiamato questo metodo.

Tabella 7. Metodi dell'interfaccia ObjectGrid (Continua)

Metodo	Descrizione
void setTxTimeout(int timeout);	*setTxTimeout*: utilizzare questo metodo per impostare il tempo, espresso in secondi, che una transazione avviata da una sessione creata da un'istanza di questo ObjectGrid, impiega per il completamento. Se una transazione non viene completata in questo periodo di tempo, la sessione che ha avviato la transazione viene contrassegnata come ″Scaduta″. Contrassegnando una sessione in questo modo, il successivo metodo ObjectMap richiamato dalla sessione scaduta restituirà una eccezione `com.ibm.websphere.objectgrid.TransactionTimeoutException` . La sessione viene contrassegnata come *solo rollback*, il che provoca il rollback della transazione anche se l'applicazione richiama il metodo commit invece che il metodo rollback dopo che l'eccezione `TransactionTimeoutException` è stata rilevata dall'applicazione. Un valore di timeout pari a `0` indica che la transazione non ha alcun limite di tempo per essere completata. La transazione non scade mai se viene utilizzato il valore `0`. Se questo metodo non viene richiamato, allora per impostazione predefinita, qualsiasi sessione restituita dal metodo getSession di questa interfaccia ha un valore di timeout di transazione impostato su `0`. Un'applicazione può sovrascrivere l'impostazione del timeout di transazione su una base per sessione utilizzando il metodo setTransactionTimeout dell'interfaccia com.ibm.websphere.objectgrid.Session.
int getTxTimeout();	*getTxTimeout*: restituisce il valore di timeout di transazione espresso in secondi. Questo metodo restituisce lo stesso valore inviato come parametro di timeout sul metodo setTxTimeout. Se il metodo setTxTimeout non è stato richiamato, allora il metodo restituisce `0` per indicare che la transazione non ha alcun limite di tempo per il completamento.
// Parole chiave.	

Tabella 7. Metodi dell'interfaccia ObjectGrid (Continua)

Metodo	Descrizione
void associateKeyword(Serializable parent, Serializable child);	*associateKeyword*: la parola chiave ObjectGrid fornisce un meccanismo di invalidazione flessibile basato sulle parole chiave. Per ulteriori informazioni sulle parole chiave, fare riferimento a "Parole chiave" a pagina 120. Questo metodo collega due parole chiave in una relazione direzionale. Se viene invalidata una voce principale, allora verrà invalidata anche la voce secondaria. L'invalidazione dell'elemento secondario non ha invece alcun impatto sulla voce principale.
// Sicurezza	
void setSecurityEnabled()	*setSecurityEnabled*: abilita la sicurezza. Per impostazione predefinita, la sicurezza è disabilitata.
void setPermissionCheckPeriod(long period);	*setPermissionCheckPeriod*: questo metodo utilizza un unico parametro che indica la frequenza con cui controllare le autorizzazioni utilizzate per consentire un accesso del client. Se il parametro è `0`, tutti i metodi richiedono il meccanismo di autorizzazione, sia esso JAAS o l'autorizzazione personalizzata, per verificare se l'oggetto corrente ha le autorizzazioni. Questa strategia potrebbe provocare problemi di prestazioni a seconda dell'implementazione dell'autorizzazione. Tuttavia, questo tipo di autorizzazione è disponibile, se richiesto. In alternativa, se il parametro è minore di `0`, allora indica il numero di millisecondi per cui memorizzare nella cache una serie di autorizzazioni prima di tornare al meccanismo di autorizzazione per l'aggiornamento. Questo parametro fornisce migliori prestazioni, ma se le autorizzazioni del backend vengono modificate, allora l'ObjectGrid potrebbe consentire o impedire l'accesso anche se il provider di sicurezza del backend è stato modificato.
void setAuthorizationMechanism(int authMechanism);	*setAuthorizationMechanism*: imposta il meccanismo di autorizzazione. Il valore predefinito è `SecurityConstants.JAAS_AUTHORIZATION`.

Tabella 7. Metodi dell'interfaccia ObjectGrid (Continua)

Metodo	Descrizione
setMapAuthorization(MapAuthorization ma);	*setMapAuthorization*: imposta il plug-inMapAuthorization per l'istanza di questo ObjectGrid. Questo plug-in può essere utilizzato per autorizzare gli accessi di ObjectMap o JavaMap ai principali contenuti nell'oggetto Subject. Una tipica implementazione di questo plug-in consiste nel richiamare i principali dall'oggetto Subject e verificare se sono state concesse le autorizzazioni specificate consultando politiche specifiche.
setSubjectSource(SubjectSource ss);	*setSubjectSource*: imposta il plug-in SubjectSource. Questo plug-in può essere utilizzato per richiamare un oggetto Subject che rappresenta il client ObjectGrid. Questo oggetto può quindi essere utilizzato per l'autorizzazione dell'ObjectGrid. Il metodo SubjectSource.getSubject viene richiamato dal runtime ObjectGrid quando il metodo ObjectGrid.getSession viene utilizzato per richiamare una sessione ed è abilitata la sicurezza. Questo plug-in è utile per un client già autenticato: esso può richiamare l'oggetto Subject autenticato e inviarlo all'istanza ObjectGrid. Non è necessaria un'ulteriore autenticazione.
setSubjectValidation(SubjectValidation sv);	*setSubjectValidation*: imposta il plug-in SubjectValidation per l'istanza di questo ObjectGrid. Questo plug-in può essere utilizzato per verificare che un oggetto javax.security.auth.Subject, inviato a ObjectGrid o richiamato dal plug-in SubjectSource, è un oggetto valido che non è stato modificato. Un'implementazione di questo plug-in richiede il supporto dal creatore dell'oggetto Subject in quanto solo costui è a conoscenza se è stata eseguito un accesso non consentito all'oggetto Subject. Tuttavia, il creatore dell'oggetto potrebbe non essere a conoscenza di questo fatto. In questo caso, il plug-in non va utilizzato.

Interfaccia ObjectGrid: plug-in

L'interfaccia BackingMap ha diversi plug-in opzionali per interazioni più estensibili.

```
void addEventListener(ObjectGridEventListener cb);
void setEventListeners(List cbList);
void removeEventListener(ObjectGridEventListener cb);
void setTransactionCallback(TransactionCallback callback);
int reserveSlot(String);
// Plug-in relativi alla sicurezza
void setSubjectValidation(SubjectValidation subjectValidation);
void setSubjectSource(SubjectSource source);
void setMapAuthorization(MapAuthorization mapAuthorization);
```

- *ObjectGridEventListener*: un'interfaccia ObjectGridEventListener viene utilizzata per ricevere le notifiche relative a quando si verifica un evento significativo sull'ObjectGrid. Tra questi eventi vi sono l'inizializzazione di ObjectGrid, l'inizio o la fine di un a transazione e l'eliminazione di un ObjectGrid. Per ascoltare questi eventi, creare una classe che implementi l'interfaccia ObjectGridEventListener e che la aggiunga all'ObjectGrid. Tali listener sono associati a ogni sessione. Per ulteriori informazioni, consultare "Listener" a pagina 182 e "Interfaccia Session" a pagina 112.
- *TransactionCallback*: l'interfaccia del listener TransactionCallback consente l'invio di eventi transazionali quali begin, commit e rollback a questa interfaccia. Di solito, un'interfaccia del listener TransactionCallback viene utilizzata con un programma di caricamento. Per ulteriori informazioni, consultare "Plug-in TransactionCallback" a pagina 213 e "Programmi di caricamento" a pagina 197. Tali eventi possono quindi essere utilizzati per coordinare le transazioni con una risorsa esterna o all'interno di più programmi di caricamento.
- *reserveSlot*: consente ai plug-in su questoObjectGrid di riservare slot per l'uso nelle istanze degli oggetti che hanno slot come TxID.
- *SubjectValidation*. Se la sicurezza è abilitata, questo plug-in può essere utilizzato per convalidare una classe `javax.security.auth.Subject` che viene inviata a ObjectGrid.
- *MapAuthorization*. Se la sicurezza è abilitata, questo plug-in può essere utilizzato per autorizzare gli accessi di ObjectMap ai principali che sono rappresentati dall'oggetto Subject.
- *SubjectSource*. Se la sicurezza è abilitata, questo plug-in può essere utilizzato per richiamare un oggetto Subject che rappresenta il client ObjectGrid. Questo oggetto può quindi essere utilizzato per l'autorizzazione dell'ObjectGrid.

Interfaccia BackingMap

Ogni istanza ObjectGrid contiene una raccolta di oggetti BackingMap.

Ogni BackingMap viene denominato e aggiunto a un'istanza di ObjectGrid utilizzando il metodo defineMap dell'interfaccia ObjectGrid. Questo metodo restituisce un'istanza di BackingMap che viene quindi utilizzata per definire il comportamento di una singola mappa. Per ulteriori informazioni, consultare "Interfaccia ObjectGrid" a pagina 103.

L'interfaccia Session viene utilizzata per avviare una transazione e per ottenere l'ObjectMap o la JavaMap richiesta per l'esecuzione dell'interazione transazionale tra un'applicazione e un oggetto BackingMap. Tuttavia, le modifiche alle transazioni non vengono applicate all'oggetto BackingMap fino a che viene eseguito il commit della transazione. Una BackingMap può essere considerata come cache in memoria dei dati di cui è stato eseguito il commit per una singola mappa. Per ulteriori informazioni sull'interfaccia Session, fare riferimento a Interfaccia Session.

L'interfaccia com.ibm.websphere.objectgrid.BackingMap fornisce i metodi per l'impostazione degli attributi BackingMap. Alcuni dei metodi set consentono l'estensione di BackingMap mediante una serie di plug-in personalizzati. Di seguito è riportato un elenco dei metodi set per l'impostazione degli attributi e per fornire il supporto dei plug-in personalizzati:

```
// Per l'impostazione degli attributi 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);
// Per l'impostazione di un plug-in personalizzato facoltativo
   fornito dall'applicazione.
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);
```

Un metodo get corrispondente esiste per ognuno dei metodi set riportati.

Attributi di BackingMap

Ogni BackingMap è dotato dei seguenti attributi che possono essere impostati per modificare o controllare il funzionamento di BackingMap:

- Attributo *ReadOnly*. Questo attributo indica se la mappa è una mappa di sola lettura o una di lettura e scrittura. Se questo attributo non è mai impostato per la mappa, allora la mappa è automaticamente una mappa di lettura e scrittura. Quando BackingMap è impostato su sola lettura, ObjectGrid ottimizza le prestazioni per la lettura.
- Attributo *NullValuesSupported*. Questo attributo indica se è possibile inserire un valore null nella mappa. Se questo attributo non è impostato, la mappa non supporta valori null. Se sono supportati i valori null, un'operazione get che restituisce un valore null può significare sia che il valore è realmente null sia che la mappa non contiene la chiave specificata dall'operazione get.
- Attributo *LockStrategy*. Questo attributo determina se un gestore di blocchi viene utilizzato da BackingMap. Se viene utilizzato un gestore blocchi, allora l'attributo LockStrategy viene utilizzato per indicare se viene utilizzato un approccio di blocco ottimistico o pessimistico per il blocco delle voci della mappa. Se questo attributo non è impostato, allora viene utilizzato l'attributo LockStrategy ottimistico. Fare riferimento alla sezione "Blocco" a pagina 127 topic per maggiori dettagli sulle strategie di blocco supportate.
- Attributo *CopyMode*. Questo attributo determina se una copia di un oggetto di valore viene eseguita da BackingMap quando viene letto un valore dalla mappa o se viene inserita in BackingMap durante il ciclo di commit di una transazione. Sono supportate diverse modalità di copia per consentire all'applicazione di trovare un compromesso tra prestazioni e integrità dei dati. Se questo attributo non è impostato, allora viene utilizzata la modalità di copia `COPY_ON_READ_AND_COMMIT`. Questa modalità di copia non fornisce le migliori prestazioni, ma offre le prestazioni ottimali rispetto ai problemi di integrità dei dati. Per ulteriori informazioni sulle modalità di copia, fare riferimento a Procedure ottimali per il metodo copyMode.
- Attributo *CopyKey*. Questo attributo determina se la BackingMap esegue una copia dell'oggetto key quando viene creata una voce nella mappa. L'azione

predefinita consiste nel non creare una copia degli oggetti key in quanto di solito le chiavi sono oggetti che non possono essere modificati.

- Attributo *NumberOfBuckets*. Questo attributo indica il numero di contenitori hash da utilizzare con BackingMap. L'implementazione di BackingMap utilizza una mappa hash per questa implementazione. Se esiste un numero elevato di voci in BackingMap, allora un numero elevato di contenitori corrisponde a migliori prestazioni. Il numero di chiavi che hanno lo stesso contenitore diminuisce man mano che il numero di contenitori aumenta. Un numero elevato di contenitori implica anche una maggiore simultaneità. Questo attributo risulta utile per l'ottimizzazione delle prestazioni. Il valore predefinito `503` viene utilizzato se l'applicazione non ha impostato l'attributo NumberOfBuckets.
- Attributo *NumberOfLockBuckets*. Questo attributo indica il numero di contenitori di blocco utilizzati dal gestore blocco per questo BackingMap. Quando LockStrategy è impostato su OPTIMISTIC o PESSIMISTIC, viene creato un gestore blocchi per BackingMap. Tale gestore utilizza una mappa hash per tenere traccia delle voci che vengono bloccate da una o più transazioni. Se nella mappa hash sono presenti molte voci, allora un numero elevato di contenitori di blocco porta a prestazioni elevate in quanto il numero di chiavi nello stesso contenitore diminuisce man mano che il numero di contenitori aumenta. Un numero elevato di contenitori di blocco implica anche una maggiore simultaneità. Quando l'attributo LockStrategy è impostato su NONE, nessun gestore di blocchi è utilizzato da BackingMap. In questo caso, l'impostazione di numberOfLockBuckets non ha alcun effetto. Se questo attributo non è impostato, viene utilizzato il valore predefinito `383`.
- Attributo *LockTimeout*. Questo attributo viene utilizzato quando BackingMap utilizza un gestore di blocchi. BackingMap un gestore di blocchi nel caso in cui l'attributo LockStrategy è impostato su OPTIMISTIC o PESSIMISTIC. Il valore dell'attributo è espresso in secondi e determina quanto tempo il gestore deve attendere per concedere un blocco. Se questo attributo non è impostato, allora viene utilizzato il valore di LockStrategy pari a 15 secondi. Fare riferimento a Blocco pessimistico per maggiori dettagli sulle eccezioni di timeout di attesa blocco che si possono verificare.
- Attributo *TtlEvictorType*. Ogni BackingMap ha un proprio programma di esclusione integrato che utilizza un algoritmo temporale per determinare quali voci della mappa escludere. Per impostazione predefinita, il programma di esclusione per il tempo integrato non è attivo. È possibile attivare tale programma richiamando il metodo setTtlEvictorType con uno dei seguenti tre valori: `CREATION_TIME`, `LAST_ACCESS_TIME` o `NONE`. Un valore `CREATION_TIME` indica che il programma di esclusione aggiunge l'attributo TimeToLive all'ora in cui è stata creata la voce della mappa in BackingMap per determinare quando il programma di esclusione deve escludere la voce della mappa da BackingMap. Un valore `LAST_ACCESS_TIME` indica che il programma di esclusione aggiunge l'attributo TimeToLive all'ora in cui una qualsiasi transazione eseguita dall'applicazione per determinare quando si deve verificare l'esclusione della voce, ha eseguito l'accesso alla voce della mappa. La voce della mappa viene eliminata solo se non è mai stato eseguito l'accesso alla voce da parte di una transazione per un periodo di tempo specificato dall'attributo TimeToLive. Un valore uguale a NONE indica che il programma di esclusione deve rimanere inattivo e non deve escludere alcuna voce della mappa. Se questo attributo non è impostato, allora viene utilizzato il valore NONE come valore predefinito e il programma di esclusione non è attivo. Fare riferimento a Evictor per maggiori dettagli relativo all'evictor time to live integrato.
- Attributo *TimeToLive*. Questo attributo viene utilizzato per specificare il numero di secondi che il programma di esclusione deve aggiungere all'ora di creazione o dell'ultimo accesso come descritto precedentemente per l'attributo TtlEvictorType.

Se questo attributo non è impostato, allora viene utilizzato il valore zero per indicare che il tempo da attendere non ha un valore limite. In questo caso, le voci della mappa non vengono eliminate dal programma di esclusione.

Nel seguente esempio viene illustrata la definizione di The someMap BackingMap nell'istanza someGrid ObjectGrid e l'impostazione di vari attributi di BackingMap utilizzando i metodi set dell'interfaccia 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 );
// sovrascrive il valore predefinito di lettura/scrittura
bm.setNullValuesSupported(false);
// sovrascrive il valore predefinito che consente valori null
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
// sovrascrive il valore predefinito OPTIMISTIC
bm.setLockTimeout( 60 );
// sovrascrive il valore predefinito di 15 secondi.
bm.setNumberOfBuckets(251);
// sovrascrive il valore predefinito (i numeri primi sono preferibili)
bm.setNumberOfLockBuckets(251);
// sovrascrive il valore predefinito (i numeri prima sono preferibili)
...
```

Plug-in di BackingMap

L'interfaccia BackingMap ha diversi punti di collegamento opzionali per un'interazione più estensibile con ObjectGrid:

- **Plug-in ObjectTransformer**: per alcune operazioni di associazione, un BackingMap potrebbe aver bisogno di serializzare, deserializzare o copiare una chiave o un valore di una voce in BackingMap. BackingMap può eseguire queste azioni fornendo un'implementazione predefinita dell'interfaccia ObjectTransformer. Un'applicazione può migliorare le prestazioni fornendo un plug-in ObjectTransformer personalizzato che può essere utilizzato da BackingMap per serializzare, deserializzare o copiare una chiave e un valore di una voce in BackingMap. Per ulteriori informazioni, consultare "Plug-in ObjectTransformer" a pagina 209.
- **Plug-in Evictor**: il programma di esclusione integrato utilizza un algoritmo temporale per decidere quando deve essere esclusa una voce in BackingMap. Alcuni applicazioni potrebbero dover utilizzare un algoritmo differente per decidere quando escludere una voce in un BackingMap. Il plug-in Evictor rende un programma di esclusione personalizzato disponibile per l'utilizzo con BackingMap. Il plug-in Evictor si aggiunge al programma di esclusione integrato, ma non lo sostituisce. ObjectGrid fornisce un plug-in Evictor personalizzato che implementa gli algoritmi noti con ″least recently used″ o ″least frequently used″. Le applicazioni possono utilizzare i plug-in Evictor forniti oppure utilizzare i propri. Per ulteriori informazioni, consultare "Programmi di esclusione" a pagina 187.
- **Plug-in MapEventListener**: un'applicazione potrebbe aver bisogno di conoscere gli eventi BackingMap come l'eliminazione delle voci della mappa o il precaricamento di un completamento di BackingMap. BackingMap richiama i metodi sul plug-in MapEventListener per inviare una notifica a un'applicazione relativamente agli eventi BackingMap. Un'applicazione può ricevere una notifica di vari eventi BackingMap mediante il metodo setMapEventListener per fornire

uno o più plug-in MapEventListener personalizzati a BackingMap. L'applicazione può modificare gli oggetti MapEventListener riportati utilizzando il metodo addMapEventListener o il metodo removeMapEventListener. Per ulteriori informazioni, consultare "Interfaccia MapEventListener" a pagina 185.

- **Plug-in Loader**: un BackingMap è una cache in memoria di una mappa. Un plug-in Loader è un'opzione utilizzata da BackingMap per spostare i dati sulle memoria ed è utilizzata per la memorizzazione permanente per BackingMap. Ad esempio, un programma di caricamento JDBC (Java database connectivity) può essere utilizzato per spostare i dati su un BackingMap e su una o più tabelle relazionali di un database relazionale. Un database relazionale non deve essere utilizzato come archivio permanente per un BackingMap. Il programma di caricamento può essere utilizzato per spostare i dati tra unBackingMap e un file, tra un BackingMap e una mappa di ibernazione, tra un BackingMap e un bean di entità Java 2 Platform, Enterprise Edition (J2EE), tra unBackingMap e un altro server delle applicazioni e così via. L'applicazione deve fornire un plug-in Loader personalizzato per spostare i dati tra BackingMap e l'archivio permanente per ogni tecnologia utilizzata. Se non è fornito un programma di caricamento, BackingMap diventa una cache in memoria semplice. Consultare "Programmi di caricamento" a pagina 197 per ulteriori informazioni su questo plug-in.
- **Plug-in OptimisticCallback**: quando l'attributo LockStrategy per BackingMap è impostato su OPTIMISTIC, BackingMap o il plug-in Loader devono eseguire le operazioni di confronto per i valori della mappa. Il plug-in OptimisticCallback è utilizzato da BackingMap e dal programma di caricamento per eseguire le operazioni di confronto per la creazione di una versione ottimistica. Per ulteriori informazioni, consultare "Interfaccia OptimisticCallback" a pagina 220.
- **MapIndexPlugin plug-in**: un plug-in MapIndexPlugin, o un Indice in breve, è un'opzione utilizzata da BackingMap per creare un indice in base all'attributo specificato dell'oggetto memorizzato. L'indice consente alle applicazioni di trovare gli oggetti in base a un valore o a un intervallo di valori specifico. Esistono due tipi di indice: l'indice statico e l'indice dinamico. Fare riferimento a "Indicizzazione" a pagina 234 per maggiori informazioni.

Interfaccia Session

In questa sezione viene descritto il modo in cui le applicazioni iniziano e terminano le transazioni utilizzando l'interfaccia Session. Questa interfaccia fornisce anche un accesso alle interfacce ObjectMap e JavaMap basate sull'applicazione.

Introduzione

Ogni istanza ObjectMap o JavaMap è collegata direttamente a un determinato oggetto Session. Ogni thread che desidera accedere a un ObjectGrid deve prima ottenere un oggetto Session dall'oggetto ObjectGrid. Un'istanza di Session non può essere condivisa tra thread. ObjectGrid non utilizza alcuna memoria locale dei thread, ma alcune limitazioni relative alla piattaforma potrebbero limitare la possibilità di inviare un oggetto Session da un thread a un altro.

Metodi

I seguenti metodi sono disponibili con l'interfaccia Session. Fare riferimento alla documentazione dell'API per ulteriori informazioni sui seguenti metodi:

```
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();
}
```

Get method

Un'applicazione ottiene un'istanza Session da un oggetto ObjectGrid mediante il metodo ObjectGrid.getSession. Il seguente frammento di codice dimostra come ottenere un'istanza di Session:

```
ObjectGrid objectGrid = ...;
Session sess = objectGrid.getSession();
```

Una volta ottenuto l'oggetto Session, il thread conserva un riferimento alla sessione. Il richiamo del metodo getSession per più volte restituisce un nuovo oggetto Session ogni volta.

Metodi di transazioni e sessioni

Un oggetto Session può essere utilizzato per iniziare, eseguire il commit o eseguire il rollback delle transazioni. Le operazioni rispetto ai BackingMap che utilizzano ObjectMaps e JavaMaps vengono eseguite più efficientemente all'interno di una transazione Session. Una volta avviata una transazione, qualsiasi modifica a uno o più BackingMap nell'ambito della transazione viene memorizzata in una cache speciale per le transazioni fino a che viene eseguito il commit della transazione. Quando viene eseguito il commit, le modifiche in sospeso vengono applicate ai BackingMap e ai Loader e diventano visibili per gli altri client dell'ObjectGrid.

ObjectGrid consente anche il commit automatico delle transazioni, a volte detto auto-commit. Se vengono effettuate delle operazioni di ObjectMap all'esterno del contesto di una transazione attiva, prima dell'operazione viene avviata una transazione implicita e viene eseguito il commit automatico della transazione prima di restituire il controllo all'applicazione.

```
Session session = objectGrid.getSession();
ObjectMap objectMap = session.getMap("someMap");
session.begin();
objectMap.insert("key1", "value1");
objectMap.insert("key2", "value2");
session.commit();
objectMap.insert("key3", "value3"); // auto-commit
```

Metodo Session.flush

Il metodo Session.flush può essere utilizzato solo se un Loader è associato a un BackingMap. Il metodo flush richiama il Loader con la serie di modifiche correnti nella cache della transazione. Il Loader applica le modifiche al backend. Tali modifiche non vengono applicate quando viene richiamato il metodo flush. Se viene eseguito il commit di una transazione Session in seguito a un richiamo del metodo flush, soltanto gli aggiornamenti che si sono verificati in seguit o al richiamo verranno applicati al Loader. Se viene eseguito il rollback di una transazione Session in seguito al richiamo del metodo flush, le modifiche verranno eliminate insieme a tutte le altre modifiche in sospeso della transazione. Non utilizzare frequentemente il metodo Flush in quanto limita la possibilità di eseguire operazioni batch rispetto a un Loader. Di seguito è riportato un esempio di utilizzo del metodo Session.flush:

```
Session session = objectGrid.getSession();
session.begin();
// apporta alcune modifiche
...
session.flush();
// inserisce le modifiche sul Loader, ma non esegue ancora il commit
// apporta altre modifiche
...
session.commit();
```

Metodo no write through

Alcune mappe di ObjectGrid vengono salvate da un Loader, che fornisce una memoria permanente per i dati nella mappa. A volte è utile eseguire il commit dei dati solo sulla mappa ObjectGrid e non inserire i dati sul Loader. L'interfaccia Session fornisce il metodo beginNoWriteThough a questo scopo. Il metodo beginNoWriteThrough avvia una transazione come il metodo begin. Con il metodo beginWriteThrough, quando viene eseguito il commit della transazione, viene eseguito il commit dei dati solo per la mappa in memoria di ObjectGrid e non sulla memoria permanente fornita dal Loader. Questo metodo è particolarmente utile quando si esegue un precaricamento dei dati sulla mappa.

Quando si utilizza un'istanza ObjectGrid distribuita, il metodo beginNoWriteThrough risulta utile quando si apportano modifiche solo alla cache prossima, senza modificare la cache sul server. Se i dati sono inattivi nella cache prossima, l'utilizzo del metodo beginNoWriteThrough consente l'invalidazione delle voci sulla cache prossima senza doverle invalidare anche sul server.

L'interfaccia Session fornisce inoltre il metodo isWriteThroughEnabled per determinare il tipo di transazione correntemente attiva.

```
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
// apporta alcune modifiche
session.commit(); // tali modifiche non verranno inserite sul Loader
```

Ottenimento del metodo dell'oggetto TxID

L'oggetto TxID è un oggetto non visibile identifica la transazione attiva. Utilizzare questo oggetto TxID per:

- Un confronto quando si ricerca una determinata transazione.
- Memorizzare i dati condivisi tra gli oggetti TransactionCallback e Loader.

Fare riferimento alle sezioni “Plug-in TransactionCallback” a pagina 213 e “Programmi di caricamento” a pagina 197 per maggiori informazioni sulla funzione Object.

Impostazione del tipo di transazione per il metodo di controllo delle prestazioni

Se si utilizza ObjectGrid su un server delle applicazioni WebSphere Application Server, potrebbe essere necessario reimpostare il tipo di transazione per il controllo delle prestazioni. È possibile impostare il tipo mediante il metodo setTransactionType. Fare riferimento alla sezione “Controllo delle prestazioni di ObjectGrid con la PMI (performance monitoring infrastructure) di WebSphere Application Server” a pagina 291 per ulteriori informazioni sul metodo setTransactionType.

Elaborazione di un metodo LogSequence completo

ObjectGrid può propagare le serie di modifiche alla mappa ad altri listener ObjectGrid come mezzo per la distribuzione delle mappe da una Java Virtual Machine (JVM) a un’altra. Per semplificare l’elaborazione delle LogSequence ricevute da parte del listener, l’interfaccia Session fornisce il metodo processLogSequence. Questo metodo esamina ogni elemento LogElement all’interno di LogSequence ed esegue l’operazione appropriata, ad esempio insert, update, invalidate e così via, sul BackingMap identificato da LogSequence MapName. Un oggetto Session ObjectGrid deve essere attivo già prima di richiamare il metodo processLogSequence. L’applicazione è responsabile del commit o del rollback per completare l’oggetto Session. L’elaborazione di un commit automatico non è disponibile per questo metodo.

Una normale elaborazione ricevendo ObjectGridEventListener sulla JVM remota avvierebbe una sessione mediante il metodo beginNoWriteThrough, il che impedisce la propagazione infinita delle modifiche, quindi verrebbe richiamato il metodo processLogSequence e avviato il commit o il rollback della transazione.

```
// Utilizza l'oggetto Session inviato durante
//ObjectGridEventListener.initialization...
session.beginNoWriteThrough();
// elabora la LogSequence ricevuta
try {
 session.processLogSequence(receivedLogSequence);
} catch (Exception e) {
 session.rollback(); throw e;
}
// esegue il commit delle modifiche
session.commit();
```

Metodo markRollbackOnly

Questo metodo è utilizzato per contrassegnare la transazione corrente come ″solo rollback″. Contrassegnando una transazione in questo modo, viene garantito che anche se il metodo commit viene richiamato dall’applicazione, viene eseguito il rollback della transazione. Questo metodo viene di solito utilizzato dall’ObjectGrid stesso o dall’applicazione quando si è a conoscenza del fatto che potrebbe verificarsi un danneggiamento dei dati se può essere eseguito il commit della transazione.

Una volta richiamato questo metodo, l’oggetto Throwable inviato al metodo viene concatenato all’eccezione `com.ibm.websphere.objectgrid.TransactionException`

restituita dal metodo commit se questo viene richiamato su una sessione che era precedentemente contrassegnata come ″solo rollback″. Qualsiasi chiamata successiva a questo metodo per una transazione già contrassegnata come ″solo rollback″ verrà ignorata. In altre parole, viene utilizzata solo la prima chiamata che invia un riferimento Throwable non null. Una volta completata la transazione contrassegnata, l'indicatore ″solo rollback″ viene rimosso così che possa essere eseguito il commit della transazione successiva avviata dalla sessione.

Metodo isMarkedRollbackOnly

Viene restituito se la sessione è contrassegnata come ″solo rollback″. Il valore booleano true viene restituito dal metodo se e solo se il metodo markRollbackOnly è stato precedentemente richiamato su questa sessione e la transazione avviata dalla sessione è ancora attiva.

Metodo setTransactionTimeout

Impostare il valore di timeout della transazione per la successiva transazione avviata da questa sessione su un numero specificato di secondi. Questo metodo non influenza il valore di timeout delle transazioni precedentemente avviate da questa sessione. Esso influenza solo le transazioni avviate dopo aver richiamato il metodo. Se il metodo non viene richiamato, allora verrà utilizzato il valore di timeout inviato al metodo setTxTimeout del metodo com.ibm.websphere.objectgrid.ObjectGrid.

Metodo getTransactionTimeout

Questo metodo restituisce il valore di timeout di transazione espresso in secondi. L'ultimo valore che è stato inviato come valore di timeout al metodo setTransactionTimeout viene restituito da questo metodo. Se il metodo setTransactionTimeout non viene richiamato, allora verrà utilizzato il valore di timeout inviato al metodo setTxTimeout del metodo com.ibm.websphere.objectgrid.ObjectGrid.

transactionTimedOut

Questo metodo restituisce il valore booleano true se la transazione corrente avviata da questa sessione è scaduta.

Metodo isFlushing

Questo metodo restituisce il valore booleano true se e solo se tutte le modifiche alle transazioni sono state eliminate dal plug-in del programma di caricamento come risultato del metodo flush dell'interfaccia Session richiamato. Un plugin del programma di caricamento può trovare utile questo metodo quando è necessario sapere il motivo per cui è stato richiamato il metodo batchUpdate.

Metodo isCommitting

Questo metodo restituisce il valore booleano true se e solo se viene eseguito il commit di tutte le modifiche alle transazioni come risultato del metodo commit dell'interfaccia Session richiamato. Un plug-in del programma di caricamento può trovare utile questo metodo quando è necessario sapere il motivo per cui è stato richiamato il metodo batchUpdate.

Interfacce ObjectMap e JavaMap

In questa sezione viene descritto il modo in cui le applicazioni interagiscono con ObjectGrid utilizzando le interfacce ObjectMap e JavaMap. Queste due interfacce sono utilizzate per una interazione a livello di transazioni tra le applicazioni e BackingMaps.

Interfaccia ObjectMap

Un'istanza di ObjectMap viene ottenuta da un oggetto Session che corrisponde al thread corrente. L'interfaccia ObjectMap è il veicolo principale che le applicazioni utilizzano per apportare delle modifiche alle voci in un BackingMap.

Ottenimento di un'istanza di ObjectMap

Un'applicazione richiama un'istanza di ObjectMap da un oggetto Session mediante il metodo Session.getMap(String). Il seguente frammento di codice dimostra come ottenere un'istanza di ObjectMap:

```
ObjectGrid objectGrid = ...;
BackingMap backingMap = objectGrid.defineMap("mapA");
Session sess = objectGrid.getSession();
ObjectMap objectMap = sess.getMap("mapA");
```

Ogni istanza di ObjectMap corrisponde a un determinato oggetto Session. Richiamando più volte il metodo getMap su un determinato oggetto Session con lo stesso nome BackingMap, viene restituita sempre la stessa istanza ObjectMap.

Commit automatico delle transazioni

Come già accennato in precedenza, le operazioni rispetto ai BackingMap che utilizzano ObjectMaps e JavaMaps vengono eseguite più efficientemente all'interno di una transazione Session. ObjectGrid fornisce un supporto per il commit automatico quando i metodi sulle interfacce ObjectMap e JavaMap vengono richiamati all'esterno di una sessione di transazioni. I metodi avviano una transazione implicita, eseguono l'operazione richiesta ed eseguono il commit della transazione implicita.

Sintassi dei metodi

Di seguito è riportata la spiegazione della sintassi di ogni metodo delle interfacce ObjectMap e JavaMap. Il metodo setDefaultKeyword, il metodo invalidateUsingKeyword e i metodi che hanno un argomento serializzabile sono dsicussi nella sezione "Parole chiave" a pagina 120. Il metodo setTimeToLive è discusso nella sezione "Programmi di esclusione" a pagina 187. Fare riferimento alla documentazione dell'API per ulteriori informazioni su questi metodi.

Metodo containsKey
: Determina se una chiav ha un valore in BackingMap o nel programma di caricamento. Se i valori null sono supportati da un'applicazione, questo metodo può essere utilizzato per determinare se un riferimento null restituito da un'operazione get fa riferimento a un valore null o se indica che BackingMap e Loader non contengono la chiave.

Metodo flush
: La sintassi di questo metodo è simile a quella del metodo flush dell'interfaccia Session. L'unica differenza è che il metodo flush di Session applica le modifiche in sospeso per tutte le mappe che sono state modifiche

nella sessione corrente. Con questo metodo, soltanto le modifiche in questo ObjectMap vengono inviate al programma di caricamento.

Metodo get

Legge la voce dal BackingMap. Se la voce non viene trovata sul BackingMap ma un programma di caricamento è associato al BackingMap, questo prova a leggere la voce dal programma. Il metodo getAll è fornito per consentire l'elaborazione di apertura batch.

Metodo getForUpdate

Questo metodo è uguale al metodo get, ma utilizza il metodo getForUpdate che indica a BackingMap e al Loader che si sta per aggiornare la voce. Un programma di caricamento può utilizzare questo suggerimento per emettere una query `SELECT for UPDATE` al backend di database. Se è stata definita una strategia Pessimistic LockingStrategy per BackingMap, il gestore blocchi bloccherà la voce. Il metodo getAllForUpdate è fornito per consentire l'elaborazione di apertura batch.

Metodo insert

Inserisce una voce in BackingMap e nel Loader. Questo metodo indica al BackingMap e al Loader che si desidera inserire una voce che in precedenza non esisteva. Quando si richiama questo metodo, si verifica una eccezione quando viene richiamato il metodo o quando viene eseguito il commit della transazione corrente.

Metodo invalidate

La sintassi del metodo invalidate dipende dal valore del parametro **isGlobal** che viene inviato al metodo. Il metodo invalidateAll è fornito per consentire l'elaborazione di invalidazione batch.

Una invalidazione locale viene specificata quando viene inviato il valore *false* come parametro **isGlobal** del metodo invalidate. L'invalidazione locale elimina qualsiasi modifica alla voce nella cache delle transazioni. Se l'applicazione emette un metodo `get`, la voce viene letta dall'ultimo valore di cui è stato eseguito il commit in BackingMap. Se non è presente alcuna voce in BackingMap, la voce viene letta dal valore dell'ultima eliminazione o dal valore di cui è stato eseguito il commit sul programma di caricamento. Quando viene eseguito il commit di una transazione, qualsiasi voce contrassegnata come invalidata in locale non avrà alcun impatto su BackingMap. Verrà comunque eseguito il commit di qualsiasi modifica eliminata sul programma di caricamento anche se la voce è stata invalidata.

Una invalidazione globale viene specificata quando *true* viene inviato come parametro **isGlobal** del metodo invalidate. L'invalidazione globale elimina le modifiche in sospeso su una voce nella cache delle transazioni e ignora il valore di BackingMap su operazioni successive che vengono eseguite sulla voce. Quando viene eseguito il commit di una transazione, tutte le voci contrassegnate come invalidate in globale vengono eliminate da BackingMap.

Si consideri il seguente esempio di invalidazione: il BackingMap viene salvato da una tabella di database che ha una colonna di incremento automatico. Le colonne di incremento sono utili per assegnare i numeri univoci ai record. L'applicazione inserisce una voce. In seguito all'inserimento, l'applicazione deve conoscere il numero di sequenza per la riga inserita. L'applicazione sa che la copia dell'oggetto è obsoleta, pertanto utilizza l'invalidazione globale per richiamare il valore dal programma di caricamento. Il seguente codice dimostra questo caso di utilizzo:

```
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();
```

Questo codice di esempio aggiunge una voce per *Billy*. L'attributo della versione di Person viene impostato mediante una colonna di incremento automatico nel database. L'applicazione esegue prima un comando insert. Quindi emette un flush, che provoca l'invio dell'inserimento al programma di caricamento e al database. Il database imposta la colonna della versione sul numero successivo della sequenza, il che rende l'oggetto Person della transazione obsoleto. Per aggiornare l'oggetto, l'applicazione esegue un'invalidazione globale. Il successivo metodo get emesso richiama la voce dal programma di caricamento ignorando il valore della transazione. La voce viene quindi letta dal database con il valore della versione aggiornato.

Metodo put

La sintassi del metodo put dipende dal metodo get che è stato richiamato nella transazione per la chiave. Se l'applicazione emette un'operazione get che restituisce una voce esistente nel BackingMap o sul Loader, il richiamo del metodo put viene interpretato come un aggiornamento e restituisce il valore precedente nella transazione. Un metodo put senza un precedente metodo get o con un metodo get che non trova una voce viene interpretato come un insert. La sintassi dei metodi insert e update si applicano quando viene eseguito il commit dell'operazione put. Il metodo putAll è fornito per abilitare l'elaborazione di inserimento e aggiornamento batch.

Metodo remove

Rimuove la voce dal BackingMap e dal Loader, se ne esiste uno collegato. Il valore dell'oggetto che è stato rimosso viene restituito da questo metodo. Se l'oggetto non esiste, questo metodo restituitsce un valore null. Il metodo removeAll è fornito per abilitare l'elaborazione di eliminazione batch senza i valori di restituzione.

Metodo setCopyMode

Specifica un CopyMode per questo ObjectMap. Con questo metodo, un'applicazione può sovrascrivere la CopyMode specificata in BackingMap. La modalità CopyMode specificata avrà effetto fino a che viene richiamato il metodo clearCopyMode. Entrambi i metodi sono richiamati all'esterno dei confini della transazione. Una modalità CopyMode non può essere modificata nel mezzo di una transazione.

Metodo touch

Aggiorna l'ora dell'ultimo accesso per una voce. Questo metodo non richiama il valore da BackingMap. Utilizzare il metodo nella propria transazione. Se la chiave fornita non esiste in BackingMap a causa di una invalidazione o di una rimozione, si verifica una eccezione durante l'elaborazione del commit.

Metodo update

Aggiorna esplicitamente una voce in BackingMap e nel programma di caricamento. L'utilizzo di questo metodo indica a BackingMap e la programma di caricamento che si desidera aggiornare una voce esistente.

Se si richiama questo metodo su una voce che non esiste quando viene richiamato il metodo o durante l'elaborazione del commit, viene restituita una eccezione.

Metodo getIndex
Prova a ottenere un indice specificato creato sulla BackingMap. l'indice non può essere condiviso tra thread e operazioni sulle stesse regole di Session. È necessario che sia eseguito il casting dell'indice restituito sull'interfaccia corretta, come MapIndex o MapRangeIndex o un'interfaccia di indice personalizzata.

Interfaccia JavaMap

Un'istanza JavaMap viene ottenuta da un oggetto ObjectMap. L'interfaccia JavaMap utilizza le stesse firme del metodo di ObjectMap, ma con una diversa gestione delle eccezioni. JavaMap estende l'interfaccia java.util.Map, in modo che tutte le eccezioni siano istanze della classe java.lang.RuntimeException. Poiché JavaMap estende l'interfaccia java.util.Map, è possibile utilizzare ObjectGrid con un'applicazione esistente che utilizza un'interfaccia java.util.Map per la memorizzazione nella cache degli oggetti.

Ottenimento di un'istanza JavaMap

Un'applicazione richiama un'istanza di JavaMap da un oggetto ObjectMap che utilizza il metodo ObjectMap.getJavaMap. Il seguente frammento di codice dimostra come ottenere un'istanza di JavaMap.

```
ObjectGrid objectGrid = ...;
BackingMap backingMap = objectGrid.defineMap("mapA");
Session sess = objectGrid.getSession();
ObjectMap objectMap = sess.getMap("mapA");
java.util.Map map = objectMap.getJavaMap();
JavaMap javaMap = (JavaMap) javaMap;
```

Una JavaMap viene salvato dall'ObjectMap da cui è stata ottenuta. Richiamando getJavaMap più volte mediante un determinato ObjectMap viene restituita sempre la stessa istanza di JavaMap.

Metodi supportati

L'interfaccia JavaMap supporta solo una serie di metodi sull'interfaccia java.util.Map. L'interfaccia java.util.Map supporta i seguenti metodi:

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

Tutti gli altri metodi ereditati dall'interfaccia java.util.Map provocano una eccezione `java.lang.UnsupportedOperationException`.

Parole chiave

ObjectGrid fornisce un meccanismo di invalidazione flessibile basato su parole chiave. Una *parola chiave* è un'istanza non null di qualsiasi oggetto serializzabile. È possibile associare le parole chiave alle voci BackingMap in qualsiasi mappa selezionata.

Associazione di parole chiave alle voci

Se si desidera, è possibile associare una serie di voci alle parole chiave. I metodi di ObjectMap e JavaMap che modificano le voci, tra cui i metodi `get`, `update`, `put`, `insert` e `touch`, hanno delle versioni che consentono l'associazione di una singola parola chiave a tutte le voci modificate dal metodo. Le nuove associazioni di parole chiave sono visibili solo nella transazione corrente fino a c he viene eseguito il commit della transazione. In seguito a un commit, la nuova associazione viene applicata a BackingMap e diventa visibile alle altre transazioni. Se si verifica un errore durante l'elaborazione del commit che provoca un rollback o se un utente esegue il rollback di una transazione attiva, allora viene eseguito il rollback anche per le associazioni delle parole chiave. Il seguente codice dimostra il modo in cui una nuova voce viene associata a una parola chiave.

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

In questo esempio viene inserita una nuova voce in BackingMap e viene quindi associata alla parola chiave `"New York"`. Un'applicazione che inserisce voci deve anche associare le parole chiave alle voci richiamate. Tale associazione va effettuata ogni volta che le voci vengono richiamate. Considerare il seguente codice di esempio:

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

Esso garantisce che la voce richiamata sia associata alla parola chiave `"New York"`. Un'applicazione può associare più parole chiave a una voce, ma soltanto una parola chiave per richiamo di un metodo. Per associare più parole chiave, è necessario emettere un altro richiamo del metodo, ad esempio:

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

Parole chiave predefinite

Il metodo `setDefaultKeyword` sulle interfacce ObjectMap e JavaMap fornisce un modo per associare le voci a una determinata parola chiave senza utilizzare la relativa versione della parola chiave dei metodi `get`, `insert`, `put`, `update` o `touch`. Se viene utilizzata una versione della parola chiave di un metodo, la parola chiave predefinita verrà ignorata e verrà utilizzata la parola chiave fornita.

```
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();
```

Nell'esempio precedente, la voce `Billy` viene associata alla parola chiave predefinita, "New York". La voce `Bob` non viene associata alla parola chiave predefinita in quanto è stata inviata una parola chiave esplicita al richiamo del metodo get per richiamare la voce Bob. Nessuna parola chiave è associata a "Jimmy" in quanto è stata reimpostata la parola chiave e non è stato inviato alcun

argomento esplicito al richiamo del metodo `get`.

Invalidazione delle voci con le parole chiave

L'utilizzo del metodo `invalidateUsingKeyword` con le interfacce ObjectMap e JavaMap invalida tutte le voci associate a una parola chiave nel corrispondente BackingMap. Con questo approccio, è possibile invalidare efficacemente tutte le voci relative in un'unica operazione.

```
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();
```

Nell'esempio precedente, la voce per ″Billy″ viene invalidata e pertanto non viene inserita nel BackingMap. La voce ″Bob″ non viene invalidata in quanto è stata inserita soltanto dopo il richiamo del metodo `invalidateUsingKeyword`. Il metodo `invalidateUsingKeyword` invalida le voci basate sulle associazioni alle parole chiave quando viene richiamato il metodo.

Raggruppamento di parole chiave

Le parole chiave possono essere raggruppate insieme in una relazione parent-child. Una parola chiave principale può avere più elementi secondari e una parola chiave secondaria può avere più parole chiave principali. Ad esempio, se un'applicazione utilizza le parole chiave ″Dublin″, ″Paris″, ″New York″ e ″Los Angeles″, è possibile aggiungere i seguenti raggruppamenti:

- ″USA″ groups ″New York″ and ″Los Angeles″
- ″Europe″ groups ″Dublin″ and ″Paris″
- ″World″ groups ″USA″ and ″Europe″

L'invalidazione della parola chiave ″USA″ invalida tutte le voci che sono associate alle parole chiave ″New York″ e ″Los Angeles″. L'invalidazione della parola chiave ″World″ invalida tutte le voci associate ai raggruppamenti ″USA″ e ″Europe″. Le associazioni delle parole chiave sono definite mediante il metodo `associateKeyword` sull'interfaccia ObjectGrid. L'aggiunta di parole chiave secondarie a una parola chiave primaria dopo un richiamo al metodo `invalidateUsingKeyword` non provoca l'invalidazione delle voci associate alla parola chiave secondaria. Il segunete esempio di codice definisce la serie di associazioni di parole chiave descritte:

```
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");
```

Oggetti LogElement e LogSequence

Quando un'applicazione apporta delle modifiche a una mappa durante una transazione, un oggetto LogSequence tiene traccia di tali modifiche. Se l'applicazione modifica una voce nella mappa, esiste un corrispondente elemento LogElement che fornisce i dettagli della modifica. Ai programmi di caricamento è fornito un oggetto LogSequence per una determinata mappa se un'applicazione

richiede un'eliminazione o un commit della transazione. Il programma di caricamento itera sugli elementi LogElements all'interno di LogSequence e applica ogni LogElement al backend.

Anche gli ObjectGridEventListeners registrati con un ObjectGrid utilizzano i LogSequence. A tali listener è assegnato un LogSequence per ogni mappa in una transazione di cui viene eseguito il commit. Le applicazioni possono utilizzare questi listener per attendere le modifiche di determinate voci, come ad esempio un trigger in un normale database.

In questa sezione sono descritte le interfacce di log o le classi fornite dalla struttura 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`

Interfaccia LogElement

Un LogElement rappresenta un'operazione su una voce eseguita durante una transazione. Un oggetto LogElement ha i seguenti attributi. Gli attributi più utilizzato sono gli attributi *type* e *current value*:

Attributo *type*
: Un elemento di log *type* indica il tipo di operazione che l'elemento rappresenta. L'attributo *type* può essere una delle seguenti costanti definite nell'interfaccia `LogElement`: `INSERT`, `UPDATE`, `DELETE`, `EVICT`, `FETCH` o `TOUCH`.

Attributo *undo type*
: Restituisce l'operazione che deve essere eseguita per ″annullare″ una modifica precedente apportata dalla transazione alla voce della mappa.

Attributo *current value*
: L'attributo *current value* rappresenta il nuovo valore per l'operazione `INSERT`, `UPDATE` o `FETCH`. Se l'operazione è `TOUCH`, `DELETE` o `EVICT`, il valore corrente è null. Questo valore può essere emesso su ValueProxyInfo quando viene utilizzato ValueInterface.

Attributo *CacheEntry*
: È possibile richiamare un riferimento all'oggetto CacheEntry da LogElement e utilizzare i metodi definiti sull'oggetto CacheEntry per richiamare le informazioni necessarie.

Attributo *pending state*
: Se l'attributo *pending state* è impostato su `true`, la modifica rappresentata da questo elemento di log non è stata ancora applicata al programma di caricamento. Se impostato su `false`, la modifica è stata applicata al programma di caricamento, molto probabilmente dall'operazi one di eliminazione.

Attributo *versioned value*
: Il valore della versione è un valore che può esser utilizzato per creare la versione.

Attributo *new keywords*
: La raccolta delle nuove parole chiave contiene le parole chiave che sono state associate a questa voce.

Attributo *last access time*
Rappresenta l'ora dell'ultimo accesso per la voce.

Attributi *before image / after image*
I metodi getter sono disponibili per ottenere l'immagine del valore prima o dopo l'applicazione delle modifiche alla mappa.

Interfaccia LogSequence

Nella maggior parte delle transazioni, si verificano delle operazioni su più di una voce della mappa, pertanto vengono creati più oggetti LogElement. È quindi possibile avere un oggetto che funziona come elemento composto da più oggetti LogElement. L'interfaccia LogSequence ha questo scopo e contiene una serie di oggetti LogElement. L'interfaccia LogSequence utilizza i seguenti metodi:

Metodo size
Restituisce il numero di oggetti LogElement nella sequenza specificata.

Metodo getAllChanges
Restituisce un iteratore di tutte le modifiche nella sequenza di log specificata.

Metodo getPendingChanges
Restituisce un iteratore di tutte le modifiche in sospeso. Questo metodo viene utilizzato per lo più da un programma di caricamento per applicare soltanto le modifiche in sospeso all'archivio permanente.

Metodo getChangesByKeys
Restituisce un iteratore degli oggetti LogElement che hanno la chiave di destinazione basata sul parametro di input.

Metodo getChangesByTypes
Restituisce un iteratore degli oggetti LogElement che sono del tipo di LogElement specificato.

Metodo getMapName
Restituisce il nome della mappa di backup a cui vengono applicate le modifiche. Il chiamante può utilizzare questo nome come input per il metodo `Session.getMap(string)`.

Metodo isDirty
Restituisce se questo LogSequence ha elementi LogElements che potrebbero *sporcare* una mappa. In altre parole, se LogSequence contiene qualsiasi oggetto LogElement di un tipo diverso da Fetch o Get, allora l'oggetto LogSequence è considerato ″sporco″.

Metodo isRollback
Viene restituito se questo LogSequence è stato generato per eseguire il rollback di una transazione.

Metodo getObjectGridName
Restituisce il nome dell'ObjectGrid che contiene una mappa a cui si applicano queste modifiche.

LogElement e LogSequence sono utilizzati di frequente in ObjectGrid e dai plug-in di ObjectGrid scritti dagli utenti quando le operazioni vengono propagate da un componente o un server a un altro. Ad esempio, un oggetto LogSequence può essere utilizzato dalla funzione di propagazione delle transazioni distribuite ObjectGrid per propagare le modifiche su altri server oppure può essere applicato all'archivio permanente da parte di un programma di caricamento. LogSequence è utilizzato principalmente dalle seguenti interfacce:

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

Per maggiori dettagli su tali interfacce, fare riferimento alla documentazione dell'API.

Esempio di programma di caricamento

In questa sezione viene dimostrato il modo in cui gli oggetti LogSequence e LogElement sono utilizzati in un programma di caricamento. Un *programma di caricamento* è utilizzato per caricare i dati in un archivio permanente. Il metodo `batchUpdate` dell'interfaccia del programma di caricamento utilizza LogSequence:

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

Il metodo `batchUpdate` viene richiamato quando ObjectGrid deve applicare tutte le modifiche correnti al programma di caricamento. Al programma di caricamento è assegnato un elenco di oggetti LogElement per la mappa, incapsulati in un oggetto LogSequence. L'implementazione del metodo `batchUpdate` deve iterare le modifiche e le deve applicare sul backend. Il seguente frammento di codice mostra il modo in cui un programma di caricamento utilizza un oggetto LogSequence. Il frammento itera la serie di modifiche e crea tre database JDBC batch (Java database connectivity): uno per gli inserimenti, uno per gli aggiornamento e uno per le eliminazioni:

```
public void batchUpdate(TxID tx, LogSequence sequence)
throws LoaderException
{
 // Richiama una connessione SQL da utilizzare.
 Connection conn = getConnection(tx);
try
{
  // Elabora l'elenco di modifiche e crea una serie di
  // istruzioni preparate per l'esecuzione di un aggiornamento batch,
     un inserimento o un'eliminazione
  // delle operazioni SQL. Le istruzioni sono memorizzate nella cache in stmtCache.
  Iterator iter = sequence.getPendingChanges();
 while ( iter.hasNext() )
{
   LogElement logElement = (LogElement)iter.next();
   Object key = logElement.getCacheEntry().getKey();
   Object value = logElement.getCurrentValue();
switch ( logElement.getType().getCode() )
{
   case LogElement.CODE_INSERT:
   buildBatchSQLInsert( key, value, conn );
break;
   case LogElement.CODE_UPDATE:
   buildBatchSQLUpdate( key, value, conn );
break;
   case LogElement.CODE_DELETE:
   buildBatchSQLDelete( key, conn );
break;
}
}
 // Esegue le istruzioni batch create dal loop precedente.
 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;
}
}
```

Il precedente esempio illustra la logica di livello elevato di elaborazione dell'argomento LogSequence mentre i dettagli del modo in cui viene creata un'istruzione SQL `insert`, `update` o `delete` non sono riportati. Questo esempio illustra che il metodo `getPendingChanges` viene richiamato sull'argomento LogSequence per ottenere un iteratore degli oggetti LogElement che devono essere elaborati dal programma di caricamento e il metodo `LogElement.getType().getCode()` viene utilizzato per determinare se un LogElement è per una operazione SQL `insert`, `update` o `delete`.

Esempio di programma di esclusione

In questo esempio viene riportato il modo in cui vengono utilizzati LogSequence e LogElement in un programma di esclusione. Un programma di esclusione viene utilizzato per eliminare le voci da una mappa di backup in base a determinati criteri. Il metodo apply dell'interfaccia del programma di esclusione utilizza LogSequence:

```
/**
* Esso viene richiamato durante il commit della cache per consentire al programma
* di esclusione di tenere traccia dell'utilizzo dell'oggetto
* in una mappa di backup. Verranno quindi riportate tutte le voci che sono state
* correttamente escluse.
*
* @param sequence LogSequence delle modifiche alla mappa
*/
void apply(LogSequence sequence);
```

Per ulteriori informazioni sul modo in cui il metodo apply utilizza LogSequence, fare riferimento all'esempio di codice riportato nella sezione "Programmi di esclusione" a pagina 187.

Interfacce LogSequenceFilter e LogSequenceTransformer

A volte, è necessario filtrare gli oggetti LogElement in modo che vengano accettati solo gli oggetti LogElement con determinati criteri, mentre gli altri verranno rifiutati. Ad esempio, si assuma che si desideri serializzare un determinato LogElement in base a certi criteri. LogSequenceFilter risolve questo problema con il seguente metodo:

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

Questo metodo restituisce un valore true se nell'operazione deve essere utilizzato il determinato LogElement e restituisce un elemento false se LogElement non deve essere utilizzato.

LogSequenceTransformer è una classe che utilizza la funzione LogSequenceFilter descritta in precedenza. Essa utilizza LogSequenceFilter per filtrare alcuni oggetti LogElement e serializzare quindi gli oggetti LogElement accettati. Questa classe ha due metodi. Di seguito è riportato il primo metodo:

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

Questo metodo consente al chiamante di fornire un filtro per determinare gli elementi LogElements da includere nel processo di serializzazione. Il parametro **DistributionMode** consente al chiamante di controllare il processo di serializzazione. Ad esempio, se la modalità di distribuzione è solo di invalidazione, allora non è necessario serializzare il valore. Il secondo metodo di questa classe è:

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

Questo metodo legge il modulo serializzato della sequenza di log, creata dal metodo `serialize` dal flusso di input degli oggetti fornito.

Blocco

In questa sezione viene descritta la strategia di blocco supportata da un ObjectGrid BackingMap.

Ogni BackingMap può essere configurata in modo da utilizzare una delle seguenti strategie di blocco:

- Blocco pessimistico
- Blocco ottimistico
- None

Di seguito è riportato un esempio di come è possibile impostare la strategia di blocco sulle BackingMap `map1`, `map2` e `map3`, dove ciascuna mappa utilizza una diversa strategia di blocco:

```
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 );
```

Per evitare un'eccezione `java.lang.IllegalStateException`, è necessario richiamare il metodo `setLockStrategy` prima di richiamare i metodi `initialize` o `getSession` sull'istanza di ObjectGrid.

Quando viene utilizzata una strategia di blocco `PESSIMISTIC` o `OPTIMISTIC`, viene creato un gestore blocchi per il BackingMap. Tale gestore utilizza una mappa hash per tenere traccia delle voci che vengono bloccate da una o più transazioni. Se sono presenti numerose voci nella mappa hash, allora un numero elevato di contenitori di blocchi implica migliori prestazioni. Il rischio di collisioni della sincronizzazione Java è meno probabile se aumenta il numero di contenitori. Un numero elevato di contenitori di blocco implica anche una maggiore simultaneità. Nel seguente esempio viene riportato il modo in cui un'applicazione imposta il numero di contenitori di blocchi da utilizzare per un determinato BackingMap:

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

Allo stesso modo, per evitare un'eccezione `java.lang.IllegalStateException`, è necessario richiamare il metodo `setNumberOfLockBuckets` prima di richiamare i metodi `initialize` o `getSession` sull'istanza di ObjectGrid. Il parametro del metodo `setNumberOfLockBuckets` è un numero intero primitivo Java che specifica il numero di contenitori di blocchi da utilizzare. L'utilizzo di un numero primo garantisce una distribuzione uniforme delle voci della mappa su tutti i contenitori. Un buon punto di partenza per delle prestazioni elevate è l'impostazione del numero di contenitori di blocchi su circa il 10% del numero previsto di voci di BackingMap.

Blocco pessimistico

Utilizzare la strategia di blocco pessimistico per leggere e scrivere mappe quando altre strategie di blocco non sono utili.

Quando un ObjectGrid Map è configurato per utilizzare una strategia di blocco PESSIMISTIC, un blocco pessimistico delle transazioni per una voce della mappa viene ottenuto quando la transazione richiama per la prima volta la voce da BackingMap. Il blocco pessimistico viene mantenuto fino a che l'applicazione completa la transazione. Di solito, una strategia di blocco pessimistico viene utilizzata nelle seguenti situazioni:

- Un BackingMap viene configurato con o senza un programma di caricamento e non sono disponibili le informazioni sulla versione.
- Il BackingMap viene utilizzato direttamente da un'applicazione che richiede assistenza da parte dell'ObjectGrid per il controllo della simultaneità.
- Le informazioni sulla versione sono disponibili, ma le transazioni degli aggiornamenti collidono frequentemente, provocando degli errori negli aggiornamenti ottimistici.

Poiché la strategia di blocco pessimistico ha il maggiore impatto sulle prestazioni e sulla scalabilità, questa strategia deve essere utilizzata soltanto per la lettura e la scrittura di mappe quando le altre strategie non sono utili. Ad esempio, nel caso in cui gli errori degli aggiornamenti ottimistici di verificano frequentemente oppure il ripristino da un errore ottimistico è difficile da gestire.

Metodi ObjectMap e modalità di blocco

Quando un'applicazione utilizza i metodi dell'interfaccia ObjectMap, ObjectGrid prova automaticamente ad acquisire un blocco pessimistico per la voce della mappa a cui si sta eseguendo l'accesso. ObjectGrid utilizza le seguenti modalità di blocco uin base a quali metodi vengono richiamati dall'applicazione sull'interfaccia ObjectMap:

- I metodi `get` e `getAll` acquisiscono una modalità di *blocco S*, o blocco condiviso, per la chiave della voce di una mappa. Il blocco S viene mantenuto fino a che la transazione viene completata. Una modalità di blocco S consente una simultaneità tra le transazioni ma provano ad acquisire una modalità di blocco S o di blocco aggiornabile (blocco U) per la stessa chiave, ma blocca le altre transazioni che invece provano ad ottenere un blocco esclusivo (blocco X) per la chiave.
- I metodi `getForUpdate` e `getAllForUpdate` acquisiscono una modalità di *blocco U*, o aggiornabile, per la chiave della voce di una mappa. Il blocco U viene mantenuto fino a che la transazione viene completata. Una modalità di blocco U consente una simultaneità tra le transazioni che acquisiscono una modalità di blocco S per la stessa chiave, ma blocca le altre transazioni che invece provano ad ottenere un blocco U o un blocco X.

- I metodi `put`, `putAll`, `remove`, `removeAll`, `insert`, `update` e `touch` acquisiscono un *blocco X*, o esclusivo, per la chiave della voce di una mappa. Il blocco X viene mantenuto fino a che la transazione viene completata. Una modalità di blocco X garantisce che solo una transazione inserisca, aggiorni o rimuova la voce di una mappa di un determinato valore della chiave. Un blocco X blocca tutte le altre transazioni che provano ad acquisire una modalità di blocco S, U o X per la stessa chiave.
- I metodi `global invalidate` e `global invalidateAll` acquisiscono un blocco X per ogni voce della mappa che viene invalidata. Il blocco X viene mantenuto fino a che la transazione viene completata. Nessun blocco viene acquisito per i metodi `local invalidate` e `local invalidateAll` in quanto nessuna delle voci BackingMap viene invalidata dalle chiamate al metodo invalidate locale.

Dalle definizioni precedenti, risulta chiaro che una modalità di blocco S è più debole di una modalità di blocco U in quanto consente l'esecuzione simultanea di più transazioni quando accedono alla stessa voce della mappa. La modalità di blocco U è molto più forte in quanto blocca le altre transazioni che richiedono una modalità di blocco U o X. La modalità di blocco S blocca soltanto le transazioni che richiedono una modalità di blocco X. Questa piccola differenza è molto importante nell'impedire che si verifichino delle condizioni di stallo. La modalità di blocco X è la modalità di blocco più forte in quanto blocca tutte le altre transazioni che provano a ottenere una modalità di blocco S, U o X per la stessa voce della mappa. L'interesse di una modalità di blocco X sta nel garantire che soltanto una transazione possa inserire, aggiornare o rimuovere una voce della mappa e nell'impedire la perdita di aggiornamenti quando più di una transazione prova ad aggiornare la stessa voce.

La seguente tabella è una matrice di compatibilità tra le modalità di blocco che riepiloga le modalità di blocco descritte e viene utilizzata per determinare quali modalità di blocco sono compatibili tra loro. Per leggere tale matrice, è necessario tenere presente che la riga della matrice indica una modalità di blocco che è già concessa. La colonna indica la modalità di blocco richiesta da un'altra transazione. Se nella colonna viene visualizzato **Sì**, la modalità di blocco richiesta dall'altra transazione viene concessa in quanto è compatibile con la modalità già concessa. **No** indica invece che la modalità di blocco non è compatibile e l'altra transazione deve attendere che la prima transazione rilasci il blocco da lei definito.

Tabella 8. Compatibilità e forza delle modalità di blocco

blocco	**blocchi compatibili**			**forza**
	S (shared, condiviso)	U (upgradeable, aggiornabile)	X (exclusive, esclusivo)	
S (Shared)	Sì	Sì	No	più debole
U (Upgradeable)	Sì	No	No	normale
X (Exclusive)	No	No	No	più forte

Timeout di attesa del blocco

Ogni ObjectGrid BackingMap ha un valore di timeout di attesa per il blocco predefinito. Il valore di timeout viene utilizzato per garantire che un'applicazione non debba attendere troppo a lungo perché sia concessa una modalità di blocco a causa delle condizioni di stallo che si verificano per colpa di un errore delle applicazioni. L'applicazione può utilizzare l'interfaccia BackingMap per sovrascrivere il valore di timeout di attesa del blocco predefinito. Il seguente esempio illustra come impostare il valore di timeout di attesa del blocco per map1 su 60 secondi:

```
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 );
```

Per evitare un'eccezione `java.lang.IllegalStateException`, richiamare entrambi i metodi setLockStrategy e setLockTimeout prima di richiamare i metodi initialize o getSession sull'istanza di ObjectGrid. Il parametro del metodo setLockTimeout è un numero intero di una primitiva Java che specifica il numero di secondi che ObjectGrid deve attendere perché venga concessa una modalità di blocco. Se una transazione attende più del valore di timeout configurato per il BackingMap, ObjectGrid emette un'eccezione `com.ibm.websphere.objectgrid.LockTimeoutException`.

Quando si verifica un'eccezione LockTimeoutException, l'applicazione deve determinare se il timeout si verifica a causa dell'applicazione che viene eseguita più lentamente di quanto dovuto o se il timeout si è verificato a causa di una condizione di stallo. In quest ultimo caso, allora l'aumento del valore di timeout non elimina l'eccezione. L'aumento del valore di timeout provoca l'emissione di una eccezione ancora più lunga. Tuttavia, se l'aumento del valore di timeout non elimina l'eccezione, allora il problema si è verificato a causa del fatto che l'applicazione è in esecuzione più lentamente del previsto. L'applicazione in questo caso deve determinare il motivo per cui le prestazioni sono ridotte. Per ulteriori informazioni, consultare Capitolo 14, "Risoluzione dei problemi", a pagina 345 e Capitolo 11, "Procedure ottimali delle prestazioni di ObjectGrid", a pagina 325.

Condizioni di stallo

Considerare la sequenza di richieste di modalità di blocco riportata:

```
Un blocco X viene concesso alla transazione 1 per key1.
Un blocco X viene concesso alla transazione 2 per key2.
Un blocco X viene richiesto dalla transazione 1 per key2.
 (La transazione 1 blocca l'attesa del blocco di proprietà della transazione 2.)
Un blocco X viene richiesto dalla transazione 2 per key1.
 (La transazione 2 blocca l'attesa del blocco di proprietà della transazione 1.)
```

La sequenza precedente è un classico esempio di condizione di stallo di due transazioni che provano ad acquisire più di un blocco e ogni transazione acquisisce i blocchi in diverse modalità. Per evitare questa condizione di stallo, ogni transazione deve ottenere più blocchi nello stesso ordine. Se viene utilizzata la strategia di blocco OPTIMISTIC e il metodo flush sull'interfaccia ObjectMap non viene mai utilizzato dall'applicazione, allora le modalità di blocco vengono richieste dalla transazione soltanto durante il ciclo di commit. Durante questo ciclo, l'ObjectGrid determina le chiavi per le voci della mappa che devono essere bloccate e richiede le modalità di blocco nella sequenza delle chiavi. Con questo metodo, ObjectGrid impedisce gran parte delle condizioni di stallo classiche. Tuttavia, ObjectGrid non previene tutte le condizioni di stallo possibili. Esistono un paio di scenari che devono essere considerati. Di seguito sono riportati gli scenari che un'applicazione deve considerare in modo da poter intraprendere un'azione preventiva.

Esiste uno scenario in cui ObjectGrid è in grado di rilevare una condizione di stallo senza dover attendere che si raggiunga il valore di timeout. In questo caso, viene restituitaun'eccezione `com.ibm.websphere.objectgrid.LockDeadlockException`. Considerare il seguente frammento di codice:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Billy");
// È il compleanno di Billy, che pertanto è di un anno più vecchio.
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
sess.commit();
```

In questa situazione, la moglie di Billy lo voglia far diventare più vecchio di quello che è, quindi Billy e sua moglie eseguono la transazione contemporaneamente. In questa situazione, entrambe le transazioni hanno una modalità di blocco S sulla voce **Billy** della mappa PERSON come risultato del richiamo del metodo person.get("Billy"). Come risultato del richiamo al metodo person.put ("Billy", p), entrambe le transazioni provano ad aggiornare la modalità di blocco S a una modalità di blocco X. Entrambe le transazioni vengono bloccate in attesa che l'altra transazione rilasci il proprio blocco S. Come risultato, si verifica una condizione di stallo in quanto esiste una condizione di attesa circolare tra le due transazioni. Una condizione di attesa circolare si verifica quando più di una transazione prova a passare da una modalità di blocco più debole a una più forte per la stessa voce. In questo scenario, l'ObjectGrid emette una eccezione `LockDeadlockException` piuttosto che un'eccezione `LockTimeoutException`. Per ulteriori informazioni, consultare "LockDeadlockException" a pagina 349.

L'applicazione può impedire che venga emessa l'eccezione `LockDeadlockException` per l'esempio precedente utilizzando la strategia di blocco OPTIMISTIC piuttosto che la strategia di blocco PESSIMISTIC. L'utilizzo della strategia di blocco OPTIMISTIC rappresenta la soluzione preferita quando la mappa viene principalmente letta e gli aggiornamenti sono molto poco frequenti. Fare riferimento a "Blocco ottimistico" a pagina 133 per maggiori dettagli sulla strategia ottimistica. Se deve essere utilizzata la strategia di blocco PESSIMISTIC, è possibile utilizzare il metodo getForUpdate invece del metodo get. In questo modo, la prima transazione a richiamare il metodo getForUpdate acquisisce una modalità di blocco U invece che di blocco S. Questa modalità di blocco implica che la seconda transazione viene bloccata quando richiama il metodo getForUpdate in quanto la modalità di blocco U viene concessa soltanto a una transazione alla volta. Poiché la seconda transazione è bloccata, questa non avrà alcun blocco sulla voce della mappa Billy. La prima transazione non viene bloccata fino a che prova ad aggiornare la modalità di blocco U su una modalità di blocco X come risultato della chiamata al metodo put dalla prima transazione. Questa funzione mostra il motivo per cui la modalità di blocco U è detta modalità di blocco ″aggiornabile″ (upgradeable). Una volta completata la prima transazione, la seconda transazione viene sbloccata e le viene concessa la modalità di blocco U. Un'applicazione può impedire che si verifichi questo scenario della condizione di stallo dovuta al passaggio a un altro blocco utilizzando il metodo getForUpdate invece del metodo get quando viene utilizzata la strategia di blocco PESSIMISTIC.

Importante: Questa soluzione non impedisce che le transazioni di sola lettura possono leggere una voce della mappa. Le transazioni di sola lettura richiamano il metodo get, ma non richiamano mai i metodi put, insert, update o remove. Il livello di simultaneità è elevato quando viene utilizzato il metodo get. L'unica riduzione nella simultaneità si verifica

quando il metodo getForUpdate viene richiamato da più di una transazione per la stessa voce della mappa.

È necessario prestare attenzione quando una transazione richiama il metodo getForUpdatesu più di una voce della mappa in modo da essere certi che i blocchi U vengano acquisiti nello stesso ordine da ogni transazione. Ad esempio, si assuma che la prima transazione richiami il metodo getForUpdate per la chiave 1 e il metodo getForUpdate per la chiave 2. Un'altra transazione simultanea richiama il metodo getForUpdate per le stesse chiavi, ma in ordine inverso. Questa sequenza provoca il verificarsi della condizione di stallo in quanto sono ottenuti più blocchi da diverse transazioni. L'applicazione deve essere certa che ogni transazione che accede a più voci della mappa lo faccia in sequenza in modo che non si verifichino le condizioni di stallo. Poiché il blocco U è ottenuto nel momento in cui viene richiamato il metodo getForUpdate piuttosto che in fase di commit, ObjectGrid non può ordinare le richieste di blocco come farebbe durante il ciclo del commit. L'applicazione in questo caso deve controllare l'ordine dei blocchi.

L'utilizzo del metodo flush sull'interfaccia ObjectMap prima di un commit consente di prendere in considerazione altri fattori importanti per l'ordinamento dei blocchi. Il metodo flush è di solito utilizzato per forzare le modifiche apportate alla mappa sul backend mediante il plug-in Loader. In questa situazione, il backend utilizza il proprio gestore blocchi per controllare la simultaneità, pertanto la condizione di attesa del blocco e la condizione di stallo possono verificarsi sul backend piuttosto che sul gestore blocchi di ObjectGrid. Si consideri la seguente transazione:

```
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();
}
```

Si supponga che un'altra transazione abbia aggiornato la voce Tom, che abbia richiamato il metodo flush e che quindi abbia aggiornato la voce Billy. In questo caso, la seguente interlacciatura delle due transazioni provoca una condizione di stallo nel database:

```
Un blocco X viene concesso alla transazione 1 per "Billy"
quando viene eseguito il metodo flush.
Un blocco X viene concesso alla transazione 2 per "Tom"
quando viene eseguito il metodo flush.
Un blocco X viene richiesto dalla transazione 1 per "Tom"
durante l'elaborazione del commit.
 (La transazione 1 blocca l'attesa del blocco di proprietà della transazione 2.)
Un blocco X viene richiesto dalla transazione 2 per "Billy"
durante l'elaborazione del commit.
 (La transazione 2 blocca l'attesa del blocco di proprietà della transazione 1.)
```

Questo esempio dimostra che l'uso del metodo flush può far sì che si verifichi una condizione di stallo nel database piuttosto che in ObjectGrid. Questa condizione di stallo può verificarsi indipendentemente dalla strategia di blocco utilizzata. L'applicazione deve poter impedire questo tipo di condizioni di stallo quando si utilizza il metodo flush e quando un programma di caricamento è collegato a BackingMap. L'esempio precedente illustra inoltre un altro motivo per cui ObjectGrid utilizza un meccanismo di timeout di attesa del blocco. Una transazione che attende un blocco del database potrebbe essere in attesa mentre possiede un blocco sulla voce della mappa ObjectGrid. Di conseguenza, i problemi a livello di database possono provocare tempi di attesa eccessivi per una modalità di blocco di ObjectGrid e restituire quindi un'eccezione `LockTimeoutException`.

Gestione delle eccezioni

Gli esempi riportati in questa sezione non comprendono alcuna gestione delle eccezioni. Per evitare che i blocchi vengano mantenuti per troppo tempo quando si verifica un'eccezione `LockTimeoutException` o `LockDeadlockException`, un'applicazione deve verificare che vengano rilevate le eccezioni non previste e richiami il metodo rollback quando si verifica una condizione imprevista. Modificare il seguente frammento di codice come dimostrato nell'esempio seguente:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
sess.begin();
 activeTran = true;
Person p = (IPerson)person.get("Billy");
// È il compleanno di Billy, che pertanto è di un anno più vecchio.
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
sess.commit();
 activeTran = false;
}
finally
{
 if ( activeTran ) sess.rollback();
}
```

Il blocco `finally` nel frammento di codice garantisce che venga eseguito il rollback di una transazione quando si verifica una eccezione non prevista. Non solo gestisce un'eccezione `LockDeadlockException`, ma qualsiasi altra eccezione che si può verificare. Il blocco `finally` gestisce il caso in cui si verifica una eccezione durante un richiamo al metodo commit. Questo esempio non è l'unico che tratta di eccezioni non previste e potrebbero verificarsi dei casi in cui un'applicazione desideri rilevare le eccezioni impreviste emesse ed emetterne una propria. È possibile aggiungere blocchi catch in base alle necessità, ma l'applicazione deve garantire che il frammento di codice non venga interrotto prima del completamento della transazione.

Blocco ottimistico

La strategia di blocco ottimistico consiste nel credere che due transazioni non possono provare ad aggiornare la stessa voce della mappa se sono in esecuzione simultaneamente. Per questo motivo, non è necessario mantenere una modalità di blocco per l'intera durata della transazione in quanto è molto poco probabile che più di una transazione possa aggiornare la voce della mappa nello stesso momento.

La strategia di blocco ottimistico viene di solito utilizzata se:

- Un BackingMap è configurato con o senza un programma di caricamento e sono disponibili le informazioni sulla versione.
- Un BackingMap è principalmente di lettura. In altre parole, le transazioni leggono di frequente le voci della mappa e solo occasionalmente ne inseriscono, aggiornano o rimuovono una.
- Un BackingMap viene inserito, aggiornato o rimosso più frequentemente di quanto venga letto, ma le transazioni difficilmente collidono sulla stessa voce della mappa.

Come per la strategia di blocco pessimistico, i metodi sull'interfaccia ObjectMap determinano il modo in cui ObjectGrid prova ad acquisire automaticamente una modalità di blocco per la voce della mappa a cui si esegue l'accesso. Tuttavia, di seguito sono riportate alcune differenze importanti tra le due strategie:

- Come la strategia di blocco pessimistico, una *modalità di blocco a S* viene acquisita dai metodi `get` e `getAll` quando viene richiamato il metodo. Tuttavia, con il blocco ottimistico, la modalità di blocco a S non viene mantenuta fino a che la transazione viene completata. La modalità di blocco a S viene invece rilasciata prima che il metodo venga restituito all'applicazione. Lo scopo di acquisizione della modalità di blocco è che in questo mdoo ObjectGrid può garantire che solo i dati di cui è stato eseguito il commit da altre transazioni siano visibili per la transazione corrente. Una volta che ObjectGrid ha verificato il commit dei dati, la modalità di blocco a S viene rilasciata. In fase di commit, viene eseguito un controllo della versione ottimistico per essere certi che nessuna altra transazione abbia modificato la voce della mappa dopo che la transazione corrente ha rilasciato la modalità di blocco a S. Se una voce non viene utilizzata dalla mappa prima che venga aggiornata, invalidata o eliminata, allora il runtime ObjectGrid utilizza implicitamente tale voce. Questa operazione get implicita viene eseguita per ottenere il valore corrente nel momento in cui viene richiesta la modifica della voce.
- A differenza della strategia di blocco pessimistica, i metodi `getForUpdate` e `getAllForUpdate` vengono gestiti esattamente come i metodi `get` e `getAll` quando viene utilizzata la strategia di blocco ottimistico. In altre parole, una modalità di blocco a S viene acquisita all'inizio del metodo e viene rilasciata prima di tornare all'applicazione.
- Tutti gli altri metodi di ObjectMap vengono gestiti esattamente come per la strategia di blocco pessimistico. Ovvero, quando viene richiamato un metodo per il commit, una modalità di blocco a X viene ottenuta per ogni voce della mappa che viene inserita, aggiornata, rimossa o invalidata e tale modalità viene conservata fino a che mode la transazione completa l'elaborazione del commit.

Questa strategia di blocco è detta ottimistica in quanto si basa su un punto di vista ottimistico. La strategia di blocco ottimistico consiste nel credere che due transazioni non possono provare ad aggiornare la stessa voce della mappa se sono in esecuzione simultaneamente. Per questo motivo, non è necessario mantenere una modalità di blocco per l'intera durata della transazione in quanto è molto poco probabile che più di una transazione possa aggiornare la voce della mappa nello stesso momento. Tuttavia, poichè non è stata mantenuta alcuna modalità di blocco, una transazione simultanea potrebbe aggiornare potenzialmente la voce della mappa in seguito al rilascio della modalità di blocco a S da parte della transazione corrente. Per gestire questa possibilità, ObjectGrid richiama un blocco a X in fase di commit ed esegue un controllo della versione ottimistico per verificare che nessun'altra transazione abbia modificato la voce della mappa dopo la lettura della voce da BackingMap da parte della transazione corrente. Se un'altra transazione modifica la voce della mappa, il controllo della versione riporta un errore e si verifica una eccezione `OptimisticCollisionException`. Questa eccezione forza il

rollback della transazione corrente e l'applicazione deve richiamare l'intera transazione. La strategia di blocco ottimistico è molto utile quando una mappa viene principalmente letta ed è molto improbabile che si verifichino degli aggiornamenti per la stessa voce della mappa.

Strategia di blocco BackingMap None

Quando un BackingMap viene configurato per utilizzare una strategia di blocco NONE, non viene ottenuto alcun blocco delle transazioni per una voce della mappa. Uno scenario in cui questa strategia è particolarmente utile è dato da un'applicazione che è un gestore permanente, come ad esempio un contenitore EJB (Enterprise JavaBeans) Java 2 Platform, Enterprise Edition (J2EE) o da un'applicazione che utilizza Hibernate per ottenere i dati permanenti. In questo scenario, il BackingMap è configurato senza un programma di caricamento e viene utilizzato come cache dei dati dal gestore permanente. Il gestore in questo scenario fornisce il controllo della simultaneità tra le transazioni che accedono alle stesse voci della mappa di ObjectGrid. L'ObjectGrid non deve ottenere alcun blocco delle transazioni per il controllo della simultaneità. Ciò assume che il gestore permanente non rilasci i blocchi delle transazioni prima di aver aggiornato la mappa ObjectGrid con le modifiche salvate. Se questo non è il caso, allora è necessario utilizzare una strategia di blocco PESSIMISTIC o OPTIMISTIC. Ad esempio, si assuma che il gestore permanente di un contenitore EJB aggiorni la mappa ObjectGrid con i dati salvati nella transazione gestita dal contenitore EJB. Se l'aggiornamento della mappa ObjectGrid si verifica prima che il gestore rilasci i blocchi delle transazioni, allora è possibile utilizzare la strategia di blocco NONE. Se l'aggiornamento della mappa theObjectGrid si verifica una volta che i blocchi del gestore permanente siano rilasciati, allora è richiesta la strategia di blocco OPTIMISTIC o PESSIMISTIC.

Un altro scenario in cui è possibile utilizzare la strategia NONE è quando l'applicazione utilizza un BackingMap direttamente e un programma di caricamento è configurato per la mappa. In questo scenario, il programma di caricamento utilizza il supporto di controllo della simultaneità fornito da RDBMS (relational database management system) utilizzando Java database connectivity (JDBC) o Hibernate per accedere ai dati in un database relazionale. L'implementazione del programma di caricamento può utilizzare sia un approccio ottimistico che uno pessimistico.

Un programma di caricamento che utilizza un blocco ottimistico o un approccio di determinazione della versione consente di ottenere una maggiore simultaneità e migliori prestazioni. Per ulteriori informazioni sull'implementazione di una strategia di blocco ottimistico, fare riferimento alla sezione "OptimisticCallback" a pagina 207 in "Considerazioni sul programma di caricamento" a pagina 203.

Un programma di caricamento che utilizza il supporto di un approccio pessimistico del backend potrebbe aver bisogno di utilizzare il parametro **forUpdate** inviato al metodo get sull'interfaccia del Loader. Questo parametro è impostato su *true* se l'applicazione ha utilizzato il metodo `getForUpdate` dell'interfaccia ObjectMap per richiamare i dati. Il programma di caricamento può utilizzare questo parametro per determinare se richiedere un blocco aggiornabile sulla riga letta. Ad esempio, DB2 ottiene un blocco aggiornabile quando una istruzione SQL select contiene una clausola `for update`. Questo approccio offre la stessa possibilità di impedire una condizione di stallo descritta nella sezione Blocco pessimistico.

Sicurezza di ObjectGrid

Utilizzare i meccanismi di sicurezza ObjectGrid per proteggere l'accesso ai dati delle mappe e alle attività di gestione mediante un metodo di configurazione o uno di programmazione.

ObjectGrid fornisce due meccanismi di sicurezza per per proteggere l'accesso ai dati di una mappa e le attività d i gestione. La sicurezza ObjectGrid si basa sul meccanismo Java Authentication and Authorization Services (JAAS). JAAS è parte integrante di Java 2 Security.

In questa sezione vengono descritti i meccanismi di sicurezza ObjectGrid e viene riportato come utilizzare le API di sicurezza di ObjectGrid.

- "Panoramica sulla sicurezza di ObjectGrid" fornisce una panoramica della sicurezza ObjectGrid.
- "Sicurezza server client" a pagina 141 descrive la sicurezza server client per il modello di programmazione ObjectGrid distribuito.
- "Sicurezza ObjectGrid locale" a pagina 160 descrive la sicurezza ObjectGrid locale.
- "Autorizzazione" a pagina 166 descrive il meccanismo di autorizzazione e i plug-in correlati che si applicano al modello di programmazione ObjectGrid distribuito e locale.
- "Sicurezza del cluster ObjectGrid" a pagina 176 descrive i meccanismi di sicurezza del cluster ObjectGrid e i relativi plug-in.
- "Sicurezza gateway" a pagina 179 descrive la sicurezza del gateway.
- "Integrazione della sicurezza con WebSphere Application Server" a pagina 181 evidenzia l'integrazione con WebSphere Application Server.

La maggior parte del codice di esempio riportato in questa sezione proviene da esempi di ObjectGrid. Una panoramica sugli esempi di sicurezza è riportata in Capitolo 5, "Esempi di ObjectGrid", a pagina 59.

Panoramica sulla sicurezza di ObjectGrid

ObjectGrid è un sistema di memorizzazione nella cache distribuito. L'accesso ai dati nella cache può essere protetto. In generale, la sicurezza è basata su tre concetti chiave:

- *Autenticazione attendibile*: stabilisce in modo sicuro l'identità del richiedente.
- *Autorizzazione*: concedere diritti di accesso al richiedente con autorizzazioni.
- *Trasporto sicuro*: trasmissione sicura di dati sulle reti.

ObjectGrid fornisce sicurezza sui seguenti aspetti:

- "Sicurezza del server client" a pagina 137 risolve l'autenticazione e la sicurezza di comunicazione del server client utilizzando SSL(Secure Sockets Layer).
- "Autorizzazione" a pagina 137 i meccanismi garantiscono che solo i client autorizzati possono accedere ai dati della mappaObjectGrid e alle attività di gestione.
- "Sicurezza del cluster ObjectGrid" a pagina 137 verifica che solo i server autorizzati possono unirsi al cluster ObjectGrid.
- "Sicurezza Gateway" a pagina 138 risolve l'autenticazione del client gateway.
- "Sicurezza ObjectGrid locale" a pagina 138 fornisce il meccanismo di sicurezza quando l'applicazione istanzia direttamente l'istanza ObjectGrid.

La sicurezza ObjectGrid è costruita su un'architettura aperta e fornisce diversi punti di plug-in per la personalizzazione. Il meccanismo di plug-in gioca un ruolo importante. ObjectGrid fornisce inoltre un'implementazione integrata per questi plug-in. Alcune implementazioni sono per un uso con produzione predefinita e altre sono per la prova o per scopi generici. Fare riferimento a "Plug-in di sicurezza" a pagina 138 per un riepilogo dei plug-in e delle implementazioni integrate.

Sicurezza del server client

ObjectGrid supporta la struttura server client distribuita. Un'infrastruttura di sicurezza server client server a proteggere l'accesso ai server ObjectGrid.

Un client ObjectGrid può utilizzare la credenziale desiderata per l'autenticazione del server ObjectGrid. Un contratto deve essere stabilito tra i client e i server in modo che tale credenziale sia intesa dal meccanismo di autenticazione del server. Quando SSL (Secure Sockets Layer) viene utilizzato, il client può inoltre utilizzare i certificati SSL per l'autenticazione nel server ObjectGrid.

Per proteggere la comunicazione del server client, ObjectGrid supporta SSL. Il protocollo SSL fornisce la sicurezza a livello di trasporto con autenticità, integrità e riservatezza, per un collegamento protetto tra un client e un server ObjectGrid. Alcune delle funzioni di sicurezza che sono fornite da SSL sono: crittografia dei dati per evitare l'esposizione di informazioni sensibili durante il flusso dei dati, la firma dei dati per impedire la modifica non autorizzata dei dati durante il flusso dei dati e l'autenticazione client e server per garantire di parlare con la persona o con la macchina appropriata. SSL può essere efficace nella protezione di ambiente aziendale.

Per ulteriori informazioni, consultare "Sicurezza server client" a pagina 141.

Autorizzazione

Le autorizzazioni ObjectGrid sono basate sugli oggetti e le autorizzazioni. In ObjectGrid, esistono due categorie di autorizzazioni: autorizzazioni per accesso ai dati e autorizzazioni per le attività di gestione. È possibile utilizzare JAAS (Java Authentication and Authorization Services) per consentire l'accesso o l'inserimento nei meccanismi per la gestione delle autorizzazioni.

Per ulteriori informazioni, consultare "Autorizzazione" a pagina 166.

Sicurezza del cluster ObjectGrid

In un ambiente protetto, un server deve essere in grado di verificare l'autenticità di un altro server. ObjectGrid utilizza un meccanismo di stringa con chiave segreta condivisa per questo scopo. Questo meccanismo di chiave segreta è simile a una password condivisa. Tutti i server ObjectGrid sono d'accordo su un segreto condiviso. Quando un server entra in un cluster, viene chiesto di presentare la stringa segreta. Se la stringa segreta del server corrisponde a quella nel server principale, il server può entrare nel cluster; diversamente la richiesta di entrata viene rifiutata.

L'invio di un testo segreto chiaro non è protetto. L'infrastruttura di sicurezza ObjectGrid fornisce un plug-in SecureTokenManager per consentire al server di "proteggere" questo segreto prima dell'invio. La modalità di implementazione

dell'operazione "protetta" è aperta. ObjectGrid fornisce un'implementazione predefinita, nella quale il funzionamento "protetto" viene implementato per la codifica e la firma del segreto.

Fare riferimento a "Sicurezza del cluster ObjectGrid" a pagina 176 per maggiori informazioni

Sicurezza Gateway

Un gateway ObjectGrid serve come un punto di delega delle richieste di gestione del client al server ObjectGrid. Il gateway di gestione presenta una serie di mbeans. Il client gateway invoca tali mbean per la gestione o il controllo dei server ObjectGrid.

Il gateway di gestione e la comunicazione del server utilizza il meccanismo di comunicazione del server client ObjectGrid nel quale il gateway viene trattato come un client ObjectGrid. Il client gateway e la comunicazione gateway (MBean server) possono essere protette da SSL. Questa capacità viene fornita dal livello di connettore JMX, che viene implementata da un progetto open source mx4j. ObjectGrid richiede mx4j in posizione per far funzionare il gateway.

Per l'autenticazione, il gateway propaga la credenziale presentata dal client gateway al server ObjectGrid. Sia l'autenticazione che l'autorizzazione sono eseguite sui server ObjectGrid.

Per ulteriori informazioni, consultare "Sicurezza gateway" a pagina 179.

Sicurezza ObjectGrid locale

In WebSphere Extended Deployment Server versione 6.0, il modello di programmazione ObjectGrid locale è stato introdotto. In questo modello, l'applicazione istanzia e utilizza direttamente un'istanza ObjectGrid. L'applicazione e le istanzw ObjectGrid sono nella stessa JVM (Java virtual machine). Nessun concetto di client o server esiste in questo modello.

L'autenticazione non è supportata nel modello di programmazione ObjectGrid. Le applicazioni devono gestire la popria autenticazione e quindi passare l'oggetto Soggetto in ObjectGrid.

Lo stesso meccanismo di autorizzazione viene utilizzato per il modello di programmazione ObjectGrid locale come quello utilizzato per il modello del server client.

Per ulteriori informazioni, consultare "Sicurezza ObjectGrid locale" a pagina 160.

Plug-in di sicurezza

La struttura di sicurezza ObjectGrid viene supplementata dai plug-in di sicurezza. Tali plug-in possono essere implementati per estendere o personalizzare la struttura di sicurezza.

Un esempio è il plug-in CredentialGenerator, rappresentato dall'interfaccia com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator. Quando un client ObjectGrid si collega a un server ObjectGrid, il metodo getCredential() di questo plug-in viene chiamato per generare un oggetto Credenziale. Questo oggetto Credenziale viene quindi inviato al server. Il server utilizza quindi tale

oggetto Credenziale per l'autenticazione utilizzando il plug-in Authenticator, che è presentato dall'interfaccia thecom.ibm.websphere.objectgrid.security.plugins.CredentialGenerator.Authenticator.

I plug-in giocano un ruolo importante nella struttura di sicurezza ObjectGrid. È possibile implementare il CredentialGenerator per generare una credenziale specifica, ad esempio, coppia ID utente e password, un'etichetta kerberos o un token di sicurezza. È possible implementare il plug-in Authenticator plug-in per autenticare il client. Volendo, è possibile implementare il plug-in Authenticator per supportare sia la password dell'utente che un token di sicurezza.

Tutti i plug-in per la sicurezza che è possibile utilizzare sono presenti nella seguente tabella:

Tabella 9. Plug-in di sicurezza

Categoria	Nome classe plug-in	Instanza
Autenticazione	`com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator`	client
	`com.ibm.websphere.objectgrid.security.plugins.Credential`	client
	`com.ibm.websphere.objectgrid.security.plugins.Authenticator`	server
Autorizzazione	`com.ibm.websphere.objectgrid.security.plugins.MapAuthorization`	ObjectGrid
	`com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization`	Cluster
Sicurezza cluster ObjectGrid	`com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager`	server
altro	`com.ibm.websphere.objectgrid.security.plugins.SubjectSource`	ObjectGrid locale
	`com.ibm.websphere.objectgrid.security.plugins.SubjectValidation`	ObjectGrid locale

Il seguente diagramma mostra i plug-in e le istanze applicate. Ad esempio, il plug-in MapAuthorization si applica sulle istanze ObjectGrid, ma AdminAuthorization si applica sulle instanze del server.

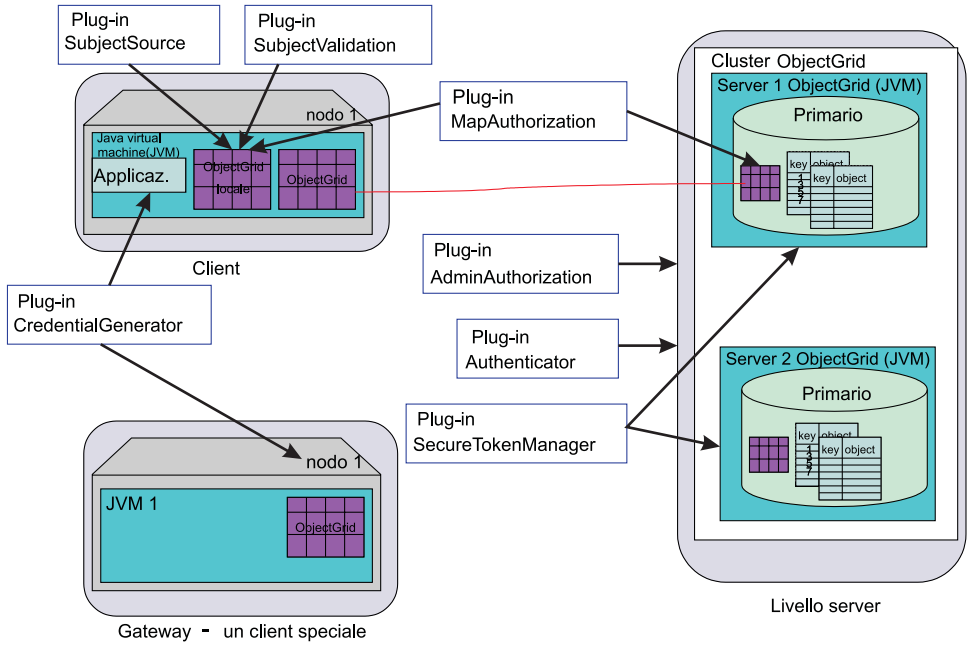


Figura 17. Plug-in di sicurezza

La seguente tabella mostra l'implementazione integrata. La colonna obiettivo mostra lo scopo. Lo scopo può essere per la produzione o la prova predefinita.

Tabella 10. Implementazioni integrate di sicurezza

Plug-in	nome classe integrato	scopo
Credenziale Generatore credenziali	com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator	produzione
	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator	produzione
	com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential	produzione
	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential	produzione
	com.ibm.websphere.objectgrid.security.plugins.builtins.ClientCertificateCredential	produzione
Authenticator	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator	produzione
	com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator	prova
	com.ibm.websphere.objectgrid.security.plugins.builtins.CertificateMappingAuthenticator	prova
	com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator	prova
Autorizzazione della mappa	com.ibm.websphere.objectgrid.security.plugins.builtins.JAASMapAuthorizationImpl	produzione
	com.ibm.websphere.objectgrid.security.plugins.builtins.TAMMapAuthorizationImpl	prova
SubjectSource	com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl	produzione

Tabella 10. Implementazioni integrate di sicurezza (Continua)

Plug-in	nome classe integrato	scopo
Convalida soggetto	com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl	produzione

Sicurezza server client

In questa sezione viene descritto il meccanismo di autenticazione e il modo in cui proteggere la comunicazione server client.

La sicurezza server client copre i seguenti aspetti importanti:

- Come “Abilitazione della sicurezza server client ”
- Come ottenere le credenziali del client con il “Credenziali e generatore di credenziali ” a pagina 142
- Come configurare i parametri utilizzati per la configurazione SSL mediante “Comunicazione sicura ” a pagina 158
- Come autenticare il client sul server mediante un “Authenticator” a pagina 148

Abilitazione della sicurezza server client

Abilitazione della sicurezza client

Per abilitare la sicurezza client, impostare la proprietà securityEnabled nel file `security.ogclient.props` su **true**. ObjectGrid fornisce un file di maschera delle proprietà di sicurezza client, il file `security.ogclient.props`, nella directory `[WAS_HOME]/optionalLibraries/ObjectGrid/properties` per un'installazione WebSphere o nella directory `/ObjectGrid/properties` per un'installazione server mista. È possibile modificare questo file di maschera con i valori appropriati.

Di seguito è riportata la descrizione della proprietà securityEnabled:

securityEnabled (true, false+)

Questa proprietà indica se la sicurezza è abilitata. Quando un client si collega a un server, i valori securityEnabled sul client e sul server devono entrambi essere true o false. Ad esempio, se la sicurezza del server collegato è abilitata, il client deve impostare questa proprietà su true per potersi collegare.

L'interfaccia com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration rappresenta il file `security.ogclient.props`. È possibile utilizzare l'API pubblica com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory per creare un'istanza di questa interfaccia con i valori predefiniti oppure creare un'istanza mediante il file delle proprietà di sicurezza client di ObjectGrid. Il file `security.ogclient.props` contiene altre proprietà.

Abilitazione della sicurezza server

Per abilitare la sicurezza sul server, impostare la proprietà securityEnabled nell'XML del cluster su true. Segue un esempio:

```
<cluster>
<objectGrid name="cluster" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300">
```

Credenziali e generatore di credenziali

Quando ci si collega a un server, un client deve presentare le proprie credenziali. Una credenziale client è rappresentata da una interfaccia com.ibm.websphere.objectgrid.security.plugins.Credential. Le credenziali possono contenere una coppia di password utente, un'etichetta Kerberos e così via.

Di seguito è riportata l'interfaccia delle credenziali:

```
package com.ibm.websphere.objectgrid.security.plugins;
import java.io.Serializable;
/**
* Questa interfaccia rappresenta le credenziali utilizzate da un client ObjectGrid. Inoltre,
* rappresenta un'identità client. Queste credenziali sono inviate
* al server ObjectGrid per l'autenticazione. Esse devono essere serializzabili.
*
* Le credenziali devono implementare i metodi equals(Object) e
* hashCode(). Due oggetti delle credenziali sono considerati uguali
* se e solo se rappresentano la stessa identità e le stesse informazioni sulla sicurezza. Ad
* esempio, nel caso in cui le credenziali contengono ID utente e password, queste
* saranno uguali solo se gli ID utente e le password sono uguali.
*
* ObjectGrid fornisce tre implementazioni integrate per questa interfaccia:
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* ClientCertificateCredential:
* Le credenziali contenente una catena di certificati SSL.
* com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential:
* Le credenziali contenenti una coppia di ID utente e password.
* com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential:
* Le credenziali contenenti i token di autorizzazione e autenticazione specifici
* di WebSphere Application Server.
*
* Fare riferimento alla documentazione dell'API per maggiori dettagli.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see CredentialGenerator
*/
public interface Credential extends Serializable {
/**
* Controlla le credenziali per verificare se sono uguali.
*
* Due oggetti delle credenziali sono considerati uguali
* se e solo se rappresentano la stessa identità e le stesse informazioni sulla sicurezza.
*
* @param o l'oggetto che si verifica per l'uguaglianza.
*
* @return true se le credenziali sono equivalenti.
*/
boolean equals(Object o);
/**
* Restituisce il codice hash delle credenziali
*
* @return il codice hash delle credenziali
*/
int hashCode();
}
```

Questa interfaccia definisce esplicitamente i metodi equals(Object) e hashCode(). Tali metodi sono importanti per garantire il corretto funzionamento. Gli oggetti dei soggetti autenticati vengono memorizzati nella cache in base alle credenziali sul server.

ObjectGrid fornisce tre diverse implementazioni predefinite per le interfacce delle credenziali:

1. L'implementazione com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential. Queste credenziali contengono una coppia di ID utente e password.
2. L'implementazione com.ibm.websphere.objectgrid.security.plugins.builtins.ClientCertificateCredential. Queste credenziali contengono una catena di certificati. Esse non possono essere utilizzate per l'autenticazione dei certificati client di ObjectGrid. Non è possibile creare queste credenziali sul client. Esse devono essere generate sul server come parte di un handshake SSL.
3. L'implementazionecom.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential. Queste credenziali contengono i token di autenticazione e autorizzazione di WebSphere Application Server. Tali token possono essere utilizzati per propagare gli attributi di sicurezza sui server delle applicazioni nello stesso dominio di sicurezza.

Per ulteriori informazioni, fare riferimento alla documentazione API.

ObjectGrid fornisce anche un plug-in per generare le credenziali. Questo plug-in è rappresentato dall'interfaccia com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator. Di seguito sono riportate le interfacce di CredentialGenerator:

```
/**
* Questo plug-in è utilizzato per ottenere le credenziali che rappresentano il client. Questo
* è una produzione per le credenziali.
* Una implementazione di esempio consiste nel restituire le credenziali
* contenenti una coppia ID utente e password. L'implementazione delle credenziali
* generate da un'implementazione di questa classe devono essere comprese dal
* plug-in di autenticazione del server.
*
* Una classe di implementazione di questa interfaccia deve avere un costruttore predefinito.
* Quando si avvia il client in un ambiente sicuro, impostare il
* nome della classe di implementazione (credentialGeneratorClass) nel file delle proprietà
* di configurazione della sicurezza client. Il runtime del client costruisce un oggetto
* di questa classe di implementazione e richiama getCredential() per ottenere le credenziali
* per collegarsi a un cluster ObjectGrid.
*
* Gli utenti possono specificare anche ulteriori proprietà per questa produzione
* mediante la proprietà credentialGeneratorProps nel file delle proprietà
* di configurazione della sicurezza client.
* Tali proprietà devono essere inviate a questa
* produzione utilizzando il metodo setProperties(String). In questo modo, è possibile
* personalizzare la produzione.
*
* È inoltre possibile impostare CredentialGenerator in maniera programmatica richiamando
* il metodo ClientSecurityCinfiguration.setCredentialGenerator
* (CredentialGenerator).
*
*
* Ad esempio, è possibile avere le seguenti impostazioni nel
* file delle proprietà di configurazione della sicurezza client:
* credentialGeneratorClass=com.myco.CredGenFactory
*
* credentialGeneratorProps=user1 password1
*
*
* , una stringa "user1 password1" viene inviata al metodo setProperties(String)
* con "user1" che indica il nome utente e "password1"
* che indica la password.
```

```
*
* ObjectGrid fornisce due implementazioni integrate per questa interfaccia:
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* UserPasswordCredentialGenerator:
* Un programma di generazione delle credenziali che crea UserPasswordCredential
* contenenti una coppia ID utente e password.
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* WSTokenCredentialGenerator:
* Un programma di generazione delle credenziali che crea un WSTokenCredential contenente
* i token di WebSphere Application Server specifici per
* l'autenticazione e l'autorizzazione.
*
*
* La relazione tra CredentialGenerator e le credenziali può essere
* una relazione uno a uno o uno a molti. Ad esempio,
* UserPasswordCredentialGenerator
* ha una relazione uno a uno con UserPasswordCredential, ma
* WSTokenCredentialGenerator
* ha una relazione uno a molti con WSTokenCredential in quanto potrebbe generare
* un WSTokenCredential differente in base al soggetto associato
* al thread corrente.
*
* Fare riferimento alla documentazione dell'API per maggiori dettagli.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see Authenticator
* @see ClientSecurityConfiguration#setCredentialGenerator(CredentialGenerator)
* @see Credential
* @see CredentialGeneratorFactory#getCredentialGenerator()
*/
public interface CredentialGenerator {
/**
* Ottiene le credenziali che rappresentano il client.
*
* @return le credenziali che rappresentano il client
*
* @throws CannotGenerateCredentialException se si verifica un errore quando
* si generano le credenziali per il client.
*
* @see Credential
*/
Credential getCredential() throws CannotGenerateCredentialException;
/**
* Imposta le proprietà definite dall'utente sul valore di produzione
*
* Questo metodo viene utilizzato per aggiungere ulteriori proprietà CredentialGenerator
* all'oggetto. Tali proprietà possono essere impostate mediante
* la proprietà credentialGeneratorProps nel file delle proprietà di
* configurazione della sicurezza client.
* In questo modo,  è possibile personalizzare la produzione.
*
* @param properties proprietà definite dall'utente
*/
void setProperties(String properties);
}
```

ObjectGrid fornisce due implementazioni integrate predefinite:

1. Il costruttore com.ibm.websphere.objectgrid.security.plugins.builtins. UserPasswordCredentialGenerator utilizza un ID utente e una password. Quando viene richiamato il metodo getCredential(), questo restituisce un oggetto UserPasswordCredential che contiene l'ID utente e la password.
2. Il costruttore com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator rappresenta un generatore di credenziali (token di

sicurezza) se in esecuzione su WebSphere Application Server. Quando viene richiamato il metodo getCredential(), viene richiamato l'oggetto associato al thread corrente. Quindi, le informazioni sulla sicurezza in questo oggetto vengono convertite in un oggetto WSTokenCredential. È possibile specificare se richiamare un oggetto runAs o un oggetto caller dal thread utilizzando la costante WSTokenCredentialGenerator.RUN_AS_SUBJECT o WSTokenCredentialGenerator.CALLER_SUBJECT.

Per ulteriori informazioni, fare riferimento alla documentazione API.

Connessione

Se un client ObjectGrid desidera connettersi a un server in maniera sicura, è possibile utilizzare il metodo any connect nell'interfaccia ObjectGridManager. Di seguito è riportato un esempio del metodo connect:

```
/**
* Consente al client di collegarsi a un ObjectGrid remoto
* Il RemoteObject Grid è specificato dai parametri
* @param clusterName: il nome del cluster a cui il client
* si collega
* @param host: l'host a cui ci si collega
* @param port: la porta clientAceess utilizzata
* @param ClientSecurityConfiguration: la configurazione di sicurezza. Questa può essere null
* se la sicurezza non è abilitata.
* @param overRideObjectGrid xml. Questo parametro può essere null. Se non è
* null, la configurazione client del plug-in objectgrid viene sovrascritta.
* Non tutti i plug-in possono essere sovrascritti. Per maggiori dettagli fare
* riferimento alla documentazione ObjectGrid
* @throws ConnectException
* @ibm-api
*/
public ClientClusterContext connect(String clusterName, String host, String port,
ClientSecurityConfiguration securityProps, URL overRideObjectGrid) throws
ConnectException ;
```

Questo metodo utilizza un parametro di tipo ClientSecurityConfiguration. Questa interfaccia rappresenta una configurazione di sicurezza client. È possibile utilizzare l'API pubblicacom.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory per creare un'istanza con i valori predefiniti oppure è possibile creare un'istanza mediante il file delle proprietà di sicurezza client di ObjectGrid. Il file `security.ogclient.props` contiene le seguenti proprietà relative all'autenticazione. Il valore contrassegnato con + è il valore predefinito.

- `securityEnabled (true, false+)`: questa proprietà indica se la sicurezza è abilitata. Quando un client si collega a un server, i valori securityEnabled sul client e sul server devono entrambi essere true o false. Ad esempio, se la sicurezza del server collegato è abilitata, il client deve impostare questa proprietà su true per potersi collegare.
- `credentialAuthentication (Never, Supported+, Required)`: questa proprietà indica se il client supporta l'autenticazione delle credenziali.
  - Se il valore delle proprietà è **Never**, l'autenticazione delle credenziali non è supportata da questo client.
  - Se il valore delle proprietà è **Supported**, allora l'autenticazione client viene eseguita quando si comunica con un server che supporta o richiede l'autenticazione delle credenziali. L'autenticazione delle credenziali client trasmette le credenziali o un token SSO (single sign-on).
  - Se il valore della proprietà è **Required**, il client deve inviare le credenziali al server per l'autenticazione.

- `authenticationRetryCount (an integer value, 0+)`. Questa proprietà determina il numero di tentativi provati per il login quando le credenziali sono scadute. Se il valore è `0`, non viene eseguito alcun tentativo. Il tentativo di autenticazione viene applicato solo nei casi in cui le credenziali sono scadute. Se le credenziali non sono valide, allora non viene eseguito alcun tentativo. L'applicazione sarà responsabile del tentativo di collegamento.
- `clientCertificateAuthentication (Never+, Supported, Required)`: questa proprietà indica se il client supporta l'autenticazione dei certificati client.
  - Se il valore della proprietà è **Never**, l'autenticazione di certificati client non è supportata sul client.
  - Se il valore della proprietà è **Supported**, l'autenticazione del client a livello di trasporto può essere eseguita e il client invia i certificati digitali alserver durante le operazioni di autenticazione.
  - Se il valore della proprietà è **Required**, il client viene autenticato solo su server che supportano l'autenticazione client a livello di trasporto.
- `transportType (TCP/IP, SSL-Supported+, SSL-Required)`: indica il protocollo di trasporto con cui il client si collega al server. Il protocollo utilizzato dipende anche dall'impostazione transportType sul server. Fare riferimento a "Comunicazione sicura " a pagina 158 per maggiori dettagli. .
  - Se il valore è **TCP/IP**, il client deve utilizzare TCP/IP per collegarsi al server.
  - Se il valore è **SSL-Supported**, il client può utilizzare TCP/IP o SSL per collegarsi al server. Il client prova prima a utilizzare SSL per collegarsi al server. Se la connessione SSL non riesce, allora il client prova a utilizzare TCP/IP.
  - Se il valore è **SSL-Required**, allora il client deve utilizzare SSL.
- `SSOEnabled`: specifica se il client supporta l'invio di token SSO (single sign-on) al server. Impostare questa proprietà su `false` se il client viene autenticato su ogni server. Impostare la proprietà su `true` se il client viene autenticato solo su un server. Se si imposta SSOEnabled true sul client, verificare che la proprietà single-sign-on-enabled nella configurazione XML del cluster sia impostata su `true`.

È inoltre possibile impostare queste proprietà utilizzando i metodi setter nell'interfaccia ClientSecurityConfiguration.

Una volta creato un oggetto di tipo ClientSecurityConfiguration, impostare credentialGenerator sull'oggetto utilizzando il seguente metodo:

```
/**
* Impostare l'oggetto {@link CredentialGenerator} per il client.
* @param generator l'oggetto CredentialGenerator associato a questo client
*/
void setCredentialGenerator(CredentialGenerator generator);
```

È possibile impostare CredentialGenerator anch nel file delle proprietà di sicurezza del client ObjectGrid. Di seguito sono riportate le proprietà:

- **credentialGeneratorClass**: il nome dell'implementazione della classe per CredentialGenerator. Tale nome deve avere un costruttore predefinito.
- **credentialGeneratorProps**: le proprietà per la classe CredentialGenerator. Se il valroe non è null, esso sarà impostato sull'oggetto CredentialGenerator costruito mediante il metodo setProperties(String).

Di seguito è riportato un esempio per istanziare ClientSecurityConfiguration e per utilzizarlo quindi per collegarsi al server.

```
/**
* Richiamare un ClientClusterContext sicuro
* @return un oggetto ClientClusterContext sicuro
*/
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 viene richiamato il metodo connect, il client ObjectGrid richiama il metodo CredentialGenerator.getCredential() per ottenere le credenziali del client. Queste credenziali vengono quindi inviate insieme alla richiesta di connessione al server per l'autenticazione.

Utilizzare un CredentialGenerator differente per ogni sessione.

In alcuni casi, un client ObjectGrid rappresenta solo una identità client, mentre in altri casi può rappresentare più identità. Di seguito è riportato uno scenario per il secondo caso: viene creato un client ObjectGrid condiviso in un server Web. Tutti i servlet in questo server Web utilizzano questo client ObjectGrid. Poiché ogni servlet rappresenta un client Web differente, utilizzare credenziali differenti quando si inviano le richieste ai server ObjectGrid.

ObjectGrid consente la modifica delle credenziali a livello di sessione. In altre parole, ogni sessione può utilizzare un CredentialGenerator differente. Pertanto, gli scenari precedenti possono essere eseguiti lasciando al servlet l'ottenimento di una sessione con un CredentialGenerator differente. Di seguito è riportato il metodo nell'interfaccia ObjectGridManager.

```
/**
* Richiamare una sessione con un CredentialGenerator. Questo metodo può essere
* richiamato soltanto dal client ObjectGrid in un ambiente server client.
*
* Se ObjectGrid viene utilizzato in un modello principale, ovvero all'interno della stessa JVM
* senza client o server, è preferibile utilizzare getSession(Subject) per
* proteggere l'ObjectGrid.
*
* @since WAS XD 6.0.1
*/
Session getSession(CredentialGenerator credGen) throws
ObjectGridException, TransactionCallbackException;
```

Segue un esempio:

```
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
CredentialGenerator credGenManager = new UserPasswordCredentialGenerator
("manager", "xxxxxx");
CredentialGenerator credGenEmployee = new UserPasswordCredentialGenerator
("employee", "xxxxxx");
ObjectGrid og = ogManager.getObjectGrid(ctx, "accounting");
// Richiamare una sessione con CredentialGenerator;
Session session = og.getSession(credGenManager );
// Richiamare la mappa di impegati
ObjectMap om = session.getMap("employee");
// avviare una transazione.
session.begin();
Object rec1 = map.get("xxxxxx");
session.commit();
// Richiamare un'altra sessione con un CredentialGenerator differente;
session = og.getSession(credGenEmployee );
```

```
// Richiamare la mappa di impegati
om = session.getMap("employee");
// avviare una transazione.
session.begin();
Object rec2 = map.get("xxxxx");
session.commit();
```

Se si utilizza il metodo ObjectGird.getSession() per ottenere una sessione, la sessione utilizzerà il CredentialGenerator impostato sull'oggetto ClientConfigurationSecurity. Pertanto, il CredentialGenerator inviato al metodo ObjectGird.getSession(CredentialGenerator) sovrascriverà il CredentialGenerator impostato nell'oggetto ClientConfigurationSecurity.

Se è possibile riutilizzare la sessione, si ottiene un miglioramento delle prestazioni. Tuttavia, l'utilizzo del metodo ObjectGrid.getSession(CredentialGenerator) non è costoso; il carico maggiore sia ha durante il periodo di raccolta dei dati non utilizzati dell'oggetto. Verificare di rilasciare i riferimenti una volta terminata la sessione. In breve, se la sessione può condividere un'identità, provare a riutilizzare la sessione, se invece non è in grado di condividerla, utilizzare il metodo ObjectGrid.getSession(CredentialGenerator).

Authenticator

Dopo che il client ObjectGrid ha richiamato le credenziali mediante l'oggetto CredentialGenerator, le credenziali vengono inviate insieme alla richiesta client al server ObjectGrid. Il server ObjectGrid autentica le credenziali prima di elaborare la richiesta. Se le credenziali vengono autenticate correttamente, viene restituito un oggetto per rappresentare le credenziali stesse. Questo oggetto viene quindi utilizzato per autorizzare la richiesta.

Esso viene inoltre memorizzato nella cache. L'oggetto scade una volta raggiunto il valore di timeout della sessione. Il valore di timeout della sessione di login può essere impostato mediante la proprietà loginSessionExpirationTime nel file XML del cluster. Ad esempio, impostando loginSessionExpirationTime=″300″ l'oggetto scadrà in 300 secondi.

Il server ObjectGrid utilizza il plug-in Authenticator per autenticare le credenziali. Di seguito è riportata l'interfaccia di Authenticator:

```
/**
* Questo plug-in può essere utilizzato per autenticare un client ObjectGrid
* su un server ObjectGrid
* in base alle credenziali fornite dal client. Un oggetto
* viene restituito come risultato del processo di autenticazione.
*
* Questo plug-in è utilizzato su un server ObjectGrid. Esso può essere configurato
* nel file XML del cluster ObjectGrid.
*
* Le credenziali inviate al metodo authenticate(Credential)
* possono contenere tutte le informazioni sulle credenziali necessarie. Ad esempio,
* le credenziali possono contenere una coppia ID utente e password.
*
* ObjectGrid fornisce diverse implementazioni integrate per questa interfaccia:
* * com.ibm.websphere.objectgrid.security.plugins.builtins.
* CertificateMappingAuthenticator:
* Un programma di autenticazione che associa semplicemente un certificato
* SSL *  a un oggetto.
* com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator:
* Un programma di autenticazione che autentica un ID utente e una password a
* un file delle * chiavi.
* com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator:
```

```
* Un programma di autenticazione che autentica un ID utente e una password a
* un server LDAP.
* com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator:
* Un programma di autenticazione che autentica un token di sicurezza WebSphere
* Application Server.
*
* Fare riferimento alla documentazione dell'API per maggiori dettagli.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see Credential
*/
public interface Authenticator {
/**
* Autentica un utente rappresentato dalle credenziali.
*
* @param credential le credenziali utente
*
* @return a Subject un oggetto che rappresenta l'utente
*
* @throws InvalidCredentialException se le credenziali non sono valide
* @throws ExpiredCredentialException se le credenziali sono scadute
*
* @see Credential
*/
Subject authenticate(Credential credential)
throws InvalidCredentialException, ExpiredCredentialException;
}
```

Questo è il punto in cui l'implementazione ottiene le credenziali e le autentica con un registro utente, ad esempio un server LDAP (Lightweight Directory Access Protocol) e così via. ObjectGrid non fornisce una configurazione del registro utente predefinita. Il collegamento a un registro utente e la relativa autenticazione deve essere implementata in questo plug-in.

Ad esempio, una implementazione di un Authenticator estrae l'ID utente e la password dalle credenziali, le utilizza per connettersi e convalidarsi su un server LDAP, quindi crea un oggetto come risultato del processo di autenticazione. L'implementazione può utilizzare i moduli di login JAAS. Un oggetto viene restituito come risultato dell'autenticazione.

Questo metodo emette due eccezioni: `InvalidCredentialException` and `ExpiredCredentialException`. L'eccezione `InvalidCredentialException` indica che le credenziali non sono valide. L'eccezione `ExpiredCredentialException` indica che le credenziali sono scadute. Se una di queste due eccezioni viene restituita dal metodo di autenticazione, allora vengono restituite al client. Tuttavia, il runtime del client tratta le due eccezioni in maniera differente:

- Se l'eccezione è una eccezione InvalidCredentialException, il runtime del client visualizza questa eccezione. L'applicazione dovrà poi gestirla. È possibile correggere l'eccezione CredentialGenerator, ad esempio, e provare a eseguire di nuovo l'operazione.
- Se l'eccezione è una eccezione ExpiredCredentialException e il numero di tentativi non è 0, il runtime del client richiama di nuovo il metodo CredentialGenerator.getCredential() e invia le nuove credenziali al server. Se l'autenticazione con le nuove credenziali riesce, il server elabora la richiesta. Se l'autenticazione invece non riesce, l'eccezione viene restituita al client. Se il numero di tentativi di autenticazione raggiunge il valore massimo consentito e il

client continua a ricevere l'eccezione ExpiredCredentialException, l'eccezione ExpiredCredentialException sarà il risultato finale. L'applicazione dovrà gestire tale applicazione.

L'interfaccia Authenticator fornisce una elevata flessibilità. È possibile implementare questa interfaccia in diversi modi. Ad esempio, è possibile implementare questa interfaccia in modo da eseguire sia l'autenticazione delle credenziali che l'autenticazione dei certificaticlient, in modo da supportare entrambi i tipi di autenticazione. In alternativa, è possibile implementare l'interfaccia in modo da supportare due diversi registri utente.

ObjectGrid supporta due tipi di autenticazione: l'autenticazione delle credenziali e l'autenticazione dei certificati client. Il meccanismo da utilizzare dipende dall'impostazione delle proprietà di sicurezza sul client e sul server. Di seguito sono riportate le proprietà:

- credentialAuthentication nel file`security.ogclient.props`
- credentialAuthentication nel file`security.ogserver.props`
- clientCertificateAuthentication nel file`security.ogclient.props`
- clientCertificateAuthentication nel file `security.ogserver.props`

È possibile impostare queste proprietà anche utilizzando le API di programmazione.

Nelle due tabelle seguenti sono riportati i meccanismi di autenticazione utilizzati in base a impostazioni differenti.

Tabella 11. Autenticazione delle credenziali in base alle impostazioni del client e del server

Client credentialAuthentication	**Server credentialAuthentication**	**Risultato**
No	Mai	disabilitato
	Supportato	disabilitato
	Richiesto	Errore
Supportato	Mai	disabilitato
	Supportato	abilitato
	Richiesto	abilitato
Richiesto	Mai	Errore
	Supportato	abilitato
	Richiesto	abilitato

Quando non è presente l'autenticazione delle credenziali (ovvero il risultato è disabilitato), allora è necessario utilizzare l'autenticazione dei certificati client.

La seguente tabella riporta se l'autenticazione dei certificati client è utilizzata con impostazioni differenti. L'autenticazione dei certificati client è possibile solo se come protocollo di comunicazione è utilizzato SSL e non è utilizzata l'autenticazione delle credenziali.

Tabella 12. Autenticazione dei certificati client in base alle impostazioni del client e del server

Autenticazione clientCertificate client	Autenticazione clientCertificate server	Risultato
No	Mai	disabilitato
	Supportato	disabilitato
	Richiesto	Errore
Supportato	Mai	disabilitato
	Supportato	abilitato*
	Richiesto	abilitato*
Richiesto	Mai	Errore
	Supportato	abilitato*
	Richiesto	abilitato*

* ClientCertificateAuthentication si verifica solo se SSL è utilizzato come protocollo e non viene utilizzato CredentialAuthentication.

Quando viene utilizzata sia l'autenticazione delle credenziali che l'autenticazione dei certificati client, ma le credenziali inviate dal client sono null, verrà utilizzata l'autenticazione dei certificati client.

Il programma di autenticazione può essere configurato nel file XML del cluster. Di seguito viene riportato un esempio:

```
<cluster name="cluster1" securityEnabled="true" singleSignOnEnabled="true"
loginSessionExpirationTime="300" statisticsEnabled="true"
statisticsSpec="map.all=enabled">
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12500" workingDirectory="" traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12501" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<authenticator
className ="com.ibm.websphere.objectgrid.security.plugins.builtins.
WSTokenAuthenticator">
</authenticator>
</cluster>
```

ObjectGrid fornisce quattro tipi di implementazione integrata per l'autenticazione: autenticazione con ID utente e password su un registro utente del file delle chiavi, autenticazione con ID utente e password su un server LDAP, autenticazione con associazione dei certificati client SSL e un meccanismo di sicurezza di WebSphere Application Server. Tranne per il meccanismo di sicurezza di WebSphere Application Server, le implementazioni integrate sono solo a scopo di verifica. Lo scopo principale di queste implementazioni integrate consiste nel consentire una semplice verifica senza dover scrivere del codice. L'implementazione del programma di autenticazione WebSphere Application Server è un'implementazione predefinita che può essere collegata quando sia i client che i server ObjectGrid si trovano nello stesso dominio di sicurezza.

Per i server ObjectGrid che desiderano utilizzare i registri utente WebSphere Application Server, è possibile utilizzare le API di WebSphere Application Server per ottenere il registro utente configurato sul server delle applicazioni e utilizzare quindi

l'implementazione del proprio programma di autenticazione. Tuttavia, questa implementazione non viene trattata in questa guida alla programmazione.

Implementazione del programma di autenticazione del registro del file delle chiavi

È possibile memorizzare un ID utente e una password in un file denominato file di archivio chiavi. Per creare questo file e le relative voci, è possibile utilizzare lo strumento keytool. Ad esempio, il seguente comando crea una voce con l'alias user1:

```
keytool -genkey -v -keystore ./keys.jks -storepass password -alias user1
-keypass password -dname CN=user1,O=MyCompany,L=MyCity,ST=MyState
```

A scopo di verifica, ObjectGrid fornisce l'implementazione predefinita com.ibm.websphere.objectgrid.security.plugins.builtins. KeyStoreLoginAuthenticator per questo plug-in per gestire l'autenticazione con nome utente e password. Questa implementazione utilizza un nome di login KeyStoreLogin per collegare l'utente a un file di archivio chiavi.

Di seguito è riportato un frammento di codice che mostra l'implementazione del metodo authenticate(Credential) nella 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();
```

Questo frammento di codice per prima cosa esegue il casting delle credenziali su UserPasswordCredential, che è un'implementazione dell'interfaccia delle credenziali, poiché ha un contratto con il client che il client può inviare soltanto a un oggetto di tipo UserPasswordCredential. Quindi richiama il modulo di login KeyStoreLogin per collegarsi.

ObjectGrid fornisce un modulo di login com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule per questo scopo. È necessario fornire un file di archivio chiavi che contiene il nome utente e la password per ogni utente. Il file di archivio chiavi è configurato come opzione per il modulo di login.

Di seguito è riportato un frammento di codice che mostra il modo in cui il modello di login si collega al file delle chiavi.

```
/**
* Autentica un utente in base al file keystore.
*
* @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) {
// treat a NULL password as an empty password
tempPwd = new char[0];
}
pwd = new char[tempPwd.length];
System.arraycopy(tempPwd, 0, pwd, 0, tempPwd.length);
pwdCallback.clearPassword();
if (debug) {
System.out.println("[KeyStoreLoginModule] login: "
+ "user entered user name: " + name);
}
if (ObjectGridManagerImpl.isTraceEnabled && TC.isDebugEnabled())
Tr.debug(TC, "login", "userName="+name);
// Validate the user name and password
try {
validate(name, pwd);
}
catch (SecurityException se) {
principals.clear();
publicCreds.clear();
privateCreds.clear();
LoginException le = new LoginException(
"Exception encountered during login");
le.initCause(se);
throw le;
}
if (debug) {
System.out.println("[KeyStoreLoginModule] login: exit");
}
return true;
}
/**
* Convalida il nome utente e la password basati sul keystore.
*
* @param userName user name
* @param password password
* @throws SecurityException if any exceptions encountered
*/
protected void validate(String userName, char password[])
throws SecurityException {
PrivateKey privateKey = null;
// Richiama la chiave privata dal keystore
try {
privateKey = (PrivateKey) keyStore.getKey(userName, password);
}
catch (NoSuchAlgorithmException nsae) {
SecurityException se = new SecurityException();
se.initCause(nsae);
throw se;
}
catch (KeyStoreException kse) {
SecurityException se = new SecurityException();
se.initCause(kse);
throw se;
}
catch (UnrecoverableKeyException uke) {
SecurityException se = new SecurityException();
```

```
se.initCause(uke);
throw se;
}
if (privateKey == null) {
throw new SecurityException("Invalid name: " + userName);
}
// Controlla i certificati
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 il primo certificato è un X509Certificate
if (certs[0] instanceof X509Certificate) {
try {
// Richiama il primo certificato che rappresenta l'utente
X509Certificate certX509 = (X509Certificate) certs[0];
// Crea un principal
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;
}
}
}
}
```

È necessario creare un nome di login “KeyStoreLogin” nel file di configurazione di autenticazione JAAS. Se non si è esperti del file di configurazione di autenticazione JAAS, fare riferimento a JAAS Authentication Tutorial per maggiori dettagli.

```
KeyStoreLogin {
com.ibm.websphere.objectgrid.jaas.KeystoreLoginModule required
keyStoreFile="${user.dir}${/}security${/}.keystore";
};
```

Questa implementazione è solo a scopo di verifica.

Implementazione del programma di autenticazione LDAP

ObjectGrid fornisce l’implementazione predefinita com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator per questo plug-in per gestire l’autenticazione con nome utente e password a un server LDAP. Questa implementazione utilizza il modulo di login LDAPLogin per collegare l’utente a un server LDAP.

Il seguente frammento di codice dimostra come viene implementato il metodo di autenticazione:

```
/**
* @see com.ibm.ws.objectgrid.security.plugins.Authenticator#
* authenticate(LDAPLogin)
*/
```

```
public Subject authenticate(Credential credential) throws
InvalidCredentialException, ExpiredCredentialException {
UserPasswordCredential cred = (UserPasswordCredential) credential;
LoginContext lc = null;
try {
lc = new LoginContext("LDAPLogin",
new UserPasswordCallbackHandlerImpl(cred.getUserName(),
cred.getPassword().toCharArray()));
lc.login();
Subject subject = lc.getSubject();
return subject;
}
catch (LoginException le) {
throw new InvalidCredentialException(le);
}
catch (IllegalArgumentException ile) {
throw new InvalidCredentialException(ile);
}
}
```

ObjectGrid fornisce un modulo di login com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule per questo scopo. È necessario fornire le seguenti due opzioni nel file di configurazione di login JAAS:

- **providerURL**: l'URL del provider del server LDAP
- **factoryClass**: la classe di implementazione della produzione di contesto LDAP

LDAPLoginModule richiama il metodo com.ibm.websphere.objectgrid.security.plugins.builtins. LDAPAuthentcationHelper.authenticate. Il seguente frammento di codice mostra il modo in cui viene implementato il metodo di autenticazione diLDAPAuthentcationHelper:

```
/**
* Autentica l'utente sulla directory LDAP.
* @param user l'ID utente, ad esempio uid=xxxxxx,c=us,ou=bluepages,o=ibm.com
* @param pwd la password
*
* @throws NamingException
*/
public String[] authenticate(String user, String pwd)
throws NamingException {
   Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, factoryClass);
env.put(Context.PROVIDER_URL, providerURL);
env.put(Context.SECURITY_PRINCIPAL, user);
env.put(Context.SECURITY_CREDENTIALS, pwd);
env.put(Context.SECURITY_AUTHENTICATION, "simple");
InitialContext initialContext = new InitialContext(env);
// Ricerca l'utente
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("");
// Controlla l'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 l'autenticazione riesce correttamente, l'ID e la password sono considerati validi. Il modulo di login richiama quindi le informazioni sull'UID e le informazioni sul reparto da questo metodo di autenticazione. Il modulo di login crea due principal: SimpleUserPrincipal e SimpleDeptPrincipal. È possibile utilizzare l'oggetto non autenticato per l'autorizzazione del gruppo (in questo caso il reparto è un gruppo) e l'autorizzazione dei singoli.

Di seguito è riportato un modulo di login di esempio utilizzato per collegarsi al server LDAP:

```
LDAPLogin { com.ibm.websphere.objectgrid.security.plugins.builtins.
LDAPLoginModule required
providerURL="ldap://directory.acme.com:389/"
factoryClass="com.sun.jndi.ldap.LdapCtxFactory";
};
```

Nella precedente configurazione, il server LDAP fa riferimento al server `ldap://directory.acme.com:389/`. Modificare le impostazioni sul server LDAP. Questo modulo di login utilizza l'ID utente e la password forniti per collegarsi al server LDAP. Questa implementazione è solo a scopo di verifica.

Implementazione del programma di autenticazione di WebSphere Application Server

ObjectGrid fornisce anche l'implementazione integratacom.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator per utilizzare l'infrastruttura di sicurezza di WebSphere Application Server. Questa implementazione integrata può essere utilizzata quando si verificano le seguenti condizioni:

- La sicurezza globale di WebSphere Application Server è attivata.
- Sia i client ObjectGrid che i server ObjectGrid sono avviati sulle JVM di WebSphere Application Server.
- Questi server delle applicazioni si trovano nello stesso dominio di sicurezza.
- Il client ObjectGrid è già autenticato con WebSphere Application Server.

Il client ObjectGrid può utilizzare la classe com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator per generare le credenziali e il server ObjectGrid utilizza questa classe di implementazione del programma di autenticazione per autenticare le credenziali. Se il token viene autenticato correttamente, viene restituito un oggetto.

Questo scenario sfrutta il fatto che il client ObjectGrid è già stato autenticato. Poiché i server delle applicazioni che hanno i server ObjectGrid nello stesso dominio di sicurezza dei server delle applicazioni su cui sono presenti i client ObjectGrid, i token di sicurezza possono essere propagati dal client ObjectGrid al server ObjectGrid pertanto lo stesso registro utente non deve essere autenticato di nuovo.

Implementazione del programma di autenticazione di associazione certificati semplice

ObjectGrid fornisce un'implementazione integrata com.ibm.websphere.objectgrid.security.plugins.builtins. CertificateMappingAuthenticator per associare il certificato a un oggetto. L'implementazione estrae il DN (Distinguished Name) del primo certificato nella catena e crea un principal con il nome. Questa implementazione è solo a scopo di verifica.

Implementazione del programma di autenticazione di Tivoli Access Manager

Tivoli Access Manager viene diffuso come server di sicurezza. È inoltre possibile implementare il programma di autenticazione mediante i moduli di login forniti da Tivoli Access Manager.

Per autenticare un utente mediante Tivoli Access Manager, il modulo di login fornito da Tivoli, com.tivoli.mts.PDLoginModule, richiede che l'applicazione chiamante fornisca:

- Il nome di un principal, specificato come nome breve o nome X.500 (DN)
- Una password

Il LoginModule autentica il principal e restituisce le credenziali di Tivoli Access Manager. LoginModule si aspetta che l'applicazione chiamante fornisca le seguenti informazioni:

- Il nome utente, mediante un javax.security.auth.callback.NameCallback
- La password, mediante javax.security.auth.callback.PasswordCallback.

Una volta richiamate le credenziali di Tivoli Access Manager, il modulo di login JAAS crea un oggetto e un PDPrincipal. Non viene fornita alcuna autenticazione TAM in quanto inutile con PDLoginModule. Fare riferimento al manuale IBM Tivoli Access Manager Authorization Java Classes Developer Reference per maggiori dettagli.

SSO (single sign-on)

Dopo che un client ObjectGrid è stato correttamente autenticato sul server, il server ObjectGrid crea un oggetto. Se sia il client che il server supportano SSO (single sign-on), questo oggetto viene quindi convertito in un token SSO. Questo token viene restituito al client per l'associazione al socket. Questo token SSO può essere inviato a un nuovo server per l'autenticazione pertanto non è necessario eseguire la riautenticazione su un server differente.

Il token SSO viene implementato mediante il meccanismo di gestione dei token sicuro di ObjectGrid. Per maggiori dettagli sul gestore dei token sicuri, fare riferimento a "Sicurezza del cluster ObjectGrid" a pagina 176. In sostanza, il meccanismo dei token sicuri utilizza le chiavi di crittografia (chiavi segrete) per crittografare e decrittare i dati utente trasferiti tra server e le chiavi pubbliche e private per firmare i dati.

Il token SSO contiene anche un tempo di scadenza. Tutti i server del prodotto participanti a un dominio di protezione devono avere l'ora, la data e il fuso orario sincronizzati. In caso contrario, i token SSO appaiono come scaduti in anticipano e provocano un errore di autenticazione o di convalida. Ciò non è necessario se viene utilizzato il fuso orario UTC.

Quando un client ObjectGrid si collega a un server differente, questo token SSO può essere inviato a un nuovo server. Tale server convalida il token SSO per

verificare che non sia stato eseguito un accesso non autorizzato mediante l'annullamento della firma e la decrittografia. Inoltre, viene controllato il formato data/ora per verificare che non sia scaduto. Se il token è valido, il client non deve essere autenticato sul server.

Se un token SSO è scaduto, il server deve riautenticare il client. I server richiede quindi al client di fornire di nuovo le credenziali.

Abilitazione di SSO (single-sign-on) per il client

È possibile abilitare SSO per il client in due modi:

- **Configurazione**. Utilizzare la proprietà SSOEnabled nel file `security.ogclient.props` per abilitare SSO sul client.
- **Programmazione**. Utilizzare ClientSecurityConfiguration per abilitare SSO con il seguente metodo.

```
/**
* Impostare se SSO è abilitato.
* @param enabled se SSO è abilitato per questo
* client o meno.
*/
void setSingleSignOnEnabled(boolean enabled);
```

Abilitazione di SSO (single-sign-on) per il server

Per abilitare SSO sul server, impostare l'attributo singleSignOnEnabled su true nel file XML del cluster. Di seguito viene riportato un esempio:

```
<cluster>
<objectGrid name="cluster" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300">
```

SSO è abilitato solo se la sicurezza è abilitata.

Comunicazione sicura

ObjectGrid supporta sia TCP/IP che SSL per la comunicazione sicura. SSL fornisce la comunicazione sicura tra il client e il server. Il meccanismo di comunicazione utilizzato dipende dall'impostazione delle seguenti proprietà:

- La proprietà transportType nel file security.ogclient.props
- La proprietà transportType nel file security.ogserver.props

Tabella 13. Protocollo di trasporto da utilizzare con le impostazioni di trasporto client e server

transportType client	transportType server	Protocollo restituito
TCP/IP	TCP/IP	TCP/IP
	SSL supportato	TCP/IP
	SSL richiesto	Errore
SSL supportato	TCP/IP	TCP/IP
	SSL supportato	SSL (se SSL riporta un errore, allora TCP/IP)
	SSL richiesto	SSL
SSL richiesto	TCP/IP	Errore
	SSL supportato	SSL
	SSL richiesto	SSL

Quando viene utilizzato SSL, la configurazione SSL deve essere fornita sia sul client e sul server.

Configurazione dei parametri SSL per i client ObjectGrid

I parametri SSL sul client possono essere configurati nei seguenti modi:

- Creare un oggetto com.ibm.websphere.objectgrid.security.config.SSLConfiguration utilizzando la classe di produzione com.ibm.websphere.objectgrid.security.config. ClientSecurityConfigurationFactory. Per maggiori dettagli, fare riferimento alla documentazione dell'API.
- Configurare i parametri nel file `security.ogclient.props`, quindi utilizzare il metodo ClientSecurityConfigurationFactory.getClientSecurityConfiguration(String) per inserire i dati nell'istanza dell'oggetto.

Le seguenti proprietà sono relative alle configurazioni SSL nel file`security.ogclient.props`.

- **provider**: specifica il provider JSSE SSL. I valori possibili sono IBMJSSE+, IBMJSSE2, SunJSSE e così via. Impostare questo valore in base al Java Development Kit (JDK) utilizzato.
- **protocol**: specifica il protocollo SSL. I valori possibili sono SSL+, SSLV2, SSLV3, TLS, TLSv1 e così via. Impostare il valore di questo protocollo in base al provider Java Secure Socket Extension (JSSE) utilizzato.
- **alias**: la stringa rappresenta l'alias nell'archivio chiavi. Non esiste un valore predefinito. Questa proprietà viene utilizzata se l'archivio chiavi ha più certificati di coppie di chiavi e si desidera selezionare uno di questi certificati.
- **keyStoreType**: specifica il tipo di archivio chiavi SSL. I valori possibili sono JKS+, JCEK, PKCS12 ecc. Impostare questo valore in base al provider Java Secure Socket Extension (JSSE) utilizzato.
- **keyStore**: specifica il file del percorso dell'archivio chiavi che ha i certificati pubblici del client e le chiavi private. Ad esempio, `[OBJECTGRID_HOME]/properties/DummyClientKeyFile.jks`. In questo rilascio, il supporto hardware non è supportato.
- **keyStorePassword**: specifica la password per proteggere il percorso dell'archivio chiavi. La password viene codificata utilizzando l'algoritmo "xor" da ObjectGrid. Utilizzare lo strumento PropFilePasswordEncoder per codificare questo file delle proprietà. Di seguito è riportato un esempio di password codificata: `{x0r}CDo9Hgw\\`.
- **trustStoreType**: specifica il tipo di archivio sicuro. I valori possibili sono JKS+, JCEK, PKCS12 ecc. È possibile impostare questo valore in base al provider JSSE utilizzato.
- **trustStore**: specifica il nome file del percorso dell'archivio sicuro che ha i certificati pubblici del server. Ad esempio,`[OBJECTGRID_HOME]/properties/DummyClientTrustFile.jks`
- **trustStorePassword**: specifica la password per proteggere il percorso dell'archivio sicuro. La password viene codificata utilizzando l'algoritmo xor da ObjectGrid. Utilizzare lo strumento PropFilePasswordEncoder per codificare questo file delle proprietà. Di seguito è riportato un esempio di password codificata: `{x0r}CDo9Hgw\`
- **certReqSubjectDN**: questa è la stringa richiesta nel nome distinto dell'oggetto del certificato dal server. Un client può collegarsi al server solo se il DN del certificato del server contiene questa stringa. Se il

valore è null, il client non richiede un determinato DN dell'oggetto nel certificato del server. Ad esempio, se il DN dell'oggetto del certificato è ″CN=Server1, OU=Your Organizational Unit, O=Your Organization, S=Your State,C=Your Country″, then ″CN=server1″, ″O=Your Organization″, ″OU=Your Organizational Unit, O=Your Organization, S=Your State,C=Your Country″ restituiscono una corrispondenza ma ″CN=server2″ and ″OU=Your Organizational Unit, L=smething, O=Your Organization, S=Your State,C=Your Country″ non corrispondono. La corrispondenza di caratteri jolly non è supportata.

Configurazione dei parametri SSL per i server Object

I parametri SSL sul client possono essere configurati nel file `security.ogserver.props`. Questo file delle proprietà può essere passato come parametro quando viene avviato un server ObjectGrid.

Oltre alle precedenti proprietà SSL, la configurazione SSL del server ha una proprietà aggiuntiva:

- **clientAuthentication** (true+, false). Se questa proprietà è impostata su true, il client SSL deve essere autenticato. Questa autenticazione è differente dall'autenticazione dei certificati client. L'autenticazione dei certificati client implica l'autenticazione di un client su un registro utenti in base alla catena di certificati, mentre questa proprietà garantisce che il server si colleghi al client corretto.

Sicurezza ObjectGrid locale

In questa sezione viene descritta la sicurezza del modello di programmazione ObjectGrid locale. In questo modello di programmazione, la funzione di sicurezza principale è l'autorizzazione. Il modello non supporta alcun tipo di autenticazione. È necessario eseguire l'autenticazione all'esterno di ObjectGrid. Tuttavia, ObjectGrid fornisce i plug-in per ottenere e convalidare gli oggetti.

L'abilitazione della sicurezza ObjectGrid può essere effettuata in due modi:

- **Configurazione**. È possibile utilizzare un file XML per definire un ObjectGrid e abilitare la sicurezza per tale ObjectGrid. Di seguito è riportato il file `secure-objectgrid-definition.xml` utilizzato nell'applicazione enterprise di esempio di ObjectGridSample. In questo file XML, la sicurezza è abilitata impostando l'attributo securityEnabled su`true`.

```
<objectGrids>
<objectGrid name="secureClusterObjectGrid" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_JASS">
<bean id="TransactionCallback"
classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
...
</objectGrids>
```

- **Programmazione**. Se si desidera creare un ObjectGrid utilizzando le API, richiamare il seguente metodo sull'interfaccia ObjectGrid per abilitare la sicurezza:

```
/**
* Abilita la sicurezza ObjectGrid
*/
void setSecurityEnabled();
```

Nel modello di programmazione ObjectGrid locale, non esiste alcun meccanismo di autenticazione. Quando si imposta la sicurezza con questo metodo, viene soltanto configurata l'autorizzazione. Questa condizione si verifica anche con il

modello server client. L'abilitazione della sicurezza per un ObjectGrid in un modello server client consente l'autorizzazione soltanto su quella istanza di ObjectGrid.

Autenticazione

Nel modello di programmazione ObjectGrid locale, non esiste alcun meccanismo di autenticazione. ObjectGrid si basa su un ambiente di autenticazione, sia esso rappresentsato da server delle applicazioni che dalle applicazioni stesse. Quando un ObjectGrid viene utilizzato in un ambiente WebSphere Application Server o WebSphere Extended Deployment, le applicazioni possono utilizzare il meccanismo di autenticazione di sicurezza di WebSphere Application Server. Quando invece ObjectGrid viene eseguito in un ambiente Java 2 Platform, Standard Edition (J2SE), allora l'applicazione deve gestire le altre applicazioni utilizzare un meccanismo di autenticazione JAAS o di un altro tipo. Per ulteriori informazioni sull'utilizzo dell'autenticazione JAAS, fare riferimento a JAAS reference guide.

Il contratto tra un'applicazione e un'istanza di ObjectGrid è stabilito dall'oggetto javax.security.auth.Subject. Una volta che il client è stato autenticato dal server delle applicazioni o dall'applicazione, l'applicazione può richiamare l'oggetto javax.security.auth.Subject e utilizzare questo oggetto Subject per richiamare una sessione da dall'istanza ObjectGrid richiamando il metodo ObjectGrid.getSession(Subject). Questo oggetto Subject è utilizzato per autorizzare l'accesso ai dati di una mappa. Questo contratto è detto meccanismo di invio dell'oggetto. Di seguito è riportata l'API ObjectGrid.getSession(Subject):

```
/**
* Questa API consente alla cache di utilizzare un oggetto specifico piuttosto che uno
* configurato sull'ObjectGrid per richiamare una sessione.
* @param subject
* @return Un'istanza della sessione
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws InvalidSubjectException l'oggetto inviato si basa su un
* meccanismo SubjectValidation non valido.
*/
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;
```

Il metodo getSession nell'interfaccia ObjectGrid può essere utilizzato anche per richiamare un oggetto Session:

```
/**
* Restituisce un oggetto Session che può essere utilizzato da un unico thread alla volta.
* Non è possibile condividere questo oggetto Session tra più thread senza inserire
* una sezione critica su di esso. Mentre la struttura principale consente all'oggetto
* di essere spostato
* tra i thread, TransactionCallback e Loader potrebbero impedire questo utilizzo,
* specialmente in ambienti J2EE. Se la sicurezza è abilitata, allora verrà utilizzato
* SubjectSource per richiamare un oggetto Subject.
*
* Se il metodo initialize() non è stato richiamato prima del primo
* richiamo di getSession, allora si verifica una inizializzazione implicita.
* Ciò garantisce che la configurazione sia completa prima che sia richiesto
* un qualsiasi utilizzo del runtime.
*
* @see #initialize()
* @return Un'istanza della sessione
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws IllegalStateException Se questo metodo viene richiamato dopo
* il richiamo del metodo destroy().
```

```
*/
public Session getSession()
throws ObjectGridException, TransactionCallbackException;
```

Come specificato nella documentazione dell'API, quando è abilitata la sicurezza questo metodo utilizza il plug-in di SubjectSource per richiamare un oggetto Subject. Il plug-in SubjectSource è uno dei plug-in di sicurezza definiti in ObjectGrid per supportare la propagazione degli oggetti. Per ulteriori informazioni, consultare "Plug-in relativi alla sicurezza ".

Il metodo getSession(Subject) può essere richiamato soltanto sull'istanza ObjectGrid locale. Se il metodo getSession(Subject) viene richiamato su un client in una configurazione ObjectGrid distribuita, viene restituita una eccezione.

Plug-in relativi alla sicurezza

ObjectGrid fornisce due plug-in di sicurezza che sono relativi al meccanismo di invio dell'oggetto: il plug-in SubjectSource e il plug-in SubjectValidation.

Plug-in SubjectSource

Il plug-in SubjectSource è rappresentato dall'interfacciacom.ibm.websphere.objectgrid.security.plugins.SubjectSource ed è utilizzato per richiamare un oggetto Subject da un ambiente ObjectGrid in esecuzione. Questo ambiente ObjectGrid potrebbe essere un'applicazione che utilizza ObjectGrid o un server delle applicazioni su cui è presente l'applicazione. Di seguito è riportata l'interfaccia:

```
/**
* Questo plug-in può essere utilizzato per richiamare un oggetto che rappresenta
* il client ObjectGrid.
* Questo oggetto può quindi essere utilizzato per l'autorizzazione dell'ObjectGrid.
* Il metodo getSubject viene richiamato dal runtime ObjectGrid quando viene utilizzato
* il metodo ObjectGrid.getSession() per richiamare una sessione e
* la sicurezza è abilitata.
*
* Questo plug-in è utile per un client già autenticato: esso
* richiama l'oggetto autenticato e lo invia
* all'istanza ObjectGrid. Pertanto, non è necessaria
* una ulteriore autenticazione.
*
* Ad esempio, utilizzare
* Subject.getSubject(AccessControlContext)
* per richiamare l'oggetto associato a AccessControlContext e
* restituirlo nell'implementazione getSubject.
*
* Questo plug-in può essere utilizzato in un dominio sicuro, come su
* un server ObjectGrid.
*
* @ibm-api
* @since WAS XD 6.0
*/
public interface SubjectSource {
/**
* Richiama un oggetto che può rappresentare il client ObjectGrid.
*
* @return a Subject object
* @throws ObjectGridSecurityException any exception during the subject
* retrieving
*/
Subject getSubject() throws ObjectGridSecurityException;
}
```

Il plug-in SubjectSource può essere considerato come alternativa al meccanismo di invio dell'oggetto. Mediantequesto meccanismo, l'applicazione richiama l'oggetto Subject e lo utilizza per richiamare l'oggetto della sessione ObjectGrid. Con il plug-in SubjectSource, il runtime di ObjectGrid richiama l'oggetto Subject e lo utilizza per richiamare l'oggetto della sessione. Il meccanismo di invio dell'oggetto offre il controllo degli oggetti Subject alle applicazioni, mentre il meccanismo del plug-in SubjectSource libera le applicazioni dalle operazioni di richiamo dell'oggetto Subject.

Questo plug-in SubjectSource può essere utilizzato per richiamare un oggetto Subject che rappresenta un client ObjectGrid, che viene quindi utilizzato per l'autorizzazione di ObjectGrid. Quando viene richiamato il metodo ObjectGrid.getSession(), l'oggetto Subject getSubject() emette il metodo ObjectGridSecurityException () che viene richiamato dal runtime di ObjectGrid (se è abilitata la sicurezza).

ObjectGrid fornisce un'implementazione predefinita del plug-in SubjectSource: com.ibm.websphere.objectgrid.security.plugins.builtins. WSSubjectSourceImpl. Questa implementazione può essere utilizzata per richiamare un oggetto caller subject o un oggetto RunAs dal thread quando un'applicazione viene eseguita su WebSphere Application Server. È possibile configurare questa classe come classe di implementazione SubjectSource quando si utilizza ObjectGrid con WebSphere Application Server. Di seguito è riportato un frammento di codice che mostra il flusso principale del metodo WSSubjectSourceImpl.getSubject():

```
Subject s = null;
try {
 if (finalType == RUN_AS_SUBJECT) {
  // get the RunAs subject
  s = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
}
 else if (finalType == CALLER_SUBJECT) {
  // get the callersubject
  s = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
}
}
catch (WSSecurityException wse) {
 throw new ObjectGridSecurityException(wse);
}
return s;
```

Per maggiori dettagli, fare riferimento alla documentazione dell'API relativa al plug-in e all'implementazione di WSSubjectSourceImpl.

Plug-in SubjectValidation

Il plug-in SubjectValidation, rappresentato dall'interfaccia com.ibm.websphere.objectgrid.security.plugins.SubjectValidation, è un altro plug-in di sicurezza. Il plug-in SubjectValidation può essere utilizzato per verificare che un oggetto javax.security.auth.Subject, inviato a ObjectGrid o richiamato dal plug-in SubjectSource, è un oggetto valido che non è stato modificato. Di seguito è riportata l'interfaccia:

```
/**
* Questo plug-in può essere utilizzato per verificare che
* un javax.security.auth.Subject inviato all'ObjectGrid è un oggetto valido
* per cui non è stato eseguito alcun accesso non autorizzato.
*
* Un'implementazione di questo plug—in richiede un supporto dal programma
* di creazione dell'oggetto, in quanto solo questo sa perfettamente se è
* stato eseguito un accesso non autorizzato all'oggetto. Tuttavia, è possibile
```

```
* che lo stesso programma di creazione non sia a conoscenza di questa cosa.
* In questo caso, il plug-in non va utilizzato.
*
* Questo plug-in può essere utilizzato in un dominio sicuro, come su
* un server delle applicazioni. Non inserire il plug-in sul client, in quanto
* verrà ignorato.
*
* @ibm-api
*
* @since WAS XD 6.0
*/
public interface SubjectValidation {
/**
* Verificare che non sia stato eseguito un accesso non autorizzato all'oggetto.
* @param subject un oggetto da convalidare
* @return l'oggetto convalidato
* @throws InvalidSubjectException
*/
Subject validateSubject(Subject subject) throws
InvalidSubjectException;
}
```

L'oggetto Subject validateSubject(Subject subject) emette InvalidSubjectException; il metodo nell'interfaccia SubjectValidation utilizza un oggetto Subject e restituisce un oggetto Subject. Se un oggetto Subject è considerato valido e viene restituito un oggetto what, allora sono presenti tutti gli elementi per le proprie implementazioni. Se l'oggetto Subject non è valido, il risultato è un'eccezione `InvalidSubjectException`.

È possibile utilizzare questo plug-in se non si considera sicuro l'oggetto Subject inviato a questo metodo. Tuttavia, di solito gli sviluppatori dell'applicazione che creano il codice per richiamare l'oggetto Subject possono essere considerati sicuri.

Un'implementazione di questo plug-in richiede il supporto dal creatore dell'oggetto Subject in quanto solo costui è a conoscenza se è stata eseguito un accesso non consentito all'oggetto Subject. Tuttavia, il creatore dell'oggetto potrebbe non essere a conoscenza di questo fatto. In questo caso, questo plug-in non è di alcuna utilità.

ObjectGrid fornisce un'implementazione predefinita del plug-in SubjectValidation: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl. Questa implementazione può essere utilizzata per convalidare l'oggetto autenticato di WebSphere. Gli utenti possono configurare questa classe come la classe di implementazione SubjectValidation quando si utilizza ObjectGrid con WebSphere Application Server. L'implementazione di WSSubjectValidationImpl considera un oggetto Subject valido se il token delle credenziali associato al Subject non è stato modificato. In altre parole, gli utenti possono modificare altre parti dell'oggetto Subject. L'implementazione WSSubjectValidationImpl richiede a WebSphere Application Server l'oggetto Subject originale che corrisponde al token delle credenziali e restituisce l'oggetto Subject originale come oggetto Subject convalidato. Pertanto, le modifiche apportate al contenuto dell'oggetto Subject oltre al token delle credenziali non hanno alcun effetto. Il seguente frammento di codice mostra il flusso di base di WSSubjectValidationImpl.validateSubject(Subject):

```
// Crea un LoginContext con schema WSLogin e
// invia un handler di richiamata.
LoginContext lc = new LoginContext("WSLogin",
```

```
new WSCredTokenCallbackHandlerImpl(subject));
// Quando viene richiamato questo metodo, i metodi dell'handler di richiamata
// vengono richiamati per il collegamento dell'utente.
lc.login();
// Richiama l'oggetto da LoginContext
return lc.getSubject();
```

Nel frammento di codice precedente, un oggetto dell'handler di richiamata del token delle credenziali, l'oggetto WSCredTOkenCallbackHandlerImpl, viene creato con l'oggetto Subject da convalidare. Quindi, viene creato un oggetto LoginContext con lo schema di login ″WSLogin″. Viene richiamato il metodo lc.login(), la sicurezza di WebSphere Application Server richiama il token delle credenziali dall'oggetto Subject e restituisce quindi il corrispondente Subject come oggetto Subject convalidato.

Per maggiori dettagli, fare riferimento alla documentazione dell'API per SubjectValidation e WSSubjectValidationImpl.

Configurazione dei plug-in

I plug-in SubjectValidation e SubjectSource possono essere configurati in due modi:

- **Configurazione**. È possibile utilizzare un file XML per definire un ObjectGrid e impostare i due plug-in. Di seguito è riportato un esempio di tale file, in cui la classe WSSubjectSourceImpl viene configurata come plug-in SubjectSource e la classe WSSubjectValidation viene configurata come 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>
  ```

- **Programmazione.** Se si desidera creare un ObjectGrid mediante le API, è possibile richiamare i seguenti metodi per impostare i plug-in SubjectSource o SubjectValidation.

  ```
  /**
  * Impostare il plug-in SubjectValidation per questa istanza di ObjectGrid. Un
  * plug-in SubjectValidation può essere utilizzato per verificare che
  * l'oggetto Subject
  * inviato sia un Subject valido. Fare riferimento a {@link SubjectValidation}
  * per maggiori dettagli.
  * @param subjectValidation il plug-in SubjectValidation
  */
  void setSubjectValidation(SubjectValidation subjectValidation);
  /**
  * Imposta il plug-in SubjectSource. Un plug-in SubjectSource può essere
  * utilizzato
  per richiamare un oggetto Subject dall'ambiente
  * per rappresentare
  * il client ObjectGrid.
  *
  * @param source Il plug-in SubjectSource
  */
  void setSubjectSource(SubjectSource source);
  ```

Scrittura del proprio codice di autenticazione JAAS

È possibile scrivere il proprio codice di autenticazione JAAS in modo da gestire i processi di autenticazione. È necessario scrivere i propri moduli di login e configurarli quindi per il modulo di autenticazione.

Il modulo di login riceve informazioni su un utente e lo autentica. Tali informazioni possono essere un qualsiasi dato che consenta di identificare l'utente. Ad esempio, tali informazioni possono includere un ID utente e una password, il certificato client e così via. Dopo aver ricevuto le informazioni, il modulo di login verifica che rappresentano un oggetto valido e crea quindi un oggetto Subject. Sono disponibili diverse implementazioni di moduli di login.

Una volta scritto un modulo di login, è necessario configurarlo in modo che possa essere utilizzato dal runtime. È inoltre necessario configurare un file di configurazione del modulo di login JAAS. Questo modulo di login contiene il modulo di login e il relativo schema di autenticazione. Ad esempio:

```
FileLogin
{
com.acme.auth.FileLoginModule required
};
```

Lo schema di autenticazione è ″FileLogin″ e il modulo di login è com.acme.auth.FileLoginModule. Il token richiesto indica che il modulo FileLoginModule deve verificare se il login o lo schema riporta un errore.

L'impostazione del file di configurazione del modulo di login JAAS può essere effettuata in uno dei seguenti modi:

- Impostare il file di configurazione del modulo di login JAAS in**login.config.url** nel file `java.security`, ad esempio `login.config.url.1=file:${java.home}/lib/security/file.login`
- Impostare il file di configurazione del modulo di login JAAS dalla riga comandi utilizzando gli argomenti JVM **-Djava.security.auth.login.config**, ad esempio `-Djava.security.auth.login.config ==$JAVA_HOME/lib/security/file.login`

Per ulteriori informazioni su come scrivere e configurare i moduli di login, fare riferimento a JAAS Authentication Tutorial.

Se il codice viene eseguito su WebSphere Application Server, è necessario configurare il login JAAS nella console di gestione e memorizzare quindi la configurazione di login nella configurazione del server delle applicazioni. Fare riferimento a Login configuration for Java Authentication and Authorization Service per maggiori dettagli.

Autorizzazione

Una volta autenticato il client, è possibile utilizzare i meccanismi di autorizzazione di ObjectGrid per autorizzare l'accesso ai dati delle mappe ObjectGrid e alle attività di gestione. L'autorizzazione ObjectGrid si basa sull'oggetto Subject. ObjectGrid supporta due tipi di meccanismi di autorizzazione: l'autorizzazione Java Authentication and Authorization Service (JAAS) e l'autorizzazione personalizzata.

Classe di autorizzazioni

Esistono due tipi di autorizzazioni ObjectGrid: l'autorizzazione dei dati nella mappa e l'autorizzazione per le attività di gestione. Ogni autorizzazione utilizza una classe di autorizzazioni. Le autorizzazioni per accedere alla mappa sono rappresentate

dalla classe MapPermission mentre le autorizzazioni per eseguire le attività di gestione sono rappresentate dalla classe AdminPermission.

Classe MapPermission

In ObjectGrid, viene fornita la classe pubblica com.ibm.websphere.objectgrid.security.MapPermission per rappresentare le autorizzazioni per le risorse ObjectGrid, in particolare per i metodi delle interfacce ObjectMap o JavaMap. ObjectGrid definisce le seguenti stringhe di autorizzazione per accedere ai metodi di ObjectMap e JavaMap:

- **read**: concede le autorizzazioni per leggere i dati dalla mappa. La costante intera è definita come MapPermission.READ.
- **write**: concede le autorizzazioni per aggiornare i dati nella mappa. La costante intera è definita come MapPermission.WRITE.
- **insert**: concede le autorizzazioni per inserire i dati nella mappa. La costante intera è definita come MapPermission.INSERT.
- **remove**: concede le autorizzazioni per rimuovere i dati dalla mappa. La costante intera è definita come MapPermission.REMOVE.
- **invalidate**: concede le autorizzazioni per invalidare i dati dalla mappa. La costante intera è definita come MapPermission.INVALIDATE.
- **all**: concede tutte le autorizzazioni, read, write, insert, remote e invalidate. La costante intera è definita come MapPermission.ALL.

È possibile creare un oggetto MapPermission inviando il nome completo della mappa ObjectGrid nel formato [nome_ObjectGrid].[nome_ObjectMap] e la stringa o il valore intero dell'autorizzazione. Una stringa di autorizzazione può essere una stringa delimitata da virgola delle stringhe di autorizzazione quali ″read, insert″ oppure può essere ″all″ che significa che sono concesse tutte le autorizzazioni. Un valore intero dell'autorizzazione può essere qualsiasi costante intera che un valore matematico "or" di diverse costanti di autorizzazione, come ad esempio MapPermission.GET|DGMapPermission.PUT.

L'autorizzazione si verifica quando un client richiama un metodo di ObjectMap o JavaMap. Il runtime di ObjectGrid controlla diverse autorizzazioni per metodi diversi. Se le autorizzazioni richieste non sono concesse al client, viene restituita un'eccezione AccessControlException.

Tabella 14. Elenco di metodi e relative autorizzazioni

	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)

Tabella 14. Elenco di metodi e relative autorizzazioni (Continua)

	com.ibm.websphere.objectgrid.ObjectMap **com.ibm.websphere.objectgrid.JavaMap**
write	Object put(Object key, Object value)
	void put(Object, Object, Serializable)
	void putAll(Map)
	void putAll(Map, Serializable)
	void update(Object, Object)
	void update(Object, Object, Serializable)
insert	public void insert (Object, Object)
	void insert(Object, Object, Serializable)
	remove Object remove (Object)
	void removeAll(Collection)
invalidate	public void invalidate (Object, boolean)
	void invalidateAll(Collection, boolean)
	void invalidateUsingKeyword(Serializable)
	int setTimeToLive(int)

L'autorizzazione si basa sul metodo utilizzato e non sulle operazioni eseguite dal metodo. Ad esempio, un metodo put può inserire o aggiornare un record in base al fatto se il record esiste o meno. Tuttavia, l'inserimento o l'aggiornamento non vengono distinti in questo momento.

Di tenga presente che un tipo di operazione può essere ottenuto combinando diversi tipi. Ad esempio, un aggiornamento può essere ottenuto mediante una rimozione e quindi un inserimento. Progettare le proprie politiche di autorizzazione tenendo presente questo concetto.

AdminPermission

L'autorizzazione di gestione è rappresentata dalla classe com.ibm.websphere.objectgrid.security.AdminPermission. ObjectGrid definisce due azioni di autorizzazione per le autorizzazioni di gestione:

- **admin**: concede le autorizzazioni per eseguire le attività di gestione.
- **monitor**: concede le autorizzazioni per le azioni che sono attività di gestione di sola lettura.

Le operazioni dettagliate concesse agli utenti con autorizzazioni differenti sono riportate nella seguente tabella. Tali operazioni corrispondono ai metodi nell'interfaccia ManagementMBean:

Tabella 15. Relazione tra le attività di gestione e le autorizzazioni di gestione

operazioni	**admin**	**monitor**
startServer	A	N
stopServer	A	N
forceStopServer	A	N
setServerTrace	A	N
retrieveServerStatus	A	A
getMapStats	A	A
getOGStats	A	A

Tabella 15. Relazione tra le attività di gestione e le autorizzazioni di gestione (Continua)

operazioni	admin	monitor
getReplicationStats	A	A

Se il client ha le autorizzazioni di gestione, allora può eseguire l'attività startServer, mentre se ha le autorizzazioni monitor allora non potrà eseguire l'attività startServer.

Meccanismi di autorizzazione

ObjectGrid supporta due tipi di meccanismi di autorizzazione: l'autorizzazione JAAS e l'autorizzazione personalizzata. Ciò è vero sia per l'autorizzazione di accesso ai dati delle mappe che per l'autorizzazione di gestione. L'autorizzazione JAAS si aggiunge alle politiche di sicurezza Java con i controlli agli accessi utente. Le autorizzazioni possono essere concesse in base al codice in esecuzione e all'utente di tale codice. Ciò fa parte di JDK 1.4.

ObjectGrid supporta anche l'autorizzazione personalizzata con il plug-in com.ibm.websphere.objectgrid.security.plugins.MapAuthorization e il plug-in com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization. È possibile implementare il proprio meccanismo di autorizzazione se non si desidera utilizzare l'autorizzazione JAAS. Utilizzando un meccanismo di autorizzazione personalizzato, è possibile utilizzare i database delle politiche, i server delle politiche o Tivoli Access Manager per gestire le autorizzazioni ObjectGrid.

Il meccanismo di autorizzazione di ObjectGrid può essere configurato in due modi:

- **Configurazione**. È possibile utilizzare un file XML per definire un ObjectGrid e impostare il meccanismo di autorizzazione su AUTHORIZATION_MECHANISM_JAAS o AUTHORIZATION_MECHANISM_CUSTOM. Di seguito è riportato il file `secure-objectgrid-definition.xml` utilizzato nell'applicazione enterprise di esempio di ObjectGridSample.

```
<objectGrids>
<objectGrid name="secureClusterObjectGrid" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
<bean id="TransactionCallback"
classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
...
</objectGrids>
```

- **Programmazione**. Se si desidera creare un ObjectGrid utilizzando le API, è possibile richiamare il seguente metodo per impostare il meccanismo. Questo metodo si applica solo al modello di programmazione ObjectGrid locale quando viene istanziata direttamente l'istanza ObjectGrid.

```
/**
* Imposta il meccanismo di autorizzazione. Il valore predefinito è
* com.ibm.websphere.objectgrid.security.SecurityConstants.
* AUTHORIZATION_MECHANISM_JAAS.
* @param authMechanism Il meccanismo di autorizzazione della mappa
*/
void setAuthorizationMechanism(int authMechanism);
```

Autorizzazione JAAS

Un oggetto javax.security.auth.Subject rappresenta un utente autenticato. Un oggetto è costituito da una serie di principali e ogni principale rappresenta un'identità per l'utente. Ad esempio, un oggetto può avere un principale di nome (″Joe Smith″) e un principale di gruppo (″manager″).

Mediante la politica di autorizzazione JAAS, è possibile concedere le autorizzazioni a principali specifici. ObjectGrid associa l'oggetto al contesto di controllo degli accessi corrente. Per ogni chiamata del metodo a ObjectMap o JavaMap, il runtime Java determina se la politica concede l'autorizzazione richiesta solo a un determinato principale e, in questo caso, l'operazione è consentita solo se l'oggetto associato al contesto di controllo degli accessi contiene il principale desiderato.

È necessario conoscere la sintassi della politica del file delle politiche. Per una descrizione dettagliata dell'autorizzazione JAAS, fare riferimento a JAAS Authorization Tutorial.

ObjectGrid ha una base di codice speciale utilizzata per il controllo dell'autorizzazione JAAS per le chiamate ai metodi ObjectMap e JavaMap. Questa base di codice speciale è`http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction`. Utilizzare tale codice per concedere autorizzazioni ObjectMap o JavaMap ai principali. Questo codice è stato creato in quanto il file JAR (Java archive) di ObjectGrid viene concesso con tutte le autorizzazioni.

La maschera della politica per concedere l'autorizzazione MapPermission è:

```
grant codeBase "http://www.ibm.com/com/ibm/ws/objectgrid/security/
PrivilegedAction"
<Principal field(s)>{
permission com.ibm.websphere.objectgrid.security.MapPermission
"[ObjectGrid_name].[ObjectMap_name]", "action";
....
permission com.ibm.websphere.objectgrid.security.MapPermission
"[ObjectGrid_name].[ObjectMap_name]", "action";
};
```

Un campo del principale ha il seguente aspetto:

```
Principal Principal_class "nome_principale"
```

In altre parole, la parola ″Principal″ è seguita dal nome completo di una classe del principale e da un nome del principale. Il valore di nome_mappa è il nome completo della mappa in formato [nome ObjectGrid].[nome mappa], ad esempio ″secureClusterObjectGrid.employees″. L'azione è una stringa delimitata da virgola delle stringhe di autorizzazione definite nella classe MapPermission, come ad esempio “read, insert o all”.

È supportata anche una funzione limitata dei caratteri jolly. È possibile sostituire il nome ObjectGrid o il nome della mappa con un simbolo asterisco (“*”) per indicare una qualsiasi mappa. Tuttavia, ObjectGrid non supporta la sostituzione di una parte del nome di ObjectGrid o del nome di una mappa con un asterisco. Pertanto, “*.employees”, “clusterObjectGrid.*” e “*.*” sono tutti nomi validi, mentre “cluster*.employees” non è valido.

Ad esempio, nell'applicazione di esempio `ObjectGridSample.ear` sono definiti due file delle politiche di autorizzazione, `fullAccessAuth.policy` e `readInsertAccessAuth.policy`. Di seguito è riportato il contenuto della politica `readInsertAccessAuth.policy`:

```
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";
};
```

In questa politica, vengono concesse solo le autorizzazioni ″insert″ e ″read″ alle quattro mappe di un determinato principale. L'altro file delle politiche, `fullAccessAuth.policy`, concede invece le autorizzazioni ″all″ di queste mappe a un principale. Prima di eseguire l'applicazione, modificare nome_principale e classe_principale nei valori appropriati. Il valore di nome_principale dipende dal registro utente. Ad esempio, se viene utilizzato il registro utente SO locale, il nome della macchina sarà MACH1, l'ID utente sarà user1 e il nome del principale sarà ″MACH1/user1″.

Una politica di autorizzazione JAAS può essere inserita direttamente nel file delle politiche Java o in un file di autorizzazione JAAS separato, quindi può essere impostato utilizzando l'argomento JVM `-Djava.security.auth.policy=file:[JAAS_AUTH_POLICY_FILE]` oppure l'impostazione `auth.poliyc.url.x= file:[JAAS_AUTH_POLICY_FILE]` nel file `java.security`.

La descrizione dell'autorizzazione JAAS è valida anche nel caso in cui si desidera scrivere e configurare le politiche per l'accesso delle autorizzazioni alle attività di gestione. L'unica differenza sta nel fatto che invece che utilizzare il formato ″com.ibm.websphere.objectgrid.security.MapPermission map name, actions;″, verrà utilizzato ″com.ibm.websphere.objectgrid.security.AdminPermission action;″. L'azione può essere ″admin″ o ″monitor″.

Autorizzazione delle mappe personalizzate

ObjectGrid supporta anche un meccanismo di autorizzazione delle mappe personalizzate mediante il plug-in MapAuthorization. Di seguito è riportata l'interfaccia:

```
/**Questo plugin può essere utilizzato per autorizzare ObjectMap/JavaMap ad accedere
* ai principali rappresentati dall'oggetto Subject.
*
* Un'implementazione tipica di questo plug-in consiste nel richiamare
* i principali dall'oggetto Subject e controllare quindi se
* le autorizzazioni specificate sono state concesse ai principali.
*
*
* @ibm-api
* @since WAS XD 6.0
*/
public interface MapAuthorization {
/**
* Controlla se i principali rappresentati dall'oggetto Subject
* nell'oggetto hanno l'autorizzazione MapPermission specificata. Se
* le autorizzazioni sono state concesse, allora viene restituito true,
* altrimenti viene restituito
* un valore false.
*
* @param subject l'oggetto
* @param permission le autorizzazioni per accedere a ObjectMap
*
* @return true se l'autorizzazione è concessa, altrimenti false.
*/
boolean checkPermission(Subject subject, MapPermission permission);
}
```

Questo plug-in può essere utilizzato per autorizzare gli accessi di ObjectMap o JavaMap ai principali contenuti nell'oggetto Subject. Il metodo è il seguente:

```
boolean checkPermission(Subject subject, MapPermission permission)
```

L'interfacciaMapAuthorization viene richiamata dal runtime ObjectGrid per controllare se l'oggetto inviato ha l'autorizzazione specificata. L'implementazione dell'interfaccia MapAuthorization restituisce true in questo caso, altrimenti restituisce false.

Una tipica implementazione di questo plug-in consiste nel richiamare i principali dall'oggetto Subject e verificare se sono state concesse le autorizzazioni specificate consultando politiche specifiche. Tali politiche possono essere definite dagli utenti. Ad esempio, le politiche possono essere definite in un database, in un file semplice o su un server delle politiche di Tivoli Access Manager.

ObjectGrid fornisce due diverse implementazioni predefinite per questo plug-in. La classe com.ibm.websphere.objectgrid.security.plugins.builtins. JAASMapAuthorizationImpl è una implementazione di MapAuthorization che utilizza il meccanismo JAAS per l'autorizzazione. Un'altra classe di implementazione è la classe com.ibm.websphere.objectgrid.security.plugins.builtins.TAMMapAuthorizationImpl. Essa mostra il modo in cui è possibile utilizzare Tivoli Access Manager per gestire le autorizzazioni ObjectGrid. Di seguito è riportato un frammento di codice che mostra il flusso di base del metodo JAASMapAuthorizationImpl.checkPermission(Subject, MapPermission):

```
// Crea un'azione PrivilegedExceptionAction per controllare le autorizzazioni.
PrivilegedExceptionAction action =
  MapPermissionCheckAction.getInstance(permission);
Subject.doAsPrivileged(subject, action, null);
```

Fare riferimento a IBM Tivoli Access Manager Authorization Java Classes Developer Reference per maggiori dettagli.

Non utilizzare questo plug-in TAMMapAuthorizationImpl in uno scenario predefinito. Utilizzare il plug-in solo a scopo di verifica. Esso richiede determinate condizioni restrittive:

- L'oggetto Subject contiene un principale com.tivoli.mts.PDPrincipal.
- Il server delle politiche TAM ha definite le seguenti autorizzazioni per l'oggetto ObjectMap o JavaMap. L'oggetto definito sul server delle politiche deve avere lo stesso nome di ObjectMap o JavaMap in formato [NOME_OBJECTGRID].[NOME_MAPPA]. L'autorizzazione è il primo carattere delle stringhe di autorizzazione definite in MapPermission. Ad esempio, l'autorizzazione ″r″ definita sul server delle politiche rappresenta l'autorizzazione di lettura ″read″ per ObjectMap.

Il seguente frammento di codice dimostra come implementare il metodo 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) {
// Gestione dell'eccezione
}
}
return false;
}
```

Il plug-in MapAuthorization può essere configurato nei seguenti modi:

- **Configurazione**. È possibile utilizzare ilfile XML ObjectGrid per definire un plug-in MapAuthorization. Segue un esempio:

  ```
  <objectGrids>
  <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
  ...
  <bean id="MapAuthorization"
  className="com.ibm.websphere.objectgrid.security.plugins.builtins.
  JAASMapAuthorizationImpl" />
  </objectGrids>
  ```

- **Programmazione**. Se si desidera creare un ObjectGrid utilizzando le API, è possibile richiamare il seguente metodo per impostare il plug-in di autorizzazione. Questo metodo si applica solo al modello di programmazione ObjectGrid locale quando viene istanziata direttamente l'istanza ObjectGrid.

  ```
  /**
  * Imposta il plug-in MapAuthorization per questa istanza di ObjectGrid.
  *
  * Un plug-in {@link MapAuthorization} può essere utilizzato per autorizzare
  * l'accesso alle mappe. Fare riferimento a {@link MapAuthorization}
  * per maggiori dettagli.
  * @param mapAuthorization il plug-in MapAuthorization
  */
  void setMapAuthorization(MapAuthorization mapAuthorization);
  ```

Autorizzazione di gestione personalizzata

Come il supporto dell'autorizzazione di accesso ai dati delle mappe personalizzate, ObjectGrid supporta l'autorizzazione di gestione personalizzata. Il plug-in ècom.ibm.websphere.objectgrid.security.plugins.AdminAuthorization.

```
/**
* Questo plug-in può essere utilizzato per autorizzare le operazioni di gestione
* per i principali contenuti nell'oggetto Subject. Le autorizzazioni per le
* operazioni di gestione sono rappresentate dall'oggetto
* AdminPermission.
*
* Questo plug-in viene utilizzato su un server ObjectGrid. Esso può
* essere configurato nel file XML del cluster ObjectGrid.
*
```

```
* Un'implementazione tipica di questo plug-in consiste nel richiamare
* la serie di principali dall'oggetto Subject e controllare quindi se
* le autorizzazioni specificate sono state concesse ai principali.
*
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see AdminPermission
*/
public interface AdminAuthorization {
/**
* Controlla se l'utente rappresentati dall'oggetto Subject
* ha l'autorizzazione AdminPermission specificata o meno.
*
* Se le autorizzazioni sono state concesse, allora viene restituito true,
* altrimenti viene restituito
* false.
*
* @param subject l'oggetto Subject che rappresenta l'utente
* @param permission l'autorizzazione di gestione da controllare
*
* @return true se l'autorizzazione è concessa, altrimenti false.
*
* @see AdminPermission
*/
boolean checkPermission(Subject subject, AdminPermission permission);
}
```

Questo plug-in può essere utilizzato per autorizzare gli accessi di gestione ai principali contenuti nell'oggetto Subject. Il metodo

```
boolean checkPermission(Subject subject, AdminPermission permission)
```

nell'interfaccia AdminAuthorization viene richiamato dal runtime ObjectGrid per controllare se l'oggetto Subject ha le autorizzazioni di gestione specificate. L'implementazione dell'interfaccia AdminAuthorization restituisce true in questo caso, altrimenti restituisce false.

È possibile implementare questa interfaccia in base ai requisiti di sicurezza. ObjectGrid non fornisce una classe di implementazione per questa interfaccia.

È possibile impostare il plug-in AdminAuthorization al livello cluster nell'XML del cluster. Di seguito viene riportato un esempio:

```
<cluster name="cluster1" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300"
statisticsEnabled="true"
statisticsSpec="map.all=enabled">
<serverDefinition name="server1" host="localhost"
clientAccessPort="12503" peerAccessPort="12500" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="localhost"
clientAccessPort="12504" peerAccessPort="12501" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<authenticator className ="com.ibm.websphere.objectgrid.security.plugins.
builtins.WSTokenAuthenticator"></authenticator>
<adminAuthorization className= "com.ibm.ws.objectgrid.test.security.util.
TestAdminAuthorization"></adminAuthorization>
</cluster>
```

Periodo di controllo delle autorizzazioni

ObjectGrid supporta la memorizzazione nella cache dei risultati per il controllo delle autorizzazioni delle mappe. Senza questo meccanismo, quando un metodo riportato in Tabella 14 a pagina 167 viene richiamato, il runtime ObjectGrid richiama il meccanismo di autorizzazione configurato per autorizzare l'accesso. Con questo periodo di controllo delle autorizzazioni impostato, il meccanismo di autorizzazione viene richiamato periodicamente in base al valore specificato.

Le informazioni sulle autorizzazioni vengono memorizzate nella cache in base all'oggetto Subject. Quando un client prova ad accedere ai metodi, il runtime ObjectGrid ricercherà la cache in base all'oggetto Subject. Se la cache non viene trovata, il runtime controllerà le autorizzazioni concesse per l'oggetto Subject e le memorizzerà quindi nella cache.

Il periodo di controllo delle autorizzazioni deve essere definito prima che venga inizializzato l'ObjectGrid. Il periodo di controllo delle autorizzazioni può essere configurato in due modi:

- **Configurazione**. È possibile utilizzare un file XML per definire un ObjectGrid e impostare il periodo di controllo delle autorizzazioni. Di seguito è riportato un esempio perimpostare il periodo di controllo su 45 secondi.

```
<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>
```

- **Programming**. Se si desidera creare un ObjectGrid utilizzando le API, è possibile richiamare il seguente metodo per impostare il periodo di controllo delle autorizzazioni. Questo metodo può essere richiamato soltanto prima dell'inizializzazione dell'istanza ObjectGrid. Questo metodo si applica solo al modello di programmazione ObjectGrid locale quando viene istanziata direttamente l'istanza ObjectGrid.

```
/**
* Questo metodo utilizza un unico parametro che indica la frequenza
* con cui il cliente desidera controllare le autorizzazioni utilizzate
* per consentire l'accesso a un client.
* Se il parametro
* è 0 allora ogni singola chiamata get/put/update/remove/evict
* richiede al meccanismo di autenticazione, sia esso l'autorizzazione JAAS che
* l'autorizzazione personalizzata, di controllare se l'oggetto corrente
* ha le autorizzazioni. Ciò è particolarmente costoso da un punto di vista delle
* prestazioni in base all'implementazione dell'autorizzazione ma, se richiesto,
* è necessario effettuarlo.
* In alternativa, se il parametro è > 0 allora indica il numero
* di secondi richiesti per memorizzare nella cache
* una serie di autorizzazioni prima
* di restituire al meccanismo di autorizzazione l'istruzione per aggiornarle.
* Ciò fornisce
* prestazioni migliori, ma si corre il rischio
* che, se le autorizzazioni di backend, vengono modificate durante questo
* periodo, allora l'ObjectGrid può consentire o impedire l'accesso anche
* se il provider di sicurezza di backend è stato modificato.
*
* @param period Il periodo di controllo delle autorizzazioni
* espresso in secondi.
*/
void setPermissionCheckPeriod(int period);
```

Sicurezza del cluster ObjectGrid

La sicurezza del cluster ObjectGrid garantisce che un server abbia le credenziali corrette in modo che un server non autorizzato non possa unirsi al cluster. ObjectGrid utilizza un meccanismo di stringa con segreto condiviso per questo scopo.

Tutti i server ObjectGrid confermano la stringa del segreto condiviso. Quando un server si unisce al cluster, viene richiesto di presentare la stringa segreta. Se la stringa del segreto del server che si unisce al cluster corrisponde a quella del server presidente, il server si può unire al cluster, altrimenti la richiesta di unione viene rifiutata.

L'invio di un segreto di testo semplice non è sicura. L'infrastruttura di sicurezza di ObjectGrid fornisce un plug-in di gestione dei token sicuri per consentire al server di "proteggere" il segreto prima di inviarlo. È soltanto necessario decidere come implementare l'operazione di protezione. ObjectGrid fornisce un'implementazione predefinita in cui l'operazione di protezione viene implementata per crittografare e firmare il segreto.

La stringa del segreto (authenticationSecret) è impostata nel file `security.ogserver.props`:

- `authenticationSecret`: la stringa del segreto da richiedere al server. Quando un server viene avviato, deve presentare questa stringa al server presidente. Se la stringa del segreto corrisponde a quella del server presidente, il server potrà unirsi al cluster.

Plug-in SecureTokenManager

Un plug-in del gestore di token sicuri è rappresentato dall'interfaccia com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager. Di seguito è riportata l'interfaccia:

```
package com.ibm.websphere.objectgrid.security.plugins;
import com.ibm.websphere.objectgrid.security.ObjectGridSecurityException;
import com.ibm.websphere.objectgrid.security.SecurityConstants;
/**
* Questa interfaccia è utilizzata dai server ObjectGrid per trasformare un oggetto
* in un token sicuro e viceversa. Un token sicuro è un'array di byte.
* Di seguito è riportato un esempio di possibile utilizzo: quando un server
* si unisce al cluster,
* il server deve presentare una password al server presidente nel
* cluster. Prima di inviare la password, il server che si unisce richiama
* il metodo generateToken(Object) per generare un token per questa
* password. Il token non deve essere semplice da indovinare in modo
* che la password sia protetta
* in maniera sicura. Il token verrà quindi inviato. Di solito il token è
* associato un formato orario in modo da ridurre gli attacchi non autorizzati.
* Sul lato di ricezione, il server richiama il metodo verifyToken(byte[])
* per verificare il token e ricostruire il corrispondente oggetto
* dal token.
*
* ObjectGrid utilizza JCE per fornire un'implementazione predefinita di
* questa interfaccia. In questa implementazione, durante la generazione del token, l'oggetto è
* crittografato con un formato orario e quindi firmato. Per verificare un token, la
* firma del token viene verificata e quindi decrittata. Questa implementazione avrà bisogno
* di un archivio chiavi configurato sui server ObjectGrid per supportare la
* crittografia e la decrittografia dei dati e la firma e la verifica. Utilizzare
* security.ogserver.props per le impostazioni delle chiavi dei token sicuri.
*
* Una classe di implementazione deve avere un costruttore predefinito.
* Gli utenti possono impostare la proprietà CustomSecureTokenManagerProps
```

```
* nel file delle proprietà di configurazione della sicurezza del server.
* Questa proprietà viene impostata sull'oggetto mediante
* il metodo 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 {
/**
*
Impostare le proprietà definite dall'utente sulla produzione
*
*
* Questo metodo viene utilizzato per impostare ulteriori
* proprietà SecureTokenManager sull'oggetto. Tali proprietà
* possono essere impostate mediante la proprietà SecureTokenManagerProps
* nel file delle proprietà di configurazione della sicurezza server.
* In questo modo,  è possibile personalizzare la produzione.
*
* @param properties proprietà definite dall'utente
*/
void setProperties(String properties);
/**
* Genera il token per l'oggetto specificato.
*
* Il token generato deve essere difficile da indovinare.
*
* @param o l'oggetto da proteggere
*
* @return un token che rappresenta l'oggetto da proteggere
*
* @throws ObjectGridSecurityException se si verifica un'eccezione durante
* la generazione dell'array di byte del token
*/
byte[] generateToken(Object o) throws ObjectGridSecurityException;
/**
* Verifica il token e ricostruisce l'oggetto.
*
* @param bytes l'array di byte del token che rappresenta l'oggetto protetto.
*
* @return l'oggetto protetto
*
* @throws ObjectGridSecurityException if qualsiasi eccezione che si verifica durante
* la verifica dell'array di byte del token
*/
Object verifyToken(byte[] bytes) throws ObjectGridSecurityException;
}
```

Il metodo generateToken(Object) considera un oggetto da proteggere e genera quindi un token che non può essere interpretato da altri. Il metodo verifyTokens(byte[]) non inverte il processo: esso converte il token sull'oggetto originale.

Una semplice implementazione SecureTokenManager consiste nell'utilizzare un un algoritmo di codifica (come l'algoritmo XOR) per codificare l'oggetto in forma serializzata e utilizzare quindi l'algoritmo di decodifica corrispondente per decodificare il token. Questa implementazione non è sicura ed è semplice da interrompere.

ObjectGrid fornisce un'implementazione predefinita per questa interfaccia. L'implementazione non è un'API pubblica ed è trasparente all'utente.

L'implementazione predefinita utilizza una coppia di chiavi per firmare e verificare la firma e utilizza quindi una chiave segreta per crittografare il contenuto. Per far ciò, ogni server ha un archivio chiavi di tipo JCKES per memorizzare la coppia di chiavi (chiave pubblica e chiave privata) e una chiave segreta. L'archivio chiavi deve essere di tipo JCKES per memorizzare le chiavi segrete.

Queste chiavi sono utilizzate per crittografare e firmare o verificare la stringa del segreto sull'estremità di invio. Inoltre, il token è associato a un orario di scadenza, pertanto scade dopo un certo periodo di tempo. Sul lato di ricezione, i dati vengono verificati, decrittati e confrontati alla stringa del segreto del ricevente. I protocolli di comunicazione come SSL non sono richiesti tra una coppia di server per l'autenticazione, in quanto le chiavi private e le chiavi pubbliche hanno lo stesso scopo. Tuttavia, se la comunicazione server non è crittografata, i dati possono essere intercettati durante la comunicazione. Poiché il token scade presto, la minaccia di attacchi non autorizzati è ridotta al minimo. Questa possibilità è notevolmente ridotta se tutti i server utilizzano un firewall.

Lo svantaggio di questo approccio è che gli amministratori di ObjectGrid devono generare le chiavi e trasportarle sui server, il che può provocare dei problemi alla sicurezza.

Configurazioni

Per utilizzare il gestore dei token sicuri, è necessario configurare le seguenti proprietà nel file `security.ogserver.props`:

- Proprietà **secureTokenManagerType**: questa proprietà indica il gestore dei token sicuri da utilizzare.
  - Se il valore è **none**, non viene utilizzato alcun gestore di token sicuri.
  - Se il valore è **default**, viene utilizzato il gestore di token sicuri predefinito.
  - Se il valore è**custom**, viene utilizzato il gestore di token sicuri fornito dall'utente.
- Proprietà **customSecureTokenManagerClass**: questa proprietà specifica la classe di implementazione SecureTokenManager. Essa viene utilizzata solo se il valore di secureTokenManagerType è ″custom″. La classe di implementazione deve avere un costruttore predefinito da istanziare.
- Proprietà **customSecureTokenManagerProps**: questa proprietà specifica la proprietà personalizzata SecureTokenManager. Essa viene utilizzata solo se il valore di secureTokenManagerType è ″custom″. Il valore è impostato su SecureTokenManager con il metodo setProperties(String).
- Se il valore disecureTokenManagerType è impostato su default, allora sono necessarie le seguenti configurazioni per le chiavi di firma e cifratura:
  - `secureTokenKeyStore`: specifica il nome del file per l'archivio chiavi che memorizza la coppia di chiavi pubblica e privata e la chiave segreta.
  - `secureTokenKeyStoreType`: specifica il tipo di archivio chiavi, ad esempio JCKES. È possibile impostare questo valore in base al providerJava Secure Socket Extension (JSSE) utilizzato. Tuttavia, questo archivio chiavi deve essere in grado di supportare le chiavi segrete.
  - `secureTokenKeyStorePassword`: specifica la password per proteggere l'archivio chiavi.
  - `secureTokenKeyPairAlias`: specifica l'alias della coppia di chiave pubblica e privata per la firma e la verifica.
  - `secureTokenKeyPairPassword`: specifica la password per proteggere l'alias della coppia di chiavi per la firma e la verifica.

- `secureTokenSecretKeyAlias`: specifica l'alias della chiave segreta utilizzata per la cifratura.
- `secureTokenSecretKeyPassword`: specifica lapassword per proteggere la chiave segreta.
- `secureTokenCipherAlgorithm`: specifica l'algoritmo utilizzato per la cifratura. È possibile impostare questo valore in base al provider JSSE utilizzato.
- `secureTokenSignAlgorithm`: specifica l'algoritmo utilizzato per la firma dell'oggetto. È possibile impostare questo valore in base al provider JSSE utilizzato.

Sicurezza gateway

Un gateway di gestione ObjectGrid funziona da punto di delega delle richieste di gestione client al server ObjectGrid. In questa sezione viene descritto come proteggere l'accesso al gateway.

Il seguente diagramma rappresenta un esempio. Se il client ObjectGrid desidera raccogliere le statistiche da un cluster, questo invia prima una richiesta al gateway. Il gateway invia quindi la richiesta a entrambi i server per ottenere le statistiche, quindi le combina. Le statistiche combinate vengono quindi restituite al client.

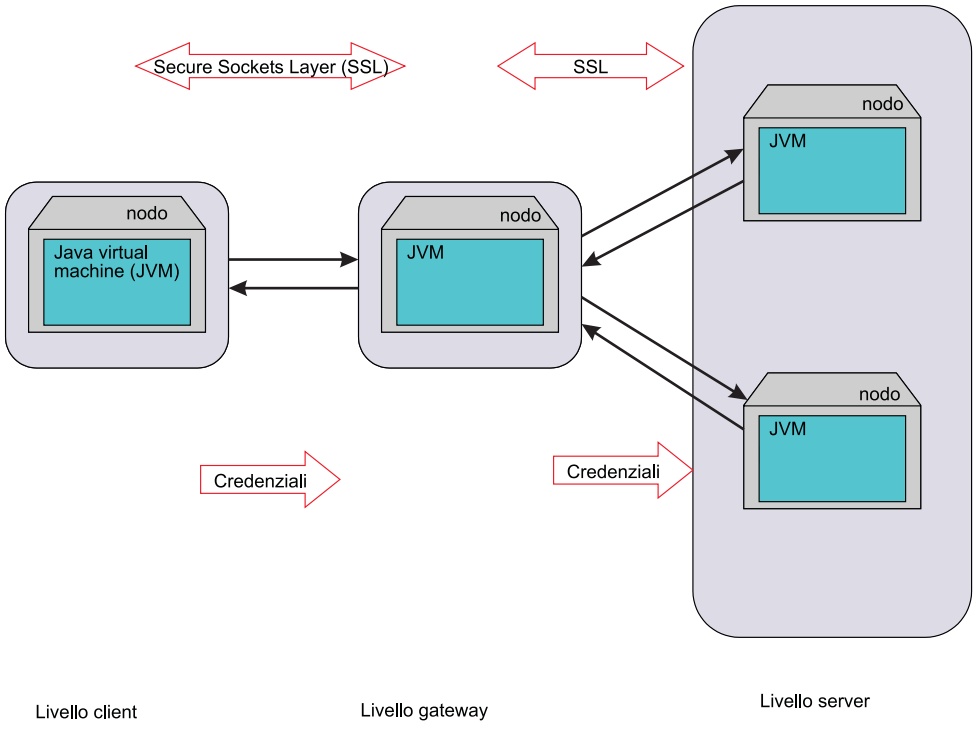


Figura 18. Sicurezza gateway

La comunicazione tra il gateway e il server utilizza il meccanismo di comunicazione client-server di ObjectGrid. Il gateway viene trattato come un client ObjectGrid. La comunicazione tra il client e il gateway può essere protetta mediante SSL. Questa funzione è fornita dal livello di connettore JMX, che è un progetto open source mx4j. Perché il gateway funzioni, ObjectGrid richiede mx4j.

Per l'autenticazione, il gateway propaga le credenziali, ad esempio un ID utente e una password, presentate dal client al server. Sui server ObjectGrid viene eseguita sia l'autenticazione che l'autorizzazione.

L'autenticazione con certificato client per il client del gateway non è supportata.

Sicurezza server gateway

Un server gateway è un client ObjectGrid. Tutti gli aspetti della sicurezza sono gli stessi su un client ObjectGrid. Fare riferimento a "Avvio del server gateway di gestione" a pagina 86 per maggiori dettagli su come avviare un server gateway da una riga comandi.

Il seguente frammento di codice dimostra come avviare il gateway sicuro in maniera programmatica:

```
// Richiamare ClientSecurityConfiguration dal file delle proprietà di sicurezza del client
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration("etc/test/security/security.client.props");
CredentialGenerator creGen = new UserPasswordCredentialGenerator("admin",
"xxxxxx");
csConfig.setCredentialGenerator(credGen);
// Inizializzare il 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");
// Impostare l'oggetto ClientSecurityConfiguration
gateway.setCsConfig(csConfig);
// Avviare il gateway
gateway.startConnector();
```

Nel codice precedente, viene creato un oggetto ClientSecurityConfiguration che viene poi impostato sull'istanza ManagementGateway.

Sicurezza client gateway

Il client gateway deve inviare le credenziali a un server gateway in fase di connessione. Il seguente frammento di codice dimostra come inviare le credenziali:

```
/**
* richiamare lo stato del server dal 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);
// Creare JMXCconnectorServer
JMXConnector cntor = JMXConnectorFactory.newJMXConnector(url, null);
// La mappa dell'ambiente di connessione
Map environment = new HashMap();
// Creare le credenziali
UserPasswordCredential gatewayClientCred =
new UserPasswordCredential("admin", "admin1");
environment.put(JMXConnector.CREDENTIALS, gatewayClientCred);
// Connect and invoke an operation on the remote MBeanServer
try {
cntor.connect(environment);
}
catch (SecurityException x) {
// Uh-oh ! Bad credentials !
```

```
throw x;
}
// Ottenere uno stub per il 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;
}
```

In questo frammento di codice, viene creato un oggetto gatewayClientCred che viene inserito nell'ambiente. Questo ambiente è quindi utilizzato per collegarsi al server gateway.

Se si desidera utilizzare SSL per collegarsi dal client gateway al server gateway, è necessario utilizzare le proprietà di sistema per memorizzare l'archivio sicuro e la relativa password. Ad esempio, è possibile inviare le seguenti proprietà quando si avvia un client gateway.

- `-Djavax.net.ssl.trustStore=etc/test/security/client.public`
- `-Djavax.net.ssl.trustStorePassword=public`

Fare riferimento al sito WebMX4J - Open Source Java Management Extensions per ulteriori informazioni.

Integrazione della sicurezza con WebSphere Application Server

ObjectGrid fornisce diverse funzioni di sicurezza da integrare con l'infrastruttura di sicurezza di WebSphere Application Server.

Integrazione della sicurezza di ObjectGrid distribuito con WebSphere Application Server

Per il modello ObjectGrid distribuito, l'integrazione della sicurezza può essere eseguita utilizzando le seguenti classi:

- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator.`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential`

  .

Tali classi sono descritte in "Sicurezza server client" a pagina 141. Di seguito è riportato un esempio di come utilizzare la classe WSTokenCredentialGenerator.

```
/**
* connessione al server ObjectGrid.
*/
protected ClientClusterContext connect() throws ConnectException {
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration(proFile);
CredentialGenerator gen = getWSCredGen();
csConfig.setCredentialGenerator(gen);
```

```
return objectGridManager.connect(csConfig, null);
}
/**
* Richiamo di un WSTokenCredentialGenerator
*
private CredentialGenerator getWSCredGen() {
WSTokenCredentialGenerator gen = new WSTokenCredentialGenerator(
WSTokenCredentialGenerator.RUN_AS_SUBJECT);
return gen;
}
```

Sul server, WSTokenAuthentication può essere utilizzato come programma di autenticazione per autenticare l'oggetto WSTokenCredential.

Integrazione della sicurezza di ObjectGrid localecon WebSphere Application Server

Per il modello ObjectGrid locale, l'integrazione della sicurezza può essere eseguita utilizzando le seguenti due classi:

- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl`

Per ulteriori informazioni su tali classi, fare riferimento a "Sicurezza ObjectGrid locale" a pagina 160. È possibile configurare la classe WSSubjectSourceImpl come plug-in SubjectSource e la classe WSSubjectValidationImpl come plug-in SubjectValidation.

Listener

ObjectGrid fornisce due interfacce di tipo listener estensibili. Le estensioni fanno sempre riferimento a un'interfaccia e descrivono le operazioni eseguite su un'istanza di ObjectGrid o un'istanza di una mappa.

Interfaccia ObjectGridEventListener

Utilizzare l'interfaccia ObjectGridEventListener per ricevere le notifiche relative a quando si verifica un evento significativo su un ObjectGrid. Tra questi eventi vi sono l'inizializzazione di ObjectGrid, l'inizio o la fine di un a transazione e l'eliminazione di un ObjectGrid. Per ascoltare questi eventi, creare una classe che implementi l'interfaccia ObjectGridEventListener e che la aggiunga all'ObjectGrid.

L'interfaccia ObjectGridEventListener

L'interfaccia ObjectGridEventListener utilizza i metodi riportati di seguito. Tali metodi vengono richiamati quando si verificano determinati eventi sull'ObjectGrid.

```
/**
* Questo metodo viene richiamato quando viene inizializzato l'ObjectGrid stesso.
* Un'istanza di sessione utilizzabile viene inviata al listener per consentire
* una risposta facoltativa al LogSequence ricevuto in una mappa.
*
* @param session L'istanza della sessione a cui è associato questo listener.
*/
void initialize(Session session);
/**
* Questo evento segnala l'inizio di una transazione (sessione).
* Una versione stringata del TxID è fornita per
* la correlazione con la fine della transazione
* (sessione), se desiderato. Il tipo di
```

```
* transazione (sessione) è
* fornito anche mediante il parametro booleano isWriteThroughEnabled.
*
* @param txid Versione stringata del TxID
* @param isWriteThroughEnabled Indicatore booleano che definisce se
* la sessione è stata avviata via beginNoWriteThrough
*/
void transactionBegin(String txid, boolean isWriteThroughEnabled);
/**
* Questo evento segnala la fine di una transazione (sessione). Una versione stringata
* del TxID è fornita per la correlazione con
* l'inizio della transazione
* (sessione), se desiderato. Vengono riportate anche le modifiche. L'uso tipico di
* questi eventi sono per i clienti che eseguono un'invalidazione del peer
* personalizzata o un push del commit del peer. Questo listener di eventi
* restituisce le modifiche. I richiami a questo metodo vengono effettuati
* in seguito al commit e vengono riportati in sequenza in modo che
* possano essere distribuiti uno a uno,
* non in parallelo. L'ordine degli eventi è l'ordine del commit.
*
* @param txid Versione stringata del TxID
* @param isWriteThroughEnabled Un indicatore booleano che definisce
* se la sessione
* è stata avviata via beginNoWriteThrough
* @param committed Un indicatore booleano che definisce se per la sessione
* è stato eseguito un commit
* (true) o un rollback (false)
* @param changes Una raccolta di LogSequences che sono state
* elaborate per la sessione corrente.
*/
void transactionEnd(String txid, boolean isWriteThroughEnabled,
boolean committed, Collection /*/* <LogSequence> */*/ changes);
/**
* Questo metodo verrà richiamato quando l'ObjectGrid viene eliminato. Esso
* è l'opposto di initialize. Quando viene richiamato questo metodo
* ObjectGridEventListener può liberare le risorse che utilizza.
*/
void destroy();
```

Aggiunta e rimozione di oggetti ObjectGridEventListeners

Un ObjectGrid può avere più ObjectGridEventListeners. Sono presenti due metodi su ObjectGrid che consentono l'aggiunta di ObjectGridEventListeners. Gli ObjectGridEventListeners che sono stati aggiunti possono essere rimossi da un ObjectGrid.

Il metodo addEventListener può essere utilizzato per aggiungere ObjectGridEventListener a un ObjectGrid.

```
/**
* Aggiungere un EventListener alla sessione. Gli eventi significativi
* verranno comunicati ai listener interessati mediante questa richiamata.
* È possibile registrare più listener degli eventi senza
* che questo influenzi l'ordine delle notifiche eventi.
*
* Nota, questo metodo può essere richiamato prima e dopo
* il metodo {@link ObjectGrid#initialize()}.
*
* @param cb Un'istanza di ObjectGridEventListener
*/
void addEventListener(ObjectGridEventListener cb);
```

Per aggiungere un elenco diObjectGridEventListeners, utilizzare il metodo setEventListeners:

```
/**
* Questo metodo sovrascrive l'elenco corrente di richiamate e lo sostituisce con
* l'elenco fornito di richiamate.
```

```
*
* Nota, questo metodo può essere richiamato prima e dopo
* il metodo {@link ObjectGrid#initialize()}.
* @param callbacks
*/
void setEventListeners(List callbacks);
```

Per rimuovere un ObjectGridEventListener da un ObjectGrid utilizzare il metodo removeEventListener:

```
/**
* Rimuove un EventListener dalla sessione. Se l'EventListener desiderato
* non si trova sulla sessione, non verrà restituito alcun errore.
*
* Nota, questo metodo può essere richiamato prima e dopo
* il metodo {@link ObjectGrid#initialize()}.
* @param cb Un'istanza di ObjectGridEventListener
*/
void removeEventListener(ObjectGridEventListener cb);
```

Creazione di un listener di eventi ObjectGrid personalizzato

Per utilizzare un listener di eventi ObjectGrid personalizzato, creare prima una classe che implementi l'interfaccia com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener. Aggiungere il listener personalizzato a un ObjectGrid per ricevere la notifica relativa a eventi significativi. Un ObjectGridEventListener può essere configurato in maniera programmatica o con XML:

- **In maniera programmatica.** Si assuma che il nome della classe del listener di eventi di ObjectGrid sia com.company.org.MyObjectGridEventListener. Questa classe implementa l'interfaccia ObjectGridEventListener. Il seguente frammento di codice crea la classe ObjectGridEventListener personalizzata e l'aggiunge a ObjectGrid:

  ```
  ObjectGridManager objectGridManager =
   ObjectGridManagerFactory.getObjectGridManager();
  ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
  MyObjectGridEventListener myListener = new MyObjectGridEventListener();
  myGrid.addEventListener(myListener);
  ```

- **Con XML.** Un ObjectGridEventListner può essere configurato anche mediante XML. Il seguente codice XML crea una configurazione che è equivalente a quella descritta dal listener di eventi ObjectGrid creata dal programma. Il seguente testo deve trovarsi nel file `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>
  ```

  Fornire questo file a ObjectGridManager per facilitare la creazione di questa configurazione. Il seguente frammento di codice dimostra come creare un ObjectGrid utilizzando questo file XML. L'ObjectGrid creato ha una classe ObjectGridEventListener impostata su myGrid ObjectGrid.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
objectGridManager.createObjectGrid("myGrid", new URL(
"file:etc/test/myGrid.xml"), true, false);
```

Visualizzazione delle modifiche in una mappa

Il metodo ObjectGridEventListener#transactionEnd è particolarmente utile per le applicazioni che desiderano visualizzare le voci nelle mappe locali. Un'applicazione può aggiungere uno di questi listener e utilizzare quindi il metodo transactionEnd per verificare se le voci sono state modificate. Ad esempio, se ObjectGrid funziona in modalità distribuita, l'applicazione può visualizzare le modifiche in arrivo. Si assuma che le voci replicate siano relative a prezzi della merce più recenti. Questo listener visualizza l'arrivo delle modifiche a l'aggiornamento di una seconda mappa che conserva il valore di una posizione in un portfolio. Il listener deve apportare tutte le modifiche utilizzando la sessione fornita al listener nel metodo ObjectGridEventListener#initialize. Il listener distingue tra modifiche locali e modifiche remote in ingresso di solito verificando se la transazione è di scrittura. Le modifiche in ingresso da ObjectGrids del peer sono sempre in scrittura.

Interfaccia MapEventListener

Utilizzare l'interfaccia MapEventListener per ricevere gli eventi significativi relativi a una mappa. Gli eventi vengono inviati a MapEventListener quando una voce viene esclusa dalla mappa e viene completato il precaricamento di una mappa.

Interfaccia MapEventListener

L'interfaccia MapEventListener utilizza i metodi riportati di seguito. Implementare l'interfaccia com.ibm.websphere.objectgrid.plugins.MapEventListener per creare un MapEventListener personalizzato.

```
/**
* Questo metodo viene richiamato quando la voce specificata viene esclusa
* dalla mappa. L'esclusione può verificarsi a causa del programma di esclusione
* oppure richiamando uno dei metodi di invalidazione di
* ObjectMap.
*
* @param key La chiave per la voce della mappa che è stata esclusa.
* @param value Il valore della voce della mappa che è stata esclusa. Il valore
* non deve essere modificato.
*
*/
void entryEvicted(Object key, Object value);
/**
* Questo metodo viene richiamato una volta completato
* il precaricamento di questa mappa.
*
* @param t Un oggetto Throwable che indica se il precaricamento è stato completato
* senza alcuna eccezione Throwable durante l'elaborazione. Un riferimento null
* indica un precaricamento completato senza che si sia verificato
* alcun oggetto Throwable
* durante il precaricamento della mappa.
*/
void preloadCompleted( Throwable t );
```

Aggiunta e rimozione di MapEventListeners

I seguenti metodi BackingMap consentono l'aggiunta e la rimozione di MapEventListeners da una mappa:

```
/**
* Aggiunge un MapEventListener a questo BackingMap.
*
* Nota, questo metodo può essere richiamato prima e dopo
* il metodo ObjectGrid.initialize().
* @param eventListener Un riferimento non na un MapEventListener da aggiungere
* all'elenco.
*
* @throws IllegalArgumentException se eventListener è null.
*
* @see MapEventListener
*/
public void addMapEventListener(MapEventListener
 eventListener );
/**
* Imposta l'elenco di oggetti MapEventListener.
*
* Se questo BackingMap ha già un elenco di
* MapEventListeners, allora tale elenco viene sostituito
* dall'elenco inviato come argomento dal richiamo corrente
* di questo metodo. Questo metodo può essere richiamato
* dopo il metodo ObjectGrid.initialize().
*
* @param eventListenerList Un riferimento non null a un elenco
* di oggetti MapEventListener.
*
* @throws IllegalArgumentException viene emesso se
* eventListenerList è null
* o se eventListenerList contiene un riferimento null
* o un oggetto che non è un'istanza di
* MapEventListener.
*
* @see MapEventListener
*/
public void setMapEventListeners( List /*MapEventListener*/
 eventListenerList );
/**
* Rimuove un MapEventListener da questo BackingMap.
*
* Nota, questo metodo può essere richiamato prima e dopo
* il metodo ObjectGrid.initialize().
*
* @param eventListener Un riferimento non null a un listener degli eventi
* precedentemente aggiunto richiamando il
* metodo addMapEventListener(MapEventListener) o
* setMapEventListeners(List) di questa interfaccia.
*
* @throws IllegalArgumentException se eventListener è null.
*
* @see MapEventListener
*/
public void removeMapEventListener(MapEventListener eventListener );
```

Creazione di un MapEventListener

Per creare un MapEventListener personalizzato, implementare l'interfaccia com.ibm.websphere.objectgrid.plugins.MapEventListener. Per utilizzare MapEventListener, aggiungerla a un BackingMap. Un MapEventListener può essere creato e configurato in maniera programmatica o mediante XML:

- **In maniera programmatica**. Il nome della classe per il MapEventListener personalizzato è com.company.org.MyMapEventListener. Questa classe implementa l'interfaccia MapEventListener. Il seguente frammento di codice crea il MapEventListener personalizzato e lo aggiunge a un BackingMap:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap myMap = myGrid.defineMap("myMap");
MyMapEventListener myListener = new MyMapEventListener();
myMap.addMapEventListener(myListener);
```

- **Creazione XML**. Un MapEventListner può essere configurato anche mediante XML. Il seguente codice XML crea una configurazione che è equivalente a quella della precedente creazione programmatica. Il seguente testo deve trovarsi nel file `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>
```

  Fornire questo file a ObjectGridManager per facilitare la creazione di questa configurazione. Il seguente frammento di codice dimostra come creare un ObjectGrid utilizzando questo file XML. L'ObjectGrid appena creato ha un MapEventListener impostato su myMap BackingMap.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
objectGridManager.createObjectGrid("myGrid", new URL(
"file:etc/test/myGrid.xml"), true, false);
```

Programmi di esclusione

ObjectGrid fornisce un meccanismo di esclusione predefinito. È inoltre possibile fornire un meccanismo del programma di esclusione collegabile.

Un *programma di esclusione* controlla l'appartenenza delle voci in ogni BackingMap. Il programma di esclusione predefinito utilizza una politica *time to live* per ogni BackingMap. Se si utilizza un meccanismo del programma di esclusione collegabile, questo di solito utilizza una politica di esclusione basata sul numero di voci invece che sul tempo. In questa sezione vengono descritti entrambi i programmi di esclusione.

Programma di esclusione TTL predefinito

ObjectGrid fornisce un programma di esclusione TTL (time to live) per ogni BackingMap. Il programma di esclusione TTL imposta un'ora di scadenza per ogni voce che viene creata. Quando si raggiunge il valore della scadenza di una voce, il programma di esclusione elimina la voce da BackingMap. Per ridurre l'impatto sulle prestazioni, il programma di esclusione TTL attende il raggiungimento dell'orario di scadenza per escludere una voce e non la elimina mai prima che la voce scada.

BackingMap ha gli attributi utilizzati per controllare il modo in cui il programma di esclusione TTL elabora il valore di scadenza per ogni voce. Le applicazioni impostano l'attributo ttlType per specificare il modo in cui il programma di esclusione TTL calcola il valore di scadenza. L'attributo ttlType può essere impostato su uno dei seguenti valori:

- **Nessuno** indica che una voce in BackingMap non scade mai. Il programma di esclusione TTL non elimina tali voci.
- **Ora di creazione** indica che al calcolo dell'ora di scadenza contribuisce l'ora in cui la voce è stata creata.
- **Ora dell'ultimo accesso** indica che al calcolo dell'ora di scadenza contribuisce l'ora in cui è stato eseguito l'accesso alla voce.

Se l'attributo ttlType non viene impostato su un BackingMap, verrà utilizzato il valore predefinito **Nessuno** in modo che il programma di esclusione TTL non elimina alcuna voce. Se l'attributo ttlType è impostato su **ora di creazione** o **ora dell'ultimo accesso**, allora l'attributo TTL su BackingMap viene aggiunto a questi due valori temporali per elaborare l'ora di scadenza. Il valore orario dell'attributo della mappa TTL è espresso in secondi. Un valore uguale a `0` per l'attributo time to live è un valore speciale cyhe viene utilizzato per indicare che la voce della mappa non scade e che rimane sulla mappa fino a che l'applicazione la rimuove o la invalida esplicitamente.

Specifica degli attributi per i programmi di esclusione TTL

I programmi di esclusione TTL sono associati alle istanze BackingMap. Il seguente frammento di codice illustra come utilizzare l'interfaccia BackingMap per impostare gli attributi necessari in modo da creare ciascuna voce, che ha un valore di scadenza impostato su dieci minuti dopo la creazione.

```
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 );
```

L'argomento del metodo setTimeToLive è `600` e indica il valore time to live in secondi. Il codice precedente deve essere eseguito prima di richiamare il metodo initializesull'istanza ObjectGrid. Tali attributi BackingMap non possono essere modificati dopo aver inizializzato l'istanza ObjectGrid. Una volta eseguito il codice, qualsiasi voce che viene inserita in myMap BackingMap ha un valore di scadenza. Quando viene raggiunto questo valore, il programma di esclusione TTL elimina la voce.

Se un'applicazione richiede che l'ora di scadenza deve essere impostata sull'ora dell'ultimo accesso più dieci minuti, è necessario modificare una riga del codice precedente. L'argomento inviato al metodo setTtlEvictorType viene modificato da `TTLType.CREATION_TIME` a `TTLType.LAST_ACCESS_TIME`. Con questo valore, l'ora di scadenza viene elaborata come l'ora dell'ultimo accesso più 10 minuti. Quando viene creata una voce, l'ora dell'ultimo accesso coincide con l'ora di creazione.

Quando viene utilizzato il valore `TTLType.LAST_ACCESS_TIME`, è possibile utilizzare le interfacce ObjectMap e JavaMap per sovrascrivere il valore time to live di BackingMap. Questo meccanismo consente a un'applicazione di utilizzare un valore

time to live diverso per ogni voce che viene creata. Si assuma che il precedente frammento di codice sia stato utilizzato per impostare l'attributo ttlType su LAST_ACCESS_TIME e che il valore time to live sia stato impostato su 10 minuti su BackingMap. Un'applicazione può sovrascrivere il valore time to live per ogni voce eseguendo il codice riportato di seguito prima di creare o modificare una voce:

```
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" );
```

In questo frammento di codice, la voce con la chiave key1 ha un'ora di scadenza uguale all'ora di inserimento più 30 minuti come risultato del richiamo del metodo setTimeToLive(1800) sull'ObjectMap. La variabile oldTimeToLive1 è impostata su 600 in quanto il valore time to live da BackingMap viene utilizzato come valore predefinito se il metodo setTimeToLive non è stato precedentemente richiamato su ObjectMap.

La voce con la chiave key2 ha un'ora di scadenza uguale all'ora di inserimento più 20 minuti come risultato di un richiamo al metodo setTimeToLive(1200) sull'ObjectMap. La variabile oldTimeToLive2 viene impostata su 1800 in quanto il valore time to live dal precedente metodo ObjectMap.setTimeToLive la ha impostata su 1800.

Il precedente esempio mostra due voci della mappa inserite nella mappa myMap map per i valori delle chiavi key1 e key2. Successivamente, l'applicazione da un nuovo thread potrebbe aggiornare tali voci con i nuovi valori della mappa. Tuttavia, l'applicazione può conservare i valori time-to-live utilizzati in fase di inserimento per ogni voce. Il seguente esempio illustra come conservare i valori time-to-live utilizzando una costante definita nell'interfaccia ObjectMap per questo scopo:

```
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();
```

Poiché il valore speciale ObjectMap.USE_DEFAULT è utilizzato sulla chiamata del metodo setTimeToLive, key1 conserva il proprio valore time-to-live di 1800 secondi e key2 conserva il valore time-to-live di 1200 secondi in quanto tali valori sono stati utilizzati quando le voci della mappa sono state inserite dalla transazione precedente.

L'esempio precedente mostra anche l'inserimento di una nuova voce di mappa per key3. In questo caso, il valore speciale USE_DEFAULT indica l'utilizzo dell'impostazione predefinita del valore time-to-live per questa mappa. Il valore predefinito è definito dall'attributo time-to-live di BackingMap. Fare riferimento a "Attributi di BackingMap" a pagina 109 per ulteriori informazioni su come è definito l'attributo time-to-live su BackingMap.

Fare riferimento alla documentazione dell'API per il metodo setTimeToLive sulle interfacce ObjectMap e JavaMap. Essa riporta che viene restituita un'eccezione IllegalStateException se il metodo BackingMap.getTtlEvictorType() restituisce un

valore diverso da TTLType.LAST_ACCESS_TIME. ObjectMap e JavaMap possono essere utilizzati per sovrascrivere il valore time to live solo se si utilizza il tipo di programma di esclusione TTL LAST_ACCESS_TIME. Questi metodi non possono essere utilizzati per sovrascrivere il valore time to live se si utilizza il tipo di programma di esclusione TTL CREATION_TIME o il tipo NONE.

Utilizzo di un file XML per specificare gli attributi per il programma di esclusione TTL

Invece che utilizzare l'interfaccia BackingMap per impostare in maniera programmatica gli attributi BackingMap che devono essere utilizzati dal programma di esclusione TTL, è possibile utilizzare un file XML per configurare ogni BackingMap. Il seguente codice illustra come impostare questi attributi per tre diversi 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>
```

L'esempio precedente riposta che la map1 BackingMap utilizza un tipo di programma di esclusione TTL NONE. map2 BackingMap utilizza un tipo LAST_ACCESS_TIME e un valore time to live uguale a 1800 secondi (30 minuti). Infine, la mappa map3 BackingMap è definita in modo da utilizzare un tipo di programma di esclusione TTL CREATION_TIME e ha un valore time to live pari a 1200 secondi (20 minuti).

Programmi di esclusione collegabili opzionali

Il programma di esclusione TTL predefinito utilizza una politica di esclusione basata sul tempo e pertanto il numero di voci in BackingMap non ha alcun effetto sull'ora di scadenza di una voce. Un programma di esclusione collegabile opzionale può essere utilizzato per escludere le voci in base al numero di voci presenti invece che in base al tempo. I seguenti programmi di esclusione collegabili opzionali forniscono alcuni algoritmi utilizzati di frequente per decidere quali voci escludere quando un BackingMap va oltre il suo limite di dimensione.

- **LRUEvictor** è un programma di esclusione che utilizza un algoritmo *least recently used* per decidere quali voci escludere quando un BackingMap supera un numero massimo di voci consentite.
- **LFUEvictor** è un programma di esclusione che utilizza un algoritmo *least frequently used* per decidere quali voci escludere quando un BackingMap supera un numero massimo di voci consentite.

BackingMap indica a un programma di esclusione quando le voci vengono create, modifi cate o rimosse da una transazione. BackingMap tiene traccia di queste voci e decide quando eliminare delle voci da BackingMap.

Un BackingMap non contiene informazioni di configurazione per la dimensione massima consentita. Per contro, le proprietà del programma di esclusione sono impostate per controllare il funzionamento del programma stesso. Sia LRUEvictor che LFUEvictor hanno una proprietà relativa alla dimensione massima che viene

utilizzata per far sì che il programma di esclusione comnci ad eliminare le voci dopo aver raggiunto questa dimensione massima. Come il programma di esclusione TTL, i programmi di esclusione LRU e LFU potrebbero non escludere immediatamente una voce quando viene raggiunto il numero massimo di voci per ridurre l'impatto sulle prestazioni.

Se l'algoritmo del programma di esclusione LRU o LFU non è adeguato a una determinata applicazione, è possibile scrivere i propri programmi di eliminazione in modo da utilizzare la strategia di eliminazione desiderata.

Specifica di un programma di esclusione collegabile

Poiché i programmi di esclusione sono associati ai BackingMap, l'interfaccia BackingMap è utilizzata per specificare il programma di esclusione collegabile da utilizzare. Il seguente frammento di codice è un esempio di specifica di un programma di esclusione LRUEvictor per map1 BackingMap e di un programma di esclusione LFUEvictor per map2 BackingMap:

```
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);
```

Il frammento precedente mostra un programma di esclusione LRUEvictor utilizzato per map1 BackingMap con un numero massimo di voci pari a `1000`. Il programma di esclusione LFUEvictor è utilizzato da map2 BackingMap con un numero massimo di voci pari a `2000`. Sia il programma di esclusione LRU che il programma LFU hanno una proprietà sleep time che indica per quanto tempo il programma di esclusione deve rimanere inattivo prima di controllare se vi sono voci da eliminare. Il valore sleep time è specificato in secondi. Un valore di 15 secondi è un buon compromesso per ottenere buone prestazioni e impedire che la dimensione di BackingMap diventi troppo elevata. Lo scopo è utilizzare il maggior valore sleep time possibile senza che BackingMap raggiunga una dimensione eccessiva.

Il metodo setNumberOfLRUQueues imposta la proprietà LRUEvictor che indica il numero di code LRU che il programma di esclusione utilizza per gestire le informazioni LRU. Viene utilizzata una raccolta di code in modo che ogni voce non conservi le stesse informazioni LRU sulla stessa coda. Questo approccio consente di migliorare le prestazioni riducendo il numero di voci di una mappa che devono essere sincronizzate sulla stessa coda. L'aumento del numero di code è un buon modo per minimizzare l'impatto che un programma di esclusione LRU può avere sulle prestazioni. Un buon punto di partenza è avere il 10% del numero massimo di voci come numero di code. L'utilizzo di un numero primo è preferibile.

Il metodo setNumberOfHeaps imposta la proprietà LFUEvictor per definire il numero di oggetti heap binari che il programma di esclusione LFUEvictor utilizza per gestire le informazioni LFU. Una raccolta viene utilizzata per migliorare le prestazioni. L'utilizzo del 10% del numero massimo di voci è un buon inizio ed è preferibile utilizzare un numero primo.

Utilizzo di XML per specificare un programma di esclusione collegabile

Invece che utilizzare diverse API per collegare in maniera programmatica un programma di esclusione e impostarne le relative proprietà, è possibile utilizzare un file XML per configurare ogniBackingMap come riportato nel seguente esempio:

```
<?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>
```

Scrittura di un programma di esclusione personalizzato

ObjectGrid cpuò essere esteso in modo da utilizzare qualsiasi algoritmo di esclusione. È necessario creare un programma di esclusione personalizzato che implementa l'interfaccia com.ibm.websphere.objectgrid.plugins.Evictor. Di seguito è riportata l'interfaccia:

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

- Il metodo initialize viene richiamato durante l'inizializzazione dell'oggetto BackingMap. Questo metodo inizializza un plug-in del programma di esclusione

con un riferimento a BackingMap e un riferimento a un oggetto che implementa l'interfaccia com.ibm.websphere.objectgrid.plugins.EvictionEventCallback.

- Il metodo apply viene richiamato quando vengono eseguite le transazioni che accedono a una o più voci di BackingMap. Al metodo apply viene inviato un riferimento a un oggetto che implementa l'interfaccia com.ibm.websphere.objectgrid.plugins.LogSequence. L'interfaccia LogSequence consente a un plug-in Evictor di determinare le voci di BackingMap che vengono create, modificate o rimosse dalla transazione. Un programma di esclusione utilizza queste informazioni per decidere quando e quali voci escludere.
- Il metodo destroy viene richiamato quando viene eliminato il BackingMap. Questo metodo consente a un programma di esclusione di terminare qualsiasi thread che potrebbe essere stato creato.

L'interfaccia EvictionEventCallback ha i seguenti metodi:

```
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);
}
```

I metodi EvictionEventCallback sono utilizzati da un plug-in Evictor per richiamare la struttura ObjectGrid come riportato di seguito:

- Il metodo setEvictorData viene utilizzato da un programma di esclusione per richiedere la struttura utilizzata per memorizzare e associare gli oggetti che il programma crea con la voce indicata dall'argomento chiave. I dati sono specifici del programma di esclusione e sono determinati dalle informazioni di cui ha bisogno il programma di esclusione per implementare l'algoritmo utilizzato. Ad esempio, nell'algoritmo LFU (least frequently used), il programma di esclusione deve mantenere un numero per i dati in modo da tenere traccia di quante volte è stato richiamato il metodo apply con un LogElement che fa riferimento a una voce per una determinata chiave.
- Il metodo getEvictorData è utilizzato da un programma di esclusione per richiamare i dati inviati al metodo setEvictorData durante un precedente richiamo al metodo apply. Se i dati del programma di esclusione per l'argomento chiave specificato non vengono trovati, viene restituito un oggetto `KEY_NOT_FOUND` speciale definito sull'interfaccia EvictorCallback.
- Il metodo evictEntries viene utilizzato da un programma di esclusione per richiedere l'esclusione di una o più voci della mappa. Un programma di caricamento fornito da un'applicazione deve implementare l'interfaccia com.ibm.websphere.objectgrid.plugins.EvictorData. Inoltre, la stessa istanza EvictorData inviata al metodo setEvictorData deve essere presente nell'elenco di parametri dei dati dell'evictor di questo metodo. Il metodo getKey dell'interfaccia EvictorData è utilizzato per determinare quale voce della mappa eliminare. La voce della mappa viene eliminata se la voce della cache contiene la stessa istanza EvictorData presente nell'elenco di dati dell'evictor per questa voce della cache.
- Il metodo evictEntries viene utilizzato da un programma di esclusione per richiedere l'esclusione di una o più voci della mappa. Questo metodo viene utilizzato solo se l'oggetto inviato al metodo setEvictorData non implementa l'interfacciacom.ibm.websphere.objectgrid.plugins.EvictorData.

ObjectGrid richiama il metodo apply dell'interfaccia Evictor *dopo* che la transazione è stata completata. Tutti i blocchi delle transazioni che sono stati acquisiti dalla

transazione completata non sono più mantenuti. Potenzialmente, più thread possono richiamare il metodo apply allo stesso tempo e ogni thread può completare la propria transazione. Poiché i blocchi delle transazioni sono già rilasciati dalla transazione completata, il metodo apply deve fornire la propria sincronizzazione per garantire che il metodoapply sia libero da thread.

Il motivo per implementare l'interfaccia EvictorData e utilizzare il metodo evictMapEntries invece che il metodo evictEntries è la chiusura di una potenziale finestra di sincronizzazione. Considerare la seguente sequenza di eventi:

1. La transazione 1 viene completata e richiama il metodo apply con un LogSequence che elimina la voce della mappa per key 1.
2. La transazione 2 viene completata e richiama il metodo apply con un LogSequence che inserisce una nuova voce di mappa per key 1. In altre parole, la transazione 2 crea di nuovo la voce della mappa eliminata dalla transazione 1.

Poiché l'evictor viene eseguito in maniera asincrona dai thread che eseguono le transazioni, è possibile che quando l'evictor decide di eliminare key 1, elimini anche la voce della mappa che esisteva prima del completamento della transazione 1 o elimini la voce della mappa ricreata dalla transazione 2. Per eliminare le finestre di sincronizzazione ed eliminare le incertezze come ad esempio la versione della voce della mappa di key 1 che l'evictor intendeva eliminare, implementare l'interfaccia EvictorData mediante l'oggetto inviato al metodo setEvictorData. Utilizzare la stessa istanza EvictorData per l'intero ciclo di una voce di mappa. Quando tale voce della mappa viene eliminata e quindi ricreata da un'altra transazione, l'evictor deve utilizzare una nuova istanza dell'implementazione EvictorData. Utilizzando l'implementazione EvictorData e il metodo evictMapEntries, l'evictor garantisce che la voce venga eliminata se la voce della cache associata alla voce della mappa contiene l'istanza EvictorData corretta.

Le interfacce Evictor e EvictonEventCallback consentono a un'applicazione di collegarsi a un programma di esclusione che implementa un algoritmo definito dall'utente per l'esclusione. Il seguente frammento di codice illustra il modo in cui è possibile implementare il metodo initialize dell'interfaccia 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;
// Variabili di istanza
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();
 // diffusione del thread del programma di esclusione
 evictorThread = new Thread( this );
String threadName = "MyEvictorForMap-" + bm.getName();
 evictorThread.setName( threadName );
 evictorThread.start();
}
```

Il codice precedente salva i riferimenti alla mappa e gli oggetti di richiamata nelle variabili delle istanze in modo che siano disponibili per i metodi apply e destroy. In

questo esempio, viene creato un elenco collegato che viene utilizzato come coda *first in, first out* per l'implementazione di un algoritmo LRU (least recently used). Un thread viene diffuso e un riferimento ad esso viene conservato come variabile dell'istanza. Mantenendo questo riferimento, il metodo destroy può essere interrotto e terminare il thread.

Ignorando i requisiti di sincronizzazione per rendere il codice più sicuro, il seguente frammento di codice illustra come implementare il metodo apply dell'interfaccia Evictor:

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.EvictorData;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
public void apply(LogSequence sequence)
{
 Iterator iter = sequence.getAllChanges();
 while ( iter.hasNext() )
{
  LogElement elem = (LogElement)iter.next();
  Object key = elem.getCacheEntry().getKey();
  LogElement.Type type = elem.getType();
  if ( type == LogElement.INSERT )
{
   // inserire qui l'elaborazione aggiungendo alla parte anteriore della coda LRU.
   EvictorData data = new EvictorData(key);
   evictorCallback.setEvictorData(key, data);
   queue.addFirst( data );
}
  else if ( type == LogElement.UPDATE || type == LogElement.FETCH ||
   type == LogElement.TOUCH )
{
   // non aggiornare l'elaborazione spostando l'oggetto EvictorData
   // nella parte anteriore della coda.
   EvictorData data = evictorCallback.getEvictorData(key);
   queue.remove(data);
   queue.addFirst(data);
}
  else if ( type == LogElement.DELETE || type == LogElement.EVICT )
{
   // non rimuovere l'elaborazione spostando l'oggetto EvictorData
   // dalal coda.
   EvictorData data = evictorCallback.getEvictorData(key);
   if ( data == EvictionEventCallback.KEY_NOT_FOUND )
{
    // Il thread del programma di esclusione asincrono
    // ha escluso la voce della mappa prima che il thread avesse potuto
    // elabora la richiesta di LogElement. In questo caso,
    // non è necessario effettuare alcuna operazione.
}
else
{
    // La chiave è stata trovata. È quindi possibile elaborare
       i dati del programma di esclusione.
    if ( data != null )
{
     // Ignora il valore null restituito dal metodo remove poiché
     // il thread del programma di esclusione potrebbe averlo già rimosso dalla coda.
     // Questo codice è comunque necessario nel caso in cui il thread
     // del programma di esclusione faccia sì che si verifichi questo LogElement.
     queue.remove( data );
}
else
{
```

```
     // A seconda del modo in cui si scrive il programma di esclusione, è possibile
     // che ciò non si verifichi oppure può indicare un difetto nel programma di esclusione
     // dovuto a una logica di sincronizzazione dei thread non corretta.
}
}
}
}
}
```

L'elaborazione dell'inserimento nel metodo apply di solito gestisce la creazione di un oggetto dati del programma di esclusione che viene inviato al metodo setEvictorData sull'interfaccia EvictionEventCallback. Poiché questo programma di esclusione illustra un'implementazione LRU, EvictorData viene aggiunto alla parte anteriore della coda creata dal metodo initialize. L'elaborazione dell'aggiornamento nel metodo apply di solito aggiorna i dati del programma di esclusione creati da un richiamo precedente del metodo apply (ad esempio, da parte dell'elaborazione di inserimento del metodo apply). Poiché questo programma di esclusione è un'implementazione LRU, è necessario spostare l'oggetto EvictorData dalla posizione della coda corrente alla parte anteriore della coda. Il thread del programma di esclusione rimuove l'ultimo oggetto EvictorData della coda in quanto tale oggetto rappresenta la voce utilizzata meno di frequente. Si assume infatti che l'oggetto EvictorData abbia un metodo getKey in modo che il thread del programma di esclusione riconosca le chiavi delle voci che devono essere eliminate. Tenere presente che in questo esempio sono stati ignorati i requisiti di sincronizzazione per rendere il codice sicuro. Un programma di esclusione personalizzato reale è molto più complesso in quanto ha a che fare con i colli di bottiglia dovuti alla sincronizzazione e alle prestazioni che si verificano come risultato dei punti di sincronizzazione.

I seguenti frammenti di codice illustrano il metodo destroy e il metodo run del thread eseguibile diffuso dal metodo initialize.

```
// Il metodo destroy interrompe semplicemente il thread diffuso dal metodo initialize.
public void destroy() {
 evictorThread.interrupt();
}
// Questo è il metodo run del thread diffuso dal metodo initialize.
 public void run()
{
 // Il loop continua fino a che il metodo destroy interrompe il thread.
 boolean continueToRun = true;
 while ( continueToRun )
{
try
{
   // Inattivo per un periodo di tempo mentre si ripulisce la coda.
   // Il valore sleepTime è un buon valore per l'impostazione della
   // proprietà del programma di esclusione.
   Thread.sleep( sleepTime );
   int queueSize = queue.size();
   // Esclude le voci se la dimensione della coda è cresciuta oltre la
   // dimensione massima. Naturalmente, la dimensione massima
   // è un'altra proprietà del programma di esclusione.
   int numToEvict = queueSize - maxSize;
if ( numToEvict > 0 )
{
    // Rimuove la parte posteriore della coda che contiene
    // le voci utilizzate meno di recente.
    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 )
{
      // La coda è vuota.
      queueSize = 0;
}
}
    // Richiede un'eliminazione se l'elenco di chiavi non è vuoto.
    if ( ! evictList.isEmpty() )
{
evictorCallback.evictMapEntries( evictList );
}
}
}
  catch ( InterruptedException e )
{
   continueToRun = false;
}
 } // fine del loop
} // fine del metodo run.
```

Interfaccia RollBackEvictor facoltativa

L'interfaccia com.ibm.websphere.objectgrid.plugins.RollbackEvictor può essere facoltativamente implementata da un plug-in dell'Evictor. Implementando questa interfaccia, un evictor può essere richiamato non solo quando viene eseguito il commit delle transazioni, ma anche quando ne viene eseguito il rollback.

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

Il metodo apply viene richiamato solo se viene eseguito il commit di una transazione. Se viene eseguito il rollback di una transazione e l'interfaccia RollbackEvictor viene implementata dall'evictor, viene richiamato il metodo rollingBack. Se l'interfaccia RollbackEvictor non viene implementata e viene eseguito il rollback della transazione, il metodo apply e il metodo rollingBack non vengono richiamati.

Programmi di caricamento

Un programma di caricamento di ObjectGrid è un componente collegabile che consente a una mappa ObjectGrid di funzionare come cache di memoria per i dati che vengono normalmente conservati in un archivio permanente sullo stesso sistema o su un sistema differente.

Di solito, come archivio permanente viene utilizzato un database o un file system. È possibile utilizzare anche una JVM (Java virtual machine) remota come origine dei dati, consentendo la creazione di cache basate su hub mediante ObjectGrid. Un programma di caricamento utilizza la logica di lettura e scrittura dei dati su un archivio permanente.

Un programma di caricamento è un plug-in opzionale per una mappa di backup di ObjectGrid. Soltanto un programma di caricamento può essere associato a una determinata mappa e ogni mappa ha la propria istanza del programma di caricamento. La mappa di backup richiede tutti i dati che non sono forniti mediante

il programma di caricamento. Qualsiasi modifica apportata alla mappa viene inserita mediante il programma di caricamento. Il plug-in del programma di caricamento fornisce un modo per spostare i dati tra la mappa e l'archivio permanente.

Collegamento di un programma di caricamento

Il seguente frammento di codice illustra il modo in cui un programma di caricamento fornito dall'applicazione viene collegato alla mappa di backup map1 utilizzando l'API 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 );
```

Si assume che `MyLoader` sia la classe fornita dall'applicazione che implementa l'interfaccia com.ibm.websphere.objectgrid.plugins.Loader. Poiché l'associazione di un programma di caricamento a una mappa di backup non può essere modificata una volta inizializzato ObjectGrid, il codice deve essere eseguito prima di richiamare il metodo `initialize` dell'interfaccia ObjectGrid richiamata. Un'eccezione `IllegalStateException` si verifica su una chiamata al metodo `setLoader` se questo viene richiamato in seguito all'inizializzazione.

Il programma di caricamento fornito dall'applicazione può avere delle proprietà impostate. Nell'esempio, il programma di caricamento MyLoader viene utilizzato per leggere e scrivere i dati da una tabella in un database relazionale. Il programma di caricamento deve avere il nome del database e il livello di isolamento SQL da utilizzare. Il programma di caricamento MyLoader ha i metodi `setDataBaseName` e `setIsolationLevel` che consentono all'applicazione di impostare queste due proprietà.

Un programma di caricamento fornito dall'applicazione può essere collegato anche utilizzando un file XML. Il seguente esempio illustra come viene collegato il programma di caricamento MyLoader alla mappa map1 con lo stesso nome del database e livello di isolamento delle proprietà del programma di caricamento impostato:

```
<?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>
```

Implementazione dell'interfaccia del programma di caricamento

Un programma di caricamento fornito da un'applicazione deve implementare l'interfaccia *com.ibm.websphere.objectgrid.plugins.Loader*. Tale interfaccia ha la seguente definizione:

```
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;
}
```

Le sezioni seguenti forniscono una spiegazione e le considerazioni relative all'implementazione di di ognuno dei metodi sull'interfaccia del programma di caricamento.

Metodo get

La mappa di backup richiama il metodo `get` del programma di caricamento per richiamare i valori associati a un elenco di chiavi inviati come argomento **keyList**. Il metodo get è richiesto per restituire un elenco java.lang.util.List di valori, uno per ogni chiave riportata nell'elenco. Il primo valore restituito nell'elenco corrisponde alla prima chiave nell'elenco, il secondo valore restituito corrisponde alla seconda chiave e così via. Se il programma di caricamento non trova il valore per una chiave nell'elenco, il programma deve restituire il valore speciale `KEY_NOT_FOUND` definito nell'interfaccia del programma di caricamento. Poiché una mappa di backup può essere configurata per consentire un valore `null` come valore valido, è molto importante che il programma di caricamento restituisca il valore speciale `KEY_NOT_FOUND` quando non rileva la chiave. Questo valore consente alla mappa di distinguere tra un valore `null` e un valore non esistente poiché la chiave non è stata trovata. Se una mappa di backup non supporta i valori `null`, un programma di caricamento che restituisce null invece che l'oggetto `KEY_NOT_FOUND` per una chiave che non esiste restituisce un'eccezione.

L'argomento **forUpdate** indica al programma di caricamento se l'applicazione ha richiamato un metodo `get` o un metodo `getForUpdate` sulla mappa. Fare riferimento all'interfacciacom.ibm.websphere.objectgrid.ObjectMap per ulteriori informazioni. Il programma di caricamento è responsabile dell'implementazione di una politica di controllo della simultaneità che controlla gli accessi simultanei all'archivio permanente. Ad esempio, molti sistemi di gestione dei database relazionali supportano la sintassi `for update` sull'istruzione SQL select utilizzata per leggere i dati da una tabella relazionale. Il programma di caricamento può decidere di utilizzare la sintassi `for update` sull'istruzione SQL `select` in base al fatto se il valore booleano `true` viene inviato come valore dell'argomento per il parametro **forUpdate** di questo metodo. Di solito, il programma di caricamento utilizza la sintassi di update solo se utilizza una politica di controllo della simultaneità pessimistica. Per un controllo ottimistico, il programma di

caricamento non utilizza mai la sintassi update sull'istruzione SQL select. Il programma di caricamento è responsabile della decisione di utilizzare l'argomento `forUpdate` basato sulla politica di controllo della simultaneità utilizzato dal programma.

Per una spiegazione del parametro **txid**, fa re riferimento alla sezione "Plug-in TransactionCallback" a pagina 213.

Metodo `batchUpdate`

Il metodo `batchUpdate` è di fondamentale importanza per l'interfaccia del programma di caricamento. Questo metodo viene richiamato quando ObjectGrid deve applicare tutte le modifiche correnti al programma di caricamento. Al programma di caricamento viene fornito un elenco delle modifiche alla mappa. Le modifiche vengono quindi iterate e applicate sul backend. Il metodo riceve il valore TxID corrente e le modifiche da applicare. Il seguente esempio itera la serie di modifiche ed raggruppa tre istruzioni JDBC (Java database connectivity), una con `insert`, un'altra con `update` e un'altra ancora con `delete`.

```
import java.util.Collection;
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
public void batchUpdate(TxID tx, LogSequence sequence)
throws LoaderException
{
 // Richiama una connessione SQL da utilizzare.
 Connection conn = getConnection(tx);
try
{
  // Elabora l'elenco di modifiche e crea una serie di
  // istruzioni preparate per l'esecuzione di un aggiornamento batch,
     un inserimento o un'eliminazione
  // dell'operazione SQL.
  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;
}
}
  // Esegue le istruzioni batch create dal loop precedente.
  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;
}
}
```

Il precedente esempio illustra la logica di livello elevato di elaborazione dell'argomento LogSequence mentre i dettagli del modo in cui viene creata un'istruzione SQL `insert`, `update` o `delete` non sono riportati. Tra i punti illustrati vi sono:

- Il metodo `getPendingChanges` viene richiamato sull'argomento LogSequence per ottenere un iteratore sull'elenco di elementi LogElements che il programma di caricamento deve elaborare.
- Il metodo `LogElement.getType().getCode()` viene utilizzato per determinare se l'elemento LogElement per un'operazione SQL `insert`, `update` o `delete`.
- Un'eccezione `SQLException` viene rilevata e viene collegata a un'eccezione `LoaderException` che stampa sul report che si è verificata un'eccezione durante l'aggiornamento batch.
- Il supporto di aggiornamento batch JDBC viene utilizzato per ridurre il numero di query che devono essere eseguite sul backend.

Metodo `preloadMap`

Durante l'inizializzazione di ObjectGrid, viene inizializzata ogni mappa di backup definita. Se un programma di caricamento viene collegato a una mappa di backup, la mappa richiama il metodo `preloadMap` sull'interfaccia del programma di caricamento per consentire al programma di leggere i dati dal backend e caricare i dati sulla mappa. Il seguente esempio assume che vengano lette le prime 100 righe di una tabella Employee dal database che vengono poi caricate sulla mappa. La classe EmployeeRecord è una classe fornita dall'applicazione che contiene i dati letti dalla tabella employee.

```
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();
  // Richiama una connessione di commit automatico da utilizzare
  // impostato su un livello di isolamento del commit di lettura.
  conn = getAutoCommitConnection(tx);
  // Precarica la mappa Employee con gli oggetti EmployeeRecord.
  // Legge tutti i valori di Employees dalla tabella, ma
  // limita il precaricamento alle prime 100 righe.
```

```
  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;
}
  // Esegue il commit della transazione.
session.commit();
  tranActive = false;
}
 catch (Throwable t)
{
throw new LoaderException("preload failure: " + t, t);
}
finally
{
  if ( tranActive )
{
try
{
session.rollback();
}
   catch ( Throwable t2 )
{
    // Tollera qualsiasi errore di rollback e
    // consente l'emissione di Throwable originale.
}
}
  // Assicurarsi di eseguire il cleanup delle risorse di altri database
  // e delle istruzioni di chiusura, delle serie di risultati, ecc.
}
}
```

Questo esempio illustra i seguenti punti chiave:

- La mappa di backup preloadMap utilizza l'oggetto Session inviato come argomento della sessione.
- Il metodo `Session.beginNoWriteThrough()` è utilizzato per iniziare la transazione al posto del metodo `begin`. Il Loader non può essere richiamato per ogni operazione `put` che si verifica nel metodo per il caricamento della mappa.
- Il Loader può associare le colonne della tabella employee a un campo nell'oggetto java EmployeeRecord.
- Il Loader rileva tutte le eccezioni che possono essere emesse ed emette una eccezione `LoaderException` che l'eccezione throwable concatenata.
- Il blocco `finally` garantisce che l'eccezione throwable che si verifica tra il richiamo del metodo `beginNoWriteThrough` e del metodo `commit` provochi l'esecuzione del rollback del blocco `finally` della transazione attiva. Questa azione è di fondamentale importanza per garantire che la transazione avviata dal metodo `preloadMap` sia completata prima che venga restituita al chiamante. Il blocco `finally` è una posizione ideale

per eseguire altre opzioni di clean up che potrebbero essere necessarie, come ad esempio la chiusura della connessione JDBC e di altri oggetti JDBC.

L'esempio `preloadMap` utilizza un'istruzione SQL select che seleziona tutte le righe della tabella. Nel programma di caricamento fornito dall'applicazione, potrebbe essere necessario impostare una o più proprietà per controllare la parte della tabella che deve essere precaricata sulla mappa.

Poiché il metodo `preloadMap` viene richiamato soltanto una volta durante l'inizializzazione di BackingMap, questo è una buona posizione per eseguire il codice di inizializzazione del programma di caricamento. Anche se un programma di caricamento sceglie di non leggere i dati dal backend e di caricare i dati nella mappa, è probabile che debba eseguire un'altra inizializzazione per rendere gli altri metodi del programma più efficienti. Di seguito è riportato un esempio di memorizzazione nella cache degli oggetti TransactionCallback e OptimisticCallback come variabili di istanze del programma di caricamento in modo che gli altri metodi non debbano eseguire delle chiamate per poter accedere a tali oggetti. Questa memorizzazione nella cache dei valori di plug-in di ObjectGrid può essere eseguita una volta inizializzato BackingMap o dopo aver modificato o sostituito gli oggetti TransactionCallback e OptimisticCallback. È possibile memorizzare nella cache i riferimenti di tali oggetti come variabili delle istanze del programma di caricamento.

```
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;
// Le variabili delle istanze del programma di caricamento.
MyTransactionCallback ivTcb; // MyTransactionCallback
// estende TransactionCallback
MyOptimisticCallback ivOcb; // MyOptimisticCallback
// implementa OptimisticCallback
...
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
 // Memorizza nella cache gli oggetti TransactionCallback e OptimisticCallback
 // in variabili di istanze di questo programma di caricamento.
 ivTcb = (MyTransactionCallback)
 session.getObjectGrid().getTransactionCallback();
 ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
 // Il promemoria del codice preloadMap
    (come riportato nell'esempio precedente).
}
```

Per ulteriori informazioni sul precaricamento e il precaricamento ripristinabile relativo al failover della replica, fare riferimento a "Programmazione della replica" a pagina 224.

Considerazioni sul programma di caricamento

Tenere presetne quanto riportato di seguito quando si implementa un programma di caricamento.

Considerazioni sul precaricamento

Ogni mappa di backup utilizza un attributo booleano preloadMode che può essere impostato per indicare se il precaricamento di una mappa è stato completato in

maniera asincrona. Per impostazione predefinita, l'attributo preloadMode è impostato su `false`, che indica che l'inizializzazione della mappa di backup non viene completata fino a che non viene completato il precaricamento. Ad esempio, l'inizializzazione della mappa non è completa fino a che il metodo `preloadMap` restituisce un valore. Se il metodo `preloadMap` viene utilizzato per leggere una grossa quantità di dati dal backend e per caricarli sulla mappa, è possibile che il completamento del processo richieda del tempo. In questo caso, è possibile configurare una mappa di backup per utilizzare il precaricamento asincrono della mappa impostando l'attributo preloadMode su true. Questa impostazione provoca la diffusione da parte del codice di inizializzazione della mappa di un thread che richiama il metodo `preloadMap`, consentendo il completamento del processo di inizializzazione mentre è ancora in corso il precaricamento della mappa.

Il seguente frammento di codice illustra il modo in cui è impostato l'attributo preloadMode per abilitare il precaricamento asincrono.

```
BackingMap bm = og.defineMap( "map1" );
bm.setPreloadMode( true );
```

L'attributo preloadMode può essere impostato anche utilizzando un file XML come illustrato nel seguente esempio:

```
<backingMap name="map1" preloadMode="true"
pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
```

TxID e utilizzo dell'interfaccia TransactionCallback

Sia il metodo `get` che il metodo `batchUpdate` sull'interfaccia del programma di caricamento ricevono un oggetto TxID che rappresenta la transazione Session che richiede l'esecuzione dell'operazione `get` o `batchUpdate`. È possibile che i metodi `get` e `batchUpdate` vengano richiamato più di una volta per ogni transazione. Pertanto, gli oggetti nell'ambito della transazione che sono necessari per il programma di caricamento sono di solito conservati in uno slot dell'oggetto TxID. Un programma di caricamento JDBC (Java database connectivity) viene utilizzato per illustrare il modo in cui un programma di caricamento utilizza le interfacce TxID e TransactionCallback.

È inoltre possibile che diverse mappe di ObjectGrid vengano memorizzate nello stesso database. Ogni mappa ha il proprio programma di caricamento e ogni programma di caricamento potrebbe collegarsi allo stesso database. Quando si collega allo stesso database, ogni programma di caricamento vuole utilizzare la stessa connessione JDBC in modo che le modifiche a ogni tabella vengano applicate come parte della stessa transazione al database. Di solito, la persona che scrive l'implementazione del programma di caricamento scrive anche l'implementazione di TransactionCallback. Il metodo migliore consiste nell'utilizzare l'interfaccia TransactionCallback estesa per aggiungere i metodi di cui ha bisogno il programma di caricamento per richiamare una connessione al database e le istruzioni preparate per la memorizzazione nella cache. Il motivo della validità di questa soluzione è dato dal modo in cui le interfacce TransactionCallback e TxID vengono utilizzate dal programma di caricamento.

Come esempio, il programma di caricamento potrebbe avere bisogno dell'interfaccia TransactionCallback che viene estesa come riportato di seguito:

```
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 );
}
```

Mediante questi nuovi metodi, i metodi `get` e `batchUpdate` del programma di caricamento può richiamare una connessione come riportato di seguito:

```
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;
}
```

Nell'esempio precedente e in quelli riportati di seguito, *ivTcb* e *ivOcb* sono variabili dell'istanza del programma di caricamento che sono state inizializzate come descritto nella sezione "Considerazioni sul precaricamento" a pagina 203. La variabile *ivTcb* è un riferimento all'istanza MyTransactionCallback mentre la variabile *ivOcb* è un riferimento all'istanza MyOptimisticCallback. La variabile *databaseName* è una variabile di istanza del programma di caricamento impostata come proprietà del Loader durante l'inizializzazione della mappa di backup. L'argomento `isolationLevel` è una delle costanti della connessione JDBC che sono definite per i vari livelli di isolamento supportati da JDBC. Se il Loader utilizza un'implementazione ottimistica, il metodo `get` di solito utilizza una connessione di commit automatico JDBC per leggere i dati dal database. In tal caso, il programma di caricamento potrebbe avere un metodo `getAutoCommitConnection` implementato come riportato di seguito:

```
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;
}
```

Tenere presente che il metodo `batchUpdate` ha la seguente istruzione 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;
}
```

Ogni metodo `buildBatchSQL` utilizza l'interfaccia MyTransactionCallback per richiamare un'istruzione preparata. Di seguito è riportato un frammento di codice

che mostra il modo in cui il metodo `buildBatchSQLUpdate` crea un'istruzione SQL update per l'aggiornamento di una voce EmployeeRecord e la relativa aggiunta all'aggiornamento 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();
}
```

Una volta che il loop `batchUpdate` ha creato tutte le istruzioni preparate, esso richiama il metodo `getPreparedStatementCollection`. Tale metodo può essere implementato come riportato di seguito:

```
private Collection getPreparedStatementCollection( TxID tx, Connection conn )
{
return ( ivTcb.getPreparedStatementCollection( tx, conn, "employee" ) );
}
```

Quando l'applicazione richiama il metodo `commit` sulla sessione, il codice dell'oggetto Session richiama il metodo `commit` sul metodo `TransactionCallback` dopo aver inserito tutte le modifiche apportate dalla transazione sul programma di caricamento per ogni mappa che è stata modificata dalla transazione. Poiché tutti i programmi di caricamento utilizzano il metodo `MyTransactionCallback` per richiamare la connessione e le istruzioni preparate necessarie, il metodo `TransactionCallback` sa quale connessione utilizzare per richiedere che il backend esegua il commit delle modifiche. Per questo motivo, l'estensione dell'interfaccia TransactionCallback con i metodi necessari a ognuno dei programmi di caricamento ha i seguenti vantaggi:

- L'oggetto TransactionCallback comprende l'utilizzo degli slot TxID per i dati dell'ambito della transazione e il programma di caricamento non richiede le informazioni sugli slot TxID. Il programma di caricamento deve conoscere soltanto i metodi aggiunti a TransactionCallback utilizzando l'interfaccia MyTransactionCallback per il supporto delle funzioni necessarie al programma di caricamento.
- L'oggetto TransactionCallback garantisce che si verifichi una condivisione della connessione tra ogni programma di caricamento che si connette allo stesso backend in modo che si possa evitare un protocollo di commit a due fasi.
- L'oggetto TransactionCallback garantisce che la connessione al backend sia portata avanti fino al completamento mediante un commit o un rollback richiamato sulla connessione quando necessario.
- TransactionCallback garantisce il cleanup delle risorse del database quando viene completata una transazione.
- TransactionCallback può essere nascosto se ottiene una connessione gestita da un ambiente gestito come WebSphere Application Server o un qualsiasi altro server delle applicazioni conforme a Java 2 Platform, Enterprise Edition (J2EE). Questo vantaggio consente l'utilizzo dello stesso codice del programma di

caricamento sia in ambienti gestiti che in ambienti non gestiti. È necessario modificare soltanto il plug-in TransactionCallback.

Per informazioni dettagliate su come l'implementazione di TransactionCallback utilizza gli slot TxID per i dati dell'ambito della transazione, fare riferimento a "Plug-in TransactionCallback" a pagina 213.

OptimisticCallback

Come già menzionato, il programma di caricamento potrebbe decidere di utilizzare un approccio ottimistico per il controllo della simultaneità. In questo caso, il metodo `buildBatchSQLUpdate` deve essere modificato leggermente per l'implementazione dell'approccio ottimistico. Esistono diversi modi per utilizzare un approccio del genere. Uno di questi consiste nell'avere una colonna in formato orario o una colonna del numero di sequenza della creazione della versione di ogni aggiornamento della riga. Si assuma che la tabella employee abbia una colonna del numero di sequenza che viene incrementato ogni volta che viene aggiornata una riga.

Quindi si modifica la forma del metodo `buildBatchSQLUpdate` in modo che all'oggetto LogElement venga inviato il metodo invece che la coppia chiave-valore. È inoltre necessario utilizzare l'oggetto OptimisticCallback che è collegato alla mappa di backup per richiamare sia la versione iniziale che l'aggiornamento della versione. Di seguito è riportato un esempio del metodo `buildBatchSQLUpdate` modificato che utilizza la variabile di istanza *ivOcb* inizializzata come descritto nella sezione di preloadMap:

```
private void buildBatchSQLUpdate( TxID tx, LogElement le,
 Connection conn )throws SQLException, LoaderException
{
 // Richiama la versione iniziale quando la voce della mappa è stata letta
 // o aggiornata nel database.
 Employee emp = (Employee) le.getCurrentValue();
 long initialVersion = ((Long) le.getVersionedValue()).longValue();
 // Richiama la versione dall'oggetto Employee aggiornato per l'operazione
//operation.
 Long currentVersion = (Long)ivOcb.getVersionedObjectForValue( emp );
 long nextVersion = currentVersion.longValue();
 // Crea SQL update che include la versione nella clausola where
 // per il controllo ottimistico.
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();
}
```

L'esempio riporta l'utilizzo di LogElement per ottenere il valore della versione iniziale. Quando la transazione accede alla voce della mappa, viene creato un elemento LogElement con l'oggetto Employee iniziale ottenuto dalla mappa. L'oggetto iniziale Employee viene inviato anche al metodo getVersionedObjectForValue sull'interfaccia OptimisticCallback che provoca il salvataggio sul LogElement. Questa elaborazione avviene prima che venga fornito

un riferimento a un'applicazione all'oggetto Employee e può richiamare un metodo per modificare lo stato dell'oggetto Employee.

L'esempio mostra che il programma di caricamento utilizza il metodo getVersiondObjectForValue per ottenere la versione per l'oggetto Employee aggiornato. Quando viene richiamato il metodo batchUpdate sull'interfaccia del programma di caricamento, ObjectGrid richiama il metodo updateVersionedObjectForValue sull'interfaccia OptimisticCallback in modo da provocare la creazione di una nuova versione dell'oggetto. Dopo aver restituito il metodo batchUpdate all'ObjectGrid, LogElement viene aggiornato con la versione corrente dell'oggetto in modo che diventi la nuova versione iniziale. Questa operazione è necessaria perché l'applicazione potrebbe aver richiamato il metodo flush sulla mappa invece che il metodo commit sulla sessione. È possibile che il programma di caricamento venga richiamato più volte da un'unica transazione per la stessa chiave. L'elemento LogElement deve essere aggiornato con la nuova versione ogni volta che una riga della tabella employee viene aggiornata.

A questo punto, il programma di caricamento ha sia la versione iniziale che la versione successive e può pertanto eseguire un'istruzione SQL update che imposta la colonna SEQNO al valore della versione successiva, utilizzando il valore della versione iniziale nella clausola `where`. Questo approccio è detto a volte istruzione `update` sovraqualificata. L'uso di una istruzione `update` sovraqualificata consente al database relazionale di verificare che la riga non è stata modificata da un'altra transazione nel periodo in cui la transazione corrente ha letto i dati dal database e il momento in cui la transazione ha aggiornato il database. Se un'altra transazione ha modificato la riga, allora l'array del conteggio che viene restituita dall'aggiornamento batch indica che zero righe sono state aggiornate per questa chiave. Il programma di caricamento è responsabile della verifica che l'operazione SQL `update` abbia realmente aggiornato la riga. In caso contrario, il programma di caricamento restituisce un'eccezione `com.ibm.websphere.objectgrid.plugins.OptimisticCollisionException` per informare l'oggetto Session che il metodo batchUpdate non è riuscito a causa di una transazione simultanea che ha provato ad aggiornare la stessa riga nella tabella del database. Questa eccezione provoca il rollback dell'oggetto Session e l'applicazione deve quindi richiamare l'intera transazione. la logica sta nel fatto che il nuovo tentativo riuscirà sicuramente e per questo motivo l'approccio è detto ottimistico. L'approccio ottimistico viene eseguito al meglio se i dati non vengono modificati molto di frequente o se le transazioni simultanee provano raramente ad aggiornare la stessa riga.

È importante che il programma di caricamento utilizzi il parametro *key* del costruttore OptimisticCollisionException constructor per identificare la chiave o la serie di chiavi che ha causato l'errore del metodo batchUpdate ottimistico. Il parametro key può essere una chiave stessa o un'array di chiavi se più di una chiave ha restituito un errore dell'aggiornamento ottimistico. ObjectGrid utilizza il metodo getKey del costruttore OptimisticCollisionException per determinare quali voci della mappa contengono dati inattivi e hanno pertanto provocato l'emissione dell'eccezione. Una parte del processo di rollback consiste nell'eliminare ogni voce inattiva dalla mappa. L'eliminazione di queste voci è necessaria in modo che le transazioni successive che accedono alle stessa chiave richiamino il metodo get dell'interfaccia Loader per aggiornare le voci della mappa con i dati correnti dal database.

Gli altri casi per cui un programma di caricamento può implementare un approccio ottimistico sono:

- Non esiste alcuna colonna dell'orario o del numero di sequenza. In questo caso, il metodo getVersionObjectForValue sull'interfaccia OptimisticCallback restituisce semplicemente il valore stesso come versione. Con questo approccio, il programma di caricamento deve creare una clausola `where` che include ognuno dei campi della versione iniziale. Questo approccio non è molto efficiente e nontutti i tipi di colonne possono essere utilizzati nella clausola `where` dell'istruzione SQL update sovraqualificata. Per questo motivo, di solito questo approccio non viene utilizzato.
- Non esiste alcuna colonna dell'orario o del numero di sequenza. Tuttavia, a differenza del caso precedente, la clausola `where` contiene soltanto i campi del valore che sono stati modificati dalla transazione. Un modo per individuare i campi che sono stati modificati consiste nell'impostare la modalità di copia sulla mappa di backup in modo che sia `CopyMode.COPY_ON_WRITE`. Questa modalità di copia richiede che venga inviata un'interfaccia del valore al metodo `setCopyMode` sull'interfaccia BackingMap. BackingMap crea quindi oggetti proxy dinamici che implementano l'interfaccia del valore fornito. Con questa modalità, il programma di caricamento può utilizzare ogni valore su un oggettocom.ibm.websphere.objectgrid.plugins.ValueProxyInfo. L'interfaccia ValueProxyInfo ha un metodo che consente al programma di caricamento di ottenere un elenco di nomi di attributi che sono stati modificati dalla transazione. Questo metodo consente al programma di caricamento di richiamare i metodi `get` sull'interfaccia del valore per i nomi di attributi in modo da ottenere i dati che sono stati modificati e di creare un'istruzione SQL update che imposta soltanto gli attributi modificati. La clausola `where` può essere creata in modo che abbia la colonna della chiave primaria più le colonne di ogni attributo modificato. Questo approccio è più efficiente dell'approccio precedente, ma richiede la scrittura di più codice sul programma di caricamento ed è inoltre possibile che ci sia bisogno di una cache delle istruzioni preparate di dimensioni maggiori per gestire modifiche differenti. Tuttavia, se le transazioni modificano di solito pochi attributi, questa limitazione può non essere un problema.
- Alcuni database relazioni possono avere un'API che consente di gestire automaticamente i dati delle colonne che risultano utili per le versioni ottimistiche. Fare riferimento alla documentazione del database per determinare se esiste questa possibilità.

Plug-in ObjectTransformer

Utilizzare il plug-in ObjectTransformer quando sono richieste prestazioni elevate. Se si verificano problemi legati alle prestazioni a causa dell'utilizzo della CPU, aggiungere un plug-in ObjectTransformer a ciascuna mappa. Se non viene fornito alcun plug-in ObjectTransformer, il 60-70% del tempo di CPU totale viene speso per la serializzazione e la copia delle voci.

Funzione

La funzione del plug-in ObjectTransformer è consentire alle applicazioni di fornire metodi personalizzati per le seguenti operazioni:

- Serializzazione o deserializzazione della chiave per una voce
- Serializzazione o deserializzazione del valore per una voce
- Copia di una chiave o di un valore per una voce

Se non viene fornito il plug-in ObjectTransformer, è necessario essere in grado di serializzare le chiavi e i valori in quanto ObjectGrid utilizza una sequenza di serializzazione e deserializzazione per copiare gli oggetti. Questo metodo è costoso, pertanto è preferibile utilizzare un plug-in ObjectTransformer quando le

prestazioni sono già critiche. La copia viene eseguita quando un'applicazione ricerca un oggetto nella transazione per la prima volta. È possibile evitare questa copia impostando la modalità di copia della mappa su `NO_COPY` o riducendo la copia impostando la modalità di copia su `COPY_ON_READ`. Ottimizzare l'operazione di copia quando lo richiede l'applicazione fornendo un metodo di copia personalizzato su questo plug-in. Tale plug-in può ridurre il sovraccarico di copia dal 65–70% al 2/3% del tempo della CPU totale.

Le implementazioni predefinite dei metodi copyKey e copyValue provano a utilizzare prima il metodo clone(), se fornito. Se non è fornita alcuna implementazione del metodo clone(), l'implementazione farà riferimento alla serializzazione.

La serializzazione degli oggetti è utilizzata anche direttamente quando l'ObjectGrid è in esecuzione in modalità distribuita. La LogSequence utilizza il plug-in ObjectTransformer per serializzare le chiavi e i valori prima di trasmettere le modifiche ai peer nell'ObjectGrid. È necessario prestare attenzione quando si fornisce un metodo di serializzazione personalizzato invece di utilizzare la serializzazione JDK integrata. La determinazione della versione di un oggetto è un problema complesso ed è possibile che si verifichino dei problemi di compatibilità delle versioni se non si verifica che i metodi personalizzati siano progettati per la determinazione della versione.

Nel seguente elenco viene riportato il modo in cui ObjectGrid prova a serializzare sia le chiavi che i valori.

- Se viene scritto e collegato un plug-in ObjectTransformer personalizzato, ObjectGrid richiama i metodi nei metodi ObjectTransformer per serializzare le chiavi e i valori e per ottenere le copie delle chiavi e dei valori degli oggetti.
- Se non viene utilizzato un plug-in ObjectTransformer personalizzato, ObjectGrid serializza e deserializza in base all'impostazione predefinita. Se viene utilizzato il valore predefinito, ogni oggetto viene implementato come esternalizzabile o come serializzabile.
  - Se l'oggetto supporta l'interfaccia Externalizable, allora viene richiamato il metodo `writeExternal`. Gli oggetti implementati come esternalizzabili portano a migliori prestazioni.
  - Se l'oggetto non supporta l'interfaccia Externalizable e implementa invece l'interfaccia Serializable, l'oggetto viene salvato mediante il metodo `ObjectOutputStream`.

Interfaccia ObjectTransformer

Fare riferimento alla documentazione dell'API per maggiori informazioni sull'interfaccia ObjectTransformer. L'interfaccia ObjectTransformer contiene i seguenti metodi che serializzano e deserializzano oppure copiano le chiavi e i valori.

```
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);
}
```

Utilizzo dell'interfaccia ObjectTransformer

È possibile utilizzare l'interfaccia ObjectTransformer nei seguenti casi:

- l'oggetto non è serializzabile
- l'oggetto è serializzabile ma migliora le le prestazioni di serializzazione
- viene eseguita la copia della chiave o del valore

Nel seguente esempio, ObjectGrid viene utilizzato per memorizzare la classe Stock:

```
/**
* Oggetto Stock per la demo di ObjectGrid
*
*
*/
public class Stock implements Cloneable {
 String ticket;
 double price;
 String company;
 String description;
 int serialNumber;
 long lastTransactionTime;
/**
 * @return Restituisce la descrizione.
*/
 public String getDescription() {
  return description;
}
/**
 * @param description La descrizione da impostare.
*/
 public void setDescription(String description) {
  this.description = description;
}
/**
 * @return Restituisce lastTransactionTime.
*/
 public long getLastTransactionTime() {
  return lastTransactionTime;
}
/**
 * @param lastTransactionTime Il valore di lastTransactionTime da impostare.
*/
 public void setLastTransactionTime(long lastTransactionTime) {
  this.lastTransactionTime = lastTransactionTime;
}
/**
 * @return Restituisce il prezzo.
*/
 public double getPrice() {
  return price;
}
/**
 * @param price Il prezzo da impostare.
*/
 public void setPrice(double price) {
  this.price = price;
}
/**
 * @return Restituisce serialNumber.
*/
 public int getSerialNumber() {
  return serialNumber;
}
/**
 * @param serialNumber Il serialNumber da impostare.
```

```
*/
 public void setSerialNumber(int serialNumber) {
  this.serialNumber = serialNumber;
}
/**
 * @return Restituisce l'etichetta.
*/
 public String getTicket() {
  return ticket;
}
/**
 * @param ticket L'etichetta da impostare.
*/
 public void setTicket(String ticket) {
  this.ticket = ticket;
}
/**
 * @return Restituisce l'azienda.
*/
 public String getCompany() {
  return company;
}
/**
 * @param company L'azienda da impostare.
*/
 public void setCompany(String company) {
  this.company = company;
}
//clone
 public Object clone() throws CloneNotSupportedException
{
  return super.clone();
}
}
```

È possibile scrivere una classe transformer per gli oggetti personalizzata per la classe Stock:

```
/**
* Implementazione personalizzata di ObjectGrid ObjectTransformer per l'oggetto 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)
{
//streamize one
}
}
/* (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;
}
}
```

Quindi, collegare questa classe MyStockObjectTransformer personalizzata al 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

Un'applicazione di solito collega un plug-in TransactionCallback e un programma di caricamento come un'unica entità. Il programma di caricamento è responsabile della lettura dei dati dal backend e dell'applicazione delle modifiche. Tali operazioni di solito si verificano all'interno del contesto di una transazione ObjectGrid.

Il plug-in TransactionCallback ha le seguenti funzioni:

- Riserva gli slot per uno stato specifico della transazione necessari per la transazione e il programma di caricamento
- Traduce o associa una transazione ObjectGrid a una transazione della piattaforma
- Imposta lo stato per transazione quando ObjectGrid inizia una transazione
- Esegue il commit della transazione quando viene eseguito il commit della transazione ObjectGrid
- Esegue il rollback della transazione quando viene eseguito il rollback della transazione ObjectGrid

ObjectGrid non è un coordinatore delle transazioni XA. Esso si basa sulla piattaforma per fornire tale capacità. I metodi `begin`, `commit` e `rollback` di ObjectGrid presentati in una sessione sono chiamate al ciclo. Il plug-in TransactionCallback deve ricevere questi eventi e costituire la piattaforma che fornisce la capacità transazionale per le risorse utilizzate dai programmi di caricamento. In questa sezione sono riportati diversi scenari e viene descritto come scrivere il plug-in TransactionCallback per tali scenari.

Panoramica sul plug-in TransactionCallback

Il plug-in TransactionCallback è un POJO che implementa l'interfaccia TransactionCallback. L'interfaccia TransactionCallback ha il seguente aspetto:

```
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);
}
```

Metodo `initialize`

Il metodo `initialize` viene richiamato quando viene inizializzato l'ObjectGrid. La richiamata riserva gli slot per l'oggetto TxID necessario. Di solito, riserva uno slot per ogni parte di stato o per ogni oggetto che si desidera creare nel metodo `begin` quando viene avviata una transazione. Ad esempio, è possibile utilizzare Persistence Manager con l'ObjectGrid come programma di caricamento. Se si assume che questo gestore permanente abbia oggetti di stato delle sessioni e delle transazioni, TransactionCallback ottiene una sessione e una transazione e fa riferimento a questi due oggetti negli slot sul TxID. In questo caso, il metodo `initialize` ha il seguente aspetto:

```
/**
* Questo viene richiamato quando viene inizializzata una griglia. Vengono
* riservati gli slot nel TxID.
*/
public void initialize(ObjectGrid objectGrid) throws
TransactionCallbackException
{
 // riserva uno slot per la transazione del gestore permanente
 TXslot = objectGrid.reserveSlot(TxID.SLOT_NAME);
 // riserva uno slot per la sessione del gestore permanente
 SessionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
}
```

Un TxID contiene degli slot. Gli slot sono voci in un'array ArrayList. I plug-in possono riservare una voce nell'array ArrayList richiamando il metodo

ObjectGrid.reserveSlot e indicando se si desidera riservare uno slot sull'oggetto TxID. Il metodo quindi restituisce l'indice della voce successiva all'applicazione. L'applicazione può quindi memorizzare le informazioni nello slot. I metodi successivi dimostrano questa tecnica.

Metodo begin

ObjectGrid richiama questo metodo quando avvia una nuova transazione. Il plug-in associa questo evento a una transazione reale che i programmi di caricamento possono utilizzare per le chiamate al metodo get e update che arrivano prima che viene richiamato il metodo commit. Di seguito è riportato un esempio del metodo begin che associa un metodo begin di ObjectGrid a un metodo begin della transazione di un gestore permanente:

```
/**
* Questo viene richiamato quando la griglia avvia una nuova transazione.
* Viene subito creata una transazione del gestore permanente e il metodo begin
* viene richiamato.
* Viene quindi memorizzata la transazione nello slot TxID in modo
* da poterla richiamare successivamente senza aver bisogno
* di ThreadLocal ecc.
*/
public void begin(TxID id) throws TransactionCallbackException
{
 Session PMsession = getPMcurrentSession();
 Transaction tx = PMsession.beginTransaction();
 id.putSlot(TXslot, tx);
 id.putSlot(SessionSlot, PMsession);
}
```

Questo esempio si basa sul fatto che il metodo initialize ha riservato due slot sull'oggetto TxID. Uno slot è per la sessione del gestore permanente mentre l'altro è per la transazione. Il metodo begin richiama il gestore permanente per ottenere una sessione, la memorizza nello slot SessionSlot indicizzato e crea una transazione sulla sessione, quindi memorizza un riferimento a tale transazione utilizzando lo slot TXSlot indicizzato.

Metodo commit

Il metodo commit viene richiamato quando viene eseguito il commit di una transazione ObjectGrid. Tutti i programmi di caricamento sono già stato ripuliti. La funzione del plug-in è comunicare questo evento di commit alla piattaforma.

```
/**
* Questo viene richiamato quando la griglia deve eseguire il commit di una transazione.
* Il metodo viene semplicemente inviato al gestore permanente.
*/
public void commit(TxID id) throws TransactionCallbackException
{
 Transaction tx = (Transaction)id.getSlot(TXslot);
   tx.commit();
}
```

Il metodo ricerca la transazione del gestore permanente memorizzata nello slot e richiama il metodo commit.

Metodo rollback

Questo metodo viene richiamato quando una transazione di ObjectGrid desidera eseguire il rollback di una transazione. Il plug-in lo inoltra al gestore delle transizioni della piattaforma. Di seguito è riportato il frammento di codice:

```
/**
* Questo viene richiamato quando la griglia deve eseguire il rollback di una transazione.
* Il metodo viene semplicemente inviato al gestore permanente.
```

```
*/
public void rollback(TxID id) throws TransactionCallbackException
{
 Transaction tx = (Transaction)id.getSlot(TXslot);
   tx.rollback();
}
```

Questo metodo è molto simile al metodo `commit`. Esso richiama un riferimento alla transazione del gestore permanente da uno slot e richiama quindi il metodo `rollback`.

Metodo `isExternalTransactionActive`

Una sessione ObjectGrid di solito opera in modalità di commit automatico o in modalità di transazione. Una modalità di commit automatico significa che viene creata una transazione implicita per ogni chiamata di un metodo alle istanze di ObjectMap per la sessione. Se nessuna transazione è attiva e un'applicazione effettua una chiamata a un metodo ObjectMap, la struttura richiama questo metodo sul plug-in TransactionCallback per verificare che sia presente una transazione attiva. Se il metodo restituisce true allora la struttura esegue un begin automatico, altrimenti effettua un commit automatico. Questo metodo consente all'ObjectGrid di essere integrato in ambienti in cui l'applicazione richiama i metodi `begin`, `commit` o `rollback` sulle API della piattaforma invece che sulle API di ObjectGrid.

Scenario: Ambiente J2SE (Java 2 Platform, Standard Edition) basato su JDBC (Java database connectivity)

Questo esempio utilizza un ambiente J2SE in cui l'applicazione ha un programma di caricamento basato su JDBC. Esistono due mappe, ognuna con un programma di caricamento che esegue il backup di ogni mappa in una tabella differente nel database. Il plug-in TransactionCallback ottiene una connessione JDBC e quindi richiama i metodi `begin`, `commit` e `rollback` sulla connessione. Di seguito è riportato un esempio di implementazione di TransactionCallback:

```
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;
}
}
```

Questo esempio mostra un plug-in TransactionCallback che converte gli eventi della transazione di ObjectGrid a una connessione JDBC. Quando viene inizializzato il plug-in, viene riservato un unico slot per conservare un riferimento alla connessione JDBC. Il metodo `begin` ottiene quindi una connessione JDBC per la nuova transazione, disattiva il commit automatico e memorizza un riferimento alla connessione nello slot TxID. I metodi `commit` e `rollback` richiamano la connessione dallo slot TxID e richiama quindi il metodo appropriato sulla connessione. Il metodo `isExternalTransaction` restituisce sempre false, indicando che l'applicazione deve utilizzare le API della transazione ObjectGrid in maniera esplicita per controllare le transazioni. Un programma di caricamento accoppiato con questo plug-in ottiene la connessione JDBC dal TxID. Un programma di caricamento ha il seguente aspetto:

```
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());
  // implementa get
return null;
}
 public void batchUpdate(TxID txid, LogSequence sequence)
 throws LoaderException, OptimisticCollisionException
{
  Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
  // aggiornamento batch implementazione TODO
}
}
```

Il programma di caricamento ottiene un riferimento all'istanza JDBCTCB quando viene richiamato il metodo `initialize`. Quindi ottiene la connessione ottenuta da JDBCTCB quando è richiesta per i metodi `get` e `batchUpdate`. Le implementazioni di TransactionCallback e i programmi di caricamento sono di solito scritti in coppie che cooperano tra loro. L'implementazione di TransactionCallback gestisce la transazione e memorizza gli oggetti necessari ai programmi di caricamento negli slot nel TxID. I programmi di caricamento implementano quindi i metodi `get` e `batchUpdate` nel contesto di una transazione gestita da TransactionCallback mediante le risorse ottenute dal TCB .

Scenario: Ambiente del motore servlet

In questo scenario, l'ObjectGrid utilizza un programma di caricamento basato su JDBC ma in un motore del servlet gestito. Il contenitore prevede l'utilizzo del metodo `UserTransaction` per iniziare ed eseguire il commit delle transazioni. Questa è una differenza con il caso J2SE in quanto la memorizzazione di un riferimento sulla connessione JDBC in uno slot TxID non è necessaria. Il contenitore gestisce la connessione JDBC. Quando la transazione di un contenitore è attiva, la connessione ricercata mediante un'origine dati diventa sempre la tessa connessione ogni volta in quanto il contenitore ricorda le connessioni utilizzate da questa transazione e restituisce sempre la stessa ogni volta che viene richiamato il metodo `DataSource.getConnection`. Si assuma che il riferimento all'origine dati sia configurato come `Shareable` nel seguente esempio:

```
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;
}
}
```

In questo esempio si ottiene un riferimento al metodo `UserTransaction` nel metodo `initialize` e si associa il metodo `begin`, `commit` e `rollback` al metodo `UserTransaction` appropriato. Gli slot non sono necessari in quanto il contenitore verifica che vengano richiamate le informazioni di connessione corrette per questa transazione. Di seguito è riportato il programma di caricamento JDBC che funziona con questa implementazione di 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();
   // implementazione TODO con questa connessione
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();
   // aggiornamento dell'implementazione TODO con questa connessione
}
  catch(SQLException e)
{
throw new LoaderException("Cannot update objects", e);
}
}
}
```

Questo esempio può risultare più semplice della versione JDBC di base in quanto il contenitore gestisce le connessioni e verifica che all'interno della stessa transazione, il metodo `DataSource.getConnection` restituisca sempre la stessa connessione se richiamato con la stessa transazione attiva ogni volta. Non provare a memorizzare nella cache la connessione in uno slot, nonostante ciò sia possibile.

Interfaccia OptimisticCallback

È possibile fornire un oggetto di richiamata ottimistica collegabile che implementa l'interfaccia com.ibm.websphere.objectgrid.plugins.OptimisticCallback.

Funzione

L'interfaccia OptimisticCallback viene utilizzata per fornire operazioni di confronto ottimistiche per i valori di una mappa. Un'interfaccia OptimisticCallback è richiesta quando viene utilizzata la strategia di blocco ottimistico, come descritto in "Blocco ottimistico" a pagina 133. ObjectGrid fornisce un'implementazione di OptimisticCallback predefinita. Tuttavia, di solito l'applicazione deve collegare la propria implementazione dell'interfaccia OptimisticCallback.

Collegamento di un oggetto OptimisticCallback fornito dall'applicazione

Il seguente esempio dimostra il modo in cui un'applicazione può collegare un oggetto OptimisticCallback per la mappa di backup employee nell'istanza grid1 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 );
```

L'oggetto EmployeeOptimisticCallbackImpl nell'esempio precedente deve implementare l'interfaccia OptimisticCallback. L'applicazione può utilizzare anche un file XML per collegare l'oggetto OptimisticCallback come riportato nel seguente esempio:

```
<?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>
```

Implementazione predefinita

La struttura di ObjectGrid fornisce un'implementazione predefinita dell'interfaccia OptimisticCallback utilizzata se l'applicazione non collega un oggetto OptimisticCallback fornito dall'applicazione, come invece riportato nell'esempio precedente. L'implementazione predefinita restituisce il valore speciale NULL_OPTIMISTIC_VERSION come oggetto della versione per il valore e non aggiorna mai tale oggetto della versione. Questa azione rende un confronto ottimistico una funzione ″nessuna operazione″. Nella maggior parte dei casi, la funzione ″nessuna operazione″ non deve essere utilizzata quando si utilizza la strategia di blocco ottimistica. Le applicazioni devono implementare l'interfaccia OptimisticCallback e devono collegare le proprie implementazioni di OptimisticCallback, in modo che non venga utilizzata l'implementazione predefinita. Tuttavia, è presente almeno uno scenario in cui l'implementazione OptimisticCallback predefinita risulta utile. Si consideri la seguente situazione:

- Un programma di caricamento viene colegato alla mappa di backup.
- Il programma di caricamento sa come eseguire il confronto ottimistico senza utilizzare un plug-in OptimisticCallback.

Ma in che modo il programma di caricamento capisce se è necessario utilizzare la versione ottimistica senza utilizzare l'oggetto OptimisticCallback? Il programma di caricamento riconosce l'oggetto della classe del valore e sa perfettamente quale campo dell'oggetto del valore è utilizzato come valore per la versione ottimistica. Ad esempio, si supponga che la seguente interfaccia venga utilizzata per l'oggetto del valore per la mappa employees:

```
public interface Employee
{
// Numero di sequenza utilizzato per la versione ottimistica.
```

```
public long getSequenceNumber();
public void setSequenceNumber(long newSequenceNumber);
// Altri metodi get/set per altri campi dell'oggetto Employee.
}
```

In questo caso, il programma di caricamento sa che può utilizzare il metodo getSequenceNumber per richiamare le informazioni sulla versione corrente per un oggetto del valore Employee. Esso incrementa il valore restituito per generare un nuovo numero di versione prima di aggiornare la memoria permanente con il nuovo valore di Employee. Per un programma di caricamento JDBC (Java database connectivity), viene utilizzato il numero di sequenza corrente nella clausola where di una istruzione di aggiornamento SQL ultra qualificata e il programma utilizza il nuovo numero di sequenza generato per impostare la colonna dei numeri di sequenza per i relativi valori. Un'altra possibilità è data dall'utilizzo di una funzione di backend che aggiorna automaticamente una colonna nascosta che può essere utilizzata per creare unaversione ottimistica. In alcuni casi, è possibile utilizzare una procedura memorizzata o un trigger per gestire una colonna in cui sono contenute le informazioni sulla versione. Se il programma di caricamento utilizza una di queste tecniche per la gestione delle informazioni sulla versione ottimistica, allora l'applicazione non deve fornire un'implementazione OptimisticCallback. In questo caso è possibile utilizzare l'interfaccia OptimisticCallback predefinita in quanto il programma di caricamento è in grado di gestire la creazione della versione ottimistica senza richiedere aiuto a un oggetto OptimisticCallback.

Implementazione dell'interfaccia OptimisticCallback

L'interfaccia OptimisticCallback contiene i seguenti metodi e valori speciali:

```
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;
}
```

Il seguente elenco riporta una descrizione o delle considerazioni per ognuno dei metodi riportati nell'interfaccia OptimisticCallback:

NULL_OPTIMISTIC_VERSION
: Questo valore speciale viene restituito dal metodo getVersionedObjectForValue se viene utilizzata l'implementazione OptimisticCallback predefinita invece di un'implementazione OptimisticCallback fornita dall'applicazione.

Metodo getVersionedObjectForValue
: Questo metodo può restituire una copia o un attributo del valore che possono essere utilizzati per creare la versione. Questo metodo viene richiamato quando un oggetto viene associato a una transazione. Quando nessun programma di caricamento è associato a una mappa di backup, la mappa utilizzare questo valore in fase di commit per eseguire un confronto ottimistico tra le versioni. Tale confronto viene utilizzato dalla mappa per verificare che la versione non sia stata modificata dal primo accesso della transazione alla voce della mappa modificata. Se un'altra transazione ha già modificato la versione della voce della mappa, il confronto tra le versioni non riesce e la mappa restituisce un'eccezione OptimisticCollisionException per imporre il rollback della transazione. Se invece un programma di

caricamento è collegato, la mappa non utilizza le informazioni sulla versione ottimistica. Il programma di caricamento è invece responsabile dell'esecuzione del confronto e dell'aggiornamento delle informazioni sulla versione, se necessario. Il programma di caricamento di solito richiama l'oggetto della versione iniziale dall'elementoLogElement inviato al metodo batchUpdate del programma, che viene richiamato quando si verifica un'operazione di eliminazione o il commit della transazione.

Il seguente codice mostra l'implementazione utilizzata dall'oggetto EmployeeOptimisticCallbackImpl:

```
public Object getVersionedObjectForValue(Object value)
{
 if (value == null)
{
return null;
}
else
{
  Employee emp = (Employee) value;
  return new Long( emp.getSequenceNumber() );
}
}
```

Come dimostrato nell'esempio precedente, l'attributo sequenceNumber viene restituito in un oggetto java.lang.Long come previsto da parte del programma di caricamento, il che implica che la stessa persona che ha scritto il programma di caricamento abbia scritto anche l'implementazione di EmployeeOptimisticCallbackImpl e ciò si capisce ad esempio dal valore restituito dal metodo getVersionedObjectForValue.

Come precedentemente descritto, l'implementazione predefinita di OptimisticCallback restituisce il valore speciale NULL_OPTIMISTIC_VERSION dell'oggetto della versione.

Metodo updateVersionedObjectForValue

Questo metodo viene richiamato quando una transazione ha aggiornato un valore ed è pertanto necessario un oggetto con una nuova versione. Se getVersionedObjectForValue restituisce un attributo del valore, questo metodo di solito aggiorna il valore dell'attributo con un oggetto con una nuova versione. Se invece getVersionedObjectForValue restituisce una copia del valore, il metodo non esegue alcuna operazione. L'implementazione predefinita di OptimisticCallback no esegue alcuna operazione in quanto l'implementazione predefinita di getVersionedObjectForValue restituisce sempre il valore speciale NULL_OPTIMISTIC_VERSION come oggetto della versione.

Di seguito è riportata l'implementazione utilizzata dall'oggetto EmployeeOptimisticCallbackImpl utilizzata nella sezione OptimisticCallback:

```
public void updateVersionedObjectForValue(Object value)
{
 if ( value != null )
{
  Employee emp = (Employee) value;
  long next = emp.getSequenceNumber() + 1;
  emp.updateSequenceNumber( next );
}
}
```

Come definito nel seguente esempio, l'attributo sequenceNumber viene incrementato di uno in modo che la vola successiva che viene richiamato il

metodo getVersionedObjectForValue, il valore java.lang.Long restituito abbia un valore lungo dato dal numero di sequenza originale più uno. Questo esempio implica che la persona che ha scritto il programma di caricamento sia la stessa che abbia scritto EmployeeOptimisticCallbackImpl.

Metodo serializeVersionedValue
Questo metodo scrive il valore della versione sul flusso specificato. A seconda dell'implementazione, il valore della versione può essere utilizzato per identificare le collisioni di aggiornamento ottimistiche. In alcune implementazioni, il valore della versione è una copia del valore originale. Altre implementazioni possono avere un numero di sequenza o un qualsiasi altro oggetto che indica la versione del valore. Poiché l'implementazione reale è sconosciuta, questo metodo è fornito per eseguire una serializzazione corretta. L'implementazione predefinita effettua una chiamata a writeObject.

Metodo inflateVersionedValue
Questo metodo utilizza la versione serializzata del valore della versione e restituisce l'oggetto del valore della versione corrente. A seconda dell'implementazione, il valore della versione può essere utilizzato per identificare le collisioni di aggiornamento ottimistiche. In alcune implementazioni, il valore della versione è una copia del valore originale. Altre implementazioni possono avere un numero di sequenza o un qualsiasi altro oggetto che indica la versione del valore. Poiché l'implementazione reale è sconosciuta, questo metodo è fornito per eseguire una deserializzazione corretta. L'implementazione predefinita esegue un readObject.

Programmazione della replica

La replica viene configurata associando una MapSet a un ReplicationGroup e agli attributi della politica di replica. ReplicationGroup definisce i membri del server che sono utilizzati per le repliche e gli standby primari e associati. Inoltre definisce il numero minimo e il numero massimo di repliche che sono richieste per questa configurazione. Gli attributi della politica di replica indicano se è richiesta una replica sincrona o asincrona e se utilizzare la compressione quando si inviano i dati di replica alle repliche. La replica ha un impatto minimo sul modello di programmazione. L'impatto principale si ha sulle applicazioni che precaricano i dati nelle mappe.

Precaricamento delle mappe

È possibile associare un programma di caricamento a ogni mappa. Un programma di caricamento viene utilizzato per utilizzare gli oggetti quando questi non vengono trovati nella mappa e per scrivere le modifiche su un backend quando viene eseguito il commit di una transazione. I programmi di caricamento possono essere utilizzati anche per pre-caricare i dati in una mappa. Il metodo di precaricamento dell'interfaccia del programma viene richiamato quando la Java virtual machine (JVM) diventa primaria per il gruppo di replica. Il metodo di precaricamento non viene richiamato su repliche o standby. Il metodo di precaricamento prova a caricare tutti i dati di riferimento dal backend alla mappa utilizzando la sessione fornita. La mappa viene quindi identificata dall'argomento BackingMap che viene inviato al metodo di precaricamento.

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

Precaricamento in una MapSet con partizioni

Le mappe possono essere divise in *N* partizioni. Esse possono essere memorizzate su più server con ogni voce identificata da una chiave memorizzata solo su uno di questi server. Mappe di grosse dimensioni possono essere conservate in un ObjectGrid poiché l'applicazione non è più limitata dalla dimensione heap di un'unica JVM che deve contenere tutte le voci di una mappa. Le applicazioni che devono essere precaricate con il metodo di precaricamento dell'interfaccia del programma di caricamento devono identificare la serie di dati che devono essere precaricati. È sempre presente un numero fisso di partizioni. Tale numero pu; essere determinato utilizzando il seguente frammento di codice:

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

Questo frammento di codice mostra il modo in cui un'applicazione può identificare la serie di dati da precaricare dal database. Le applicazioni devono sempre utilizzare questi metodi anche quando la mappa inizialmente non contiene partizioni. Questi metodi consentono una elevata flessibilità: se la mappa viene successivamente partizionata da un amministratore, allora il programma di caricamento continuerà a funzionare correttamente.

l'applicazione deve emettere delle interrogazioni per richiamare la serie myPartition dal backend. Se viene utilizzato un database, allora potrebbe risultare più semplice avere una colonna con l'identificativo della partizione per un determinato record, a meno che non sia presente una interrogazione naturale che consenta il partizionamento dei dati nella tabella.

Prestazioni

L'implementazione del precaricamento deve copiare i dati dal backend alla mappa memorizzando più oggetti nella mappa in un'unica transazione. La domanda successiva è ″Quanti record possono essere memorizzati per ogni transazione?″ e sfortunatamente, la risposta è ″Dipende.″ Se la transazione include più blocchi di 100 voci, allora le prestazioni risultano ridotte. Il numero ottimale dipende da diversi fattori, comprese la complessità e la dimensione dell'oggetto. Iniziare con 100 voci e aumentare il numero fino a che si nota che non avviene più alcun miglioramento delle prestazioni. Transazioni di grosse dimensioni provocano un miglioramento delle prestazioni di replica. Tenere presente che solo il gruppo primario esegue il codice di precaricamento. I dati precaricati vengono replicati dal gruppo primario a qualsiasi gruppo di replica che si trova in linea.

Precaricamento di MapSets

Se l'applicazione utilizza una MapSet con più mappe, allora ogni mappa avrà il proprio programma di caricamento. Ogni programma di caricamento a sua volta avrà il proprio metodo di precaricamento. Ogni mappa viene caricata in serie dall'ObjectGrid. Potrebbe essere più efficiente precaricare tutte le mappe e designando quindi una mappa in particolare come mappa di precaricamento. Questa è soltanto una convenzione per le applicazioni. Ad esempio, due mappe, department e employee, possono utilizzare il programma di caricamento di department per precaricare sia le mappe di department che le mappe di employee. Ciò garantisce che, dal punto di vista delle transazioni, se un'applicazione deve richiamare i dati di department, allora i dati di employees verranno memorizzati nella cache. Naturalmente, ciò significa che quando il programma di caricamento di department precarica i dati di department dal backend, allora utilizzerà i dati di employees per lo stessodepartment. Perché ciò sia possibile, l'oggetto department

e gli oggetti employee associati devono quindi essere aggiunti alla mappa mediante una singola transazione.

Precaricamento ripristinabile

Alcuni clienti hanno una mole notevole di dati che devono essere memorizzati nella cache. Il processo di precaricamento in questi casi può richiedere molto tempo. A volte, il precaricamento deve essere completato prima di poter far passare un'applicazione in linea. Ciò significa che potrebbe essere richiesto rendere il processo di precaricamento come ripristinabile. Si assume di avere un milione di record da precaricare. Il gruppo primario li precarica e riporta un errore all'ottocentomillesimo record. Di solito, il gruppo di replica scelto per essere il nuovo gruppo primario cancella tutti gli stati replicati e comincia di nuovo dall'inizio. ObjectGrid può avere un risultato migliore utilizzando ReplicaPreloadController. Il programma di caricamento per l'applicazione deve anche implementare l'interfaccia ReplicaPreloadController. Questa operazione aggiunge un singolo metodo al programma di caricamento:

```
Status checkPreloadStatus(Session session, BackingMap bmap);
```

Questo metodo viene richiamato dal runtime ObjectGrid prima del richiamo del metodo di precaricamento dell'interfaccia del programma di caricamento. ObjectGrid verifica il risultato di questo metodo (stato) per determinarne il comportamento quando un gruppo di replica viene promosso a gruppo primario.

Valore di stato restituito	Relativo comportamento di ObjectGrid
Status.PRELOADED_ALREADY	ObjectGrid non richiama il metodo di precaricamento in quanto questo valore di stato indica che la mappa è stata completamente precaricata.
Status.FULL_PRELOAD_NEEDED	ObjectGrid cancella la mappa e richiama il metodo di precaricamento nel solito modo.
Status.PARTIAL_PRELOAD_NEEDED	ObjectGrid lascia la mappa così com'è e richiama il precaricamento. Questa strategia consente al programma di caricamento dell'applicazione di continuare il precaricamento da quel punto in poi.

Chiaramente, mentre un gruppo primario precarica la mappa, è necessario lasciare uno stato in una mappa nella MapSet che viene replicata in modo che la replica possa riconoscere lo stato da restituire. A questo scopo, è possibile utilizzare una mappa aggiuntiva, ad esempio RecoveryMap. Questa RecoveryMap deve far parte della stessa MapSet che viene precaricata. Ciò garantisce che la mappa venga replicata in maniera coerente con i dati precaricati.

Di seguito è riportata un tipo di implementazione consigliato. Nel momento in cui il precaricamento esegue il commit di ogni blocco di record, questo deve aggiornare un contatore/valore in RecoveryMap come parte della transazione. Ciò significa che i dati precaricati e i dati RecoveryMap sono replicati atomicamente sui gruppi di replica. Quando i lgruppo di replica viene promosso a gruppo primario, può controllare lo stato di RecoveryMap. RecoveryMap può contenere un'unica voce con la chiave 'state'. Se non esiste alcun oggetto per questa chiave, allora è necessario un precaricamento completo (checkPreloadStatus restituisce FULL_PRELOAD_NEEDED). Se esiste un oggetto per la chiave 'state' allora se il valore è 'COMPLETE' il precaricamento viene eseguito e checkPreloadStatus restituisce PRELOADED_ALREADY. In caso contrario, l'oggetto del valore indica da

che punto è necessario riprendere il precaricamento e il metodo checkPreloadStatus deve restituire PARTIAL_PRELOAD_NEEDED. Il programma di caricamento può memorizzare il punto di ripristino in una variabile di istanza per il programma di caricamento in modo che, quando viene richiamato il precaricamento, se ne conosce il punto di inizio. RecoveryMap può contenere anche una voce per mappa se ogni mappa viene caricata indipendentemente.

Gestione del ripristino in modalità di replica sincrona con un programma di caricamento

Il runtime ObjectGrid è progettato per non perdere i dati di cui è stato eseguito il commit quando il gruppo primario riporta un errore. La seguente sezione mostra l'algoritmo utilizzato perché ciò si verifichi. Questi algoritmi si applicano solo quando un gruppo di replica utilizza una replica sincrona. Il programma di caricamento è facoltativo.

Il runtime ObjectGrid può essere configurato in modo da replicare tutte le modifiche da un gruppo primario ai gruppi di replica in maniera sincrona. Quando una JVM viene promossa in modo da essere un gruppo di replica, il gruppo primario invia prima un'istantanea della mappa al gruppo di replica. Una volta che il gruppo di replica ha elaborato questa istantanea, il gruppo primario comincia a inviare le modifiche (ovvero le transazioni completate) dalla creazione dell'istantanea. Eventualmente, il gruppo di replica si uniformerà al gruppo primario. Qiesta elaborazione di replica iniziale è asincrona. Una volta che il gruppo di replica raggiunge il gruppo primario allora la coppia entra in modalità peer e, alla fine, la replica sincrona inizia. da questo punto in poi, ogni transazione di cui viene eseguito il commit sul gruppo primario verrà inviata a tutti i gruppi di replica in modalità peer e il gruppo primario attende un messaggio di notifica. Questa operazione rallenta il gruppo primario quando viene confrontato con uno scenario di replica asincrona a causa della latenza coinvolta nella ricezione dei messaggi di notifica. Una sequenza di commit sincrona sul gruppo primario ha il seguente aspetto:

Operazione con il programma di caricamento	**Operazione senza il programma di caricamento**
Richiamo dei blocchi per le voci	stesso
Eliminazione delle modifiche sul programma di caricamento	NOOP
salvare le modifiche nella cache	stesso
Invio delle modifiche ai gruppi di replica e attesa per la notifica	stesso
Commit sul programma di caricamento mediante il plug-in TransactionCallback	Il commit del plug-in TransactionCallBack viene ancora richiamato ma non esegue alcuna operazione.
Rilascio dei blocchi per le voci	stesso

Si noti che le modifiche vengono inviate alla replica prima del commit al programma di caricamento. Quando viene eseguito il commit delle modifiche sulle replica? Fare riferimento a questa sequenza:

In fase di inizializzazione, inizializzare gli elenchi tx sul gruppo primario.

- Set CommitedTx = {}, RolledBackTx = {}

Durante l'elaborazione del commit sincrono:

Operazione con il programma di caricamento	Operazione senza il programma di caricamento
Richiamo dei blocchi per le voci	stesso
Eliminazione delle modifiche sul programma di caricamento	NOOP
salvare le modifiche nella cache	stesso
Inviare le modifiche con una transazione di cui è stato eseguito il commit e una transazione di cui è stato eseguito il rollback alla replica e attendere la notifica	stesso
Cancellare l'elenco di transazioni di cui è stato eseguito il commit e il rollback	stesso
Commit sul programma di caricamento mediante il plug-in TransactionCallback	Il commit del plug-in TransactionCallBack viene ancora richiamato ma non esegue alcuna operazione.
Se il commit riesce correttamente, aggiungere la transazione alle transazioni di cui è stato eseguito il commit, altrimenti aggiungerla alle transazioni di cui è stato eseguito il rollback	NOOP
Rilascio dei blocchi per le voci	stesso

Elaborazione della replica

- Ricezione delle modifiche
- Eseguire il commit di tutte le transazioni ricevute nell'elenco di transazioni con commit
- Eseguire il rollback di tutte le transazioni ricevute nell'elenco di transazioni con rollback
- Avviare una transazione o una sessione
- Applicare le modifiche alla transazione o alla sessione
- Salvare la transazione o la sessione all'elenco in sospeso
- Restituire la risposta

Si noti che sulla replica non vi sono interazioni del programma di caricamento mentre ci si trova in modalità di replica. Il gruppo primario deve inserire tutte le modifiche mediante il programma di caricamento. La replica viene eseguita automaticamente.

Un effetto collaterale di questo algoritmo è che la replica ha sempre le transazioni ma non viene eseguito il commit di queste transazioni fino a che la transazione primaria invia lo stato del commit alle transazioni stesse. Esse vengono quindi applicate oppure viene eseguito il rollback sulla replica. Fino ad allora però, non verrà eseguito il commit delle transazioni. È possibile aggiungere un timer al gruppo che invia i risultati della transazione dopo un breve periodo di tempo (pochi secondi). In questo modo verrà limitata qualsiasi inattività nella finestra, ma non eliminerà la transazione completamente. Questa inattività rappresenta un problema solo quando si utilizza la modalità di lettura della replica. In caso contrario, essa è invisibile e non ha alcun impatto sull'applicazione.

Quando il gruppo primario riporta un errore, è possibile che vi siano delle transazioni di cui è stato eseguito il commit/rollback sul gruppo primario ma il messaggio non è mai stato restituito sulla replica con questi risultati. Quando un gruppo di replica viene promosso a nuovo gruppo primario, una delle prime azioni è gestire questa condizione. Ogni transazione in sospeso viene rielaborata rispetto alla nuova serie primaria di mappe. Se è presente un programma di caricamento, allora ogni transazione viene assegnata al programma di caricamento. Tali transazioni sono applicate in ordine FIFO. Se una transazione riporta un errore, allora questa verrà ignorata. Se sono presenti 3 transazioni in sospeso, A B e C, allora viene eseguito il commit di A, viene eseguito il rollback di B e ancora il commit di C. Nessuna transazione ha un impatto sulle altre. Si assume che queste siano indipendenti.

Un programma di caricamento può utilizzare una logica leggermente differente quando in modalità'ripristino failover' rispetto alla modalità 'normale'. Il programma di caricamento riconosce facilmente quando si trova in modalità di ripristino di failover implementando l'interfaccia ReplicaPreloadController. Il metodo checkPreloadStatus viene richiamato solo una volta completato il ripristino del failover. Pertanto, se il metodo apply dell'interfaccia del programma di caricamento viene richiamato prima di checkPreloadStatus, allora si ha una transazione di ripristino. Una volta richiamato il metodo checkPreloadStatus, allora il ripristino di failover viene completato.

Singleton stateful mediante la replica

WebSphere Extended Deployment ha aggiunto un supporto per i singleton nel primo rilascio con la funzione di partizionamento. Tale supporto consentiva alle applicazioni per creare singleton in un cluster. Il runtime ObjectGrid abilita una funzione simile utilizzando le MapSets replicate. Mentre il modello singleton di ObjectGrid fornisce parecchi vantaggi, esso ha anche degli svantaggi. La funzione di partizionamento fornisce un evento all'applicazione quando il singleton o la partizione viene attivato in locale, comunicato utilizzando il metodo partitionLoad della funzione di partizionamento. Una MapSet replicata ha anche un singleton, quello primario. L'applicazione riceve una notifica quando diventa primaria mediante il metodoReplicaPreloadController#checkPreloadStatus sul programma di caricamento. Essa può essere utilizzata esattamente come una funzione di partizionamento ma ha il vantaggio di poter essere utilizzata tra versioni differenti di WebSphere Application Server o su server delle applicazioni paralleli.

La funzione di partizionamento ha un evento di disattivazione, ma il runtime ObjectGrid non fornisce questa funzione. Un gruppo primario nell'ObjectGrid di solito viene eseguito fino a che riporta un errore. Non è possibile spostarlo. Questo è un vantaggio della funzione di partizionamento in ObjectGrid. di seguito è riportata una tabella delle varie funzioni:

Tabella 16.

Funzione	**Funzione di partizionamento**	**Singleton di ObjectGrid**
Evento di avvio singleton	Sì	Sì
Evento di arresto singleton	Sì	No
Replica dello stato del singleton	No	Sì
QoS (quality of service) variabile per la replica	No	Sì

Tabella 16. (Continua)

Funzione	Funzione di partizionamento	Singleton di ObjectGrid
Inserimento flessibile del singleton	Sì	No
Spostamento del singleton in fase di runtime	Sì	No
Indirizzamento IIOP del lavoro al singleton	Sì	No
Richiesta del server Java 2 Platform, Enterprise Environment (J2EE)	Sì	No
Richiesta di una versione completa di WebSphere Extended Deployment	Sì	No
Richiesta di Enterprise JavaBeans (EJB)	Sì	No
L'applicazione può essere utilizzata su altri server delle applicazioni	No	Sì

Stato singleton

La funzione di partizionamento non forniva alcun supporto integrato per la gestione dello stato. Le applicazioni venivano lasciate sui propri dispositivi se il singleton richiedeva lo stato. Di solito, ciò significava che lo stato veniva inserito in un database. Se la partizione riportava un errore allora il server che veniva promosso a host per ripristinare la partizione doveva richiamare questo stato dal database. Se un'applicazione utilizza invece l'ObjectGrid, allora il singleton può conservare lo stato nella mappa associata al ReplicaPreloadController gestisce il singleton. Se il gruppo primario o il singleton riporta un errore, allora il gruppo di replica che viene promosso a nuovo gruppo primario ha già lo stato in locale a causa della replica. La replica sincrona deve essere utilizzata a meno che la perdita di dati non sia accettabile per l'applicazione.

Inserimento flessibile del singleton

La funzione di partizionamento utilizza il meccanismo della politica del gestore HA (high availability) per determinare se sarà presente una partizione e se tali politiche possano essere modificate durante il runtime con effetto immediato. Le politiche del gruppo di replica ObjectGrid non sono così flessibili come quelle del gestore HA e non possono essere modificate senza riavviare tutti i server. Non è più possibile spostare i singleton durante il runtime se si utilizza l'ObjectGrid.

Replica QoS della variabile

La funzione di partizionamento non offre la gestione dello stato. L'ObjectGrid offre una varietà di approcci di replica:

- Nessuna replica
- Replica asincrona
- Replica sincrona

La politica di replica della MapSet associata alla mappa che si utilizza per lo stato determina la politica. Una replica sincrona significa che non vengono persi i dati, ma che l'intero processo è più lento. La replica asincrona è più rapida ma significa che una o più transazioni con commit sul gruppo primario possono andare perse se il gruppo primario riporta un errore.

Bilanciamento del carico tra le repliche

L'ObjectGrid, a meno che non sia configurato diversamente, invia tutte le richieste di lettura e scrittura al server primario per un determinato gruppo di replica. Ciò significa che il server primario da solo deve rispondere a tutte le richieste dai client. È possibile consentire che le richieste di lettura vengano inviate alle repliche del server primario. Ciò consente al carico delle richieste di lettura la condivisione su più JVM, tuttavia, inviando le richieste di lettura alle repliche a spese della coerenza.

Ciò viene di solito utilizzato quando i client memorizzano nella cache i dati in continua modifica o quando i client utilizzano i blocchi pessimistici.

Se i dati vengono modificati continuamente e vengono quindi invalidati nelle cache del client e sul gruppo primario, allora come risultato si avrà una frequenza relativamente alta di richieste get dai client. Allo stesso modo, in modalità di blocco pessimistico, non esiste una cache locale pertanto tutte le richieste vengono inviate al server primario.

Se i dati sono relativamente statici o se non viene utilizzata la modalità pessimistica, allora la lettura della replica non ha un grosso impatto sulle prestazioni in quanto la frequenza di richieste get dai client con le cache non saranno elevate.

Tuttavia, quando viene avviato un client, la cache è vuota e le richieste della cache alla cache vuota vengono inoltrate al server primario. La cache del client richiama i dati nel tempo, provocando l'eliminazione del carico di richieste. Se esiste un numero elevato di client e molti di essi vengono avviati simultaneamente, allora questo carico potrebbe essere significativo e la lettura della replica può essere una scelta di prestazioni appropriata.

Letture da repliche e replica asincrona

Se i dati nel gruppo di replica non vengono modificati spesso allora è stato ottenuto un buon compromesso. Ciò consente alle richieste get dai client di essere indirizzate ai dati su qualsiasi replica online. Una richiesta get potrebbe essere inviata a una replica che non ha una copia e la chiave/valore potrebbe non essere replicata sul gruppo di replica a quel punto. Se i dati non sono presenti sul gruppo di replica allora la richiesta get viene reindirizzata al gruppo primario.

Se i dati vengono modificati, allora è molto probabile che le richieste get dai gruppi di replica restituiscano dati inattivi. Ciò può essere accettato o meno dall'applicazione. Se non è accettabile, allora non abilitare le letture dai gruppi di replica.

Letture da repliche e in modalità di replica sincrona

La replica sincrona prova a mantenere la replica esattamente come per il gruppo primario. Se il gruppo primario riporta un errore allora tutti i dati con commit sul gruppo primario sono disponibili per tutti i gruppi di replica che si trovavano in

modalità peer quando si è verificato l'errore. In questo caso, le letture dai gruppi di replica mostrano degli effetti collaterali degli algoritmi utilizzati.

Quando il gruppo primario esegue il commit di una transazione, una copia delle modifiche viene inviata alla replica e la replica esegue il commit della transazione nei seguenti due casi:

- Il gruppo primario riporta un errore
- La transazione successiva sul gruppo primario viene inviata

Quando il gruppo primario riporta un errore, viene eseguito il commit di tutte le transazioni in sospeso sulla replica.

Il commit delle transazioni in sospeso viene eseguito solo se viene eseguito il commit della transazione successiva sul gruppo primario. Il gruppo primario restituisce il messaggio di replica per i risultati del commit. Quando la replica riceve uno di questi messaggi, viene eseguito il commit o il rollback delle transazioni in sospeso che avevano i risultati specificati nel messaggio.

Le transazioni in sospeso diventano visibili per la lettura su una replica solo quando ne viene eseguito il commit. Naturalmente, se il gruppo primario viene caricato e sono state apportate delle modifiche regolari, allora il commit delle transazioni in sospeso viene eseguito molto rapidamente. Se il carico di modifiche sul server primario è scarso allora esistono periodi in cui non viene eseguito il commit delle transazioni in sospeso fino a che viene eseguita una modifica al gruppo primario.

Chiaramente, la replica per un gruppo primario per cui sono apportate delle modifiche è almeno una transazione del gruppo primario da un punto di vista della lettura. I dati non vengono persi, le transazioni si trovano fisicamente sulla replica, il commit delle transazioni non viene eseguito fino a che i risultati delle transazioni in sospeso vengono inviati al gruppo primario. Questo commit si verifica quando viene eseguita la successiva transazione di lettura e scrittura.

Riepilogo

Se la lettura dalla replica è abilitata, allora l'applicazione deve essere preparata a tollerare la restituzione di dati inattivi. Ciò si verifica quando viene utilizzata la replica sincrona o asincrona.

Partizionamento

Utilizzare il partizionamento quando gli oggetti nel MapSet richiedono più memoria di quella disponibile in una sola Java virtual machine (JVM) o se la JVM non è in grado di fornire la velocità di trasmissione dati richiesta per gli aggiornamenti.

Dove vengono conservate le voci

Un algoritmo hash determina il server che conserva ogni voce. L'amministratore specifica il numero di partizioni da utilizzare con la definizione PartitionSet. Questa configurazione non può essere modificata in seguito all'avvio delle JVM. Un valore hash semplice viene ottenuto dalla chiave per una voce e il risultato del modulo di questo valore (%) per il numero di partizioni indica il server che ″possiede″ la voce.

Di solito, viene utilizzato il metodo hashCode Java sull'oggetto della chiave. Sovrascrivere questo valore sovrascrivendo l'implementazione hashCode.

A volte, un'applicazione potrebbe non modificare il valore per l'hash ma continuare a utilizzare un algoritmo hash differente per la distribuzione delle voci. L'interfacciacom.ibm.websphere.objectgrid.plugins.PartitionableKey consente questa situazione. Questa interfaccia ha un unico metodo:

```
Object ibmGetPartition();
```

Se la chiave implementa questa interfaccia, il runtimeObjectGrid utilizza l'hash dell'oggetto che viene restituito da questo metodo piuttosto che l'hash sull'oggetto della chiave.

Partizionamento in fase di runtime

L'interfaccia com.ibm.websphere.objectgrid.PartitionManager fornisce delle API per consentire a un'applicazione di determinare le informazioni sul partizionamento durante il runtime. Un'applicazione può ottenere un riferimento a un'istanza di questa interfaccia utilizzando il metodo getPartitionManager dell'interfacciaBackingMap. Un riferimento al BackingMap per una mappa può essere ottenuto utilizzando il metodo getMap(String) dell'interfaccia ObjectGrid su qualsiasi istanza ObjectGrid. In alternativa, viene inviato come parametro su alcune richiamate di plug-in, come ad esempio preload(Session, BackingMap) dell'interfaccia del programma di caricamento.

Metodi dell'interfaccia PartitionManager

L'istanza PartitionManager consente a un'applicazione di determinare i seguenti fatti sul partizionamento:

Nome metodo	Descrizione
int getNumOfPartitions()	Restituisce il numero di partizioni in cui viene divisa la mappa.
int getPartition(Object key)	Restituisce il numero di partizioni basato su 0 utilizzato per la voce con la chiave specificata.
List /*Integer*/ getPartitions(List /*Object*/ keys)	Questo metodo è uguale al metodo getPartition con l'unica differenza che opera su un elenco di chiavi. L'elenco di interi restituito contiene il numero di partizioni per ogni chiave di input corrispondente.
List /*List Integer*/ getPartitionLists(List /* Object */ keys)	Questo metodo è uguale al metodo getPartitions ma restituisce un elenco ordinato di elenchi di partizioni. Ad esempio, la prima voce nell'elenco restituito contiene un elenco di chiavi di input che corrispondono alla partizione 0. La voce successiva contiene un elenco di chiavi di input che corrispondono alla partizione 1 e così via.
List /*LogSequence*/ partitionLogSequence(LogSequence ls)	Questo metodo suddivide un LogSequence in un elenco di LogSequences per le partizioni specificate. Il LogSequence di input viene esaminato e la partizione appropriata viene determinata per ogni LogElement all'interno di esso. Una volta esaminata la sequenza, allora per ogni partizione che ha un LogElement, viene restituito un LogSequence di tali LogElements.

Limitazioni del partizionamento

Una transazione può modificare soltanto le voci in una sola partizione per transazione. Se una transazione modifica più voci in una MapSet e tali voci sono suddivise su diverse partizioni, viene eseguito il rollback delle transazioni quando viene eseguito un tentativo di commit della transazione. Una transazione può leggere oggetti da partizioni differenti. Tuttavia, una transazione può modificare soltanto le voci all'interno di una sola partizione.

Eventi dell'applicazione quando viene promossa una macchina primaria per una partizione

Se viene fornito un programma di caricamento per una mappa e viene implementato ReplicaPreloadController dal programma di caricamento, allora l'applicazione può utilizzare la richiamata checkPreloadStatus per ricevere un evento che indica che JVM che riceve la chiamata al metodo è la macchina primaria per quella partizione. L'ID di partizione può essere identificato utilizzando il metodo getPartitionId dell'interfaccia BackingMap. Fare riferimento a "Programmi di caricamento" a pagina 197 per ulteriori informazioni sul precaricamento.

Partizionamento su un client rispetto all'esecuzione su un server

Il partizionamento funziona solo se l'applicazione utilizza un ObjectGrid ottenuto utilizzando i metodi connect dell'interfaccia ObjectGrid. Se l'ObjectGrid è fornito all'applicazione da una richiamata su un plug-in, allora è un ObjectGrid locale che non esegue l'instradamento. Se si esegue su un server e si desidera poter sfruttare i vantaggi delle funzioni di partizionamento, utilizzare un ObjectGrid ottenuto utilizzando un metodo connect per tutte le transazioni. Tuttavia, si verifica un calo di prestazioni se confrontato con l'utilizzo del riferimento all'ObjectGrid locale fornito dalla struttura. Se non sono necessarie le funzioni di partizionamento, allora utilizzare il riferimento locale fornito con il plug-in.

Indicizzazione

La funzione di indicizzazione può essere utilizzata per creare uno o più indici in BackingMap. Un indice viene creato da un attributo di un oggetto in BackingMap. Questa funzione consente alle applicazioni di individuare determinati oggetti più rapidamente. Senza un indice, le applicazioni individuano gli oggetti in base alle relative chiavi. La funzione di indicizzazione consente alle applicazioni di individuare gli oggetti con un valore specifico o all'interno di un intervallo di valori. Tale procedura è semplice alla query EJB (Enterprise JavaBeans) che consente di individuare gli oggetti EJB tramite richiesta con criteri specificati. L'indicizzazione fornisce alle applicazioni la convenienza di rilevare gli oggetti più facilmente e consente un miglioramento delle prestazioni nel processo di ricerca degli oggetti.

Esistono due tipi di indicizzazione: statica e dinamica. Con l'indicizzazione *statica*, è necessario configurare il plug-in dell'indice in BackingMap prima di inizializzare l'istanza ObjectGrid. È possibile creare tale configurazione con XML o tramite una configurazione programmatica di BackingMap. L'indicizzazione statica inizia la creazione di un indice durante l'inizializzazione di ObjectGrid. L'indice è sempre sincronizzato con BackingMap e pronto per l'uso. Una volta avviato il processo di indicizzazione statica, la gestione dell'indice fa parte del processo di gestione transazione di ObjectGrid. Quando le transazioni eseguono il commit delle

modifiche, tali modifiche aggiornano anche l'indice statico. Se viene eseguito il rollback della transazione, viene eseguito anche il rollback delle modifiche all'indice.

L'indicizzazione dinamica consente la creazione di un indice su BackingMap prima o dopo l'inizializzazione dell'istanza ObjectGrid che lo contiene. Le applicazioni hanno il controllo del ciclo vitale sul processo di indicizzazione dinamica. Un indice dinamico può essere rimosso quando non è più necessario. Quando un'applicazione crea un indice dinamico, l'indice potrebbe non essere pronto per un utilizzo immediato a causa del tempo necessario per completare il processo di creazione indice. Dal momento che il tempo dipende dalla quantità di dati indicizzati, viene fornita l'interfaccia DynamicIndexCallback per le applicazioni per cui si intende ricevere delle notifiche nel caso si verifichi un determinato evento di indicizzazione. Tali eventi includono ready, error e destroy. Le applicazioni possono implementare questa interfaccia di richiamata ed effettuare la registrazione con il processo di indicizzazione dinamica.

La funzione di indicizzazione è rappresentata dal plug-in MapIndexPlugin o dall'indice in forma abbreviata. MapIndexPlugin è un plug-in BackingMap. BackingMap può disporre di diversi plug-in Indice configurati purché essi seguano le regole di configurazione dell'indice.

Se BackingMap dispone di un plug-in dell'indice configurato, è possibile richiamare l'oggetto proxy dell'indice dalla corrispondente ObjectMap. La chiamata del metodo getIndex su ObjectMap e l'inoltro del nome del plug-in dell'indice restituisce l'oggetto proxy dell'indice. Occorre innanzitutto utilizzare tale oggetto in un'interfaccia appropriata dell'indice dell'applicazione, ad esempio MapIndex, MapRangeIndex, o un'interfaccia indice personalizzata.

Attualmente, la funzione di indicizzazione è supportata solo nella cache locale, non in quella distribuita. Se viene tentata un'operazione di indicizzazione rispetto a una cache distribuita, il risultato è un'eccezione `UnsupportedOperationException`.

Implementazione del plug-in dell'indice

La classe HashIndex nel pacchetto com.ibm.websphere.objectgrid.plugins.index è l'implementazione di plug-in indice incorporata che può supportare entrambe le interfacce dell'indice dell'applicazione incorporata: MapIndex e MapRangeIndex.

Le applicazioni possono fornire la propria implementazione di plug-in dell'indice per consentire la programmazione di indici più complessi. La classe di implementazione dell'indice deve implementare l'interfaccia com.ibm.websphere.objectgrid.plugins.index.MapIndexPlugin. Il MapIndexPlugin ha la seguente definizione:

```
/**
* Un'implementazione dell'indice deve implementare questa interfaccia in modo che le modifiche
* alla mappa vengano propagate ad essa e le consentano di conservare l'indice una volta
* eseguito il commit delle transazioni. Solo gli attributi che implementano l'interfaccia
* {@link java.lang.Comparable} possono essere indicizzati.
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapIndex
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex
*/
public interface MapIndexPlugin
{
/**
* Questo deve essere il nome dell'attributo da indicizzare. Se l'oggetto
* dispone di un attributo denominato EmployeeName, l'indice chiamerà il
* metodo "getEmployeeName". Il nome attributo deve essere lo stesso
```

```
* del metodo get e l'attributo deve implementare l'interfaccia
* {@link java.lang.Comparable}.
*
* @param attributeName
* Il nome dell'attributo da impostare.
*/
public void setAttributeName(String attributeName);
/**
* Nome indice.
*
* @return Il nome dell'indice.
*
* @see com.ibm.websphere.objectgrid.ObjectMap#getIndex
*/
 String getName();
/**
* Richiama un oggetto proxy dell'indice per l'esecuzione delle
* operazioni di ricerca indice. Il chiamante
* deve utilizzare l'oggetto restituito nell'oggetto MapIndex o MapRangeIndex
* per eseguire le operazioni di ricerca.
*
* @param map L'oggetto MapIndexInfo necessario per la gestione dell'indice.
* .
* @return un proxy a un oggetto che implementa MapIndex o MapRangeIndex.
*/
Object getIndexProxy(MapIndexInfo map);
/**
* Viene chiamato dal gruppo principale per consentire l'aggiornamento
* dell'indice come risultato delle modifiche applicate alla mappa
* durante il ciclo di commit di una transazione.
* Utilizzare il metodo {@link LogElement#getType()} per determinare
* quale operazione è richiesta per l'aggiornamento dell'indice.
* Utilizzare {@link LogElement#getBeforeImage()}per richiamare l'oggetto
* del valore che esisteva prima della transazione di commit che ha applicato
* una modifica alla mappa e al {@link LogElement#getAfterImage()}
* per richiamare l'oggetto del valore dopo che la transazione di commit
* ha applicato la modifica alla voce di mappa.
*
* È possibile chiamare il metodo {@link #undoBatchUpdate(TxID, LogSequence)}
* in un secondo momento per annullare tali modifiche se si verifica un'eccezione che causa il
* rollback delle transazioni di commit.
*
* @param txid La transazione per le modifiche.
* @param sequence La sequenza di log che contiene le modifiche della transazione.
*
* @throws ObjectGridRuntimeException è un errore che richiede il rollback
* della transazione.
*/
void doBatchUpdate(TxID txid, LogSequence sequence) throws
ObjectGridRuntimeException;
/**
* Viene chiamato dal gruppo principale per annullare le modifiche apportate
* all'indice come risultato di una precedente chiamata al metodo
* {@link #doBatchUpdate(TxID, LogSequence)}. Tale
* metodo viene chiamato in caso di una condizione di eccezione o di errore che richiede il
* rollback di tutte le modifiche apportate dalla transazione. Per tale motivo,
* l'implementazione di tale metodo dovrebbe rilevare tutti gli elementi Throwable
* e continuare con il successivo LogElement nella LogSequence finché tutti
* i LogElement non sono stati elaborati in modo che siano annullate il maggior
* numero possibile di modifiche all'indice. Una ObjectGridException
* dovrebbe essere emessa solo dopo l'elaborazione dell'intera LogSequence
* e nel caso in cui questo metodo non è stato in grado di annullare una
* o più modifiche nella LogSequence.
*
* Utilizzare il metodo {@link LogElement#getUndoType()} per determinare quale operazione è
* richiesta per l'annullamento delle modifiche apportate all'indice. Utilizzare
* {@link LogElement#getBeforeImage()} per richiamare l'oggetto valore che esisteva prima
```

```
* che la transazione di commit applicasse una modifica alla mappa e al {@link
* LogElement#getAfterImage()} per richiamare l'oggetto valore dopo che la transazione di commit
* ha applicato la modifica alla voce di mappa.
*
* @param txid La transazione per le modifiche.
* @param sequence La sequenza di log che contiene le modifiche della transazione.
*
*/
void undoBatchUpdate( TxID txid, LogSequence sequence) throws ObjectGridException;
}
```

I metodi setAttributeName e getName sono diretti e riconosciuti in base al nome. Gli altri metodi richiedono maggiore attenzione.

Metodo getIndexProxy

Il metodo getIndexProxy dovrebbe restituire un oggetto proxy dell'indice che implementa l'interfaccia MapIndex, MapRangeIndex o un'interfaccia dell'indice personalizzata. L'implementazione del suddetto oggetto è la parte principale del plug-in dell'indice.

Un oggetto MapIndexInfo viene passato in questo metodo per fornire informazioni sulla modifica della transazione. Questi dati sono visibili solo per la transazione corrente che richiama il metodo getIndexProxy. L'oggetto proxy dell'indice può utilizzare tale oggetto MapIndexInfo per ricercare questi dati di transazione.

Di seguito viene riportata la definizione dell'interfaccia MapIndexInfo:

```
/**
* Questa interfaccia viene utilizzata per fornire un indice con
* informazioni di modifica dettagliate per una specifica mappa in
* una transazione.
*/
public interface MapIndexInfo
{
/**
* Un indice contiene i valori chiave di una serie di voci di mappa con un
* valore di attributo specifico. Questo metodo restituisce l'ObjectMap a cui
* l'indice si riferisce e a cui è associato.
*
* @return ObjectMap a cui è associato questo indice.
*/
ObjectMap getMap();
/**
* Restituisce la serie di tutte le modifiche apportate dalla transazione corrente
* all'ObjectMap restituita dal metodo {@link #getMap()}.
*
* @param includeRemoved deve essere impostato su true per includere i tipi LogElement.DELETE
* nell'elenco restituito da questo metodo.
*
* @return un elenco di LogElement creati per ogni voce ObjectMap
* inserita, aggiornata o rimossa dalla transazione corrente.
*
* @throws ObjectGridRuntimeException
*/
List getTransactionChanges(boolean includeRemoved) throws
ObjectGridRuntimeException;
/**
* Restituisce la serie di modifiche che si applicano a una determinata serie di chiavi
* nella transazione corrente per l'ObjectMap restituito dal metodo
* {@link #getMap()}. Se non è stato fatto riferimento a una chiave
* nella transazione, viene restituito un valore nullo.
*
* @param keys L'elenco di chiavi per cui sono necessari i dati.
```

```
* @return un elenco di LogElement corrispondenti alle chiavi o un valore nullo se
* non è stato fatto riferimento alle chiavi.
*
* @throws ObjectGridRuntimeException
*
* @see com.ibm.websphere.objectgrid.plugins.LogElement
* @see com.ibm.websphere.objectgrid.ObjectMap
*/
List getTransactionChanges(List keys) throws ObjectGridRuntimeException;
}
```

Il metodo getIndexProxy è progettato per supportare il metodo getIndex(String name) dell'interfaccia ObjectMap. L'oggetto proxy dell'indice restituito sarà quello restituito dal metodo getIndex dell'ObjectMap. Ad esempio, l'applicazione richiama il metodo getIndex di ObjectMap, che richiama a sua volta questo metodo getIndexProxy e restituisce l'oggetto restituito da questo metodo getIndexProxy. L'applicazione deve utilizzare tale oggetto in un'interfaccia appropriata dell'indice dell'applicazione, ad esempio MapIndex, MapRangeIndex, o un'altra interfaccia indice personalizzata.

Il seguente esempio di codice illustra alcune delle implementazioni di oggetti proxy dell'indice che possono essere restituite dal metodo getIndexProxy:

```
/**
* Una classe utilizzata per restituire un proxy su questo indice di mappa
* in modo che le applicazioni possano eseguire le operazioni di query
* utilizzando l'interfaccia MapIndex.
*/
class Proxy implements MapIndex
{
/**
* L'oggetto MapIndexInfo associato a questo oggetto proxy dell'indice.
*/
protected MapIndexInfo ivMap;
/**
* Numero massimo di tentativi quando le transazioni contemporanee
* modificano l'indice durante un'operazione di query.
*/
protected static final int RETRY_LIMIT = 10;
/**
* comparatore EQUAL da utilizzare.
*/
final protected ProxyEQComparator ivEQ = new ProxyEQComparator();
final protected ProxyGTComparator ivGT = new ProxyGTComparator();
final protected ProxyRangeComparator ivRange = new ProxyRangeComparator();
/**
* Creare un oggetto proxy per un'ObjectMap specificata.
*
* @param map
* è l'oggetto 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(
"l'attributeValue deve essere un riferimento non nullo" );
```

```
}
// Utilizzare il comparatore maggiore di per il controllo dell'intervallo.
ivEQ.ivAttribute = (Comparable) attributeValue;
ArrayList resultList = null;
int retryCount = 0;
boolean retry;
do
{
// Variabili che è necessario reinizializzare ogni volta tramite un loop.
retry = false;
resultList = new ArrayList();
// Utilizzare l'indice per ottenere la serie di chiavi per le voci di mappa che
// contengono il valore attributo specificato.
Set s = (Set) index.get( attributeValue );
Set keySet = processSet( s, ivEQ );
if ( keySet != null )
{
resultList.addAll( keySet );
}
else
{
// Un'altra transazione ha modificato la serie ottenuta da un indice
// durante l'iterazione di quest'ultimo sulla serie per eseguire
// l'operazione addAll. Quindi è necessario riprovare partendo
// dall'inizio con il richiamo della serie dall'indice per acquisire le modifiche
// dalla transazione che ha modificato la serie.
++retryCount;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "superato il limite di tentativi query" );
}
retry = true;
}
} while ( retry );
// Restituire un iteratore per l'elenco di risultati creati dal precedente loop.
Iterator result = resultList.iterator();
return result;
}
/**
* Elaborare una serie ottenuta dall'indice per determinare quali delle chiavi
* sono relative alle voci di mappa che corrispondono ai criteri di selezione ricerca.
*
* @param s
* è la serie di valori chiave per le voci in BackingMap
* su cui è creato questo indice. Un riferimento nullo indica che
* è necessario elaborare soltanto le modifiche dalla transazione corrente.
*
* @param comparator
* è il comparatore da utilizzare per effettuare un controllo dell'intervallo.
*
* @return Serie di chiavi che corrispondevano ai criteri di selezione o un riferimento nullo
* se si verifica un'eccezione durante l'iterazione sulla serie.
*
* @throws FinderException
* se una condizione di errore impedisce l'esecuzione
* dell'elaborazione della serie.
*/
protected Set processSet(Set s, ProxyComparator comparator)
throws FinderException {
HashSet resultSet = new HashSet();
//...
//elaborare la serie s, utilizzare il comparatore e preparare la resultSet.
//...
return resultSet;
}
} // fine proxy classe
/**
```

```
* Una classe utilizzata per restituire un proxy su questo indice
* di mappa in modo che le applicazioni possano
* eseguire le operazioni di query tramite l'interfaccia MapRangeIndex.
*/
class RangeProxy extends Proxy implements MapRangeIndex
{
/**
* Vari comparatori necessari dal proxy per eseguire il
* controllo intervallo del valore di attributo.
*/
final private ProxyLTComparator ivLT = new ProxyLTComparator();
final private ProxyLEComparator ivLE = new ProxyLEComparator();
final private ProxyGEComparator ivGE = new ProxyGEComparator();
/**
* L'indice è una SortedMap sincronizzata.
*/
final SortedMap ivIndexSortedMap;
/**
* Creare un proxy MapRangeIndex.
*/
RangeProxy(MapIndexInfo map)
{
super( map );
ivIndexSortedMap = (SortedMap) index;
}
/**
* Eseguire un'operazione di query su un oggetto Map e ProxyComparator specificato.
*
* @param map
* è una sottoserie dell'indice su cui eseguire l'operazione del finder.
* @param proxyComparator
* è un comparatore utilizzato per eseguire il controllo intervallo sul valore di attributo.
*
* @return Serie di chiavi che deve essere restituita dal metodo finder.
*
* @throws FinderException
* se un errore che si verifica durante l'esecuzione della query.
*/
private Set executeQuery(Map map, ProxyComparator proxyComparator)
throws FinderException {
HashSet resultList = null;
int retryCount = 0;
boolean retry;
do
{
// Variabili che è necessario reinizializzare ogni volta tramite un loop.
retry = false;
resultList = new HashSet();
// Utilizzare l'indice per ottenere la serie di chiavi per le voci di mappa che
// contengono il valore attributo specificato.
SortedMap treeMap = (SortedMap) index;
Collection values = map.values();
if ( values.isEmpty() )
{
// Nessun elemento nell'intervallo corrente dell'indice, quindi
// è necessario soltanto verificare le modifiche della transazione corrente.
Set keySet = processSet( null, proxyComparator );
if ( keySet != null )
{
resultList.addAll( keySet );
}
}
else
{
// L'indice non contiene alcune chiavi nell'intervallo, quindi è necessario richiedere
// sia le voci di indice che le modifiche correnti alla transazione.
Iterator iter = values.iterator();
```

```
 while ( iter.hasNext() )
{
Set keySet;
try
{
Set s = (Set) iter.next();
keySet = processSet( s, proxyComparator );
}
catch (ConcurrentModificationException e)
{
// Indicare che è impossibile richiamare keySet.
keySet = null;
}
if ( keySet != null )
{
resultList.addAll( keySet );
}
else
{
++retryCount;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "superato il limite di tentativi query" );
}
retry = true;
}
}
}
} while ( retry );
return resultList;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findGreater
*/
public Iterator findGreater(Object attributeValue)
throws FinderException {
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"l'attributeValue deve essere un riferimento non nullo" );
}
// Utilizzare il comparatore maggiore di per il controllo dell'intervallo.
ivGT.ivAttribute = (Comparable) attributeValue;
SortedMap tailMap = ivIndexSortedMap.tailMap( attributeValue );
Set resultSet = executeQuery( tailMap, ivGT );
Iterator result = resultSet.iterator();
return result;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findGreaterEqual
*/
public Iterator findGreaterEqual(Object attributeValue)
throws FinderException {
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"l'attributeValue deve essere un riferimento non nullo" );
}
// Utilizzare il comparatore maggiore di per il controllo dell'intervallo.
ivGE.ivAttribute = (Comparable) attributeValue;
SortedMap tailMap = ivIndexSortedMap.tailMap( attributeValue );
Set resultSet = executeQuery( tailMap, ivGE );
Iterator result = resultSet.iterator();
```

```
return result;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findLess
*/
public Iterator findLess(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"l'attributeValue deve essere un riferimento non nullo" );
}
// Utilizzare il comparatore maggiore di per il controllo dell'intervallo.
ivLT.ivAttribute = (Comparable) attributeValue;
SortedMap headMap = ivIndexSortedMap.headMap( attributeValue );
Set resultSet = executeQuery( headMap, ivLT );
Iterator result = resultSet.iterator();
return result;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findLessEqual
*/
public Iterator findLessEqual(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"l'attributeValue deve essere un riferimento non nullo" );
}
// Utilizzare il comparatore maggiore di per il controllo dell'intervallo.
ivLE.ivAttribute = (Comparable) attributeValue;
Set resultSet;
int retryCount = 0;
boolean retry;
do
{
// Reinizializzare per ogni voce che si verifica.
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
{
// Un'altra transazione ha modificato l'indice durante l'esecuzione di
// ResultSet. Quindi è necessario ripetere l'intera query.
++retryCount;
retry = true;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "superato il limite di tentativi query" );
}
}
} while ( retry );
// Restituire un iteratore per l'elenco di risultati creati dal precedente loop.
```

```
Iterator result = resultSet.iterator();
return result;
}
/*
* (non-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(
"lowAttributeValue deve essere un riferimento non nullo" );
}
if ( highAttributeValue == null )
{
throw new IllegalArgumentException(
"highAttributeValue deve essere un riferimento non nullo" );
}
// Utilizzare il comparatore maggiore di per il controllo dell'intervallo.
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 di base astratta utilizzata per determinare se il valore dell'attributo
* è compreso nell'intervallo.
*/
abstract class ProxyComparator
{
abstract boolean inRange(Object attribute);
}
/**
* Eseguire il controllo dell'intervallo Inferiore a.
*/
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 );
}
}
}
/**
* Eseguire il controllo dell'intervallo Inferiore a o Uguale a.
*/
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 );
}
}
}
/**
* Eseguire il controllo dell'intervallo Uguale a.
*/
class ProxyEQComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
return false;
}
else
{
return ( ivAttribute.compareTo( attribute ) == 0 );
}
}
}
/**
* Eseguire il controllo dell'intervallo Superiore a.
*/
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 );
}
}
}
/**
* Eseguire il controllo dell'intervallo Superiore a o Uguale a.
*/
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 );
}
}
}
/**
* Eseguire il controllo dell'intervallo lowAttribute <= attributo < 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; // attributo < ivLowAttribute
}
else
{
// ivLowAttribute <= attributo
if ( attribute.compareTo( ivHighAttribute ) < 0 )
{
return true; // ivLowAttribute <= attributo < ivHighAttribute
}
else
{
return false; // attributo >= ivHighAttribute
}
}
}
}
```

Metodi doBatchUpdate e undoBatchUpdate

I metodi doBatchUpdate e undoBatchUpdate sono essenziali nell'interfaccia MapIndexPlugin. Il metodo doBatchUpdate viene richiamato come risultato delle modifiche applicate alla mappa durante il ciclo di commit di una transazione. Il metodo undoBatchUpdate viene utilizzato per annullare le modifiche apportate all'indice come risultato di una chiamata precedente al metodo doBatchUpdate. Esso viene chiamato in caso di una condizione di eccezione o di errore che richiede il rollback di tutte le modifiche apportate dalla transazione. Ad entrambi i metodi viene fornito il TxID corrente e un elenco di modifiche per questa mappa. Essi dovrebbero eseguire un'iterazione sulle modifiche ed elaborarle.

Il seguente esempio di codice mostra come implementare questi due metodi e i metodi di supporto.

```
/**
* La mappa sincronizzata utilizzata come implementazione dell'indice in cui
* l'oggetto del valore attributo è la chiave e java Set è il valore.
* Un membro della serie è la chiave di una voce BackingMap che corrisponde al valore di attributo.
*/
Map index; //<attributi oggetto, chiavi della serie>
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
+ " si sono verificate delle eccezioni durante il rollback delle modifiche all'indice.");
}
}
/**
* Estrae l'attributo da un valore Oggetto specificato.
*
* @param value Il valore Oggetto.
```

```
*
* attributo @return dal valore Oggetto, che potrebbe essere un riferimento nullo.
*
* @throws ObjectGridRuntimeException viene emessa se si verifica un'eccezione
* nel tentativo di estrarre il valore di attributo dal valore Oggetto.
*/
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( "rilevato Throwable imprevisto", t );
}
catch ( Throwable t )
{
throw new ObjectGridRuntimeException( "rilevato Throwable imprevisto", t );
}
}
private void updateIndex(Object key, Object oldAttribute, Object newAttribute)
{
// L'attributo è stato modificato dall'aggiornamento?
if ( newAttribute != null && oldAttribute != null &&
oldAttribute.equals( newAttribute ) )
{
// No, quindi non è necessario modificare nulla nell'indice.
return;
}
// A meno che il programma di caricamento non venga limitato ad accedere
soltanto alle tabelle con colonne che non accettano valori nulli,
// è necessario gestire la possibilità che l'attributo sia nullo.
Set oldKeys = null;
if ( oldAttribute != null )
{
// Rimuovere oldAttribute dalla voce di indice.
oldKeys = (Set) index.get( oldAttribute );
if ( oldKeys != null )
{
oldKeys.remove( key );
if ( oldKeys.isEmpty() )
{
index.remove( oldAttribute );
}
}
}
// A meno che il programma di caricamento non venga limitato ad accedere
soltanto alle tabelle con colonne che non accettano valori nulli,
// è necessario gestire la possibilità che l'attributo sia nullo.
Set keys = null;
if ( newAttribute != null )
{
keys = (Set) index.get( newAttribute );
// Aggiungere newAttribute all'indice.
if ( keys == null )
{
// Dal momento che differenti transazioni possono aggiornare differenti voci BackingMap
// e più voci di mappa possono avere lo stesso valore di attributo,
// è necessario utilizzare un oggetto Serie sincronizzato per garantire che
```

```
// una sola transazione alla volta possa apportare modifiche alla serie.
keys = Collections.synchronizedSet( new HashSet() );
index.put( newAttribute, keys );
}
// Aggiungere la chiave per questa voce di mappa alla serie di chiavi per il nuovo
   valore di attributo.
keys.add( key );
}
}
private void insertIntoIndex( Object key, Object newAttribute )
{
// A meno che il programma di caricamento non venga limitato ad accedere
soltanto alle tabelle con colonne che non accettano valori nulli,
// è necessario gestire la possibilità che l'attributo sia nullo.
if ( newAttribute != null )
{
Set keys = (Set) index.get( newAttribute );
if ( keys == null )
{
// Dal momento che differenti transazioni possono aggiornare differenti voci
// di mappa e più voci di mappa possono avere lo stesso valore di attributo,
// è necessario utilizzare un oggetto Serie sincronizzato per garantire che
// una sola transazione alla volta possa apportare modifiche alla serie.
keys = Collections.synchronizedSet( new HashSet() );
index.put( newAttribute, keys );
}
// Aggiungere la chiave per questa voce di mappa alla serie di chiavi
per il nuovo valore di attributo.
keys.add( key );
}
}
private void removeFromIndex(Object key, Object oldAttribute )
{
// Estrarre il valore di attributo obsoleto
Set oldKeys = null;
// A meno che il programma di caricamento non venga limitato ad accedere
soltanto alle tabelle con colonne che non accettano valori nulli,
// è necessario gestire la possibilità che l'attributo sia nullo.
if ( oldAttribute != null )
{
oldKeys = (Set) index.get( oldAttribute );
if ( oldKeys != null )
{
oldKeys.remove( key );
if ( oldKeys.isEmpty() )
{
index.remove( oldAttribute );
}
}
}
}
```

Interfacce dell'indice dell'applicazione

Le interfacce dell'indice dell'applicazione sono progettate per supportare i metodi di query. Attualmente, sono definite due interfacce dell'indice dell'applicazione: MapIndex e MapRangeIndex.

MapIndex

MapIndex è un indice semplice per la ricerca di oggetti in base a un valore di attributo. Esso consente l'indicizzazione di qualsiasi valore di attributo in una mappa. Ciò permette all'applicazione di rilevare rapidamente tutti gli oggetti nella mappa che hanno uno specifico valore di attributo. Di seguito viene riportata la definizione dell'interfaccia MapIndex:

```
/**
* Questo è un indice astratto che è possibile creare in una mappa vuota. L'indice
* può essere utilizzato per eseguire in modo efficiente ricerche ed altre
* operazioni, tra cui le operazioni relazionali su un attributo in una mappa.
* Il MapIndex viene fornito con tutti gli eventi di aggiornamento e conserva un
* indice che è possibile utilizzare per immettere semplici query rispetto all'indice
* in un secondo momento. L'indice può utilizzare una richiamata definita dall'indice
* stesso per creare un indice su attributi compositi.
*/
public interface MapIndex
{
/**
* Restituisce le chiavi per le voci con il valore di attributo
* specificato.
*
* @param attributeValue
* un riferimento non nullo al valore di attributo da ricercare.
*
* @return Un elenco di chiavi per le voci con tale attributo.
*
* @throws IllegalArgumentException se l'argomento attributeValue è nullo.
* @throws FinderException viene emessa se viene raggiunto
* il limite di eccezione o di tentativi in presenza di transazioni
* simultanee che aggiornano l'indice impedendo il completamento di findAll.
*/
Iterator findAll(Object attributeValue) throws FinderException;
}
```

MapRangeIndex

MapRangeIndex è un indice semplice per la ricerca di oggetti con un valore di attributo compreso in un determinato intervallo. Esso consente l'indicizzazione di qualsiasi valore di attributo in una mappa. Esso differisce da MapIndex perché consente le query utilizzando gli intervalli di valori e le operazioni di confronto tra valori. Ciò consente alle query di rilevare tutti gli oggetti con un valore di attributo inferiore o superiore rispetto a un valore specifico. Di seguito viene riportata la definizione dell'interfaccia MapRangeIndex:

```
/**
* Questo è un indice che consente le ricerche del tipo di confronto.
*/
public interface MapRangeIndex extends MapIndex
{
/**
* Rileva tutte le chiavi le cui voci hanno un attributo superiore al
* valore specificato.
*
* @param attributeValue è l'endpoint inferiore dell'intervallo, escluso il
* valore di attributo inferiore.
*
* @return La serie di chiavi con valori superiori all'attributo.
*
* @throws IllegalArgumentException se l'argomento attributeValue è nullo.
* @throws FinderException viene emessa se viene raggiunto il limite di eccezione
* o di tentativi in presenza di transazioni simultanee che aggiornano l'indice
* impedendo il completamento di findAll.
*/
Iterator findGreater(Object attributeValue) throws FinderException;
/**
* Rileva tutte le chiavi le cui voci hanno un attributo superiore o uguale al
* valore specificato.
*
* @param attributeValue è l'endpoint inferiore dell'intervallo, incluso il
* valore di attributo inferiore.
*
* @return La serie di chiavi con attributi che corrispondono ai criteri
```

```
*
* @throws IllegalArgumentException se l'argomento attributeValue è nullo.
* @throws FinderException viene emessa se viene raggiunto
* il limite di eccezione o di tentativi
* in presenza di transazioni simultanee che aggiornano l'indice impedendo
* il completamento di findAll.
*/
Iterator findGreaterEqual(Object attributeValue) throws FinderException;
/**
* Rileva tutte le chiavi le cui voci hanno un attributo inferiore al
* valore specificato.
*
* @param attributeValue è l'endpoint superiore dell'intervallo, escluso il
* valore di endpoint superiore.
*
* @return La serie di chiavi con attributi che corrispondono ai criteri
*
* @throws IllegalArgumentException se l'argomento attributeValue è nullo.
* @throws FinderException viene emessa se viene raggiunto il limite di eccezione
* o di tentativi
* in presenza di transazioni simultanee che aggiornano l'indice impedendo
* il completamento di findAll.
*/
Iterator findLess(Object attributeValue) throws FinderException;
/**
* Rileva tutte le chiavi le cui voci hanno un attributo inferiore o uguale al
* valore specificato.
*
* @param attributeValue è l'endpoint superiore dell'intervallo, incluso il
* valore di endpoint superiore.
*
* @return La serie di chiavi con attributi che corrispondono ai criteri
*
* @throws IllegalArgumentException se l'argomento attributeValue è nullo.
* @throws FinderException viene emessa se viene raggiunto il limite di eccezione
* o di tentativi
* in presenza di transazioni simultanee che aggiornano l'indice impedendo
* il completamento di findAll.
*/
Iterator findLessEqual(Object attributeValue) throws FinderException;
/**
* Restituisce tutte le chiavi per le voci con l'attributo incluso
* nell'intervallo specificato, ad esempio lowAttributeValue <= attributo
* < highAattributeValue.
*
* @param lowAttributeValue è l'endpoint inferiore dell'intervallo, incluso il
* valore di attributo inferiore.
* @param highAttributeValue è l'endpoint superiore dell'intervallo, escluso il
* valore di attributo superiore.
*
* @return L'elenco di chiavi con voci in tale intervallo, in ordine ascendente.
*
* @throws IllegalArgumentException se l'argomento lowAttributeValue o
* highAttributeValue
* è nullo o lowAttributeValue > highAttributeValue.
* @throws FinderException viene emessa se viene raggiunto il limite di eccezione o di tentat
* in presenza di transazioni simultanee che aggiornano l'indice impedendo
* il completamento di findAll.
*/
Iterator findRange(Object lowAttributeValue, Object highAttributeValue)
throws FinderException;
}
```

È necessario che le applicazioni utilizzino l'oggetto di indice ottenuto dal metodo getIndex dell'istanza ObjectMap nell'interfaccia dell'indice dell'applicazione desiderato. Se il plug-in dell'indice è progettato per

supportare l'interfaccia MapRangeIndex, è possibile utilizzare l'oggetto di indice nel tipo MapRangeIndex; altrimenti è necessario utilizzarlo nel tipo MapIndex.

È possibile definire un'interfaccia dell'indice dell'applicazione personalizzata. Implementare l'indice dell'applicazione personalizzato come oggetto proxy dell'indice che è possibile restituire tramite il metodo getIndexProxy di MapIndexPlugin. Utilizzare l'oggetto di indice ottenuto dal metodo getIndex dell'istanza ObjectMap in questa interfaccia dell'indice dell'applicazione personalizzata.

Aggiunta di plug-in dell'indice statico

Esistono due approcci per aggiungere dei plug-in statici alla configurazione BackingMap: la configurazione XML e la configurazione programmatica. Il seguente esempio illustra l'approccio Configurazione 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>
```

L'interfaccia BackingMap prevede due metodi che è possibile utilizzare per aggiungere dei plug-in di indice statici: addMapIndexPlugin e setMapIndexPlugins. Di seguito viene riportata la definizione di questi due metodi.

```
/**
* Questo metodo aggiunge un plug-in di indice a questa mappa.
* Si presuppone che l'implementazione dell'indice sia stata creata
* con il nome dell'attributo da indicizzare. Il nome dell'indice viene specificato
* al momento della creazione dell'indice stesso.
*
* Per evitare una {@link IllegalStateException}, tale metodo deve essere chiamato
* prima del metodo {@link ObjectGrid#initialize()}. Inoltre il metodo
* {@link ObjectGrid#getSession()} chiama implicitamente il
* metodo {@link ObjectGrid#initialize()} se tale metodo non è ancora stato chiamato
* dall'applicazione.
*
* @param index L'implementazione dell'indice.
*
* @throws IndexAlreadyDefinedException Questo indice esiste già.
* @throws IllegalStateException se questo metodo viene richiamato dopo
* il metodo {@link ObjectGrid#initialize()}.
*/
public void addMapIndexPlugin(MapIndexPlugin index)
throws IndexAlreadyDefinedException;
/**
* Questo metodo imposta l'elenco di oggetti MapIndexPlugin per questo BackingMap.
* Se BackingMap dispone già di un elenco di oggetti MapIndexPlugin,
* tale elenco viene sostituito dall'elenco passato come
* argomento al richiamo corrente di questo metodo.
*
* Per evitare una {@link IllegalStateException}, tale metodo deve essere chiamato
* prima del metodo {@link ObjectGrid#initialize()}. Inoltre il metodo
* {@link ObjectGrid#getSession()} chiama implicitamente il
* metodo {@link ObjectGrid#initialize()} se tale metodo non è ancora stato chiamato
* dall'applicazione.
```

```
*
* @param indexList Un riferimento non nullo a un elenco di oggetti {@link MapIndexPlugin}
* .
*
* @throws IllegalArgumentException viene emessa se indexList è nullo
* o se indexList contiene un oggetto che non è
* un'istanza di {@link MapIndexPlugin}.
*/
public void setMapIndexPlugins(List indexList );
```

Il seguente frammento di codice illustra l'approccio di configurazione programmatica:

```
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");
//utilizzare la classe HashIndex incorporata come classe di plugin dell'indice.
HashIndex mapIndexPlugin = new HashIndex();
mapIndexPlugin.setName("CODE");
mapIndexPlugin.setAttributeName("EmployeeCode");
mapIndexPlugin.setRangeIndex(true);
personBackingMap.addMapIndexPlugin(mapIndexPlugin);
```

Utilizzo di indici statici

Una volta aggiunto un plug-in di indice statico a una configurazione BackingMap e dopo avere inizializzato l'istanza ObjectGrid che lo contiene, le applicazioni possono richiamare l'oggetto dell'indice in base al nome dall'istanza ObjectMap per BackingMap. Utilizzare l'oggetto indice nell'interfaccia dell'indice dell'applicazione. Le operazioni dell'indice supportate dall'interfaccia dell'indice dell'applicazione possono ora essere eseguite. Di seguito viene riportata la definizione del metodo getIndex dell'interfaccia ObjectMap:

```
/**
* Restituisce all'indice specificato un riferimento che è possibile utilizzare con questa mappa.
* Questo indice non può essere condiviso tra thread e lavori sulle stesse regole della
* sessione. Il valore restituito deve essere utilizzato nell'interfaccia dell'indice di destra,
* ad esempio MapIndex o MapRangeIndex o in un'interfaccia dell'indice personalizzata come ad esempio
* indice geo-spaziale.
*
* @param name Il nome dell'indice
*
* @return Un riferimento all'indice, deve essere utilizzato nell'interfaccia dell'indice
* appropriata.
*
* @throws IndexUndefinedException se l'indice non è definito in BackingMap
* @throws IndexNotReadyException se l'indice non è pronto
* @throws UnsupportedOperationException se la mappa è distribuita
*/
Object getIndex(String name)
throws IndexUndefinedException, IndexNotReadyException,
UnsupportedOperationException;
```

Il seguente frammento di codice illustra le modalità di acquisizione e utilizzo degli indici statici:

```
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);
}
```

Aggiunta e rimozione di indici dinamici

Gli indici dinamici possono essere creati e rimossi da un'istanza BackingMap in modo programmatico, in qualsiasi momento. Un indice dinamico differisce da un indice statico perché può essere creato anche dopo l'inizializzazione dell'istanza ObjectGrid che lo contiene. A differenza dell'indicizzazione statica, l'indicizzazione dinamica è un processo asincrono che deve essere nello stato di pronto prima di consentire il raggiungimento del relativo scopo. Le modalità di richiamo e di utilizzo degli indici dinamici sono le stesse degli indici statici. Se un indice dinamico non è più necessario, può essere rimosso. L'interfaccia BackingMap prevede dei metodi per la creazione e la rimozione di indici dinamici. Di seguito viene riportata la definizione di questi metodi:

```
/**
* Crea un indice dinamico su BackingMap
*
* @param name Il nome dell'indice. Il nome non può essere nullo.
* @param isRangeIndex Indica se creare un MapRangeIndex o un MapIndex.
* Se impostato su true, l'indice sarà di tipo MapRangeIndex.
* @param attributeName Il nome dell'attributo da indicizzare.
* L'attributeName non può essere nullo.
* @param dynamicIndexCallback La richiamata che verrà eseguita su eventi di
* indice dinamico.
* La dynamicIndexCallback è facoltativa e può essere nulla.
*
* @throws IndexAlreadyDefinedException se un MapIndexPlugin con il
* nome specificato esiste già.
* @throws UnsupportedOperationException se la mappa è distribuita.
*
*/
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb)
throws IndexAlreadyDefinedException, UnsupportedOperationException;
/**
* Crea un indice dinamico su BackingMap.
*
* @param index L'implementazione dell'indice. L'indice non può essere nullo.
* @param dynamicIndexCallback La richiamata che verrà eseguita su eventi di
* indice dinamico.
* La dynamicIndexCallback è facoltativa e può essere nulla.
*
* @throws IndexAlreadyDefinedException se un MapIndexPlugin con il
* nome specificato esiste già.
* @throws UnsupportedOperationException se la mappa è distribuita.
*/
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback
dynamicIndexCallback)
throws IndexAlreadyDefinedException, UnsupportedOperationException;
/**
* rimuove un indice dinamico da BackingMap
*
* @param name Il nome dell'indice. Il nome non può essere nullo.
*
* @throws IndexUndefinedException se un MapIndexPlugin con il nome specificato
* non esiste.
* @throws OperationNotSupportedException se la mappa è distribuita.
*/
public void removeDynamicIndex(String name) throws IndexUndefinedException;
```

Il seguente frammento di codice illustra l'approccio di configurazione programmatica di creazione, utilizzo e rimozione di un indice dinamico:

```
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();
//creare un indice dopo l'inizializzazione di ObjectGrid senza DynamicIndexCallback.
bm.createDynamicIndex("CODE", true, "employeeCode", null);
try {
//Se non si utilizza DynamicIndexCallback, è necessario attendere che l'indice sia pronto.
//Il tempo di attesa dipende dalla dimensione corrente della mappa
Thread.sleep(3000);
} catch (Throwable t) {
//...
}
//Una volta che l'indice è pronto, le applicazioni possono tentare di richiamare l'istanza
//dell'interfaccia dell'indice.
//La applicazioni devono trovare un modo per garantire che l'indice sia pronto per l'uso,
//se non viene utilizzata l'interfaccia DynamicIndexCallback.
//Quanto segue dimostra come attendere che un indice sia pronto
//Il tempo di attesa totale deve tenere conto della dimensione della mappa
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 che l'indice non è pronto, ...
System.out.println("L''indice non è pronto. Continuare ad attendere.");
try {
Thread.sleep(3000);
} catch (Throwable tt) {
//...
}
} catch (Throwable t) {
//eccezione imprevista
t.printStackTrace();
}
}
if(!ready){
System.out.println("L''indice non è pronto. Gestire questa situazione.");
}
//utilizzare l'indice per eseguire delle query
//Consultare l'interfaccia MapIndex o MapRangeIndex per rilevare le operazioni supportate.
//L'attributo di oggetto su cui è stato creato l'indice è EmployeeCode.
//Si presuppone che l'attributo EmployeeCode sia di tipo Integer, che dovrebbe essere
//il tipo di dati del parametro passato nelle operazioni dell'indice.
Iterator iter = codeIndex.findLessEqual(new Integer(15));
//rimuovere l'indice dinamico quando non è più necessario
bm.removeDynamicIndex("CODE");
```

Interfaccia DynamicIndexCallback

L'interfaccia DynamicIndexCallback è progettata per applicazioni in cui si desidera ricevere delle notifiche per gli eventi di indicizzazione ready, error o destroy. È un parametro facoltativo per il metodo createDynamicIndex() di BackingMap. Con un'istanza registrata di DynamicIndexCallback, le applicazioni possono eseguire una logica aziendale in seguito a ricezione della notifica di un evento di indicizzazione. Ad esempio, l'evento ready indica che l'indice è pronto per l'uso. Quando si riceve una notifica per questo evento, un'applicazione può tentare di richiamare l'istanza di interfaccia dell'indice dell'applicazione e utilizzarla. Di seguito viene riportata la definizione dell'interfaccia DynamicIndexCallback:

```
/**
* Questa è l'interfaccia di richiamata per il processo di indicizzazione dinamica.
* Se si desidera che le applicazioni ricevano una notifica in caso di evento ready, error
* o destroy, è possibile implementare questa interfaccia di richiamata
* ed effettuare la registrazione con il processo di indicizzazione dinamica
* durante la creazione di indici dinamici.
*
*/
public interface DynamicIndexCallback {
/**
* Questo metodo di richiamata viene chiamato quando l'indice dinamico è pronto.
*
* @param indexName
* Il nome dell'indice
*
*/
public void ready(String indexName);
/**
* Richiamato quando il processo di indicizzazione dinamica individua un errore imprevisto.
*
* @param indexName Il nome dell'indice
* @param t Un oggetto Throwable che causa la situazione di errore nel
* processo di indicizzazione dinamica.
*/
public void error(String indexName, Throwable t);
/**
* Questo metodo di richiamata viene chiamato quando l'indice dinamico viene rimosso
*
* @param indexName
* Il nome dell'indice
*/
public void destroy(String indexName);
}
```

Il seguente frammento di codice illustra l'utilizzo dell'interfaccia 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);
}
}
```

Considerazioni sulle prestazioni

Sebbene uno degli obiettivi principali della funzione di indicizzazione sia migliorare le prestazioni complessive di BackingMap, è necessario considerare alcuni fattori prima di utilizzare questa funzione. Se l'indicizzazione non viene utilizzata correttamente, le prestazioni dell'applicazione potrebbero risultare compromesse.

- Il numero di transazioni di scrittura simultanee. L'elaborazione dell'indice può verificarsi ogni volta che una transazione scrive i dati in una BackingMap. Le prestazioni ne risentiranno se sono presenti molte transazioni che scrivono i dati nella mappa contemporaneamente quando un'applicazione tenta le operazioni di query dell'indice.
- La dimensione della serie di risultati restituita da un'operazione di query. Man mano che la dimensione della serie di risultati aumenta, le prestazioni della query diminuiscono. Un esperimento ha mostrato tale peggioramento delle prestazioni quando la dimensione della serie di risultati è il 15% o più di BackingMap.
- Il numero di indici creati nella stessa BackingMap. Ogni indice utilizza risorse di sistema. Man mano che il numero di indici creati in BackingMap aumenta, le prestazioni diminuiscono.

La conclusione è che la funzione di indicizzazione può migliorare notevolmente le prestazioni BackingMap in determinati ambienti. Un caso ideale per l'indicizzazione è quando la BackingMap è principalmente di lettura, la serie di risultati della query è una percentuale ridotta delle voci di BackingMap e pochi indici vengono creati nella BackingMap.

Configurazione di ObjectGrid

ObjectGrid può essere configurato per essere eseguito in un ambiente distribuito o come cache locale disponibile all'interno di una sola JVM. Un ObjectGrid locale può essere configurato in maniera programmatica o mediante un file XML di ObjectGrid. Il file XML di ObjectGrid è il file in cui sono definiti gli ObjectGrids, le BackingMaps e i relativi plug-in.

Un ObjectGrid locale può essere migrato in un ambiente distribuito. Per configurare un ObjectGrid distribuito, è necessario fornire l'XML di un cluster insieme all'XML dell'ObjectGrid. Il file XML del cluster definisce i server nella topologia dell'ObjectGrid e indica il modo in cui i dati ObjectGrid vengono partizionati e replicati sui server. In questa sezione viene descritto come configurare gli ObjectGrid locali e distribuiti.

Configurazione di un ObjectGrid locale

Per configurare un ObjectGrid locale, fare riferimento a "Configurazione di un ObjectGrid locale" a pagina 257.

ObjectGrid distribuito

Per configurare un ObjectGrid distribuito, fare riferimento a “Configurazione di ObjectGrid distribuita” a pagina 269.

Configurazione di un ObjectGrid locale

Un ObjectGrid locale può essere configurato in maniera programmatica o mediante XML. L'ObjectGridManager è il punto di accesso per entrambi questi tipi di configurazione.

Sono disponibili diversi metodi sull'ObjectGridManager che possono essere utilizzati per creare un ObjectGrid locale. Per una descrizione completa di ogni metodo, fare riferimento a “Interfaccia ObjectGridManager” a pagina 89.

Utilizzare le informazioni riportate nelle seguenti sezioni per configurare un ObjectGrid locale:

- “Configurazione di base di ObjectGrid” descrive come creare un file XML molto semplice con un ObjectGrid e un BackingMap definiti.
- In “Configurazione completa di ObjectGrid” a pagina 258 sono definiti tutti gli elementi e gli attributi del file XML e viene descritto come ottenere gli stessi risultati del file XML in maniera programmatica.
- In “Configurazione di ObjectGrid in modalità mista” a pagina 268 è riportato come utilizzare una combinazione dei metodi di configurazione XML e programmatico.

Configurazione di base di ObjectGrid

In questa sezione viene dimostrato come creare un file XML molto semplice, il file `bookstore.xml`, con un ObjectGrid e un BackingMap definiti.

Le prime righe del file sono l'intestazione richiesta per ogni file XML di ObjectGrid. Il seguente file XML definisce il *bookstore* ObjectGrid con *book* BackingMap:

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>
```

Il file XML è introdotto sull'interfaccia ObjectGridManager per creare un'istanza di ObjectGrid basata sul file. Il seguente frammento di codice convalida il file `bookstore.xml` rispetto allo schema XML e crea un ObjectGrid denominato *bookstore*. L'istanza di ObjectGrid appena creata non viene memorizzata nella cache.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore",
new URL("file:etc/test/bookstore.xml"), true, false);
```

Il seguente codice esegue la stessa attività senza utilizzare il codice XML. Utilizzare questo codice per definire in maniera programmatica un BackingMap su un ObjectGrid. Questo codice crea il registro BackingMap su bookstoreGrid ObjectGrid:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
```

Configurazione completa di ObjectGrid

Questa sezione rappresenta una guida completa per la configurazione di un ObjectGrid. Ogni elemento e attributo del file XML viene definito. Sono forniti i file XML di esempio insieme al codice che esegue la stessa attività in maniera programmatica.

Il seguente file XML, `bookstore.xml`, viene utilizzato come esempio per tutta la sezione. Gli elementi e gli attributi di questo file sono descritti dettagliatamente con questo esempio.

File *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

Numero di ricorrenze: una

Elementi secondari: l'elemento objectGrids e l'elemento backingMapPluginCollections

L'elemento objectGridConfig è l'elemento di livello superiore del file XML. È necessario che venga scritto nel documento XML come riportato nel seguente esempio. Questo elemento imposta lo spazio dei nomi del file e il percorso dello schema. Lo schema è definito nel file `objectGrid.xsd`. ObjectGrid ricerca questo file nella directory root del file JAR (Java archive) di ObjectGrid.

Elemento objectGrids

Numero di ricorrenze: una

Elemento secondario: l'elemento objectGrid

L'elemento objectGrids è un contenitore per tutti gli elementi objectGrid. Nel file `sample1.xml`, l'elemento objectGrids contiene un objectGrid denominato `bookstore`.

Elemento objectGrid

Numero di ricorrenze: una a molti

Elementi secondari: l'elemento bean e l'elemento backingMap

Utilizzare l'elemento objectGrid per definire un ObjectGrid in un file XML. Ognuno degli attributi dell'elemento objectGrid corrisponde a un metodo sull'interfaccia 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"
/>
```

Attributi:

1. Attributo **name** (richiesto): specifica il nome assegnato all'ObjectGrid. Se questo attributo risulta mancante, la convalida XML non riesce.
2. Attributo **securityEnabled** (opzionale, il valore predefinito è `false`): l'impostazione di questo attributo su `true` abilita la sicurezza per l'ObjectGrid. L'abilitazione della sicurezza a livello di ObjectGrid implica l'abilitazione delle autorizzazioni di accesso ai dati nella mappa. Per impostazione predefinita, la sicurezza è abilitata.
3. Attributo **authorizationMechanism** (opzionale, il valore predefinito è `AUTHORIZATION_MECAHNISM_JAAS`): imposta il meccanismo di autorizzazione per questo ObjectGrid. Questo attributo può essere impostato su uno dei due valori: `AUTHORIZATION_MECHANISM_JAAS` o `AUTHORIZATION_MECHANISM_CUSTOM`. Impostare l'attributo per `AUTHORIZATION_MECHANISM_CUSTOM` quando si utilizza un plug-in MapAuthorization personalizzato. Questa impostazione ha effetto solo se l'attributo securityEnabled è impostato su `true`.
4. Attributo **permissionCheckPeriod** (opzionale, il valore predefinito è `0`): specifica un valore intero in secondi che indica la frequenza con cui controllare le autorizzazioni utilizzate per consentire un accesso client. Se il valore dell'attributo è `0`, allora ogni metodo get, put, update, remove o evict richiama il meccanismo di autorizzazione, l'autorizzazione JAAS o l'autorizzazione personalizzata, per verificare che l'oggetto corrente abbia le autorizzazioni. Un valore maggiore di `0` indica il numero di secondi per cui memorizzare nella cache una serie di autorizzazioni prima di tornare al meccanismo di autorizzazione per l'aggiornamento. Questa impostazione ha effetto solo se l'attributo securityEnabled è impostato su `true`. Per ulteriori informazioni, vedere "Sicurezza di ObjectGrid" a pagina 136.
5. Attributo **txTimeout** (opzionale, il valore predefinito è `0`): il periodo di durata di ogni voce della mappa, espresso in secondi. Se una transazione non viene completata in questo periodo di tempo, la transazione viene contrassegnata per il rollback e viene restituita una eccezione `TransactionTimeoutException`. Se il valore è impostato su `0`, le transazioni non scadono mai.

Il seguente file XML di esempio, il file `bookstoreObjectGridAttr.xml`, dimostra un modo per configurare gli attributi di un objectGrid. In questo esempio, la sicurezza è abilitata, il meccanismo di autorizzazione è impostato su JAAS e il periodo di controllo delle autorizzazioni è impostato su 45 secondi.

File *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>
```

Il seguente codice dimostra l'approccio programmatico per ottenere la stessa configurazione del file `bookstoreObjectGridAttr.xml` utilizzato nel precedente esempio.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory
.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore", false);
bookstoreGrid.setSecurityEnabled();
bookstoreGrid.setAuthorizationMechanism(
SecurityConstants.AUTHORIZATION_MECHANISM_JAAS);
bookstoreGrid.setPermissionCheckPeriod(45);
```

Elemento backingMap

Numero di ricorrenze: da zero a molte

Elementi secondari: nessuno

L'elemento backingMap viene utilizzato per definire un BackingMap su un ObjectGrid. Ognuno degli attributi dell'elemento backingMap corrisponde a un metodo sull'interfaccia 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"
/>
```

Attributi:

1. Attributo **name** (richiesto): specifica il nome assegnato al BackingMap. Se questo attributo risulta mancante, la convalida XML non riesce.
2. Attributo **readOnly** (opzionale, il valore predefinito è `false`): l'impostazione di questo attributo su `true` rende un BackingMap di sola lettura. L'impostazione su `false` rende il BackingMap di lettura e scrittura. Se non viene specificato alcun valore, verrà utilizzato il valore di lettura e scrittura predefinito.
3. Attributo **pluginCollectionRef** (opzionale): specifica un riferimento a un plug-in backingMapPluginCollection. Il valore di questo attributo deve corrispondere

all'attributo id di un plug-in backingMapCollection. Se non esiste alcun id corrispondente, la convalida non riesce. Questo riferimento è progettato per poter riutilizzare i plug-in BackingMap.

4. Attributo **numberOfBuckets** (opzionale, il valore predefinito è `503`): il numero di contenitori che devono essere utilizzati dal BackingMap. L'implementazione di BackingMap utilizza una mappa hash per l'implementazione. Se in BackingMap esistono molte voci, allora un numero elevato di contenitori porta a migliori prestazioni in quanto il rischio di collisioni diminuisce man mano che il numero di contenitori aumenta. Un numero elevato di contenitori implica anche una maggiore simultaneità.
5. Attributo **preloadMode** (opzionale, il valore predefinito è `false`): imposta la modalità di precaricamento se un plug-in del Loader è impostato per questo BackingMap. Se l'attributo è impostato su `true`, il metodo Loader.preloadMap(Session, BackingMap)viene richiamato in maniera asincrona. Altrimenti viene bloccata l'esecuzione del metodo durante il caricamento dei dati in modo che la cache non sia disponibile fino a che viene completato il precaricamento. Il precaricamento si verifica durante l'inizializzazione di ObjectGrid.
6. Attributo **lockStrategy** (opzionale, il valore predefinito è `OPTIMISTIC`): imposta il LockStrategy utilizzato per il BackingMap. La strategia di blocco determina se viene utilizzato un gestore blocchi ObjectGrid interno quando una transazione accede a una voce della mappa. Questo attributo può essere impostato su uno dei seguenti tre valori: `OPTIMISTIC`, `PESSIMISTIC` o `NONE`.

   `OPTIMISTIC` viene di solito utilizzato per una mappa che non ha un plug-in del programma di caricamento, la mappa viene principalmente letta e il blocco non è fornito da un gestore permanente mediante objectGrid come cache parallela o dall'applicazione. Per la strategia di blocco ottimistica, viene ottenuto un blocco esclusivo su una voce della mappa che viene inserita, aggiornata o rimossa in fase di commit. Il blocco garantisce che le informazioni sulla versione non possano essere modificate da un'altra transazione mentre la transazione di cui viene eseguito il commit esegue un controllo ottimistico della versione.

   Il valore `PESSIMISTIC` viene di solito utilizzato per una mappa che non ha un plug-in del programma di caricamento e il blocco non è fornito da un gestore permanente mediante objectGrid come cache parallela, mediante il plug-in o dall'applicazione. La strategia di blocco pessimistica viene utilizzata quando l'approccio ottimistico non riesce a causa del fatto che le transazioni di aggiornamento collidono frequentemente sulla stessa voce della mappa. L'approccio ottimistico può riportare un errore quando la mappa viene principalmente letta o se un numero elevato di client accedono a una mappa di dimensioni minime.

   Il valore `NONE` indica che l'utilizzo del LockManager interno non è necessario in quanto il controllo della simultaneità è fornito all'esterno di ObjectGrid, o dal gestore permanente che utilizza ObjectGrid come cache parallela, o dall'applicazione o dal plug-in del programma di caricamento che utilizza i blocchi al database per controllare la simultaneità degli accessi.
7. Attributo **numberOfLockBuckets** (opzionale, il valore predefinito è `383`): imposta il numero di contenitori di blocchi utilizzati dal gestore blocchi per questo BackingMap. Quando l'attributo lockStrategy è impostato su `OPTIMISTIC` o su `PESSIMISTIC`, viene creato un gestore blocchi per il BackingMap. Tale gestore utilizza una mappa hash per tenere traccia delle voci che vengono bloccate da una o più transazioni. Se esistono molte voci, allora un numero elevato di contenitori porta a migliori prestazioni in quanto il rischio di collisioni diminuisce man mano che il numero di contenitori aumenta. Un numero

elevato di contenitori di blocco implica anche una maggiore simultaneità. Quando l'attributo lockStrategy è impostato su NONE, nessun gestore di blocchi è utilizzato da BackingMap. In questo caso, l'impostazione dell'attributo numberOfLockBuckets non è necessaria.

8. Attributo **lockTimeout** (opzionale, il valore predefinito è 15): imposta il valore di timeout dei blocchi utilizzati dal gestore blocchi per questo BackingMap. Quando l'attributo lockStrategy è impostato su OPTIMISTIC o su PESSIMISTIC, viene creato un gestore blocchi per il BackingMap. Per evitare che si verifichino delle condizioni di stallo, il gestore di blocchi ha un valore di timeout predefinito per attendere che venga concesso un blocco. Se viene superato questo valore di timeout, si verifica una eccezione LockTimeoutException. Il valore predefinito di 15 secondi è sufficiente per la maggior parte delle applicazioni, ma su sistemi con un carico di lavoro elevato, è possibile che il valore di timeout venga raggiunto anche quando non esiste alcuna condizione di stallo reale. In tal caso, questo metodo può essere utilizzato per aumentare il valore di timeout di blocco dal valore predefinito a un valore desiderato per evitare che vengano restituite false eccezioni di timeout. Quando la strategia di blocco è impostata su NONE, nessun gestore di blocchi è utilizzato da BackingMap. In questo caso, l'impostazione dell'attributo lockTimeout non è necessaria.
9. Attributo **copyMode** (opzionale, il valore predefinito è COPY_ON_READ_AND_COMMIT): l'attributo copyMode determina se un'operazione get di una voce nel BackingMap restituisce il valore corrente, una copia del valore o un proxy per il valore. L'attributo copyMode può essere impostato su uno dei seguenti quattro valori: COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE o NO_COPY.

   La modalità COPY_ON_READ_AND_COMMIT garantisce che un'applicazione non abbia alcun riferimento al valore che si trova nel BackingMap e che invece utilizzi sempre una copia del valore.

   La modalità COPY_ON_READ migliora le prestazioni rispetto alla modalità COPY_ON_READ_AND_COMMIT eliminando la copia che si verifica quando viene eseguito il commit di una transazione. Per garantire l'integrità dei dati BackingMap, l'applicazione si impegna a eliminare qualsiasi riferimento che ha alla voce una volta eseguito il commit della transazione. Questa modalità fa sì che il metodo ObjectMap.get restituisca una copia del valore invece di un riferimento al valore stesso, il che garantisce che le modifiche apportate dall'applicazione al valore non influenzino il valore di BackingMap fino a che viene eseguito il commit della transazione.

   La modalità COPY_ON_WRITE migliora le prestazioni rispetto alla modalità COPY_ON_READ_AND_COMMIT eliminando la copia che viene creata quando viene richiamato il metodo ObjectMap.get per la prima volta da una transazione per una determinata chiave. Per contro, il metodo ObjectMap.get restituisce un proxy per il valore invece che un riferimento diretto al valore. Il proxy garantisce che non venga eseguita alcuna copia del valore a meno che l'applicazione non richiami il metodo set sull'interfaccia del valore.

   La modalità NO_COPY consente a un'applicazione di garantire che il valore ottenuto utilizzando il metodo ObjectMap.get non venga modificato, ottenendo in questo modo prestazioni elevate. Se viene utilizzata questa modalità, il valore non viene copiato.
10. Attributo **valueInterfaceClassName** (opzionale): quando l'attributo copyMode è impostato su COPY_ON_WRITE, è richiesto un attributo valueInterfaceClassName. Esso viene ignorato per tutte le altre modalità. L'attributo copy on write utilizza un proxy quando vengono effettuate le chiamate al metodo ObjectMap.get. Il proxy garantisce che non venga

eseguita alcuna copia del valore a meno che l'applicazione non richiami il metodo set sulla classe specificata come attributo valueInterfaceClassName.

11. Attributo **copyKey** (opzionale, il valore predefinito è `false`): questo attributo determina se la chiave deve essere copiata quando viene creata una voce della mappa. La copia della chiave consente all'applicazione di utilizzare la stessa chiave per ogni operazione di ObjectMap. L'impostazione su `true` consente la copia quando viene creata la voce.
12. Attributo **nullValuesSupported** (opzionale, il valore predefinito è `true`): il supporto di valori null implica che un valore null può essere inserito in una mappa. Se questo attributo è impostato su `true`, i valori null sono supportati nell'ObjectMap, altrimenti non lo sono. Se sono supportati i valori null, un'operazione `get` che restituisce un valore `null` può significare sia che il valore è realmente null sia che la mappa non contiene la chiave.
13. Attributo **ttlEvictorType** (opzionale, il valore predefinito è `NONE`): l'attributo ttlEvictorType determina come viene elaborato il periodo di scadenza di una voce di BackingMap. Questo attributo può essere impostato su uno dei seguenti tre valori: `CREATION_TIME`, `LAST_ACCESS_TIME` o `NONE`.

    `NONE` indica che la voce non ha alcun periodo di scadenza e può esistere sul BackingMap fino a che l'applicazione la rimuove esplicitamente o la invalida.

    `CREATION_TIME` indica che il periodo di scadenza di una voce è stato dalla somma dell'ora di creazione della voce più il valore dell'attributo timeToLive.

    `LAST_ACCESS_TIME` indica che il periodo di scadenza di una voce è stato dalla somma dell'ora di creazione della voce più il valore dell'attributo timeToLive.
14. Attributo **timeToLive** (opzionale, il valore predefinito è `0`): il periodo di durata di ogni voce della mappa, espresso in secondi. Il valore predefinito `0` significa che la voce della mappa non viene mai eliminata, a meno che l'applicazione la rimuove esplicitamente o la invalida. Se l'attributo non è `0`, allora viene utilizzato il programma di esclusione TTL per eliminare la voce della mappa in base a questo valore.

Il seguente file XML, il file `bookstoreBackingMapAttr.xml`, dimostra una configurazione di esempio di backingMap. Questo esempio utilizza tutti gli attributi opzionali tranne l'attributo pluginCollectionRef. Per un esempio che riporta come utilizzare l'attributo pluginCollectionRef, fare riferimento a "Elemento backingMapPluginCollection" a pagina 267.

File *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>
```

Il seguente codice dimostra l'approccio programmatico per ottenere la stessa configurazione del file `bookstoreBackingMapAttr.xml` utilizzato nel precedente esempio:

```
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);
// quando si imposta la modalità di copia su COPY_ON_WRITE,
   è richiesta una 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); // imposta il valore time to live su 50 minuti
```

Elemento bean

Numero di ricorrenze (all'interno dell'elemento objectGrid): da zero a molte

Numero di ricorrenze (all'interno dell'elemento backingMapPluginCollection): zero a molte

Elemento secondario: l'elemento property

Utilizzare l'elemento bean per definire i plug-in. I plug-in possono essere collegati a ObjectGrids e BackingMaps.

I plug-in di ObjectGrid sono:

- Plug-in TransactionCallback
- Plug-in ObjectGridEventListener
- Plug-in SubjectSource
- Plug-in MapAuthorization
- Plug-in SubjectValidation

I plug-in di BackingMap sono:

- Plug-in Loader
- Plug-in ObjectTransformer
- Plug-in OptimisticCallback
- Plug-in Evictor
- Plug-in MapEventListener
- Plug-in MapIndex

Attributi dell'elemento bean

```
<bean
(1) id="TransactionCallback|ObjectGridEventListener|SubjectSource|
MapAuthorization|SubjectValidation|Loader|ObjectTransformer|
OptimisticCallback|Evictor|MapEventListener|MapIndexPlugin"
(2) className="class name"
/>
```

1. Attributo **id** (richiesto): specifica il tipo di plug-in da creare. Per un bean che è un elemento secondario dell'elemento objectGrid, i valori validi sono `TransactionCallback`, `ObjectGridEventListener`, `SubjectSource`, `MapAuthorization` e `SubjectValidation`. Per un bean che è un elemento

secondario dell'elemento backingMapPluginCollection, i valori validi sono `Loader`, `ObjectTransformer`, `OptimisticCallback`, `Evictor` e `MapEventListener`. Ognuno dei valori validi per l'attributo id rappresenta un'interfaccia.

2. Attributo **className** (richiesto): specifica il nome della classe da istanziare per creare il plug-in. La classe deve implementare l'interfaccia del tipo di plug-in.

Il seguente file `bean.xml` di esempio dimostra come utilizzare l'elemento bean per configurare i plug-in. In questo file XML, un plug-in ObjectGridEventListener viene aggiunto al bookstore di ObjectGrid. L'attributo className per questo bean è la classe com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener. Questa classe implementa l'interfaccia com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener come richiesto.

Un plug-in BackingMap è definito anche nel seguente file `bookstoreBean.xml` di esempio. Un plug-in evictor viene aggiunto al book di BackingMap. Poiché l'ID del bean è *Evictor*, l'attributo className deve specificare una classe che implementa l'interfaccia com.ibm.websphere.objectgrid.plugins.Evictor. La classe com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor implementa questa interfaccia. Il backingMap fa riferimento ai suoi plug-in utilizzando l'attributo pluginCollectionRef. Fare riferimento a "Interfaccia BackingMap" a pagina 108 per ulteriori informazioni su come aggiungere i plug-in a un BackingMap.

File *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>
```

Il seguente codice dimostra l'approccio programmatico per ottenere la stessa configurazione del file `bookstoreBean.xml` utilizzato nel precedente esempio.

```
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

Numero di ricorrenze: da zero a molte

Elemento secondario: nessuno

L'elemento property viene utilizzato per aggiungere proprietà ai plug-in. Il nome della proprietà corrisponde a un metodo set sull'attributo className del bean che contiene la proprietà.

Attributi dell'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. Attributo **name** (richiesto): specifica il nome della proprietà. Il valore assegnato a questo attributo deve corrispondere a un metodo set sulla classe fornita come attributo className sull'elemento bean. Ad esempio, se l'attributo className del bean è impostato su com.ibm.MyPlugin e il nome della proprietà fornita è `size`, allora la classe com.ibm.MyPlugin deve avere un metodo setSize.
2. Attributo **type** (richiesto): specifica il tipo di proprietà. Questo è il tipo di parametro inviato al metodo set identificato dall'attributo name. I valori validi sono le primitive Java, le relative controparti `java.lang` e `java.lang.String`. Gli attributi name e type devono corrispondere a una firma del metodo sull'attributo className del bean. Ad esempio, se il nome è `size` e il tipo è `int`, allora deve esistere un metodo `setSize(int)` sulla classe specificata come attributo className per il bean.
3. Attributo **value** (richiesto): specifica il valore della proprietà. Questo valore viene convertito nel tipo specificato dall'attributo type e viene quindi utilizzato come parametro nella chiamata al metodo set identificato dagli attributi name e type. Il valore di questo attributo non viene convalidato in alcun modo. Il programma di implementazione dei plug-in deve verificare che il valore inoltrato sia valido. Tale programma di implementazione può restituire una eccezione `IllegalArgumentException` nel metodo set se il parametro non è valido.
4. Attributo **description** (opzionale): utilizzare questo attributo per scrivere una descrizione della proprietà.

Il seguente file `bookstoreProperty.xml` dimostra come aggiungere un elemento property a un bean. In questo esempio, una proprietà denominata `maxSize` di tipo `int` viene aggiunta a un plug-in Evictor. Il plug-in Evictor com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor ha una firma del metodo che corrisponde al metodo setMaxSize(int). Un valore intero uguale a `499` viene inviato al metodo setMaxSize(int) sulla classe com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor.

File *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>
```

Il seguente codice riporta la stessa configurazione ottenuta mediante il file 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 fosse stato utilizzato il file XML,
// laproprietà aggiunta avrebbe provocato la seguente chiamata
lruEvictor.setMaxSize(449);
bookMap.setEvictor(lruEvictor);
```

Elemento backingMapPluginCollections

Numero di ricorrenze: zero a una

Elemento secondario: l'elemento backingMapPluginCollection

L'elemento backingMapPluginCollections è un contenitore per tutti gli elementi backingMapPluginCollection. Nel file bookstore.xml, l'elemento backingMapPluginCollections contiene un backingMapPluginCollection con id collection1.

Elemento backingMapPluginCollection

Numero di ricorrenze: zero a molte

Elemento secondario: l'elemento bean

L'elemento backingMapPluginCollection definisce i plug-in BackingMap. Ogni elemento backingMapPluginCollection viene identificato dal relativo attributo id. Ogni backingMap deve fare riferimento ai propri plug-in utilizzando l'attributo pluginCollectionRef sull'elemento backingMap. Se sono presenti diversi BackingMaps che hanno i plug-in configurati in maniera simile, ognuno di essi può fare riferimento allo stesso elemento backingMapPluginCollection.

Attributi dell'elemento backingMapPluginCollection

```
<backingMapPluginCollection
(1) id="id"
/>
```

1. Attributo **id** (richiesto): l'identificativo di backingMapPluginCollection. Ogni id deve essere univoco. All'id fa riferimento l'attributo pluginCollectionRef dell'elemento backingMap. Se il valore di un attributo pluginCollectionRef non

corrisponde all'id di un elemento backingMapPluginCollection, la convalida XML non riesce. Qualsiasi numero di elementi backingMap può fare riferimento a un unico elemento backingMapPluginCollection.

Il seguente file `bookstoreCollection.xml` dimostra come utilizzare l'elemento backingMapPluginCollection. In questo file, sono definiti tre elementi backingMap. Gli elementi `book` e `customer` di BackingMaps utilizzano entrambi `collection1` backingMapPluginCollection. Ognuno di questi due BackingMaps ha il proprio evictor LRUEvictor. L'elemento `employee` di BackingMap fa riferimento a `collection2` backingMapPluginCollection. Questo BackingMap ha un programma di esclusione LFU impostato come plug-in Evictor e la classe EmployeeOptimisticCallbackImpl impostata come plug-in OptimisticCallback.

File *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>
```

Il seguente codice dimostra l'approccio programmatico per ottenere la stessa configurazione del file `bookstoreCollection.xml` utilizzato nel precedente esempio.

```
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);
```

Configurazione di ObjectGrid in modalità mista

ObjectGrid può essere configurato utilizzando una combinazione della configurazione XML e della configurazione programmatica.

Per realizzare una configurazione mista, è necessario prima creare un file XML da inviare all'interfaccia ObjectGridManager. Una volta creato un ObjectGrid sulla base del file XML, l'ObjectGrid può essere modificato in maniera programmatica fino a che viene richiamato il metodo ObjectGrid.initialize(). Il metodo ObjectGrid.getSession() richiama implicitamente il metodo ObjectGrid.initialize() se non è già stato richiamato dall'applicazione.

Esempio

Di seguito è riportata una dimostrazione di come realizzare una configurazione a modalità mista. Viene utilizzato il seguente file `mixedBookstore.xml`.

File *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>
```

Il seguente frammento di codice che mostra l'XML inviato a ObjectGridManager e l'ObjectGrid appena creato vengono modificati ulteriormente.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore",
new URL("file:etc/test/document/mixedBookstore.xml"), true, false);
// a questo punto, è presente l'ObjectGrid definito nell'XML
// modificare il BackingMap creato e configurato
BackingMap bookMap = bookstoreGrid.getMap("book");
// l'XML ha impostato readOnly su true
// l'attributo readOnly è stato modificato su false
bookMap.setReadOnly(false);
// l'XML non ha impostato nullValuesSupported, pertanto
// verrà utilizzato il valore predefinito true. Il
// valore è impostato su false
bookMap.setNullValuesSupported(false);
// richiama il programma di esclusione impostato nell'XML
// e ne imposta la proprietà maxSize
LFUEvictor lfuEvictor = (LFUEvictor) bookMap.getEvictor();
lfuEvictor.setMaxSize(443);
bookstoreGrid.initialize();
// un'ulteriore configurazione non è consentita
// per questo ObjectGrid in seguito alla chiamata a initialize
```

Configurazione di ObjectGrid distribuita

Per creare un ObjectGrid distribuito, creare un file XML cluster e associarlo con il file XML ObjectGrid.

Con i file XML cluster e XML ObjectGrid, è possibile avviare un server ObjectGrid.

Prima di creare un file XML cluster, creare un file XML ObjectGrid come per un ObjectGrid locale. Per i dettagli sulla costruzione di un file XML ObjectGrid, vedere “Configurazione di un ObjectGrid locale” a pagina 257. Il seguente file `university.xml` viene utilizzato come l'XML ObjectGrid per gli esempi in “Configurazione cluster” a pagina 271.

university.xml file

```
<?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>
<nome objectGrid="accademico">
<nome backingMap="facoltà" />
<nome backingMap="studente" />
<nome backingMap="corso" />
</objectGrid>
<nome objectGrid="atletica">
<nome backingMap="atleta" />
<nome backingMap="attrezzatura" />
</objectGrid>
</objectGrids>
</objectGridConfig>
```

Il seguente è il file XML cluster `universityCluster.xml` che può essere utilizzato con il file `university.xml` per avviare un server ObjectGrid. Il file `universityCluster.xml` è un file XML cluster di base con tutti gli attributi XML opzionali eliminati.

file *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">
<nome cluster="universityCluster">
<nome serverDefinition="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
</cluster>
<objectgridBinding ref="academics">
<nome mapSet="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="facoltà" />
<map ref="studente" />
<map ref="corso" />
</mapSet>
</objectgridBinding>
<objectgridBinding ref="atletica">
<nome mapSet="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="atleta" />
<map ref="attrezzatura" />
</mapSet>
</objectgridBinding>
<nome partitionSet="partitionSet1">
<nome partizione="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<nome replicationGroup="replicationGroup1">
<replicationGroupMember serverRef="server1" priorità="1" />
</replicationGroup>
</clusterConfig>
```

L'utilizzo di esempio dei molti elementi ed attributi XML opzionali si trova nella sezione “Configurazione cluster” a pagina 271.sn

Configurazione cluster

Ogni elemento ed attributo del cluster XML viene descritto in questa sezione. Gli esempi forniti mostrano inoltre come utilizzare il cluster XML con ObjectGrid XML per ottenere una configurazione. Il file `university.xml` viene utilizzato come ObjectGrid XML per questi esempi.

Elemento clusterConfig

Numero di ricorrenze: una

Elementi child: elementi cluster, objectgridBinding, partitionSet e replicationGroup

L'elemento clusterConfig è l'elemento di livello superiore del file XML cluster. Deve stare all'inizio del come dimostrato nel file `universityCluster.xml`. Questo elemento imposta lo spazio dei nomi del file e il percorso dello schema. Lo schema viene definito nel file `objectGridCluster.xsd`.

elemento cluster

Numero di ricorrenze: una

Elementi secondari: elementi serverDefinition, authenticator e adminAuthorization

L'elemento cluster viene utilizzato per definire un clusterObjectGrid. Ognuno dei server nel cluster viene definito all'interno dell'elemento cluster. L'elemento cluster viene utilizzato inoltre per definire la sicurezza e gli attributi di rete.

```
<cluster
(1) nome="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"
/>
```

Attributi:

1. **nome** attributo (richiesto): questo è il nome che è assegnato al cluster. Se questo attributo risulta mancante, la convalida XML non riesce.

2. **securityEnabled** attributo (opzionale, predefinito in **falso**): Abilita la sicurezza per il cluster quando impostato su vero. Se è impostato su falso, la sicurezza per tutto il cluster viene disabilitata. Per ulteriori informazioni, vedere “Sicurezza di ObjectGrid” a pagina 136.
3. attributo**statisticsEnabled** (opzionale, predefinito in **falso**): Abilita le statistiche per il cluster quando impostato su vero. Quando le statistiche sono abilitate, l’attributo satisticsSpec viene utilizzato per impostare la specifica delle statistiche.
4. attributo **statisticsSpec** (opzionale): specifica la stringa che viene utilizzata per impostare la specifica delle statistiche. Questa stringa determina quali statistiche vengono raccolte.
5. attributo **singleSignOnEnabled** (opzionale, predefinito in falso): l’impostazione dell’attributo singleSignOnEnabled in vero consente a un client di collegarsi a un server dopo che è stato autenticato con uno dei server. Quando questo attributo è impostato in falso, è necessario autenticare un client con ogni server prima che sia consentito collegarsi.
6. attributo **loginSessionExpirationTime** (opzionale): la quantità di tempo in secondi prima della scadenza della sessione di accesso. Se la sessione di accesso scade, il client deve essere riautenticato.
7. attributo **adminAuthorizationEnabled** opzionale, il valore predefinito è falso): Questo valore viene utilizzato per abilitare l’autorizzazione amministrativa. Se il valore è vero, tutte le attività amministrative devono essere autorizzate. Il meccanismo di autorizzazione utilizzato è specificato dal valore dell’attributo adminAuthorizationMechanism.
8. attributo **adminAuthorizationMechanism** (opzionale, il valore predefinito è AUTHORIZATION_MECHANISM_JAAS): Questo attributo indica quale meccanismo di autorizzazione viene utilizzato. ObjectGrid supporta due meccanismi di autorizzazione: Java Authentication e Authorization Service (JAAS) e personalizzato. Il meccanismo di autorizzazione JAAS utilizza l’approccio basato sulla politica JAAS standard. Per specificare JAAS come il meccanismo di autorizzazione, impostare il valore in AUTHORIZATION_MECHANISM_JAAS. Il meccanismo di autorizzazione personalizzato utilizza un’implementazione AdminAuthorization con utente collegato. Per specificare un meccanismo di autorizzazione personalizzato, impostare il valore in AUTHORIZATION_MECHANISM_CUSTOM. Per maggiori informazioni sulla modalità di utilizzo di questi due meccanismi, fare riferimento a “Sicurezza di ObjectGrid” a pagina 136.
9. attributo **clientMaxRetries** (opzionale, valore predefinito in 4): il numero massimo di volte in cui una richiesta per un server può essere automaticamente provata quando il servizio non è disponibile.
10. attributo **clientMaxForwards** (opzionale, valore predefinito in 5): Il numero massimo di volte in cui una richiesta non riuscita viene inoltrata a un altro server.
11. attributo **clientStartupRetries** (opzionale, valore predefinito in 8): il numero massimo di volte in cui una richiesta viene automaticamente riprovata mentre si attende che l’avvio del server sia completo. Dal momento in cui il gestore HA (high availability) richiede un quantitativo di tempo significativo per iniziare, impostare questo numero come sufficientemente alto. Se il numero non è sufficiente, le richieste client che sono inoltrate prima che il server sia completamente avviato falliscono.
12. attributo **clientRetryInterval** (opzionale, valore predefinito 10): l’intervallo di tempo, in secondi, tra i tentativi di un client. Questo viene utilizzato per gli attributi clientMaxRetries e clientStartupRetries.

13. attributo **tcpConnectionTimeout** (opzionale, valore predefinito 180): l'attributo tcpConnectionTimeout è il timeout di connessione socket. Il valore è in secondi.
14. attributo **tcpMinConnections** (opzionale, valore predefinito 2): il numero minimo di connessioni per il pool di connessione.
15. attributo **tcpMaxConnections** (opzionale, valore predefinito in 20): il numero massimo di connessioni per il pool di connessione.
16. attributo **tcpInactivityTimeout** (opzionale, il valore predefinito è infinito): Il numero di secondi di inattività su una connessione che deve passare prima della rimozione della connessione dal pool di connessione.
17. attributo **tcpMaxWaitTime** (opzionale, il valore predefinito è 120): il numero massimo di secondi in cui un sistema attende una connessione disponibile quando tutte le connessioni sono in uso e il numero di connessioni ha raggiunto il valore di attributo tcpMaxConnections.
18. attributo **peerHeartbeatInterval** (opzionale, il valore predefinito è 120): l'attributo peerHeartbeatInterval è l'intervallo heartbeat utilizzato dal gestore HA (high availability). Il valore è in secondi.
19. attributo **peerTransportBufferSize** opzionale, il valore predefinito è 10): gli attributi peerTransportBufferSize rappresentano la dimensione del buffer del messaggio di trasporto utilizzata dal gestore HA (high availability). Questo valore è specificato in megabyte.
20. attributo **threadPoolMinSize** (opzionale, il valore predefinito è 6): specifica la dimensione minima del pool di thread del gestore HA (high availability).
21. attributo **threadPoolMaxSize** (opzionale, il valore predefinito è 20): specifica la dimensione massima del pool di thread del gestore HA (high availability).
22. attributo **threadPoolInactivityTimeout** (opzionale, il valore predefinito è 6000): Specifica il timeout di inattività del thread per il pool di thread del gestore HA (high availability). Il valore di timeout è in secondi.
23. attributo **managementTimeout** (opzionale, il valore predefinito è 30): diverse funzioni MBean di ObjectGrid inviano messaggi ai server nel cluster per raccogliere informazioni o per effettuare operazioni sui server. Il valore managementTimeout definisce per quanto tempo il client prova a ricevere un messaggio dal server. Se esistono problemi di comunicazione tra client e server o se il server è occupato, il client riprova per il tempo specificato dal valore managementTimeout. Il valore managementTimeout è specificato in secondi.
24. attributo **threadsPerClientConnect** (opzionale, il valore predefinito è 5): il numero di thread che sono creati per ClientClusterContext. Ogni chiamata al metodo di connessione sull'interfaccia ObjectGridManager produce un nuovo ClientClusterContext.

Il seguente file `universityClusterAttr.xml` è una configurazione di esempio che utilizza i vari attributi opzionali sull'elemento cluster. In questo esempio, la sicurezza è disabilitata. I vari client, tcp, peer, e attributi relativi al thread vengono inoltre modificati. `universityClusterAttr.xml` non è una raccomandazione di quali valori assegnare agli attributi. È un esempio su come impostare i valori dell'attributo.

file *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" />
<nome serverDefinition="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectGridBinding ref="accademico">
<nome mapSet="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="facoltà" />
<map ref="studente" />
<map ref="corso" />
</mapSet>
</objectGridBinding>
<objectGridBinding ref="atletica">
<nome mapSet="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="atleta" />
<map ref="attrezzatura" />
</mapSet>
</objectGridBinding>
<nome partitionSet="partitionSet1">
<nome partizione="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<nome replicationGroup="replicationGroup1">
<replicationGroupMember serverRef="server1" priorità="1" />
</replicationGroup>
</clusterConfig>
```

elemento serverDefinition

Numero di ricorrenze: una a molte

Elementi secondari: nessuno

L'elemento serverDefinition viene utilizzato per definire un server ObjectGrid. Ogni server funziona su JVM (Java virtual machine) e richiede due porte.

Attributi:

```
<serverDefinition
(1) nome="serverName"
(2) host="hostname"
(3) clientAccessPort="portNumber"
(4) peerAccessPort="portNumber"
(5) traceSpec="traceSpecification"
(6) systemStreamToFileEnabled="true|false"
(7) workingDirectory="logsDirectory"
/>
```

1. **nome** attributo (richiesto): questo è il nome che è assegnato al server. Se questo attributo risulta mancante, la convalida XML non riesce.
2. **host** attributo (richiesto): il nome host della macchina in cui il server JVM funziona. Ogni macchina può ospitare diversi server ObjectGrid. Se questo attributo risulta mancante, la convalida XML non riesce.
3. **clientAccessPort** attributo (richiesto): la porta sul server che viene utilizzata per le connessioni client. Se questo attributo risulta mancante, la convalida XML non riesce.

4. **peerAccessPort** attributo (richiesto): la porta sul server che viene utilizzata per la comunicazione tra i server ObjectGrid. Se questo attributo risulta mancante, la convalida XML non riesce.
5. **traceSpec** attributo (opzionale, valore predefinito *=all=disabled): l'impostazione dell'attributo traceSpec abilita la traccia per il server utilizzando la stringa specificata.
6. attributo **systemStreamToFileEnabled** (opzionale, il valore predefinito è **true**): Se questo attributo è impostato su true, System.out, System.err e l'output di traccia si preparano ad andare in un file. Quando questo attributo è impostato in falso, System.out va al flusso stdout e System.err al flusso stderr. Se la traccia è abilitata, l'output di traccia passa in un file indipendentemente dal valore dell'attributo systemStreamToFileEnabled.
7. attributo **workingDirectory** (opzionale): l'attributo workingDirectory specifica dove vengono scritti i file di registro. Se un attributo workingDirectory non è specificato, i registri vengono scritti nella directory corrente.

Il file `universityClusterServerAttr.xml` dimostra l'utilizzo degli attributi serverDefinition. In questo file XML, il server server1 viene configurato per essere eseguito sull'host lion.ibm.com. La porta 12501 viene utilizzata per l'accesso del client al server e la porta 12502 viene utilizzata per le comunicazioni tra server. Dato che l'attributo systemStreamToFileEnabled è impostato su true, System.out, System.err e traccia sono emessi in un file all'interno della directory specificata con l'attributo workingDirectory. In questo esempio, i file sono nella directory `/objectgrid/`. Dato che l'attributo traceSpec è impostato in ″ObjectGrid=all=enabled″, tutta la traccia relativa a ObjectGrid viene catturata ed emessa in un file.

file *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">
<nome cluster="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" systemStreamToFileEnabled="true"
workingDirectory="/objectgrid/" traceSpec="ObjectGrid=all=enabled" />
<nome serverDefinition="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectGridBinding ref="accademico">
<nome mapSet="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="facoltà" />
<map ref="studente" />
<map ref="corso" />
</mapSet>
</objectGridBinding>
<objectGridBinding ref="atletica">
<nome mapSet="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="atleta" />
<map ref="attrezzatura" />
</mapSet>
</objectGridBinding>
<nome partitionSet="partitionSet1">
<nome partizione="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<nome replicationGroup="replicationGroup1">
<replicationGroupMember serverRef="server1" priorità="1" />
</replicationGroup>
</clusterConfig>
```

elemento objectgridBinding

Numero di ricorrenze: una a molte

Elemento secondario: elemento mapSet

L'elemento objectgridBinding viene utilizzato per collegare gli elementi objectGrid nell'XML ObjectGrid alla topologia definita nel cluster XML. Il valore assegnato all'attributo ref deve corrispondere all'attributo del nome di uno degli elementi objectGrid nell'XML ObjectGrid. Un elemento objectGrid dall'XML ObjectGrid può essere il riferimento in un solo objectgridBinding nel cluster XML.

Attributi

```
<objectgridBinding
(1) ref="objectGridReference"
(2) minThreadPoolSize="minSize"
(3) maxThreadPoolSize="maxSize"
/>
```

1. attributo **ref** (richiesto): l'attributo ref viene utilizzato per fare riferimento a un elemento objectGrid che è definito nel file ObjectGrid XML. Ogni elemento objectgridBinding deve fare riferimento a uno degli elementi objectGrid dall'XML ObjectGrid. L'attributo ref deve corrispondere all'attributo del nome di uno degli elementi objectGrid nell'XML ObjectGrid.
2. attributo **minThreadPoolSize** (opzionale, il valore predefinito è 3): L'attributo minThreadPoolSize è il numero minimo di thread consentito nel pool di thread per ogni membro del gruppo di replica. Il numero di thread è controllato da un gestore del pool di thread, ma il numero non è consentito per scendere al di sotto del valore minThreadPoolSize. In generale, più thread consentono al client di ricevere una risposta più rapida dal server. Più thread producono una maggiore contesa. Tuttavia, le macchine più veloci sono in grado di gestire ulteriori thread contemporaneamente in modo efficace.
3. attributo **maxThreadPoolSize** (opzionale, il valore predefinito è 10): l'attributo maxThreadPoolSize è il numero massimo di thread consentito nel pool di thread per ciascun membro del gruppo di replica. Il numero di thread è controllato da un gestore del pool di thread, ma il numero non è consentito per superare il valore maxThreadPoolSize. In generale, altri thread consentono al client di ricevere una risposta più rapida dal server. Maggiori thread possono produrre una disputa. Tuttavia, le macchine più veloci sono in grado di gestire ulteriori thread contemporaneamente in modo efficace.

Il file `universityClusterOGBinding.xml` dimostra come utilizzare l'elemento objectgridBinding e gli attributi. In questo esempio, un elemento objectgridBinding è definito. L'elemento objectgridBinding fa riferimento a ″accademico″ definito nel file `university.xml`. Notare che anche se l'objectGrid ″atletica″ è nel file `university.xml`, nessun elemento objectgridBinding fa riferimento all'ObjectGrid atletica nel file `universityClusterOGBinding.xml`. L'ObjectGrid ″atletica″ non è cluster perché non è incluso nel file `universityClusterOGBinding.xml`. Solo l'ObjectGrid ″accademico″ è creato e inserito nel cluster in questo caso perché è nel file `university.xml` e si fa riferimento nel file `universityClusterOGBinding.xml`.

Gli attributi minThreadPoolSize e maxThreadPoolSize sono allo stesso modo impostati in questo esempio. Il valore minThreadPoolSize è impostato su 2 e il valore maxThreadPoolSize è impostato su 11. Il gestore del pool di thread su ciascun membro del gruppo di replica conserva il numero di thread all'interno di questi limiti per tutte le mappe in questo ObjectGrid.

file *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">
<nome cluster="universityCluster">
<nome serverDefinition="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
</cluster>
<objectgridBinding ref="academics" minThreadPoolSize="2" maxThreadPoolSize="11">
<nome mapSet="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="facoltà" />
<map ref="studente" />
<map ref="corso" />
</mapSet>
</objectgridBinding>
<nome partitionSet="partitionSet1">
<nome partizione="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<nome replicationGroup="replicationGroup1">
<replicationGroupMember serverRef="server1" priorità="1" />
</replicationGroup>
</clusterConfig>
```

elemento mapSet

Numero di ricorrenze: una a molte

Elemento secondario: elemento mappa

L'elemento mapSet viene utilizzato per raggruppare le mappe insieme. Le mappe all'interno di un mapSet sono suddivise allo stesso modo. In un ObjectGrid distribuito, ogni mappa deve appartenere a un solo mapSet.

Attributi

```
<mapSet
(1) nome="mapSetName"
(2) partitionSetRef="partitionSetReference"
(3) synchronousReplication="true|false"
(4) replicaReadEnabled="true|false"
(5) replicaDeliveryRate="deliveryRate"
(6) compression="true|false"
/>
```

1. attributo **nome** (richiesto): Specifica il nome assegnato al mapSet.
2. attributo **partitionSetRef** (richiesto): Ogni mapSet deve essere associato a un partitionSet attraverso l'attributo partitionSetRef. Il valore partitionSetRef deve corrispondere al valore dell'attributo del nome di uno degli elementi partitionSet. Utilizzando l'attributo partitionSetRef e il partitionSet corrispondente, le mappe nel mapSet sono con partizioni.
3. attributo **synchronousReplication** (opzionale, il valore predefinito è **false**): Quando questo attributo è impostato in true, la replica tra i membri del gruppo di replica si verifica sincronicamente. Quando impostato in false, la replica si verifica asincronicamente.
4. attributo **replicaReadEnabled** (opzionale, il valore predefinito è **false**): Se il valore synchronousReplication è impostato in false e il valore replicaReadEnabled è true, ai client è consentito leggere i dati dalle repliche. Un maggiore sforzo viene effettuato per distribuire le richieste di lettura tra quella

primaria e le repliche. Se l'attributo synchronousReplication è impostato in true, l'attributo replicaReadEnabled viene ignorato.

5. **replicaDeliveryRate** (opzionale, valore predefinito **1000**): Il valore replicaDeliveryRate rappresenta il numero massimo di record per LogSequence che sono forniti a ogni replica.
6. attributo **compressReplicationEnabled** (opzionale, il valore predefinito è **true**): Quando compressReplicationEnabled è impostato in true, i messaggi di replica vengono compressi.

Il file `universityClusterMapSet.xml` è un po' più complesso rispetto ai precedenti esempi di file XML. In questo file l'ObjectGrid accademico viene diviso in due gruppi di mappe. Il gruppo di mappa academicsMapSet1 contiene le mappe di facoltà e di corso. Queste due mappe sono divise in partizioni in base a partitionSet1 partitionSet. Le impostazioni di replica per queste mappe sono inoltre le stesse perché sono nello stesso mapSet.

L'objectgridBinding accademico contiene inoltre il mapSet academicsMapSet2. Questo mapSet contiene solo la mappa dello studente. La mappa dello studente è divisa in partizioni differentemente dalle mappe nel mapSet academicsMapSet1. La mappa dello studente viene suddivisa in base al partitionSet studentPSet. Dato che academicsMapSet2 non ha esplicitamente dichiarato i valori per gli attributi relativi alla replica, compresi gli attributi synchronousReplication, replicaReadEnabled, replicaDeliveryRate e compressReplicationEnabled, sono assegnati i valori predefiniti. Esiste un altro modo in cui il comportamento dei due mapSets all'interno dell'objectgridBinding accademico differisce.

L'objectgridBinding atleticacontiene il mapSet athleticsMapSet. Come il mapSet academicsMapSet1 in objectgridBinding accademico, viene suddiviso in base a partitionSet1 partitionSet. La replica relativa agli attributi per questo mapSet è impostata nei valori predefiniti dal momento che non sono esplicitamente dichiarati.

file *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">
<nome cluster="universityCluster">
<nome serverDefinition="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
<nome serverDefinition="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectgridBinding ref="accademico"
<nome mapSet="academicsMapSet1" partitionSetRef="partitionSet1"
synchronousReplication="true" replicaReadEnabled="true"
replicaDeliveryRate="1500" compressReplicationEnabled="true">
<map ref="facoltà" />
<map ref="corso" />
</mapSet>
<nome mapSet="academicsMapSet2" partitionSetRef="studentPSet">
<mapRef="studente" />
</mapSet>
</objectGridBinding>
<objectgridBinding ref="atletica">
<nome mapSet="athleticsMapSet" partitionSetREf="partitionSet1">
<map ref="atleta" />
<map ref="attrezzatura" />
</mapSet>
</objectgridBinding>
```

```
<nome partitionSet="partitionSet1">
<nome partizione="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<nome partitionSet="studentPSet">
<nome partizione="studentPartition1" replicationGroupRef="replicationGroup1" />
<nome partizione="studentPartition2" replicationGroupRef="replicationGroup2" />
</partitionSet>
<nome replicationGroup="replicationGroup1">
<replicationGroupMember serverRef="server1" priorità="1" />
</replicationGroup>
<nome replicationGroup="replicationGroup2" minReplicas="1" maxReplicas="2">
<replicationGroupMember serverRef="server1" priorità="2" />
<replicationGroupMember serverRef="server2" priorità="1" />
</replicationGroup>
</clusterConfig>
```

elemento mappa

Numero di ricorrenze: una a molte

Elementi secondari: nessuno

Ogni mappa in un riferimento mapSet degli elementi backingMap definiti nel file XML ObjectGrid. Quando si definisce un ObjectGrid distribuito, viene fatto riferimento a ogni backingMap nell'elemento objectGrid dell'XML ObjectGrid da una mappa nell'XML del cluster. Ogni mappa in un ObjectGrid distribuito deve appartenere ad un unico e solo mapSet.

Attributi

```
<mappa
(1) ref="backingMapReference"
/>
```

1. attributo **ref** (richiesto): riferimento A in un backingMap nell'XML ObjectGrid. Ogni mappa in un mapSet deve far riferimento a un backingMap dal file XML ObjectGrid. Il valore assegnato a ref deve corrispondere all'attributo del nome di uno degli elementi backingMap nell'XML ObjectGrid.

Fare riferimento al file `universityClusterMapSet.xml` per un utilizzo di esempio dell'elemento della mappa. A ogni backingMap dagli objectGrid accademici in `university.xml` si fa riferimento con una mappa in un unico e solo mapSet `universityClusterMapSet.xml`. Lo stesso è vero per l'objectGrid di atletica. Un'eccezione`ObjectGridException` si produce se un objectgridBinding fa riferimento a un objectGrid dall'XML ObjectGrid, ma non include tutte le mappe in un mapSet.

elemento partitionSet

Numero di ricorrenze: una a molte

partizione **Elementi secondari:**

L'elemento partitionSet viene utilizzato per definire le partizioni per un mapSet. Ogni mappa nel mapSet è divisa in partizioni in tutte le partizioni di un partitionSet. Un mapSet è associato a un partitionSet con l'attributo partitionSetRef sull'elemento mapSet. Quando una sola partizione viene definita in un partitionSet, i dati contenuti nelle mappe di un mapSet associato non sono divisi in partizioni.

Attributi

```
<partition-set
(1) nome="partitionSetName"
/>
```

1. attributo **nome** (richiesto): Questo attributo viene utilizzato per assegnare un nome a un partitionSet. Il nome del partitionSet a cui si fa riferimento dall'attributo partitionSetRef del mapSet.

Fare riferimento al file`universityClusterMapSet.xml` per un utilizzo di esempio del partitionSet. In `universityClusterMapSet.xml`, sono definiti due partitionSet: partitionSet1 e studentPSet. Il partitionSet partitionSet1 presenta una sola partizione definita. Dal momento che solo una partizione viene definita, qualsiasi mapSet che utilizza il partitionSet partitionSet1 ha i dati divisi in partizioni. Esistono due mapSet nel file `universityClusterMapSet.xml` che sono divisi in partizioni in base al partitionSet partitionSet1. Attraverso il partitionSetRef sull'elemento mapSet, i MapSet academicsMapSet1 e athleticsMapSet sono collegati al partitionSet partitionSet1.

Il partitionSet studentPSet contiene due partizioni. Qualsiasi mapSet che utilizza questo partitionSet ha i dati della mappa divisi in due partizioni. Nel file `universityClusterMapSet.xml`, il mapSet academicsMapSet2 utilizza il partitionSet studentPSet.

elemento di partizione

Numero di ricorrenze: una a molte

Elementi secondari: nessuno

L'elemento di partizione viene utilizzato per definire le partizioni in un partitionSet. Le partizioni sono utilizzate per dividere i dati nelle mappe di un mapSet.

Attributi

```
<partizione
(1) nome="partitionName"
(2) replicationGroupRef="replicationGroupReference"
/>
```

1. attributo **nome** (richiesto): L'attributo del nome viene utilizzato per assegnare un nome a una partizione. Un nome di partizione deve essere unico nel partitionSet.
2. attributo **replicationGroupRef** (richiesto): l'attributo replicationGroupRef viene utilizzato per associare un replicationGroup con una partizione. replicationGroupRef deve corrispondere all'attributo del nome di uno degli elementi replicationGroup.

Fare riferimento al file `universityClusterMapSet.xml` per un utilizzo di esempio dell'elemento della partizione. Nel file `universityClusterMapSet.xml`, sono definite diverse partizioni. Il partitionSet partitionSet1 ha una sola partizione denominata partition1. Il partitionSet studentPSet contiene due partizioni: studentPartition1 e studentPartition2.

Ogni partizione è associata a un replicationGroup attraverso l'attributo replicationGroupRef. Nel file `universityClusterMapSet.xml`, la partizione studentPartition1 viene replicata attraverso il replicationGroup replicationGroup1. La partizione studentPartition2 viene replicata attraverso il replicationGroup replicationGroup2.

elemento replicationGroup

Numero di ricorrenze: una a molte

Elemento secondario: elemento replicationGroupMember

Un replicationGroup viene utilizzato per definire come una mappa o le partizioni sono replicate. Le partizioni di una mappa sono replicate tra i membri del gruppo di replica all'interno di un replicationGroup.

Attributi

```
<replicationGroup
(1) nome="replicationGroupName"
(2) minReplicas="minNumberOfReplicas"
(3) maxReplicas="maxNumberOfReplicas"
```

1. attributo **nome** richiesto): L'attributo nome viene utilizzato per assegnare un nome a un replicationGroup.
2. attributo **minReplicas** (opzionale, valore predefinito **0** un solo replicationGroupMember è nel replicationGroup; il valore predefinito è **1** più di un replicationGroupMember nel replicationGroup): L'attributo minReplicas viene utilizzato per indicare quanti replicationGroupMembers devono essere disponibili prima di consentire l'accesso in scrittura ai dati di mappa in questo replicationGroup. Se il numero di repliche disponibili scende al di sotto del numero di minReplicas specificato, l'accesso alle mappe consentito è di sola lettura. Se minReplicas è impostato su 0, l'accesso in scrittura è ancora consentito sul primario anche se tutte le repliche non sono disponibili.

   Per attivare una replica, devono essere disponibili almeno due membri del gruppo di replica,e l'attributo minReplicas deve essere almeno 1. È importante sapere come un replicationGroup si comporta durante la fase di ″bringup″ di un cluster ObjectGrid. Se si desidera rendere disponibili i dati di mappa dopo l'avvio di un solo server, definire un replicationGroup con un solo replicationGroupMember. In un replicationGroup con un solo replicationGroupMember, i dati non vengono replicati.

   Ecco alcune regole per l'impostazione del valore minReplicas.

   ```
   minReplicas >= 0
   minReplicas <= maxReplicas
   minReplicas <= N. di membri nel replicationGroup -1
   ```

3. attributo **maxReplicas** (opzionale, valore predefinito **0** se solo un replicationGroupMember è nel replicationGroup; valore predefinito **1** più di un replicationGroupMember è nel replicationGroup): L'attributo maxReplicas rappresenta il numero massimo di repliche che sono attivate nel replicationGroup. In un replicationGroup, la replica si verifica tra il numero di maxReplica specificato se sono disponibili molti membri. Se maxReplicas è inferiore al numero dei membri del gruppo di replica nel gruppo, i membri extra sono standby; vale a dire che, sono inattivi fino a che una delle repliche non diventa indisponibile.

   Ecco alcune regole per l'impostazione del valore maxReplicas.

   ```
   maxReplicas >= 0
   maxReplicas >= minReplicas
   ```

Fare riferimento al file `universityClusterMapSet.xml` per un utilizzo di esempio dell'elemento di partizione. In `universityClusterMapSet.xml`l, sono definiti due replicationGroup: replicationGroup1 e replicationGroup2. Il replicationGroup replicationGroup1 contiene un solo replicationGroupMember. Le partizioni che sono

collegate al replicationGroup non sono replicate perché più di un replicationGroupMember è richiesto per la replica.

Il replicationGroup replicationGroup2 contiene due replicationGroupMember. La partizione studentPartition2 del partitionSet studentPSet utilizza questo replicationGroup. Quindi, la partizione studentPartition2 viene replicata attraverso due replicationGroupMember. Il replicationGroup replicationGroup2 presenta inoltre il proprio gruppo di attributi minReplicas e maxReplicas. Dato che minReplicas è impostato su 1, i dati della mappa che si trovano in questo replicationGroup sono di sola lettura fino a che il primario e almeno una replica diventano disponibili. Il valoremaxReplicas di 1 indica che il primario di questo replicationGroup replica i propri dati al massimo di una replica. Nel caso del replicationGroup replicationGroup2, non è possibile superare una replica perché il gruppo contiene solo due membri. Un membro è il primario e l'altro è una replica.

elemento replicationGroupMember

Numero di ricorrenze: una a molte

Elementi secondari: nessuno

Un elemento replicationGroupMember viene utilizzato per fare riferimento a una definizione di server. Ogni elemento replicationGroupMember presenta inoltre una priorità associata. La priorità viene utilizzata per determinare quale replicationGroupMember è il server primario e quali membri sono repliche.

Attributi

```
<replicationGroupMember
(1) serverRef="serverDefinitionReference"
(2) priorità="priority"
/>
```

1. attributo **serverRef** (richiesto): L'attributo serverRef viene utilizzato per associare una definizione di server con un elemento replicationGroupMember. L'attributo serverRef associa ogni replicationGroupMember con un server specifico.
2. elemento **priorità** (richiesto): L'attributo di priorità viene utilizzato per determinare quale dei membri del gruppo di replica è il primario. Il campo dei valori di priorità da `1` al numero dei membri del gruppo di replica, con `1` che rappresenta la priorità più alta. ObjectGrid compie tutti gli sforzi per rispettare la priorità per ogni membro del gruppo di replica. L'elemento replicationGroupMember con una priorità `1` è il primario salvo se le circostanze lo impediscono. Se tutti i server e il replicationGroupMembers diventano disponibili quasi contemporaneamente, le impostazioni di priorità vengono rispettate. Tuttavia, se un replicationGroupMember con una priorità di `2` è disponibile prima di qualsiasi altro repliationGroupMember, questo diventa il primario.

   Se il primario si avvia correttamente e si interrompe dopo un periodo di tempo, è necessario selezionare un nuovo primario. L'elemento replicationGroupMember con la priorità più alta diventerà probabilmente il nuovo primario. Tuttavia, una diversa replica potrebbe essere selezionata come la nuova primaria se la replica con la priorità più alta viene determinata essere nella replica.

Fare riferimento al file `universityClusterMapSet.xml` per un utilizzo di esempio dell'elemento di partizione. In `universityClusterMapSet.xml`, il replicationGroup replicationGroup1 ha un solo replicationGroupMember. A causa della definizione,

questo replicationGroup ha un solo primario. Non ci sono altre repliche in questo gruppo. Il replicationGroupMember definito è attivo sul server server1 quando il valore serverRef si definisce.

Il replicationGroup replicationGroup2 presenta più di un replicationGroupMember. Il primo replicationGroupMember elencato è attivato sul server server1. Questo membro ha una priorità di 2. Il secondo replicationGroupMember elencato è attivato sul server sever2. Dato che il secondo membro ha una priorità di 1, è il primario di questo replicationGroup se i membri del gruppo diventano disponibili quasi contemporaneamente. Il primo replicationGroupMember che è elencato serve come replica perché ha una priorità di 2.

È inoltre importante comprendere come il valore minReplicas influenza il replicationGroup replicationGroup2. Considerare la situazione in cui entrambi i server server1 e server2 sono in esecuzione. In questo caso, entrambi i replicationGroupMember sono disponibili. Quindi, i valori minReplicas e maxReplicas sono entrambi soddisfatti e i dati sono replicati tra il primario e la replica di questo gruppo. Se il server1 non è più disponibile, uno dei replicationGroupMember non è più disponibile. In questa situazione, i dati nel replicationGroup replicationGroup2 diventano di sola lettura perché il valore minReplicas non è più soddisfatto.

elemento authenticator

Numero di ricorrenze: zero a una

Elemento secondario: elemento della proprietà

Un elemento authenticator viene utilizzato per autenticare i client ai server ObjectGrid nel cluster. La classe specificata dall'attributo className deve implementare l'interfaccia com.ibm.websphere.objectgrid.security.plugins.Authenticator. L'authenticator può utilizzare le proprietà per richiamare i metodi sulla classe specificata dall'attributo className. Fare riferimento a "Elemento proprietà" a pagina 285 per maggiori informazioni sull'utilizzo delle proprietà.

Attributi

```
<authenticator
(1) className="authenticatorClassName"
/>
```

1. attributo **className** (richiesto): L'attributo className viene utilizzato per specificare una classe che implementa com.ibm.websphere.objectgrid.security.plugins.Authenticator interface. Questa classe viene utilizzata per autenticare i client ai server nel cluster ObjectGrid.

Il seguente file `universityClusterSecurity.xml` dimostra come utilizzare l'elemento authenticator. In questo esempio, la classe com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator viene specifiata come authenticator. Questa classe implementa l'interfaccia com.ibm.websphere.objectgrid.security.plugins.Authenticator.

file *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">
<nome cluster="universityCluster" securityEnabled="true"
singleSignOnEnabled="true"
loginSessionExpirationTime="1800" adminAuthorizationEnabled="true"
adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
<nome serverDefinition="server1" host="lion.ibm.com"
clientAccessPort="12501" peerAccessPort="12502" />
<authenticator
className ="com.ibm.websphere.objectgrid.security.plugins.builtins.
KeyStoreLoginAuthenticator" />
<adminAuthorization className= "com.ibm.MyAdminAuthorization">
<nome proprietà="intervallo" tipo="int" valore="60" descrizione="Impostare
l'intervallo a 60 secondi" />
</adminAuthorization>
</cluster>
<objectgridBinding ref="accademico">
<nome mapSet="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="facoltà" />
<map ref="studente" />
<map ref="corso" />
</mapSet>
</objectgridBinding>
<objectgridBinding ref="atletica">
<nome mapSet="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="atleta" />
<map ref="attrezzatura" />
</mapSet>
</objectgridBinding>
<nome partitionSet="partitionSet1">
<nome partizione="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<nome replicationGroup="replicationGroup1">
<replicationGroupMember serverRef="server1" priorità="1" />
</replicationGroup>
</clusterConfig>
```

elemento adminAuthorization

Numero di ricorrenze: zero a una

Elemento secondario: elemento proprietà

Un elemento adminAuthorization viene utilizzato per impostare l'accesso di gestione al cluster ObjectGrid. Le attività di gestione possono essere effettuate dopo che l'accesso di gestione è stato fornito.

Attributi

```
<adminAuthorization
(1) className="adminAuthClassName"
/>
```

1. attributo **className** (richiesto): L'attributo className viene utilizzato per specificare una classe che implementa l'interfaccia com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization.

Fare riferimento al file `universityClusterSecurity.xml` nella sezione authenticator per un utilizzo di esempio dell'elemento adminAuthorization. In universityClusterSecurity.xml, viene utilizzato un adminAuthorization personalizzato. La classe com.ibm.MyAdminAuthorization viene utilizzata come la classe adminAuthorization. Per utilizzare un adminAuthorization personalizzato, l'attributo securityEnabled deve essere `true`, adminAuthorizationMechanism deve essere impostato in `AUTHORIZATION_MECHANISM_CUSTOM`, e un elemento adminAuthorization deve essere fornito. Questo elemento adminAuthorization utilizza inoltre una

proprietà. Per maggiori informazioni sull'utilizzo delle proprietà, vedere "Elemento proprietà".

Elemento proprietà

Numero di ricorrenze: zero a molte

Elementi secondari: nessuno

L'elemento proprietà viene utilizzato per richiamare i metodi impostati su authenticator e adminAuthorization. Il nome della proprietà corrisponde a un metodo impostato sul className dell'elemento authenticator o adminAuthorization che contiene la proprietà.

Attributi

```
<property
(1) nome="propertyName"
(2) tipo="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) valore="propertyValue"
(4) descrizione="description"
/>
```

1. attributo **nome** (richiesto): Il nome della proprietà. Il valore assegnato a questo attributo deve corrispondere a un metodo impostato sulla classe che viene fornita come className per authenticator o adminAuthorization. Ad esempio, se il className di authenticator è impostato in com.ibm.MyAuthenticator e il nome della proprietà fornita è `intervallo`, la classe com.ibm.MyAuthenticator deve avere un metodo setInterval.
2. attributo **tipo** (richiesto): Il tipo di proprietà. È il tipo di parametro che è passato al metodo impostato identificato dall'attributo del nome. I valori validi sono le primitive Java, le controparti java.lang e java.lang.String. Il nome e il tipo devono corrispondere a un metodo di firma sul className del bean. Ad esempio, se il nome è intervallo e il tipo è int, un metodo setInterval(int) deve esistere sulla classe che viene specificata come className per authenticator o adminAuthorization.
3. attributo **valore** (richiesto): Il valore della proprietà. Questo valore viene convertito nel tipo che è specificato dall'attributo del tipo e quindi utilizzato come un parametro nella chiamata nel metodo impostato che viene identificato dagli attributi di nome e tipo. È importante sapere che il valore di questo attributo non viene convalidato in nessun modo. Il programma di implementazione dei plug-in deve verificare che il valore inoltrato sia valido. Il programma di implementazione può visualizzare un'eccezione `IllegalArgumentException` nel metodo impostato se il parametro non è valido.
4. Attributo **description** (opzionale): utilizzare questo attributo per scrivere una descrizione della proprietà.

Fare riferimento al file `universityClusterSecurity.xml` per un utilizzo campione dell'elemento adminAuthorization. L'elemento di proprietà può essere utilizzato negli elementi authenticator o adminAuthorization nel cluster XML. Nel file `universityClusterSecurity.xml`, la proprietà viene utilizzata per richiamare un metodo impostato su adminAuthorization. In questo caso, un metodo setInterval viene chiamato sulla classe com.ibm.MyAdminAuthorization. Viene passato un valore intero di `60`.

Capitolo 10. Integrazione di ObjectGrid con WebSphere Application Server

Utilizzare ObjectGrid con le funzioni fornite da WebSphere Application Server per migliorare le proprie applicazioni.

Installare WebSphere Application Server e WebSphere Extended Deployment. Una volta installato WebSphere Extended Deployment, è possibile aggiungere le funzioni di ObjectGrid alle applicazioni J2EE (Java 2 Platform, Enterprise Edition).

L'API di ObjectGrid API può essere utilizzata per un'applicazione J2EE di WebSphere Application Server. Il file `wsobjectgrid.jar` si trova nella directory `\base\lib` una volta installato WebSphere Extended Deployment. Oltre a integrare l'API di ObjectGrid con il modello di programmazione delle applicazioni J2EE, è possibile sfruttare il supporto d i propagazione delle transazioni distribuite. Con questo supporto, è possibile configurare le istanze di ObjectGrid per coordinare i risultati del commit della transazione su un cluster WebSphere Application Server.

1. Effettuare le operazioni di programmazione di base riportate di seguito per abilitare un'applicazione J2EE con ObjectGrid. Per ulteriori informazioni, consultare "Integrazione di ObjectGrid in un ambiente Java 2 Platform, Enterprise Edition".
2. Monitorare i dati delle prestazioni per le applicazioni ObjectGrid. Per ulteriori informazioni, consultare "Controllo delle prestazioni di ObjectGrid con la PMI (performance monitoring infrastructure) di WebSphere Application Server" a pagina 291.
3. Se ObjectGrid è integrato, le transazioni potrebbero iniziare e terminare mediante un coordinatore delle transazioni esterne. Per ulteriori informazioni, consultare "ObjectGrid e interazione con le transazioni esterne" a pagina 298.
4. Utilizzare l'utilità di partizione con ObjectGrid. La funzione ObjectGrid consente la memorizzazione nella cache di coppie chiave-valore in maniera transazionale mentre la funzione di utilità di partizione consente l'indirizzamento in base al contesto in accordo alle caratteristiche dell'oggetto. Per ulteriori informazioni, consultare "Integrazione di ObjectGrid e dell'utilità di partizione" a pagina 301.
5. È possibile utilizzare i bean CMP (container-managed persistence) in WebSphere Application Server Versione 6.0.2 e successive, sfruttando l'ObjectGrid come cache esterna invece che come cache integrata. Per ulteriori informazioni, consultare "Configurazione di ObjectGrid per operazioni con bean CMP" a pagina 322.

È inoltre possibile utilizzare ObjectGrid con JMS per distribuire le modifiche tra livelli differenti o in ambienti a piattaforme mista. Per ulteriori informazioni, consultare "Java Message Service per la distribuzione delle modifiche alle transazioni" a pagina 340.

Integrazione di ObjectGrid in un ambiente Java 2 Platform, Enterprise Edition

ObjectGrid supporta i modelli di programmazione servlet ed Enterprise JavaBeans (EJB) nell'ambiente Java 2 Platform, Enterprise Edition (J2EE).

In questa sezione sono riportate le operazioni di programmazione comuni che consentono di abilitare un'applicazione J2EE con ObjectGrid.

Scenario ObjectGrid locale

1. Definire una configurazione ObjectGrid. Definire una configurazione ObjectGrid con i file XML, mediante un'interfaccia programmatica o utilizzando entrambi. Per ulteriori informazioni, fare riferimento alla sezione Configurazione di ObjectGrid.
2. Creare un oggetto URL. Se la configurazione di ObjectGrid è riportata in un file XML, creare un URL che faccia riferimento a tale file XML. È possibile utilizzare questo URL per creare le istanze di ObjectGrid utilizzando l'API ObjectGridManager. Se il file di configurazione XML di ObjectGrid è incluso in un file Web archive (WAR) o Enterprise JavaBeans (EJB) Java archive (JAR), è possibile accedere ad esso come se fosse una risorsa al class loader sia per il modulo Web che per il modulo EJB. Ad esempio, se il file di configurazione XML ObjectGrid si trova nella cartella WEB–INF del file WAR del modulo Web, i servlet che si trovano nel file WAR possono creare un URL con il seguente modello:

   ```
   URL url =className.class.getClassLoader().
   getResource("META-INF/objectgrid-definition.xml");
   URL objectgridUrl = ObjectGridCreationServlet.class.getClassLoader().
   getResource("WEB-INF/objectgrid-definition.xml");
   ```
3. Creare o richiamare le istanze ObjectGrid. Utilizzare l'API ObjectGridManager per richiamare e creare le istanze di ObjectGrid. Con l'API ObjectGridManager, è possibile creare le istanze ObjectGrid con XML e utilizzare i metodi dei programmi di utilità per creare rapidamente un'istanza semplice di ObjectGrid. Le applicazioni devono utilizzare l'API ObjectGridManagerFactory per richiamare un riferimento all'API ObjectGridManager. Fare riferimento al seguente esempio di codice:

   ```
   import com.ibm.websphere.objectgrid.ObjectGridManager;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory ;
   ...
   ObjectGridManager objectGridManager = ObjectGridManagerFactory.
   getObjectGridManager();
   ObjectGrid ivObjectGrid = objectGridManager.
   createObjectGrid(objectGridName, objectgridUrl, true, true);
   ```

   Per ulteriori informazioni sull'API ObjectGridManager, fare riferimento alla sezione Interfaccia ObjectGridManager.
4. Inizializzare le istanze ObjectGrid. Utilizzare il metodo initialize dell'interfaccia ObjectGrid per avviare il bootstrap delle istanze di ObjectGrid e Session. Questo metodo initialize è considerato opzionale in quanto la prima chiamata al metodo getSession esegue un'inizializzazione implicita. Una volta richiamato tale metodo, la configurazione di ObjectGrid è considerata completa ed è pronta per l'utilizzo runtime. Qualsiasi altro richiamo di un metodo di configurazione aggiuntivo, come ad esempio il metodo defineMap(String mapName), provoca una eccezione.
5. Richiamare un'istanza Session e ObjectMap. Una sessione è un contenitore di istanze ObjectMap. Un thread deve richiamare il proprio oggetto Session in modo da interagire con l'ObjectGrid principale. Questa tecnica può essere considerata come una sessione che può essere utilizzata da un unico thread alla volta. La sessione viene condivisa tra i thread se viene utilizzato soltanto un thread alla volta. Tuttavia, se viene utilizzata una infrastruttura di transazioni o una connessione J2EE, l'oggetto sessione non è condivisibile. Una buona analogia per spiegare questo oggetto è data da una connessione Java Database Connectivity (JDBC) a un database.

   Una mappa ObjectMap è un handle per una determinata mappa. Le mappe devono avere chiavi e valori omogenei. Un'istanza ObjectMap può essere

utilizzata soltanto dal thread che associato alla sessione utilizzata per richiamare questa istanza di ObjectMap. Più thread non possono condividere simultaneamente gli oggetti Session e ObjectMap. Le parole chiave sono applicate all'interno di una transazione. Un rollback delle transazioni esegue il rollback dell'associazione di parole chiave applicata durante questa transazione.Di seguito è riportato un esempio di codifica:

```
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. Avviare una sessione, leggere o scrivere gli oggetti ed eseguire il commit o il rollback della sessione. Le operazioni delle mappe devono essere all'interno di un contesto transazionale. Il metodo begin della sessione viene utilizzato per iniziare un contesto transazionale esplicito. Una volta iniziata la sessione, le applicazioni possono iniziare l'esecuzione delle operazioni di associazione. Le operazioni più comuni includo le chiamate ai metodi get, `update`, insert e remove per gli oggetti rispetto alle mappe. Alla fine delle operazioni di associazione, viene richiamate il metodo di commit o di rollback dell'oggetto Session sia per eseguire il commit che il rollback di un contesto transazionale esplicito. Di seguito viene riportato un esempio di programmazione:

```
ivSession.begin();
Integer key = new Integer(1);
if (ivCounterMap.containsKey(key) == false) {
 ivCounterMap.insert(key, new Counter(10));
}
ivSession.commit();
```

Scenario ObjectGrid distribuito

Questo scenario di ObjectGrid distribuito differisce dallo scenario di ObjectGrid locale nel modo in cui viene richiamata l'istanza ObjectGrid. Il seguente esempio di codice dimostra come richiamare un'istanza ObjectGrid distribuita:

```
// Utilizzare ObjectGridManagerFactory per richiamare il riferimento all'API ObjectGridManager
ObjectGridManager objectGridManager = ObjectGridManagerFactory.
getObjectGridManager();
// Ottenere il ClientClusterContext che rappresenta il cluster ObjectGrid da
// ObjectGridManager
// Si assuma che il server WebSphere contenga anche un server ObjectGrid
// che fa parte del cluster ObjectGrid.
ClientClusterContext context = objectGridManager.connect(null, null);
// Richiamare l'istanza ObejctGrid da ObjectGridManager a
// ClientClusterContext.
ObjectGrid ivObjectGrid= objectGridManager.getObjectGrid(context,
"objectgridName");
```

A questo punto sono state effettuate le operazioni di programmazione di base per l'abilitazione di un'applicazione J2EE con ObjectGrid.

Fare riferimento alle sezioni Creazione di applicazioni Java 2 Platform, Enterprise Edition (J2EE) abilitate per ObjectGrid e Considerazioni sull'integrazione di applicazioni Java 2 Platform, Enterprise Edition (J2EE) e ObjectGrid per ulteriori informazioni.

Creazione di applicazioni J2EE (Java 2 Platform, Enterprise Edition) abilitate per ObjectGrid

Utilizzare questa attività per configurare il percorso di build, o classpath, delle applicazioni Java 2 Platform, Enterprise Edition (J2EE) abilitate per ObjectGrid. La variabile classpath deve includere il file `wsobjectgrid.jar` che si trova nella directory `$root_install`.

Sviluppare un'applicazione J2EE abilitata per ObjectGrid. Fare riferimento alla sezione Integrazione di ObjectGrid in un ambiente J2EE per ulteriori informazioni.

Questa attività dimostra come impostare il percorso di build in modo da includere il file `wsobjectgrid.jar` in IBM Rational Software Development Platform Versione 6.0.

1. Nella vista Project Explorer della prospettiva J2EE, fare clic con il pulsante destro del mouse sul progetto **WEB** o Enterprise JavaBeans (**EJB**), quindi selezionare **Proprietà**. Viene visualizzata la finestra delle proprietà.
2. Selezionare **Percorso build Java** dal pannello sulla sinistra, selezionare la scheda **Librerie** dal pannello di destra e fare clic su **Aggiungi variabile**. Viene visualizzata la finestra **Nuova voce della variabile classpath**.
3. Fare clic su **Configura variabili** per aprire la finestra**Preferenze**.
4. Aggiungere un nuova voce alla variabile.
   a. Fare clic su **Nuovo**.
   b. Immettere`OBJECTGRID_JAR` nel campo **Nome**. Fare clic su **File** per aprire la finestra **Selezione JAR**.
   c. Passare alla directory `/lib`, selezionare il file **wsobjectgrid.jar** e fare clic su **Apri** per chiudere la finestra **Selezione JAR**.
   d. Fare clic su **OK** per chiudere la finestra**Nuova voce di variabile**.

   La variabile `OBJECTGRID_JAR` viene visualizzata nell'elenco delle variabili classpath.
5. Fare clic su **OK** per chiudere la finestra **Preferenze**.
6. Selezionare la variabile`OBJECTGRID_JAR` dall'elenco di variabili e fare clic su **OK** per chiuderela finestra **Nuova voce della variabile classpath**. La variabile `OBJECTGRID_JAR` viene visualizzata nel pannello **Librerie**.
7. Fare clic su **OK** per chiudere la finestra **Proprietà**.

A questo punto è stato impostato il percorso di build in modo da includere il file `wsobjectgrid.jar` in IBM Rational Software Development Platform Versione 6.0.

Considerazioni sull'integrazione di applicazioni J2EE (Java 2 Platform, Enterprise Edition) e ObjectGrid

Utilizzare queste considerazioni quando si integra un'applicazione Java 2 Platform, Enterprise Edition (J2EE) con ObjectGrid.

Bean di avvio e ObjectGrid

È possibile utilizzare i bean di avvio per il bootstrap da parte di un'applicazione di un'istanza di ObjectGrid all'avvio dell'applicazione e per l'eliminazione dell'istanza di ObjectGrid all'arresto dell'applicazione. Un bean di avvio è un bean di sessione stateless con una classe remota com.ibm.websphere.startupservice.AppStartUpHome e un'interfaccia remota com.ibm.websphere.startupservice.AppStartUp. Quando WebSphere Application Server trova un Enterprise JavaBean (EJB), viene subito riconosciuto il bean di

avvio. L'interfaccia remota ha due metodi, il metodo start e il metodo stop. Utilizzare il metodo start per eseguire il bootstrap della griglia e richiamare il metodo destroy con il metodo stop. L'applicazione può conservare un riferimento alla griglia utilizzando il metodo ObjectGridManager.getObjectGrid per richiamare un riferimento quando necessario. Per ulteriori informazioni, fare riferimento alla sezione Interfaccia ObjectGridManager.

Programmi di caricamento delle classi e istanze di ObjectGrid

È necessario prestare particolare attenzione quando si condivide un'istanza di ObjectGrid tra moduli delle applicazioni che utilizzano diversi programmi di caricamento delle classi. I moduli delle applicazioni che utilizzano programmi di caricamento diversi non funzionano e provocano eccezioni di utilizzo delle classi nell'applicazione. Un ObjectGrid deve essere condiviso solo dai moduli delle applicazioni che utilizzano lo stesso programma di caricamento delle classi o quando gli oggetti delle applicazioni, ad esempio i plug-in, le chiavi e i valori, si trovano su un programma di caricamento delle classi comuni.

Gestione del ciclo di vita delle istanze di ObjectGrid in un servlet

È possibile gestire il ciclo di vita delle istanze di ObjectGrid con il metodo init e il metodo destroy di un servlet. Utilizzare il metodo init per creare e inizializzare le istanze di ObjectGrid che sono necessarie all'applicazione. Una volta create le istanze di ObjectGrid e dopo averle memorizzate nella cache, è possibile richiamare le istanze in base al nome mediante l'API ObjectGridManager. Utilizzare il metodo destroy per eliminare le istanze di ObjectGrid e per rilasciare le risorse di sistema. Per ulteriori informazioni, fare riferimento alla sezione Interfaccia ObjectGridManager.

Controllo delle prestazioni di ObjectGrid con la PMI (performance monitoring infrastructure) di WebSphere Application Server

ObjectGrid supporta l'infrastruttura PMI (performance monitoring infrastructure) se viene eseguito su un server delle applicazioni WebSphere Application Server o WebSphere Extended Deployment. PMI raccoglie i dati delle prestazioni sulle applicazioni di runtime e fornisce le interfacce che consentono alle applicazioni esterne di monitorare i dati delle prestazioni.

Per ulteriori informazioni sulle statistiche fornite da ObjectGrid, fare riferimento a Statistiche di ObjectGrid.

ObjectGrid utilizza la funzione PMI personalizzata di WebSphere Application Server per aggiungere la propria struttura PMI. Con questo approccio, è possibile abilitare e disabilitare la PMI di ObjectGrid dalla console di gestione oppure mediante le interfacce Java Management Extensions (JMX). Inoltre, è possibile accedere alle statistiche ObjectGrid con le interfacce PMI e JMX standard che sono utilizzate dagli strumenti di controllo quali Tivoli Performance Viewer.

1. Abilitare la PMI di ObjectGrid. Per visualizzare le statistiche PMI, è necessario abilitare l'infrastruttura PMI. Per ulteriori informazioni, consultare "Abilitazione dell'infrastruttura PMI di ObjectGrid" a pagina 294.
2. Richiamare le statistiche PMI di ObjectGrid. È possibile visualizzare le prestazioni delle applicazioni ObjectGrid mediante Tivoli Performance Viewer. Per ulteriori informazioni, consultare "Richiamo delle statistiche PMI di ObjectGrid" a pagina 297.

Statistiche di ObjectGrid

ObjectGrid fornisce due moduli PMI (performance monitoring infrastructure): il modulo objectGridModule e il modulo mapModule.

Modulo objectGridModule

Il modulo objectGridModule contiene una statistica temporale: il tempo di risposta delle transazioni. Una transazione di ObjectGrid è definita come la durata tra la chiamata al metodo Session.begin e la chiamata al metodo Session.commit. Questa durata rappresenta il tempo di risposta di una transazione.

L'elemento root del modulo objectGridModule, l'elemento ObjectGrids, funziona come punto di immissione per le statistiche ObjectGrid. Questo elemento root ha le istanze di ObjectGrid come elementi secondari che hanno tipi di transazioni come elementi secondari relativi. La statistica del tempo di risposta è associata a ogni tipo di transazione. La struttura del modulo objectGridModule è riportata del seguente diagramma:

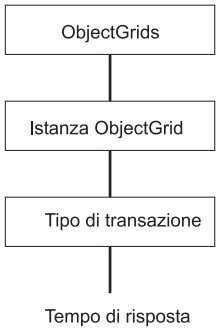


Figura 19. Struttura del modulo ObjectGridModule

Il seguente diagramma mostra un esempio della struttura del modulo PMI ObjectGrid. In questo esempio, sono presenti due istanze ObjectGrid sul sistema: objectGrid1 ObjectGrid e objectGrid2 ObjectGrid. L'istanza objectGrid1 ha due tipi di transazioni, aggiornamento e lettura, mentre l'istanza objectGrid2 ha solo un tipo di transazione, quella di aggiornamento.

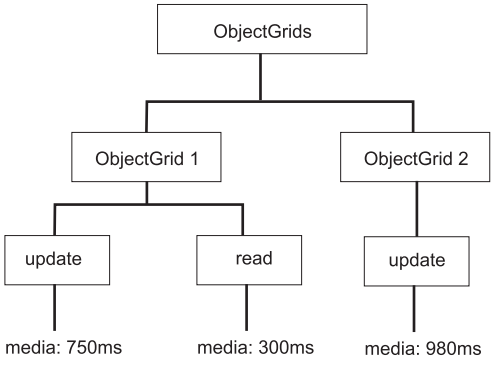


Figura 20. Struttura del modulo PMI ObjectGrid

I tipi di transazioni sono definiti dagli sviluppatori delle applicazioni in quanto conoscono già i tpi utilizzati dalle applicazioni. Il tipo di transazione viene impostato utilizzando il seguente metodo Session.setTransactionType(String):

```
/**
* Imposta il tipo di transazione per le transazioni future.
*
* Una volta richiamato il metodo, tutte le transazioni future sono dello
* stesso tipo a meno che non ne venga impostato un altro. Se non viene utilizzato
* alcun tipo do transazione, verrà utilizzato il tipo predefinito
* TRANSACTION_TYPE_DEFAULT.
*
* I tipi di transazioni sono utilizzati principalmente per la traccia dei dati statistici.
* gli utenti possono predefinire i tipi di transazioni che vengono eseguite su
* un'applicazione. L'idea sta nell'assegnare una categoria (tipo) alle transazioni
* con le stesse caratteristiche
* in modo che la statistica del tempo di risposta delle transazioni possa
* essere utilizzato per tenere traccia di ogni tipo.
*
* Questa traccia è utile quando l'applicazione ha diversi tipi di
* transazioni.
* Tra questi, vi sono alcuni tipi particolari, come le transazioni di aggiornamento,
* un processo più lungo delle altre transazioni, come le transazioni di sola lettura.
* Utilizzando il tipo di transazione, le diverse transazioni vengono controllate
* utilizzando diverse statistiche,
* in modo che queste risultino più utili.
*
* @param tranType il tipo di transazione per le transazioni future.
*/
void setTransactionType(String tranType);
```

Il seguente esempio imposta il tipo di transazione su `updatePrice`:

```
// Imposta il tipo di transazione su updatePrice
// Il tempo tra session.begin() e session.commit() verrà
// controllato nella statistica del tempo per "updatePrice".
session.setTransactionType("updatePrice");
session.begin();
map.update(stockId, new Integer(100));
session.commit();
```

La prima riga indica che il tipo di transazione successivo è `updatePrice`. Una statistica `updatePrice` è presente nell'istanza ObjectGrid che corrispone alla sessione nell'esempio. Mediante le interfacce Java Management Extensions (JMX), è possibile richiamare il tempo di risposta per le transazioni updatePrice. È inoltre possibile richiamare le statistiche aggregate per tutti i tipi di transazioni sull'ObjectGrid specificato.

Modulo mapModule

Il modulo PMI mapModule contiene tre statistiche relative alle mappe ObjectGrid:

- **Frequenza di successi mappa:** questa statistica BoundedRangeStatistic tiene traccia della frequenza di successi di una mappa. Questa frequenza è un valore a virgola mobile compreso tra `0` e `100` e rappresenta la percentuale di successi della mappa in relazione alle operazioni di richiamo mappa.
- **Numero di voci**: questa statistica CountStatistic tiene traccia del numero di voci nella mappa.
- **Tempo di risposta dell'aggiornamento batch del programma di caricamento**: questa statistica TimeStatistic tiene traccia del tempo di risposta utilizzato per l'operazione di aggiornamento batch del programma di caricamento.

L'elemento root del modulo mapModule, l'elemento ObjectGrid Maps, funziona come punto di immissione per le statistiche ObjectGrid Map. Questo elemento root ha le istanze di ObjectGrid come elementi secondari che hanno le istanze delle

mappe come elementi secondari relativi. Ogni istanza della mappa ha tre statistiche riportate. La struttura mapModule è riportata nel seguente diagramma:
Il seguente diagramma mostra un esempio della struttura mapModule:

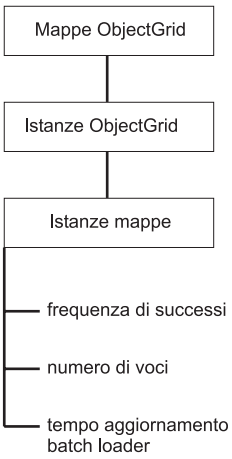


Figura 21. Struttura del modulo mapModule

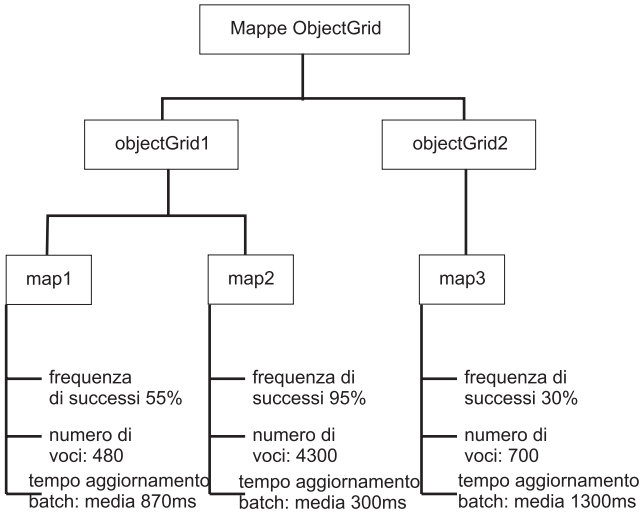


Figura 22. Esempio di struttura del modulo mapModule

Abilitazione dell'infrastruttura PMI di ObjectGrid

È possibile utilizzare l'infrastruttura PMI (Performance Monitoring Infrastructure) di WebSphere Application Server per abilitare o disabilitare le statistiche a qualsiasi livello. Ad esempio, è possibile decidere se abilitare la statistica della frequenza di successi per una determinata mappa ma non il numero di statistiche della voce o la statistica del tempo di aggiornamento batch del programma di caricamento. In questa sezione viene descritto come utilizzare la console di gestione e gli script wsadmin per abilitare la PMI di ObjectGrid.

Utilizzare la PMI di WebSphere Application Server per fornire un meccanismo granulare con cui è possibile abilitare o disabilitare le statistiche a qualsiasi livello. Ad esempio, è possibile decidere se abilitare la statistica della frequenza di successi per una determinata mappa ma non il numero di statistiche della voce o la statistica del tempo di aggiornamento batch del programma di caricamento. In

questa sezione viene descritto come utilizzare la console di gestione e gli script wsadmin per abilitare la PMI di ObjectGrid.

1. Aprire la console di gestione, ad esempio http://hostlocale:9060/ibm/console.
2. Fare clic su **Controllo e ottimizzazione > PMI (Performance Monitoring Infrastructure) > *nome_server***.
3. Verificare che la casella **Abilita PMI (Performance Monitoring Infrastructure)** sia selezionata. Per impostazione predefinita, questa impostazione è abilitata. Se l'impostazione non è abilitata, selezionare la casella di spunta e riavviare il server.
4. Fare clic su **Personalizzato**. Nella struttura di configurazione, selezionare **ObjectGrid** e il modulo **Mappe di ObjectGrid**. Abilitare le statistiche per ogni modulo.

La categoria del tipo di transazione per le statistiche ObjectGrid viene creata in fase di runtime. Dal pannello Runtime è possibile visualizzare soltanto le categorie secondarie delle statistiche di ObjectGrid e Map.

Ad esempio, è possibile effettuare le seguenti operazioni per abilitare le statistiche PMI per l'applicazione di esempio:

1. Avviare l'applicazione dall'indirizzo Web `http://host:porta/ObjectGridSample`, dove host e porta sono il nome host e il numero di porta HTTP del server su cui è installato l'esempio.
2. Nell'applicazione di esempio, fare clic su **ObjectGridCreationServlet**, quindi selezionare i pulsanti 1, 2, 3, 4 e 5 per generare alcune azioni per ObjectGrid e le mappe. Non chiudere questa pagina del servlet.
3. Tornare alla console di gestione e fare clic su **Controllo e ottimizzazione > PMI (Performance Monitoring Infrastructure) > *nome_server***. Selezionare la scheda **Runtime**.
4. Selezionare il pulsante d'opzione **Personalizzato**.
5. Espandere il modulo **Mappe di ObjectGrid** nella struttura di runtime e selezionare il link **clusterObjectGrid**. Nel gruppo **Mappe di ObjectGrid**, esiste un'istanza di ObjectGrid denominata clusterObjectGrid all'interno della quale sono presenti quattro mappe: counters, employees, offices e sites. Nell'istanza **ObjectGrids**, è presente un'istanza clusterObjectGrid, all'interno della quale è presente un tipo di transazione denominato `DEFAULT`.
6. È possibile abilitare le statistiche per cui si è interessati. A scopo dimostrativo, è possibile abilitare **numero di voci della mappa** per la mappa employees e **tempo di risposta delle transazioni** per il tipo di transazione DEFAULT.

È possibile automatizzare l'attività di abilitazione della PMI mediante gli script. Fare riferimento a Abilitazione della PMI di ObjectGrid con gli script per maggiori informazioni.

Abilitazione dell'infrastruttura PMI di ObjectGrid con gli script

È possibile automatizzare l'attività di abilitazione della PMI di ObjectGrid con lo strumento wsadmin.

Il server delle applicazioni deve essere avviato e deve avere un'applicazione abilitata per ObjectGrid installata. È inoltre necessario collegarsi e utilizzare lo strumento wsadmin. Per ulteriori informazioni sullo strumento wsadmin, fare riferimento a Utilizzo degli script (wsadmin) nel Centro informazioni di WebSphere Extended Deployment Versione 6.0.x.

Utilizzare questa attività per automatizzare l'abilitazione dell'infrastruttura PMI. Per abilitare PMI con la console di gestione, fare riferimento a Abilitazione della PMI di ObjectGrid.

1. Aprire una finestra della riga comandi. Passare alla directory *root_install*/bin. Immettere wsadmin per avviare lo strumento wsadmin.
2. Modificare la configurazione di runtime PMI di ObjectGrid. Verificare se PMI è abilitata per il server mediante i seguenti comandi:

   ```
   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 PMI non è abilitata, emettere i seguenti comandi:

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

   Se si abilita PMI, riavviare il server.
3. Impostare le variabili per la modifica di una serie di statistiche in una serie personalizzata. Eseguire questi comandi:

   ```
   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. Impostare la serie di statistiche su personalizzato: Eseguire questo comando:

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
   ```
5. Impostare le variabili per abilitare le statistiche PMI di objectGridModule. Eseguire questi comandi:

   ```
   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. Impostare la stringa di statistiche. Eseguire questo comando:

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params $sigs
   ```
7. Impostare le variabili per abilitare le statistiche PMI di mapModule. Eseguire questi comandi:

   ```
   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. Impostare la stringa di statistiche. Eseguire questo comando:

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

Queste operazioni abilitano la PMI di runtime di ObjectGrid, ma non modificano la configurazione della infrastruttura PMI. Se si riavvia il server delle applicazioni, le impostazioni di PMI vengono perse.

Una volta abilitata PMI, è possibile visualizzare le relative statistiche dalla console di gestione o mediante gli script. Per ulteriori informazioni, consultare “Richiamo delle statistiche PMI di ObjectGrid” e “Richiamo delle statistiche PMI di ObjectGrid con gli script”.

Richiamo delle statistiche PMI di ObjectGrid

Fare riferimento alle statistiche delle prestazioni delle applicazioni ObjectGrid.

Una volta abilitate le statistiche di ObjectGrid, è possibile richiamarle. Per abilitare la PMI di ObjectGrid, fare riferimento alla sezione Abilitazione di PMI di ObjectGrid.

Utilizzare questa attività per visualizzare le statistiche delle prestazioni delle applicazioni ObjectGrid.

1. Aprire la console di gestione. Ad esempio, http://hostlocale:9060/ibm/console.
2. Fare clic su **Controllo e ottimizzazione > Visualizzatore prestazioni > Attività corrente**.
3. Selezionare il server che si desidera monitorare utilizzando Tivoli Performance Viewer e abilitare il controllo.
4. Fare clic sul server per visualizzare la pagina Visualizzatore prestazioni.
5. Espandere la struttura di configurazione. Fare clic su **Mappe ObjectGrid > clusterObjectGrid**, quindi selezionare **employees**. Espandere **ObjectGrids > clusterObjectGrid**, quindi selezionare **DEFAULT**.
6. Nell’applicazione di esempio di ObjectGrid, tornare al servlet ObjectGridCreationServlet, fare clic sul pulsante **associa mappe**. È quindi possibile visualizzare le statistiche nel visualizzatore.

È possibile visualizzare le statistiche di ObjectGrid in Tivoli Performance Viewer.

È possibile automatizzare l’attività per il richiamo delle statistiche utilizzando Java Management Extensions (JMX) o lo strumento wsadmin. Consultare “Richiamo delle statistiche PMI di ObjectGrid con gli script”

Richiamo delle statistiche PMI di ObjectGrid con gli script

Utilizzare questa attività per richiamare le statistiche delle prestazioni per le applicazioni ObjectGrid.

Abilitare PMI (Performance Monitoring Infrastructure) nell’ambiente del server delle applicazioni. Fare riferimento alle sezioni Abilitazione della PMI di ObjectGrid o Abilitazione della PMI di ObjectGrid con gli script per ulteriori informazioni. È inoltre necessario collegarsi e utilizzare lo strumento wsadmin. Per ulteriori informazioni sullo strumento wsadmin, fare riferimento a Utilizzo degli script (wsadmin) nel Centro informazioni di WebSphere Extended Deployment Versione 6.0.x.

Utilizzare questa attività per richiamare le statistiche delle prestazioni per l’ambiente del server delle applicazioni. Per ulteriori informazioni sulle statistiche di ObjectGrid che possono essere richiamate, fare riferimento a “Statistiche di ObjectGrid” a pagina 292.

1. Aprire una finestra della riga comandi. Passare alla directory *root_install*/bin. Immettere `wsadmin` per avviare lo strumento wsadmin della riga comandi.
2. Impostare le variabili per l’ambiente. Eseguire questi comandi:

```
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. Impostare le variabili per ottenere le statistiche di mapModule. Eseguire questi comandi:

```
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. Richiamare le statistiche di mapModule. Eseguire questo comando:

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

5. Impostare le variabili per ottenere le statistiche di objectGridModule. Eseguire questi comandi:

```
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. Richiamare le statistiche di objectGridModule. Eseguire questo comando:

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

Fare riferimento a “Statistiche di ObjectGrid” a pagina 292 per ulteriori informazioni sulle statistiche che vengono restituite.

ObjectGrid e interazione con le transazioni esterne

Di solito, le transazioni ObjectGrid iniziano con il metodo session.begin e terminano con il metodo session.commit. Tuttavia, quando ObjectGrid è integrato, le transazioni potrebbero iniziare e terminare mediante un coordinatore delle transazioni esterne. In questo caso, non è necessario richiamare il metodo session.begin e terminare con il metodo session.commit.

Coordinamento delle transazioni esterne

Il plug-in TransactionCallback di ObjectGrid viene esteso con il metodo isExternalTransactionActive(Session session) che associa la sessione ObjectGrid con una transazione esterna. Di seguito è riportata l’intestazione del metodo:

```
public synchronized boolean isExternalTransactionActive(Session session)
```

Ad esempio, ObjectGrid può essere impostato in modo da essere integrato con WebSphere Application Server e WebSphere Extended Deployment. la chiave di questa integrazione sta nell’utilizzo dell’API ExtendedJTATransaction in WebSphere Application Server Versione 5.x e Versione 6.x. Tuttavia, se si utilizza WebSphere Application Server Versione 6.0.2, è necessario applicare l’APAR PK07848 per supportare questo metodo. Utilizzare il seguente codice di esempio per associare una sessione ObjectGrid a un ID di transazione di WebSphere Application Server:

```
/**
* Questo metodo è richiesto per associare una sessione objectGrid con un
* ID di transazione WebSphere.
*/
Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
 // tenere presente che questo ID locale implica che questa sessione
```

```
    viene salvata successivamente.
 localIdToSession.put(new Integer(jta.getLocalId()), session);
return true;
}
```

Richiamo di una transazione esterna

A volte potrebbe essere necessario richiamare un oggetto di servizio di transazioni esterne per il plug-in di ObjectGrid TransactionCallback da utilizzare. Sul server WebSphere Application Server, è possibile ricercare l'oggetto ExtendedJTATransaction dallo spazio dei nomi come riportato nel seguente esempio:

```
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");
}
}
```

Per gli altri prodotti, è possibile utilizzare un approccio simile per richiamare l'oggetto.

Commit di controllo mediante la richiamata esterna

Il plug-in TransactionCallback deve ricevere un segnale esterno per eseguire il commit o il rollback della sessione ObjectGrid. Per ricevere questo segnale esterno, è possibile utilizzare la richiamata dal servizio di transazione esterno. È necessario implementare l'interfaccia di richiamata esterna e registrarla con il servizio delle transazioni esterne. Ad esempio, nel caso di WebSphere Application Server, è necessario implementare l'interfaccia SynchronizationCallback come riportato nel seguente esempio:

```
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);
 // trova l'oggetto Session per localId
 Session session = (Session)localIdToSession.get(lid);
 if(session != null)
{
try
{
   // se viene eseguito il commit di WebSphere Application Server quando
   // di codifica la transazione su backingMap.
   // È stato già eseguito un metodo flush in beforeCompletion
   if(didCommit)
{
session.commit();
}
else
{
// otherwise rollback
session.rollback();
}
}
  catch(NoActiveTransactionException e)
{
   // impossibile in teoria
}
  catch(TransactionException e)
{
   // visto che è già stato eseguito un metodo flush,
      questo non dovrebbe riportare alcun errore
}
finally
{
   // cancellare sempre la sessione dalla mappa di associazione.
   localIdToSession.remove(lid);
}
}
}
public synchronized void beforeCompletion(int localId, byte[] arg1)
{
 Session session = (Session)localIdToSession.get(new Integer(localId));
 if(session != null)
{
try
{
   session.flush();
}
  catch(TransactionException e)
{
   // WebSphere Application Server non definisce formalmente
   // un modo per segnalare che
   // la transazione ha riportato un errore, pertanto
throw new RuntimeException("Cache flush failed", e);
}
}
}
}
```

Utilizzo delle API di ObjectGrid con il plug-in TransactionCallback

Questo plug-in, se utilizzato come il plug-in TransactionCallback per un ObjectGrid, disabilita la funzione di commit automatico. Di seguito è riportato un modello di utilizzo normale per un ObjectGrid:

```
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
ogSession.begin();
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
ogSession.commit();
```

Quando questo plug-in TransactionCallback è in uso, ObjectGrid assume che l'applicazione utilizzi l'ObjectGrid quando è presente una transazione gestita dal contenitore. Il precedente frammento di codice diventa il seguente codice in un ambiente del genere:

```
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();
}
```

Il metodo myMethod è simile a un caso di applicazione Web. L'applicazione utilizza l'interfaccia UserTransaction per iniziare, eseguire il commit ed eseguire il rollback delle transazioni. L'ObjectGrid inizia automaticamente ed esegue il commit della transazione del contenitore. Se il metodo è un metodo EJB (Enterprise JavaBeans) che utilizza l'attributo TX_REQUIRES, allora è necessario rimuovere il riferimento a UserTransaction e le chiamate per iniziare ed eseguire il commit delle transazioni. In questo caso, il contenitore è responsabile dell'avvio e della fine della transazione.

Integrazione di ObjectGrid e dell'utilità di partizione

Utilizzare l'applicazione di esempio ObjectGridPartitionCluster per imparare a utilizzare le funzioni combinate di ObjectGrid e dell'utilità di partizione (WPF).

Fare riferimento a "ObjectGrid e l'utilità di partizione" a pagina 302 per un riepilogo del modo in cui ObjectGrid e l'utilità di partizione funzionano insieme.

Per utilizzare ObjectGrid con l'utilità di partizione, è necessario che WebSphere Extended Deployment sia installato sull'ambiente.

L'esempio ObjectGridPartitionCluster dimostra la funzioni combinate di ObjectGrid e Partitioning Facility (WPF). La funzione ObjectGrid consente la memorizzazione nella cache di coppie chiave-valore in base alla transazione mentre la funzione di utilità di partizione consente l'indirizzamento in base al contesto in base alle caratteristiche dell'oggetto.

- Installare ed eseguire l'applicazione di esempio ObjectGridPartitionCluster. Per ulteriori informazioni, consultare "Installazione ed esecuzione dell'applicazione di esempio ObjectGridPartitionCluster" a pagina 304.

- Se si desidera visualizzare o modificare il codice sorgente dell'applicazione di esempio, è possibile caricare il file EAR (enterprise archive) nello strumento di sviluppo. Per ulteriori informazioni, consultare "Creazione di un ObjectGrid e di un'applicazione dell'utilità di partizione" a pagina 307.
- Imparare a utilizzare l'applicazione di esempio. Fare riferimento a "Esempio: ObjectGrid programmazione dell'utilità di partizione" a pagina 311 per una spiegazione relativa al codice presente nell'applicazione di esempio.

Fare riferimento a Capitolo 10, "Integrazione di ObjectGrid con WebSphere Application Server", a pagina 287 per ulteriori informazioni su come integrare ObjectGrid con altre funzioni di WebSphere Application Server. Per ulteriori informazioni sul modello di programmazione di ObjectGrid, fare riferimento a Capitolo 9, "Panoramica sull'interfaccia di programmazione dell'applicazione ObjectGrid.", a pagina 89.

ObjectGrid e l'utilità di partizione

Le funzioni ObjectGrid e l'utilità di partizione (WPF) possono funzionare insieme in modo da consentire la memorizzazione nella cache di coppie chiave-valore e di routing basato sul contesto in base alle caratteristiche degli oggetti.

L'esempio ObjectGridPartitionCluster dimostra le funzioni combinate di ObjectGrid e dell'utilità di partizione (WPF). Queste sono due funzioni del prodotto WebSphere Extended Deployment. La funzione ObjectGrid consente la memorizzazione nella cache di coppie chiave-valore in maniera transazionale mentre la funzione di utilità di partizione consente l'indirizzamento in base al contesto in accordo alle caratteristiche dell'oggetto.

Oltre a dimostrare le funzioni del programma di caricamento e dei plug-in TransactionCallback, questo esempio mostra come utilizzare i plug-in ObjectGridEventListener, ObjectTransformer e OptimisticCallback. In particolare, dimostra come propagare le transazioni ObjectGrid locali e come invalidare gli oggetti modificati da un server ad altri server con e senza il programma di controllo della versione.

È necessario utilizzare la funzione di indirizzamento basata sul contesto dell'utilità di partizione per garantire che le richieste di aggiornamento, inserimento e rimozione di oggetti per la stessa chiave vengano indirizzate alla stessa Java virtual machine (JVM) e che le richieste di richiamo oggetto possano essere distribuite su tutte le JVM ObjectGrid con la gestione del carico di lavoro. L'utilizzo dell'utilità di partizione garantisce l'integrità dei dati sulle istanze ObjectGrid di membri del cluster differenti.

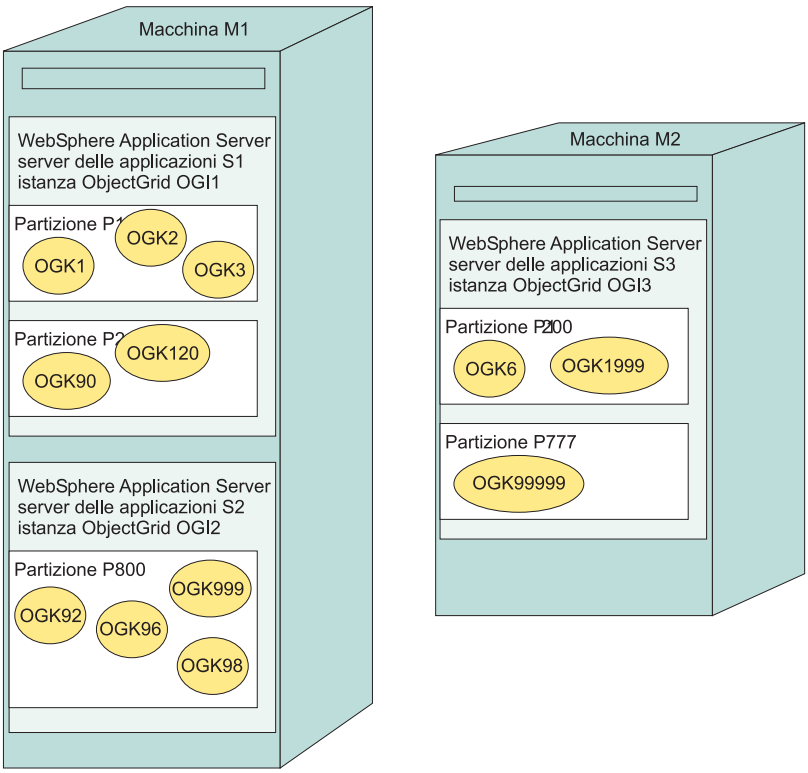
Per mantenere coerenza e integrità con ObjectGrid, è possibile utilizzare l'utilità di partizione per dividere un ObjectGrid di grosse dimensioni in più ObjectGrid partizionati a cui l'utilità di partizione indirizza le richieste in base alle chiavi ObjectGrid. Ad esempio, ObjectGrid è necessario per gestire un elevato numero di oggetti che non possono essere gestiti da un ObjectGrid della JVM. È possibile utilizzare l'utilità di partizione per caricare i dati su server differenti con il metodo partitionLoadEvent come precaricamento e l'indirizzamento dell'utilità di partizione trova quindi l'ObjectGrid corretto.

L'esempio crea una serie di partizioni basate su hash e i contesti di indirizzamento del cluster della partizione:

- È possibile partizionare e associare le chiavi ObjectGrid alle partizioni WPF con una strategia di relazione ″molti a molti″.

- Con questo tipo di strategia le partizioni WPF possono far parte del cluster WebSphere Application Server.

Il seguente diagramma mostra la configurazione e le impostazioni tipiche per l'esempio ObjectGridPartitionCluster:



Nel diagramma precedente, la macchina M1 e la macchina M2 sono utilizzate per distribuire l'esempio ObjectGridPartitionCluster. Ogni macchina fisica può ospitare uno o più WebSphere Application Server. Ad esempio, la macchina M1 ospita due server delle applicazioni, il server della applicazioni S1 e il server delle applicazioni S2. La macchina M2 ospita solo un server, il server delle applicazioni S3. Ogni server ha un'istanza ObjectGrid: l'istanza OGI1ObjectGrid per il server delle applicazioni S1, l'istanza OGI2 ObjectGrid per il server delle applicazioni S2 e l'istanza OGI3 ObjectGrid per il server delle applicazioni S3.

Ogni server delle applicazioni può avere più partizioni. Ad esempio, il server S1 ospita la partizione P1 e la partizione P2 mentre il server S3 ospita la partizione P1200 e la partizione P777.

Ogni partizione a sua volta può contenere molte chiavi ObjectGrid. Ad esempio, la partizione P1 ospita le chiavi ObjectGrid OGK1, OGK2 e OGK3 e la partizione P800 ospita le partizioni OGK92, OGK96, OGK98 e OGK9999.

Tutte le richieste di aggiornamento, inserimento e rimozione di ObjectGrid vengono indirizzate in base alle chiavi ObjectGrid. Per ogni richiamo degli oggetti sono invece disponibili due opzioni: da qualsiasi server in una strategia di carico di lavoro gestito o da una determinata partizione del server per questa chiave.

Installazione ed esecuzione dell'applicazione di esempio ObjectGridPartitionCluster

Utilizzare questa attività per installare ed eseguire l'applicazione di esempio ObjectGridPartitionCluster per verificare la funzionalità tra ObjectGrid e l'utilità di partizione.

Installare WebSphere Extended Deployment. Fare riferimento alla libreria di WebSphere Extended Deployment per le relative istruzioni.

Un buon ambiente per l'esecuzione dell'esempio ObjectGridPartitionCluster include l'installazione di WebSphere Extended Deployment su due macchine fisiche o la creazione di due nodi associati al gestore distribuzione.

1. Per dimostrare adeguatamente le funzioni di questo esempio, configurare un cluster con tre o più membri.
2. Installare il file `D_ObjectGridPartitionClusterSample.ear`. Il file distribuito dell'utilità di partizione (WPF) `D_ObjectGridPartitionClusterSample.ear` è pronto per essere installato ed eseguito. Se si modifica il codice sorgente di esempio, seguire le istruzioni integrate e wpf-deploy per creare e distribuire il file EAR (enterprise archive).

   Il modo più comune per installare i file EAR delle applicazioni consiste nell'utilizzare la console di gestione. Seguire la procedura di installazione dell'applicazione enterprise per installare il file `D_ObjectGridPartitionClusterSample.ear`. Per accedere a questa parte della console di gestione, fare clic su **Applicazioni > Installa una nuova applicazione**. Non distribuire il file EAR durante l'installazione. Utilizzare le impostazioni predefinite tranne nel passo in cui viene chiesto di selezionare il percorso di installazione. In questo passo, selezionare il cluster definito invece che il server predefinito server1.
3. Eseguire il client ObjectGridPartitionClusterSample.
   a. Avviare il cluster. Nella console di gestione, fare clic su **Server > Cluster**. Selezionare il cluster e fare clic su **Avvia**.
   b. Eseguire lo script **WAS_INSTALL_ROOT\bin\wpfadmin balance**. Verificare che la partizione abbia uno stato attivo utilizzando il comando **WAS_INSTALL_ROOT\bin\wpfadmin list**. Per ulteriori informazioni sullo script wpfadmin e sui relativi comandi, fare riferimento al manuale Partitioning Facility Guide nel Centro informazioni di WebSphere Extended Deployment.
   c. Per eseguire il client ObjectGridPartitionClusterSample, emettere il seguente comando:

      ```
      WAS_INSTALL_ROOT/bin/launchClient.bat|sh \
      WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear \
      -CCBootstrapPort=PORT
      ```

      PORT è la porta RMI del server che si trova nel file `SystemOut.log` dopo aver avviato il server. Di solito il valore di questa porta è `9810`, `9811` o `9812`.

      Ad esempio, è possibile emettere il seguente comando:

      ```
      WAS_INSTALL_ROOT/bin/launchClient.bat|sh
      WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear
        -CCBootstrapPort=9811
      ```

      Per ulteriori informazioni su questo script, fare riferimento a "Opzioni del client applicativo per ObjectGridPartitionClusterSample" a pagina 305.

4. Modificare il numero di partizioni. Modificare il numero di partizioni create dal bean enterprise della sezione ObjectGridPartitionCluster; il numero di partizioni creato dal bean di sessione PFClusterObjectGridEJB è deciso dalla variabile d'ambiente NumberOfPartitions che si trova nel file `META-INF\ejb-jar.xml`. Il valore predefinito è `10`. Per creare un numero diverso di partizioni, è necessario modificare il valore di questa variabile d'ambiente e reinstallare l'applicazione. Impostare il numero di partizioni su un valore inferiore a `999999`.
5. Modificare le opzioni del listener distribuito. È possibile modificare le seguenti opzioni del listener distribuito di ObjectGrid:

Tabella 17. Opzioni del listener distribuito

Nome variabile	Descrizione
enableDistribution,	È possibile abilitare un listener distribuito di ObjectGrid con la variabile d'ambiente enableDistribution che si trova nel descrittore di distribuzione EJB. Il valore predefinito è `true` , il che significa che l'opzione è abilitata. Impostare il valore su `false` per disattivare il listener distribuito.
propagationMode	È possibile modificare la modalità di propagazione con la variabile d'ambiente propagationMode che si trova nel descrittore di distribuzione EJB. Il valore predefinito è `update`. È possibile modificare il valore su `invalidate` se non si desidera utilizzare il valore predefinito.
propagationVersionOption,	È possibile modificare l'opzione della versione di propagazione con la variabile d'ambiente propagationVersionOption che si trova nel descrittore di distribuzione EJB. Il valore predefinito è `enable`. È possibile impostare questo valore su `disable`.
compressionMode	È possibile modificare la modalità di compressione con la variabile d'ambiente compressionMode che si trova nel descrittore di distribuzione EJB. Il valore predefinito è `enable`. È possibile impostare questo valore su `disable`.

   Il valore predefinito è propagare gli aggiornamenti con il controllo della versione. È possibile impostare il valore in modalità di invalidazione senza il controllo della versione.

A questo punto è stata installata ed eseguita l'applicazione di esempioObjectGridPartitionCluster.

Opzioni del client applicativo per ObjectGridPartitionClusterSample

Utilizzare queste opzioni per un uso avanzato mediante l'esecuzione del file `D_ObjectGridPartitionClusterSample.ear`.

Utilizzo avanzato di esempio

Fare riferimento a "Installazione ed esecuzione dell'applicazione di esempio ObjectGridPartitionCluster" a pagina 304 per ulteriori informazioni sull'installazione e l'esecuzione del file `D_ObjectGridPartitionClusterSample.ear`.

Per un utilizzo avanzato dell'esempio, fare riferimento alla seguente guida all'utilizzo:

```
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]
```

Le variabili sono:

- HOSTNAME: Specifica il nome host del server delle applicazioni in esecuzione.
- SERVER_RMI_PORT: Specifica la porta Bootstrap del server delle applicazioni.
- LOOP: Specifica il numero di loop eseguiti dal client. Questo parametro è facoltativo. Il valore predefinito Þ 1.
- NUMBER_OF_THREADS: Specifica il numero di thread eseguiti dal client. Questo parametro è facoltativo. Il valore predefinito Þ 1.
- NUMBER_OF_STOCKS_PER_PARTITION: Specifica il numero di memorie per ogni partizione da aggiungere. Questo parametro è facoltativo. Il valore predefinito è 3.
- SECONDS_TO_WAIT_FOR_PROPAGATION: Specifica i secondi da attendere per gli oggetti ObjectGrid appena aggiunti o aggiornati prima che vengano propagati su altri server. Il valore predefinito è 2 secondi.
- NUMBER_OF_ITERATION_PER_OGKEY: Specifica il numero di iterazioni di richiamo degli oggetti in ObjectGrid in un modo gestito dal carico di lavoro. Il valore predefinito è 6. Con un numero maggiore di iterazioni specificato, è disponibile un semplice modello per gli oggetti della stessa chiave su diversi server WebSphere Application Server.

Output di esempio

L'output di questo comando ha un aspetto simile a quello riportato nel seguente esempio:

```
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: elaborazione degli argomenti della riga comandi.
WSCL0013I: inizializzazione dell'ambiente del client applicativo J2EE in corso.
WSCL0035I: inizializzazione dell'ambiente client applicativo J2EE completata.
WSCL0014I: richiamo dell'applicazione
Client class com.ibm.websphere.samples.objectgrid.partitionclust
er.client.PartitionObjectGrid
La partizione ObjectGrid di esempio ha 10 partizioni
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**********
------Operazioni di ObjectGrid: Stock Ticket=Stock000104 --------
get on partition for ticket: Stock000104->clusterdevNode02/s2
update: Stock000104->clusterdevNode02/s2
inattività per 2 secondi.....
```

```
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
------Operazioni di ObjectGrid: Stock Ticket=Stock000114 --------
get on partition for ticket: Stock000114->clusterdevNode01/s2
update: Stock000114->clusterdevNode02/s2
inattività per 2 secondi.....
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
------Operazioni di ObjectGrid: Stock Ticket=Stock000124 --------
get on partition for ticket: Stock000124->clusterdevNode02/s2
update: Stock000124->clusterdevNode01/s2
inattività per 2 secondi.....
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>
```

Creazione di un ObjectGrid e di un'applicazione dell'utilità di partizione

Aprire, modificare e installare l'applicazione di esempio per il partizionamento di ObjectGrid.

Effettuare queste operazioni per modificare, esportare ed installare il file `ObjectGridPartitionSample.ear` in un ambiente WebSphere Extended Deployment. Se non si desidera apportare delle modifiche al file di esempio, è possibile utilizzare il file `D_ObjectGridPartitionClusterSample.ear` abilitato per l'utilità di partizione e già distribuito. Se si utilizza il file `D_ObjectGridPartitionClusterSample.ear`, è possibile installare ed eseguire il file senza dover effettuare le operazioni riportate di seguito. Entrambi i file EAR (enterprise archive) si trovano nella directory `WAS_INSTALL_ROOT/installableApps`.

1. Impostare il file `ObjectGridPartitionSample.ear` nell'ambiente build, come IBM Rational Application Developer Versione 6.0.x o Application Server Toolkit Versione 6.0.x. Per ulteriori informazioni, consultare "Introduzione alla creazione di un ObjectGrid e di un'applicazione dell'utilità di partizione" a pagina 308.

2. Modificare il codice di origine nell'esempio.
3. Esportare l'applicazione ObjectGridPartitionClusterSample dall'ambiente di build come file EAR. "Esportazione del file ObjectGridPartitionClusterSample.ear in IBM Rational Application Developer" a pagina 309 per ulteriori informazioni.
4. Distribuire l'applicazione in modo da poter utilizzare l'utilità di partizione. Per ulteriori informazioni, consultare "Distribuzione del file ObjectGridPartitionClusterSample.ear per operare con l'utilità di partizione" a pagina 310.
5. Installare il file`ObjectGridPartitionClusterSample.ear` in WebSphere Extended Deployment. Il modo più comune per installare i file EAR dell'applicazione consiste nell'utilizzare la console di gestione di WebSphere Application Server. Seguire la procedura di installazione dell'applicazione enterprise nella console di gestione per installare il file `D_ObjectGridPartitionClusterSample.ear`. Non distribuire il file durante l'installazione, ma utilizzare il file predefinito. Utilizzare le impostazioni predefinite per ogni passo tranne quando viene richiesto di selezionare il percorso di installazione. In questo passo, selezionare il cluster definito invece del server predefinito server1.

A questo punto è stato installato il file `ObjectGridPartitionClusterSample.ear` in un ambiente WebSphere Extended Deployment.

Per ulteriori informazioni sulla programmazione con ObjectGrid, sull'utilità di partizione e sulle applicazioni di esempio, fare riferimento a "Esempio: ObjectGrid programmazione dell'utilità di partizione" a pagina 311.

Introduzione alla creazione di un ObjectGrid e di un'applicazione dell'utilità di partizione

Utilizzare Application Server Toolkit Versione 6.0.x o IBM Rational Application Developer Versione 6.0.x per creare di nuovo l'applicazione di esempio.

Il file `ObjectGridPartitionClusterSample.ear` nella directory `WAS_INSTALL_ROOT/installableApps` contiene tutto il codice sorgente. È possibile utilizzare Application Server Toolkit Versione 6.0.x o IBM Rational Application Developer Versione 6.0.x per creare di nuovo l'applicazione di esempio. Questa attività utilizza Rational Application Developer come esempio per ristabilire l'ambiente di creazione del file `ObjectGridPartitionClusterSample.ear`. È inoltre possibile utilizzare l'Application Server Toolkit, uno strumento di assemblaggio gratuito rilasciato con WebSphere Application Server su un CD separato.

Il file EAR (enterprise archive) abilitato per WPF, `D_ObjectGridPartitionClusterSample.ear` che si trova nella directory `WAS_INSTALL_ROOT/installableApps`, è pronto per essere installato ed eseguito.

1. Importare il file `ObjectGridPartitionClusterSample.ear` in Rational Application Developer.
   a. Avviare Rational Application Developer.
   b. **Opzionale:** Aprire la prospettiva Java 2 Platform, Enterprise Edition (J2EE) per operare sui progetti J2EE. Fare clic su **Finestra > Apri prospettiva > Altro > J2EE**.
   c. **Opzionale:** Aprire la vista Explorer progetto. Fare clic su **Finestra > Mostra vista > Explorer progetto**. Un'altra vista utile è la vista Navigator: **Finestra > Mostra vista > Navigator**.
   d. Importare il file `ObjectGridPartitionClusterSample.ear`. Fare clic su **File > Importa > File EAR**, quindi selezionare **Avanti**.

e. Selezionare il file `ObjectGridPartitionClusterSample.ear` dalla directory `WAS_INSTALL_ROOT/installableApps`.

f. **Opzionale:** Fare clic su **Nuovo** per aprire la procedura guidata Nuovo runtime server e seguire le istruzioni.

g. Nel campo Server di destinazione, selezionare il tipo di runtime del server di **WebSphere Application Server V6.0**.

h. Fare clic su **Fine**.

I progetti ObjectGridPartitionClusterSample, ObjectGridPartitionClusterSampleEJB e ObjectGridPartitionClusterSampleClient devono essere creati e visibili nella vista Project Explorer.

2. Impostare il progetto ObjectGridPartitionClusterSampleEJB.

a. Nella vista Project Explorer della prospettiva J2EE, fare clic con il pulsante destro del mouse sul progetto **ObjectGridPartitionClusterSampleEJB** tra i progetti EJB e selezionare **Proprietà**. Viene visualizzata la finestra delle proprietà.

b. Selezionare **Percorso build Java** nel pannello di sinistra, fare clic sulla scheda **Librerie** nel pannello di destra e selezionare **Aggiungi variabile**. Viene visualizzata la finestra Nuova voce della variabile classpath.

c. Fare clic su **Configura variabili** per aprire la finestra Preferenze.

d. Fare clic su **Nuovo** per aprire la finestra Nuova voce di variabile.

e. Immettere `ObjectGridPartitionCluster_JAR` nel campo **Nome** e fare clic su **File** per aprire la finestra Selezione JAR.

f. Passare alla directory`WAS_INSTALL_ROOT/lib` e selezionare **wsobjectgrid.jar**. Fare clic su **Apri** per chiudere la finestra Selezione JAR.

g. Fare clic su **OK** per chiudere la finestraNuova voce di variabile. La variabile ObjectGridPartitionCluster_JAR viene visualizzata nell'elenco di variabili classpath.

h. Fare clic su**OK** per chiudere la finestra Preferenze.

i. Selezionare la variabile**ObjectGridPartitionCluster_JAR** dall'elenco di variabili e fare clic su **OK** per chiuderela finestra Nuova voce della variabile classpath. La variabile ObjectGridPartitionCluster_JAR viene visualizzata nel pannello Librerie.

j. Ripetere questa procedura per aggiungere il file `wpf.jar` al proprio ambiente.

k. Verificare che il file `wpf.jar` e il file `wsobjectgrid.jar` siano presenti nella variabile classpath del build.

Una volta impostato l'ambiente del build, è possibile modificare il codice sorgente e applicare altre modifiche. Per ulteriori informazioni, consultare "Creazione di un ObjectGrid e di un'applicazione dell'utilità di partizione" a pagina 307.

Esportazione del file ObjectGridPartitionClusterSample.ear in IBM Rational Application Developer

Una volta apportate le modifiche al file di esempio, è possibile esportare l'applicazione ObjectGridPartitionClusterSample per creare un file EAR (enterprise archive) che è possibile installare sui server WebSphere Extended Deployment.

Per apportare le modifiche al codice sorgente, è necessario che il file `ObjectGridPartitionSample.ear` venga importato negli strumenti di sviluppo. Per ulteriori informazioni, consultare "Introduzione alla creazione di un ObjectGrid e di un'applicazione dell'utilità di partizione" a pagina 308. Prima dell'esportazione, apportare le modifiche all'applicazione di esempio.

È possibile esportare il file ObjectGridPartitionClusterSample.ear dal progetto ObjectGridPartitionClusterSample nelle applicazioni enterprise in IBM Rational Application Developer. È possibile installare il file ObjectGridPartitionClusterSample.ear esportato su qualsiasi server WebSphere Extended Deployment Versione 6.0 dopo aver distribuito l'utilità di partizione.

1. Nella vista Project Explorer della prospettiva J2EE, fare clic con il pulsante destro del mouse sull'applicazione **ObjectGridPartitionClusterSampleEJB** che si trova in Applicazioni enterprise. Fare clic su **Esporta > File EAR**. Viene visualizzata la finestra Esporta.
2. Fare clic su**Sfoglia** per aprire la finestra **Salva con nome**. Individuare la directory di output di destinazione, specificare il nome file come ObjectGridPartitionClusterSample e fare clic su **Salva**.
3. Fare clic su **Sfoglia** per aprire la finestra Salva con nome. Individuare la directory di output di destinazione, specificare il nome file ObjectGridPartitionClusterSample. Fare clic su **Salva**.

Il file ObjectGridPartitionClusterSample.ear viene creato nella directory di output di destinazione specificata.

Dopo aver distribuito il file ObjectGridPartitionClusterSample.ear per l'utilità di partizione (WPF), è possibile eseguire il file in WebSphere Extended Deployment. Per ulteriori informazioni, consultare "Distribuzione del file ObjectGridPartitionClusterSample.ear per operare con l'utilità di partizione".

Distribuzione del file ObjectGridPartitionClusterSample.ear per operare con l'utilità di partizione

Se si desidera installare il file ObjectGridPartitionClusterSample.ear su WebSphere Extended Deployment, è necessario eseguire prima un'operazione wpf-deploy sul file.

È necessario che il file ObjectGridPartitionClusterSample.ear sia già esistente. Per modificare il file esistente, fare riferimento a "Creazione di un ObjectGrid e di un'applicazione dell'utilità di partizione" a pagina 307.

Eseguire l'operazione wpf-deploy per preparare il file EAR (enterprise archive) in un ambiente WebSphere Extended Deployment.

1. Creare una directory DEST_DIR.
2. Copiare il file ObjectGridPartitionClusterSample.ear nella directory DEST_DIR. Ridenominare il file ObjectGridPartitionClusterSample.ear in old_ObjectGridPartitionClusterSample.ear.
3. Emettere il seguente comando, dove WORKING_DIR è la directory operativa per lo strumento ejbdeloy, ad esempio c:\temp.

   ```
   WAS_HOME\bin\ejbdeploy.bat|ejbdeploy.sh
   DEST_DIR\old_ObjectGridPartitionClusterSample.ear WORKING_DIR
   DEST_DIR\ObjectGridPartitionClusterSample.ear
   ```

4. Emettere il seguente comando, dove TEMP_DIR è una directory temporanea per lo strumento. Se viene specificato l'argomento -keep, le di rectory temporanee che vengono create dal programma di utilità wpfStubUtil non verranno eliminate.

   ```
   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]
   ```

Il file `ObjectGridPartitionClusterSample.ear` è quindi pronto per essere eseguito in un ambiente WebSphere Extended Deployment. Il file di esempio `D_ObjectGridPartitionClusterSample.ear` che viene rilasciato con WebSphere Extended Deployment è già distribuito. Non è quindi necessario distribuire questo file prima di installare l'applicazione se non è stato modificato il codice sorgente.

Installare il file `ObjectGridPartitionClusterSample.ear` nell'ambiente WebSphere Extended Deployment con la console di gestione. Per ulteriori informazioni, consultare "Creazione di un ObjectGrid e di un'applicazione dell'utilità di partizione" a pagina 307.

Esempio: ObjectGrid programmazione dell'utilità di partizione

In questo esempio viene dimostrato come utilizzare le funzioni combinate di ObjectGrid e dell'utilità di partizione in un ambiente Java 2 Platform, Enterprise Edition su WebSphere Extended Deployment.

Funzione

Oltre a dimostrare le funzioni di ObjectGrid e dell'utilità di partizione (WPF), questo esempio dimostra anche la propagazione e l'invalidazione del listener distribuito ObjectGrid.

Tutte le richieste di aggiornamento, inserimento e rimozione degli oggetti vengono indirizzate a server specifici su cui sono presenti delle partizioni per le chiavi di ObjectGrid corrispondenti. Le richieste al metodo get di ObjectGrid vengono gestite su tutti i server.

Questo esempio illustra anche come partizionare un ObjectGrid di elevate dimensioni in ObjectGrid più piccoli e utilizzare il metodo partitionLoadEvent per precaricare i dati in modo che l'ObjectGrid con le partizioni possa ospitare un numero illimitato di oggetti.

Panoramica

Il file `ObjectGridPartitionClusterSampler.ear` crea unoggetto stock che illustra il modo in cui ObjectGrid e l'utilità di partizione funzionano insieme. L'oggetto stock contiene le seguenti proprietà:

- etichetta
- azienda
- serialNumber
- descrizione
- lastTransaction
- prezzo

La proprietà lastTransaction rappresenta l'ora in cui la memoria è stata modificata. Utilizzare la proprietà lastTransaction per indicare l'aggiornamento degli oggetti nell'ObjectGrid di JVM (Java virtual machines) differenti.

Nell'esempio riportato, viene creata l'istanza di ObjectGrid nel metodo setContext di Enterprise JavaBeans (EJB) con la classe ObjectGridFactory.

Definire una serie di partizioni basate su hash. Il valore predefinito è 10 partizioni, ma è possibile modificare questo numero. Inserire le etichette di memoria su queste partizioni utilizzando il file `SampleUtility.java`. Ogni partizione può contenere più coppie chiave-valore di ObjectGrid.

Nell'esempio viene riportato il modo in cui le richieste di inserimento, aggiornamento e rimozione di ObjectGrid vengono indirizzate a un determinato server e come le richieste del metodo get di ObjectGrid vengono indirizzate a un determinato server per le chiavi o a un server in un cluster. Nell'esempio vengono quindi confrontati i valori dell'oggetto per una chiave su server differenti in seguito alla modifica di un valore a causa di un'operazione di aggiornamento, inserimento o rimozione per la chiave su un determinato server.

Posizione

Utilizzare questo esempio in un ambiente cluster in cui ciascun server può ospitare più partizioni e dove ciascuna partizione può contenere molti oggetti con chiavi differenti.

Due file di esempio del cluster delle partizioni di ObjectGrid sono presenti nella directory `<root_install>\installableApps\`:

- Il file `ObjectGridPartitionClusterSampler.ear` contiene il codice sorgente. Per visualizzare il codice sorgente, espandere il file EAR sul file system o importare il codice in un ambiente di sviluppo. Per ulteriori informazioni, consultare "Creazione di un ObjectGrid e di un'applicazione dell'utilità di partizione" a pagina 307.
- Il file `D_ObjectGridPartitionClusterSample.ear` è già distribuito per l'utilità di partizione. Consultare il file readme e leggere le istruzioni riportate per eseguire questo file.

Spiegazione

Nelle seguenti sezioni è riportata la spiegazione relativa all'applicazione di esempio del cluster delle partizioni di ObjectGrid:

Interfaccia EJB per le operazioni di ObjectGrid

In questa sezione viene descritta l'interfaccia EJB (Enterprise JavaBeans) delle operazioni di ObjectGrid che esegue le operazioni di richiamo, richiamo da un server con partizioni, inserimento, aggiornamento e rimozioni.

Funzione

L'interfaccia EJB (Enterprise JavaBeans) delle operazioni di ObjectGrid esegue le operazioni di richiamo, richiamo da un server con partizioni, inserimento, aggiornamento e rimozioni. Il metodo di richiamo da un server con partizioni viene indirizzato a una partizione che corrisponde alla chiave richiesta. Il metodo get viene indirizzato a una strategia gestita del carico di lavoro su qualsiasi server.

Interfaccia PFClusterObjectGridEJB

Di seguito è riportato il contenuto dell'interfaccia PFClusterObjectGridEJB:

```
/**
* Interfaccia remota per Enterprise Bean: PFClusterObjectGridEJB
*/
public interface PFClusterObjectGridEJB extends javax.ejb.EJBObject {
public String PARTITION_PREFIX = "ObjectGridHashPartition";
/**
 * Richiama tutte le partizioni
*
 * @return Array di stringhe
 * @throws java.rmi.RemoteException
*/
 public String [] getAllPartitions() throws java.rmi.RemoteException;
/**
 * Richiama dove si trova la partizione
*
* @param partition
* @return String
 * @throws java.rmi.RemoteException
*/
 public String getServer(String partition)
 throws java.rmi.RemoteException;
/**
 * Richiama l'oggetto Stock e le relative informazioni sul server
 * (ServerIDResult) per un'etichetta di memoria
 * da qualsiasi server in un cluster (con una gestione del carico di lavoro)
*
* @param ticket
* @return
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult getStock(String ticket)
 throws java.rmi.RemoteException;
/**
 * Richiama l'oggetto Stock e le relative informazioni sul server
 * con partizioni per un'etichetta di memoria
 * dalla partizione per cui questa etichetta ha un valore hash
*
* @param ticket
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult getStockOnPartitionedServer(String ticket)
 throws java.rmi.RemoteException;
/**
 * Aggiorna la memoria su un determinato server su cui la partizione è
 * attiva per questa chiave dell'etichetta di memoria.
*
* @param stock
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult updateStock(Stock stock)
 throws java.rmi.RemoteException;
/**
 * Rimuove la memoria su un determinato server su cui la partizione è
 * attiva per questa chiave dell'etichetta di memoria.
*
* @param ticket
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult removeStock(String ticket)
 throws java.rmi.RemoteException;
```

```
/**
 * Inserisce la memoria su un determinato server su cui la partizione è
 * attiva per questa chiave dell'etichetta di memoria.
*
* @param stock
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
*/
 public ServerIDResult insertStock(Stock stock)
 throws java.rmi.RemoteException;
/**
 * Richiama i dati da tutti i server e confronta i valori
*
* @param server
 * @return ServerObjectGridVerification
 * @throws java.rmi.RemoteException
*/
 public ServerObjectGridVerification verifyObjectGrid(String server)
 throws java.rmi.RemoteException;
}
```

Classe PartitionKey

La classe PartitionKey controlla il comportamento del routing basato sul contesto dell'utilità di partizione.

Il seguente codice illustra la classe della chiave di partizione di esempio. Quando questo metodo restituisce un valore `non null`, viene indirizzato al router dell'utilità di partizione (WPF). Quando il metodo restituisce `null`, viene invece inoltrato al router di gestione del carico di lavoro (workload management, WLM).

```
/**
* PartitionKey per PSSB (Partitioned Stateless Session Bean) WPFKeyBasedPartition
*
*/
public class PFClusterObjectGridEJB_PartitionKey {
/**
 * Numero di partizioni
*
 * Il valore predefinito è 10.
*
*/
 static int numOfPartitions=10;
/**
 * Solo una volta su getPartitionNumbers
*/
 static boolean getNumOfPartitions=true;
/**
 * Richiama il numero di partizioni
*
*/
 static void getPartitionNumbers(){
 // richiama solo una volta
  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;
}
}
}
/**
* Restituisce la chiave della partizione
*
* @param partition
* @return String
*/
public static String getStock(String key) {
return null;
}
/**
* Restituisce la chiave della partizione
*
* @param key
* @return String
*/
public static String getServer(String key) {
 return key;
}
/**
* Richiama i dati ObjectGrid da un server con partizioni su cui
* si sono verificate delle modifiche ai dati (con qualità e integrità elevata)
*
* @param ticket
* @return hashcode of stock ticket
*/
public static String getStockOnPartitionedServer(String ticket) {
if (ticket==null){
return null;
}
getPartitionNumbers();
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Restituisce la chiave della partizione
*
* @param stock
* @return hashcode of stock ticket
*/
public static String updateStock(Stock stock) {
getPartitionNumbers();
 String ticket=null;
 if (stock!=null){
  ticket=stock.getTicket();
}
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Restituisce la chiave della partizione
*
* @param stock
* @return hashcode of stock ticket
```

```
*/
public static String insertStock(Stock stock) {
getPartitionNumbers();
 String ticket=null;
 if (stock!=null){
  ticket=stock.getTicket();
}
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Restituisce la chiave della partizione
*
* @param server
* @return String
*/
public static String verifyObjectGrid(String server) {
 return server;
}
/**
* Restituisce la chiave della partizione
*
* @param stock
* @return hashcode of stock ticket
*/
public static String removeStock(String ticket) {
if (ticket==null){
return null;
}
getPartitionNumbers();
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Restituisce la chiave della partizione
*
* @param partition
* @return
*/
public static String getAllPartitions() {
return null;
}
}
```

Ogni metodo remoto deve corrispondere a un metodo che restituisce una stringa valida o un valore null.

Classe SampleUtility e mappatura delle partizioni

Utilizzare il file `SampleUtility.java` per modificare chiavi, memoria, etichette, hash e partizioni. È possibile utilizzare questo file anche per associare le chiavi di ObjectGrid alle partizioni. È possibile sviluppare allo stesso modo una classe per il programma di utilità per associare le chiavi di ObjectGrid alle partizioni che rispondono alle proprie necessità aziendali. Per utilizzare l'utilità di partizione con ObjectGrid, è necessario associare diverse chiavi su diverse partizioni.

Classe SampleUtility

Di seguito è riportata la classe del programma di utilità per l'esempio ObjectGridPartitionCluster:

```
/**
* Classe di utilità per l'esempio ObjectGridPartitionCluster
*
*
*/
public class SampleUtility {
/**
```

```
 * Il contenitore per le partizioni di registrazione.
*/
 static Map serverPartitions= new HashMap();
/**
 * Il prefisso del nome della partizione
*/
public static String PARTITION_PREFIX = "ObjectGridHashPartition";
/**
 * Il prefisso del nome di memoria
*/
public static String STOCK_PREFIX="Stock";
/**
 * Richiama la parte numerica del nome della partizione
*
* @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;
}
/**
 * Richiama la parte numerica dell'etichetta di memoria
*
* @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;
}
/**
 * L'etichetta della memoria hash su una determinata base hash.
*
* @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;
}
/**
 * La chiave di memoria hash su una partizione
*
* @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;
```

```
}
/**
 * Record del server/partizione
*
* @param server
* @param partition
*/
 public static void addServer(String server, String partition){
  serverPartitions.put(server, partition);
}
/**
 * Rimuove server/partizione
*
* @param server
*/
 public static void removeServer(String server){
  serverPartitions.remove(server);
}
/**
 * Richiama tutti i server su cui le partizioni sono attive.
*
 * @return Iterator - String
*/
 public static Iterator getAllServer(){
  return serverPartitions.values().iterator();
}
}
```

È necessario utilizzare la stessa base hash globale e analizzare la variabile su hash nella base hash. Considerare il seguente esempio:

```
myKey.hashCode % hashBase
```

È necessario analizzare myKey come variabile hash e conservare lo stesso valore hash su server differenti. Nell'esempio precedente, viene ricercata la stessa variabile dall'ambiente Java. Non è possibile utilizzare `key1 % 100`, ma è possibile utilizzare `key2 % 90`.

Creazione di ObjectGrid nel metodo setContext del bean enterprise

Creare l'istanza ObjectGrid nel metodo setContext del bean enterprise bean come nel file `PFClusterObjectGridEJBBean.java` e richiamare i dati di precaricamento.

```
/**
* setSessionContext
*
* con istanza ObjectGrid
*/
public void setSessionContext(javax.ejb.SessionContext ctx) {
mySessionCtx = ctx;
try {
InitialContext ic = new InitialContext();
// richiamare PartitionManager
  ivManager = (PartitionManager)
ic.lookup("java:comp/websphere/wpf/PartitionManager");
  // richiama la configurazione enableDistribution
  boolean enableDistribution = ((Boolean)
ic.lookup("java:comp/env/enableDistribution")).booleanValue();
System.out.println("***** enableDistribution="+ enableDistribution);
  // richiama la configurazione 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;
}
  // richiama la configurazione 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;
}
  // richiama la configurazione 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 il precaricamento è abilitato
  bPreload = ((Boolean)
ic.lookup("java:comp/env/preload")).booleanValue();
System.out.println("***** enablePreload="+ bPreload);
  // se remove è abilitato
  bRemove = ((Boolean)
ic.lookup("java:comp/env/remove")).booleanValue();
System.out.println("***** enableRemove="+ bRemove);
  // se il programma di caricamento è abilitato
  boolean bLoader = ((Boolean)
ic.lookup("java:comp/env/loader")).booleanValue();
System.out.println("***** enableLoader="+ bLoader);
  // richiama il nome e il percorso del file
  String filePathandName = (String)
ic.lookup("java:comp/env/filePathandName");
System.out.println("***** fileName="+ filePathandName);
  // richiama l'istanza 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 singleton ObjectGridFactory

Un'istanza ObjectGrid viene creata con un valore predefinito personalizzato che memorizza nella cache l'istanza ObjectGrid con le impostazioni personalizzate.

Di seguito è riportato un esempio di come creare un'istanza ObjectGrid in maniera programmatica, come viene impostato l'oggetto ObjectGridTransformer, come viene configurato il listener degli eventi di propagazione e di come questo listener viene impostato per l'istanza ObjectGrid. Per eseguire questa configurazione, è possibile utilizzare anche un file XML.

```
/**
*
* Creare l'istanza ObjectGrid e configurarla.
*
*
*/
public class ObjectGridFactory {
/**
 * Nome ObjectGrid
*/
static String ogName="WPFObjectGridSample";
/**
 * Istanza ObjectGrid
*/
 static ObjectGrid og=null;
/**
 * Sessione ObjectGrid
*/
 static Session ogSession=null;
/**
 * Nome mappa
*/
static String mapName="SampleStocks";
/**
 * Cache ObjectGrid
*/
 static Map ogCache= new HashMap();
/**
 * Richiama l'istanza 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);
}
   // Define BackingMap and set the 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;
}
}
```

Precaricamento della partizione ObjectGrid

In questa sezione viene descritto come precaricare un'istanza di ObjectGrid.

Utilizzare il metodo partitionLoadEvent per caricare gli oggetti relativi a questa p0artizione solo se la partizione è attivata. Caricando gli oggetti mentre la partizione è attivata, viene eseguita una partizione su ObjectGrid in modo che possa gestire un numero elevato di oggetti.

```
/**
* Questo viene richiamato quando una determinata partizione viene assegnata
  a questo processo server
* @param partitionName
* @return
*/
public boolean partitionLoadEvent(String partitionName) {
// precaricamento dei dati
 preloadDataForPartition(partitionName);
logger.logp(
Level.FINER,
CLASS_NAME,
"partitionLoadEvent",
"Loading "+ partitionName );
return true;
}
/**
*
* preload data
*
* @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);
}
}
}
}
```

Potrebbe essere necessario disabilitare gli aggiornamenti distribuiti se si utilizza il precaricamento partizionato di ObjectGrid. La versione corrente degli aggiornamenti distribuiti non può essere partizionata. Il routing basato sul contesto dell'utilità di partizionamento (WPF) trova i dati corretti sulla partizione corretta.

Configurazione di ObjectGrid per operazioni con bean CMP

Con WebSphere Application Server Versione 6.0.2 e successive, è possibile utilizzare bean CMP (container-managed persistence) con un prodotto della cache esterna.

Utilizzare questa attività per utilizzare bean CMP, sfruttando l'ObjectGrid come cache esterna invece che come cache integrata. Questa funzione è fornita dal motore di permanenza in WebSphere Application Server.

1. Definire gli argomenti JVM per definire l'adattatore CacheFactoryManager e il percorso del file di configurazione XML di ObjectGrid. CacheFactoryManager è un adattatore tra il motore di permanenza e ObjectGrid.
   a. Fare clic su **Server > Server delle applicazioni >** *nome_server* **> Gestione processo e Java > Definizione processo > Java Virtual Machine > Argomenti JVM generici**.
   b. Aggiungere le seguenti proprietà:
      - `-Dcom.ibm.ws.pmcache.manager=com.ibm.ws.objectgrid.adapter.pm.CacheFactoryManager`
      - `-Dcom.ibm.ws.pmcache.config=file:/d:/temp/objectGrid.xml`

      La proprietà `-Dcom.ibm.ws.pmcache.config` specifica il file di configurazione per ObjectGrid. Il valore è un URL per il file di configurazione ObjectGrid.
2. Configurare il file di configurazione XML di ObjectGrid. La configurazione è riportata nel file `objectGrid.xml`. Considerare il seguente esempio. Le informazioni sull'applicazione e sul modulo sono necessarie per l'applicazione Java 2 Platform, Enterprise Edition (J2EE). Le informazioni sono riflesse nel file `objectGrid.xml`. Un'applicazione Accounts ha tre Enterprise JavaBeans CMP: Savings, Checkin e MoneyMarket. Questi Enterprise JavaBeans sono contenuti nel modulo PersonalBankingEJB. Il nome di visualizzazione è *Accounts* e *PersonalBankingEJB* è il modulo EJB del descrittore di distribuzione dell'applicazione. Savings, Checkin e MoneyMarket sono i nomi così come specificati nell'elemento ejb-name del descrittore di distribuzione Enterprise Java Bean per i bean di entità CMP. Di seguito è riportato un frammento di codice per questa configurazione:

```
<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>
```

Il file `PersonalBankingEJB.jar` è specificato all'interno delle tag EJB del descrittore di distribuzione dell'applicazione, come riportato nel seguente esempio:

```
<module id="module_1">
<ejb>PersonalBankingEJB.jar</ejb>
</module>
```

3. Abilitare l'impostazione del gestore permanenza LifeTimeInCache per ogni bean all'interno dell'applicazione per utilizzare la cache esterna. ObjectGrid richiede l'abilitazione di questa impostazione nel descrittore di distribuzione, ma ignora l'impostazione LifeTimeInCache. La configurazione di ObjectGrid ha la precedenza.
4. Configurare esplicitamente unbackingMap per eliminare gli oggetti dalla cache. Di seguito è riportato un frammento di codice XML per il file `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>
```

La proprietà lifeTime controlla il programma di eliminazione.

Fare riferimento a Capitolo 10, "Integrazione di ObjectGrid con WebSphere Application Server", a pagina 287 per ulteriori informazioni su come utilizzare ObjectGrid con WebSphere Application Server.

Capitolo 11. Procedure ottimali delle prestazioni di ObjectGrid

È possibile migliorare le prestazioni di una mappa di ObjectGrid mediante le seguenti procedure ottimali. Tali procedure sono implementate soltanto nel contesto dell'applicazione e della relativa architettura.

Ogni applicazione ed ambiente utilizza una soluzione differente per le prestazioni. ObjectGrid fornisce personalizzazioni integrate per migliorare le prestazioni è queste possono essere migliorate anche all'interno dell'architettura. Le seguenti aree mostrano miglioramenti alle prestazioni:

- "Procedure ottimali delle prestazioni di blocco"

  Scelta delle diverse strategie di blocco che influenzano le prestazioni delle applicazioni.
- "Procedure ottimali per il metodo copyMode" a pagina 326

  Scelta tra le diverse modalità di copia che possono essere utilizzate per modificare il modo in cui ObjectGrid mantiene e copia le voci.
- "Procedure ottimali per le interfacce ObjectTransformer" a pagina 330

  Utilizzo dell'interfaccia ObjectTransformer per consentire richiamate all'applicazione per fornire implementazioni personalizzate di operazioni comuni e costose come la serializzazione o la copia completa di un oggetto.
- "Procedure ottimali delle prestazioni del programma di esclusione dei plug-in" a pagina 332

  Scelta tra le strategie di esclusione LFU (least frequently used) e LRU (least recently used).
- "Procedure ottimali per il programma di esclusione predefinito" a pagina 334

  Proprietà per il programma di esclusione TTL (time to live) predefinito, il programma di esclusione predefinito creato con ogni backingMap.

Procedure ottimali delle prestazioni di blocco

Le strategie di blocco possono influenzare le prestazioni delle applicazioni.

Per maggiori dettagli sulle seguenti strategie di blocco, fare riferimento alla sezione "Blocco" a pagina 127.

Strategia di blocco pessimistico

È possibile utilizzare la strategia di blocco pessimistico per leggere e scrivere mappe nei punti in cui le chiavi collidono di frequente. Questa strategia ha un impatto notevole sulle prestazioni.

Strategia di blocco ottimistico

Un blocco ottimistico è la configurazione predefinita. Questa strategia migliora le prestazioni e aumenta la scalabilità. Utilizzare questa strategia quando le applicazioni possono tollerare alcuni errori di aggiornamento ottimistici. Questa strategia funziona al meglio per le applicazioni di lettura, aggiornate poco frequentemente.

Strategia di blocco None

Utilizzare la strategia di blocco none per le applicazioni di sola lettura. Questa strategia non utilizza alcun blocco. Pertanto, offre i livelli più elevati di simultaneità, prestazioni e scalabilità.

Procedure ottimali per il metodo copyMode

ObjectGrid crea una copia del valore in base all'impostazione di CopyMode. È possibile utilizzare il metodo setCopyMode(CopyMode, valueInterfaceClass) dell'API BackingMap per impostare la modalità di copia su uno dei seguenti campi statici definiti nella classe com.ibm.websphere.objectgrid.CopyMode.

Quando un'applicazione utilizza l'interfaccia ObjectMap per ottenere un riferimento a un valore della mappa, è preferibile utilizzare tale riferimento solo all'interno della transazione di ObjectGrid che ha ottenuto il riferimento. L'utilizzo del riferimento in una transazione ObjectGrid differente provoca degli errori. Ad esempio, se si utilizza una strategia di blocco pessimistico per il BackingMap, una chiamata al metodo get o getForUpdate acquisisce un blocco a S (shared, condiviso) o U (update, aggiornamento) rispettivamente. Il metodo get restituisce il riferimento al valore e il blocco ottenuto viene rilasciato una volta completata la transazione. La transazione deve richiamare il metodo get o getForUpdate per bloccare la voce della mappa in una transazione differente. Ogni transazione deve ottenere il proprio riferimento al valore richiamando il metodo get o getForUpdate invece che riutilizzando lo stesso riferimento in più transazioni.

Utilizzare le seguenti informazioni per scegliere la modalità di copia in base alle seguenti informazioni:

Modalità COPY_ON_READ_AND_COMMIT

La modalità COPY_ON_READ_AND_COMMIT è la modalità predefinita. Quando viene utilizzata questa modalità l'argomento valueInterfaceClass viene ignorato. Questa modalità garantisce che un'applicazione non abbia alcun riferimento al valore che si trova nel BackingMap e che invece utilizzi sempre una copia del valore. La modalità COPY_AND_COMMIT garantisce quindi che un'applicazione non danneggi mai inavvertitamente i dati memorizzati nella cache in BackingMap. Quando la transazione di un'applicazione richiama un metodo ObjectMap.get per una determinata chiave e questo è il primo accesso della voce ObjectMap per tale chiave, allora viene restituita una copia del valore. Una volta eseguito il commit della transazione, tutte le modifiche apportate dall'applicazione vengono copiate su BackingMap in modo da essere certi che l'applicazione non abbia alcun riferimento al valore salvato in BackingMap.

Modalità COPY_ON_READ

La modalità COPY_ON_READ migliora le prestazioni rispetto alla modalità COPY_ON_READ_AND_COMMIT eliminando la copia che si verifica quando viene eseguito il commit di una transazione. Quando viene utilizzata questa modalità l'argomento valueInterfaceClass viene ignorato. Per garantire l'integrità dei dati BackingMap, l'applicazione si impegna a eliminare qualsiasi riferimento che ha alla voce una volta eseguito il commit della transazione. Questa modalità fa sì che il metodo ObjectMap.get restituisca una copia del valore invece di un riferimento al valore stesso, il che garantisce che le modifiche apportate dall'applicazione al valore non influenzino il valore di BackingMap fino a che viene eseguito il commit della transazione. Tuttavia, se viene eseguito il commit della transazione, la copia

delle modifiche non viene eseguita. Invece il riferimento alla copia restituito dal metodo ObjectMap.get viene memorizzato nel BackingMap. L'applicazione elimina quindi tutti i riferimenti alle voci della mappa una volta completato il commit della transazione. Se l'applicazione non riesce a eseguire questa operazione, è possibile che i dati memorizzati nella cache di BackingMap vengano danneggiati. Se un'applicazione utilizza questa modalità e riporta dei problemi, passare alla modalità COPY_ON_READ_AND_COMMIT per verificare che il problema esista ancora. Se il problema non si ripropone, allora l'applicazione non riesce ad eliminare tutti i riferimenti una volta completato il commit della transazione.

Modalità COPY_ON_WRITE

La modalità COPY_ON_WRITE migliora le prestazioni rispetto alla modalità COPY_ON_READ_AND_COMMIT eliminando la copia che viene creata quando viene richiamato il metodo ObjectMap.get per la prima volta da una transazione per una determinata chiave. Per contro, il metodo ObjectMap.get restituisce un proxy per il valore invece che un riferimento diretto al valore. Il proxy garantisce che non venga eseguita alcuna copia del valore a meno che l'applicazione non richiami il metodo set sulla classe specificata come attributo valueInterfaceClass. Questo proxy fornisce un'implementazione *copy on write*. Quando viene eseguito il commit di una transazione, il BackingMap esamina il proxy per determinare se è stata eseguita una copia come risultato di un richiamo al metodo set. Se la copia è presente, allora il riferimento alla copia viene memorizzata nel BackingMap. Il vantaggio dell'utilizzo di questa modalità sta nel fatto che un valore non viene mai copiato in fase di lettura o di commit se la transazione non richiama un metodo set per modificare il valore.

Le modalità COPY_ON_READ_AND_COMMIT e COPY_ON_READ eseguono entrambe una copia completa quando un valore viene richiamato dall'ObjectMap. Se un'applicazione aggiorna soltanto alcuni dei valori richiamati in una transazione, allora questa modalità non è ottimale. La modalità COPY_ON_WRITE supporta questo comportamento in maniera efficiente ma richiede che l'applicazione utilizzi un modello semplice. Gli oggetti valore sono richiesti per supportare un'interfaccia. L'applicazione deve utilizzare i metodi su questa interfaccia quando interagisce con il valore in una sessione ObjectGrid. In questo caso, l'ObjectGrid crea i proxy per i valori restituiti all'applicazione. Il proxy ha un riferimento che deve essere un valore reale. Se l'applicazione effettua soltanto letture, queste vengono eseguite sempre rispetto alla copia reale. Se l'applicazione modifica un attributo sull'oggetto, il proxy effettua una copia dell'oggetto reale e apporta quindi la modifica alla copia. Il proxy quindi utilizzerà la copia da quel momento in poi. Ciò consente di evitare le operazioni di copia per gli oggetti che vengono soltanto letti da un'applicazione. Tutte le operazioni di modifica devono iniziare con il prefisso set. Gli EJB (Enterprise JavaBeans) di solito sono codificati in modo da utilizzare questo stile di denominazione dei metodi che modificano gli attributi degli oggetti. È necessario seguire questa convenzione. Tutti gli oggetti modificati vengono copiati nel momento stesso in cui vengono modificati dall'applicazione. Questo è lo scenario di lettura e scrittura più efficiente supportato da ObjectGrid. È possibile configurare una mappa per utilizzare la modalitàCOPY_ON_WRITE come riportato di seguito. In questo esempio, l'applicazione memorizza gli oggetti Person a cui è stata associata una chiave utilizzando il nome nella mappa. L'oggetto Person ha un aspetto simile al seguente frammento di codice:

```
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;
}
}
```

L'applicazione utilizza l'interfaccia IPerson solo quando interagisce con i valori che sono stati richiamanti da un ObjectMap. Modificare l'oggetto in modo da utilizzare un'interfaccia come nel seguente esempio.

```
interface IPerson
{
 void setName(String n);
 String getName();
 void setAge(int a);
 int getAge();
}
// Modifica l'oggetto Person per implementare l'interfaccia IPerson
class Person implements IPerson
{
...
}
```

L'applicazione deve quindi configurare il BackingMap in modo che utilizzi la modalità COPY_ON_WRITE, come nel seguente esempio:

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// utilizza COPY_ON_WRITE per questa mappa con
// IPerson come classe valueProxyInfo
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// L'applicazione deve quindi utilizzare il seguente
// modello quando utilizza la mappa PERSON.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// L'applicazione utilizza il valore restituito a IPerson a non Person
IPerson p = (IPerson)person.get("Billy");
p.setAge(p.getAge()+1);
...
// Crea un nuovo oggetto Person e lo aggiunge alla mappa
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// Il seguente frammento non funziona. Esso provoca un'eccezione ClassCastException
sess.begin();
// L'errore qui è che viene utilizzato Person invece che
// IPerson
Person a = (Person)person.get("Bobby");
sess.commit();
```

Nella prima sezione è riportata l'applicazione che richiama un valore denominato Billy nella mappa. L'applicazione utilizza il valore restituito all'oggetto IPerson e non l'oggetto Person in quanto il proxy che viene restituito implementa due interfacce:

- L'interfaccia specificata nella chiamata al metodo BackingMap.setCopyMode
- L'interfaccia com.ibm.websphere.objectgrid.ValueProxyInfo

È possibile utilizzare il proxy con entrambi i tipi. L'ultima parte del frammento di codice precedente dimostra cosa non è consentito in modalità COPY_ON_WRITE. L'applicazione richiama il record Bobby e prova a utilizzarlo con un oggetto Person. Questa azione non riesce e viene restituita un'eccezione di utilizzo della classe in quanto il proxy restituito non è un oggetto Person. Il proxy restituito implementa l'oggetto IPerson e ValueProxyInfo.

Interfaccia ValueProxyInfo e supporto per gli aggiornamenti parziali

Questa interfaccia consente a un'applicazione di richiamare il valore di sola lettura di cui è stato eseguito il commit e a cui fa riferimento il proxy o la serie di attributi che sono stati modificati durante questa transazione.

```
public interface ValueProxyInfo
{
List /**/ ibmGetDirtyAttributes();
Object ibmGetRealValue();
}
```

Il metodo ibmGetRealValue restituisce una copia di sola lettura dell'oggetto. L'applicazione non deve modificare questo valore. Il metodo ibmGetDirtyAttributes restituisce un elenco di stringhe che rappresentano gli attributi che sono stati modificati dall'applicazione durante questa transazione. Il caso principale di utilizzo per ibmGetDirtyAttributes è un programma di caricamento basato su JDBC (Java database connectivity) o CMP. Solo gli attributi riportati nell'elenco devono essere aggiornati sull'istruzione SQL o sull'oggetto mappato alla tabella, che porta a un SQL più efficiente generato dal programma di caricamento. Quando viene eseguito il commit di una transazione copy on write e un programma di caricamento è collegato, allora questo può utilizzare i valori degli oggetti modificati sull'interfaccia ValueProxyInfo per ottenere queste informazioni.

Gestione del metodo equals quando si utilizzano la modalità COPY_ON_WRITE o i proxy

Ad esempio, il seguente codice crea un oggetto Person e lo inserisce su un ObjectMap. Successivamente, lo stesso viene richiamato utilizzando il metodo ObjectMap.get. Il valore viene quindi utilizzato sull'interfaccia. Se il valore viene utilizzato sull'interfaccia Person, si verifica un'eccezione ClassCastException in quanto il valore restituito è un proxy che implementa l'interfaccia IPerson e non è un oggetto Person. Il controllo di uguaglianza non riesce se si utilizza l'operazione == in quanto non sono lo stesso oggetto.

```
session.begin();
// nuovo oggetto Person
Person p = new Person(...);
personMap.insert(p.getName, p);
// lo richiama di nuovo, in modo da utilizzare l'interfaccia
IPerson p2 = personMap.get(p.getName());
if(p2 == p)
{
// sono lo stesso
}
```

```
else
{
// non sono lo stesso
}
```

Un'altra considerazione da fare sta nel momento in cui è necessario sovrascrivere il metodo equals. Come illustrato nel seguente frammento di codice, il metodo equals deve verificare che l'argomento sia un oggetto che implementa l'interfaccia IPerson e l'argomento venga utilizzato in modo che sia un IPerson. Poiché l'argomento potrebbe essere un proxy che implementa l'interfaccia IPerson, è necessario utilizzare i metodi getAge e getName quando si confrontano le variabili delle istanze per l'uguaglianza.

```
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;
}
```

Modalità NO_COPY

La modalità NO_COPY consente a un'applicazione di garantire che il valore ottenuto utilizzando il metodo `ObjectMap.get` non venga modificato, ottenendo in questo modo prestazioni elevate. Quando viene utilizzata questa modalità l'argomento valueInterfaceClass viene ignorato. Se viene utilizzata questa modalità, il valore non viene copiato. Se l'applicazione modifica i valori, allora i dati BackingMap risultano danneggiati. La modalità NO_COPY è particolarmente utile per mappe di sola lettura in cui i dati non vengono modificati dall'applicazione. Se l'applicazione utilizza questa modalità e riporta dei problemi, passare alla modalità COPY_ON_READ_AND_COMMIT per verificare che il problema esista ancora. Se il problema non si ripropone, allora l'applicazione sta modificando il valore restituito dal metodo ObjectMap.get, sia durante la transazione stessa o una volta eseguito il commit.

Procedure ottimali per le interfacce ObjectTransformer

ObjectTransformer utilizza le richiamate all'applicazione per fornire implementazioni personalizzate di operazioni comuni e costose come la serializzazione o la copia completa di un oggetto.

Per maggiori dettagli sull'interfaccia ObjectTransformer, fare riferimento alla sezione "Plug-in ObjectTransformer" a pagina 209. Da un punto di vista delle prestazioni e dalle informazioni riportate sul metodo CopyMode nella sezione "Procedure ottimali per il metodo copyMode" a pagina 326, è chiaro che ObjectGrid copia i valori per tutti i casi tranne in modalità NO_COPY. Il meccanismo di copia predefinito utilizzato in ObjectGrid è la serializzazione, che è un'operazione molto costosa a livello di risorse. In questo caso, è possibile utilizzare l'interfaccia ObjectTransformer. Tale interfaccia utilizza le richiamate all'applicazione per fornire un'implementazione personalizzata di operazioni comuni e costose, quali la serializzazione o la copia completa degli oggetti.

Un'applicazione può fornire un'implementazione dell'interfaccia ObjectTransformer a una mappa. ObjectGrid quindi delega ai metodi su questo oggetto e si basa

sull'applicazione per fornire una versione ottimizzata di ogni metodo nell'interfaccia. Di seguito è riportata l'interfaccia ObjectTransformer:

```
public interface ObjectTransformer
{
 void serializeKey(Object key, ObjectOutputStream stream)
 throws IOException;
 void serializeValue(Object value, ObjectOutputStream stream)
 throws IOException;
 Object inflateKey(ObjectInputStream stream)
 throws IOException, ClassNotFoundException;
 Object inflateValue(ObjectInputStream stream)
 throws IOException, ClassNotFoundException;
 Object copyValue(Object value);
 Object copyKey(Object key);
}
```

È possibile associare un'interfaccia ObjectTransformer con un BackingMap utilizzando il seguente codice di esempio:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ottimizzazione dell'aumento e della serializzazione degli oggetti

La serializzazione è di solito il problema principale legato alle prestazioni di ObjectGrid. ObjectGrid utilizza il meccanismo di serializzazione predefinito se un plug-in di ObjectTransformer non è fornito dall'applicazione. Un'applicazione può fornire le implementazioni di Serializable readObject e writeObject oppure può far sì che gli oggetti implementino l'itnerfaccia Externalizable, che è circa 10 volte più veloce. Se gli oggetti nella mappa non possono essere modificati, allora l'applicazione può associare un ObjectTransformer all'ObjectMap. I metodi serialize e inflate sono forniti per consentire all'applicazione di fornire un codice personalizzato per ottimizzare tali operazioni visto l'impatto sulle prestazioni del sistema. Il metodo serialize serializza l'oggetto e fornisce un flusso. Esso quindi serializza il metodo sul flusso fornito. Il metodo inflate fornisce il flusso di input e aspetta che l'applicazione crei l'oggetto, aumentandolo utilizzando i dati nel flusso e quindi restituendolo. Le implementazioni dei metodi serialize e inflate devono eseguire il mirroring a vicenda.

Ottimizzazione delle operazioni di copia

Una volta che un'applicazione riceve un oggetto da un ObjectMap allora l'ObjectGrid esegue una copia sul valore in mod da essere certi che la copia nella mappa BaseMap sia protetta. L'applicazione può quindi modificare il valore dell'oggetto. Quando viene eseguito il commit della transazione, la copia del valore dell'oggetto nella mappa BaseMap viene aggiornata con il nuovo valore modificato e l'applicazione non utilizzerà più tale valore da quel momento in poi. È possibile quindi copiare di nuovo l'oggetto in fase di commit in modo da creare una copia privata, ma in questo caso le prestazioni risultano ridotte in seguito all'utilizzo del valore in seguito al commit. Il meccanismo di copia predefinito degli oggetti prova a utilizzare un clone oppure i metodi serialize e inflate per generare una copia. La coppia di metodi serialize e inflate rappresenta lo scenario della peggiore eventualità. Se il profilo rivela che serialize e inflate rappresentano un problema per l'applicazione, fornire un plug-in ObjectTransformer e implementare i metodi `copyValue` e `copyKey` utilizzando una copia dell'oggetto più efficiente.

Procedure ottimali delle prestazioni del programma di esclusione dei plug-in

Se si utilizzano i programmi di esclusione dei plug-in, questi non saranno attivi fino a che non vengono creati e fino a che si indica loro quale mappa di backup utilizzare. Utilizzare i suggerimenti relativi alle prestazioni e alle procedure ottimali da utilizzare per i programmi di esclusione LFU (least frequently used) e LRU (least recently used).

Programma di esclusione LFU (least frequently used)

Il concetto di programma di esclusione LFU si basa sulla rimozione dalla mappa delle voci che non vengono utilizzate frequentemente. Le voci della mappa sono divise su una quantità impostata di valori heap binari. Quando l'utilizzo di una determinata voce della cache aumenta, questa sale in una posizione più in alto dell'ordine dell'heap. Quando il programma di esclusione prova a eseguire delle eclusioni, questo rimuove soltanto le voci nella cache che si trovano al di sotto di un determinato punto sull'heap binario. Come risultato, vengono escluse le voci utilizzate di meno.

Programma di esclusione LRU (least recently used)

Il programma di esclusione LRU segue gli stessi concetti del programma di esclusione LFU con alcune differenze. La differenza principale consiste nel fatto che il programma di esclusione LRU utilizza una coda FIFO (first in, first out) invece di una serie di valori heap binari. Ogni volta che si accede a una voce della cache, questa passa in testa alla coda. Di conseguenza, la parte anteriore della coda contiene le voci utilizzate più recentemente mentre la fine contiene le voci della mappa utilizzate meno recentemente. Ad esempio, si assuma che la voce della cache A viene utilizzata 50 volte mentre la voce B viene utilizzata soltanto una volta. In questa situazione, la voce della cache B si troverà nella parte anteriore della coda perché è stata utilizzata più di recente mentre la voce A si troverà alla fine della coda. Il programma di esclusione LRU elimina le voci della cache che si trovano alla fine della coda e che rappresentano quindi le voci della mappa utilizzate meno recentemente.

Proprietà dei programmi di esclusione LFU e LRU e procedure ottimali per migliorare le prestazioni

Numero di heap

Quando si utilizza il programma di esclusione LFU, tutte le voci della cache per una determinata mappa vengono ordinate in base al numero di heap che si specifica, migliorando significativamente le prestazioni e evitando la sincronizzazione di tutte le eliminazioni su un heap binario che contiene l'ordinamento della mappa. Un numero maggiore di heap riduce inoltre il tempo richiesto per il riordinamento degli heap in quanto ognuno di essi ha un numero inferiore di voci. Impostare il numero di heap intorno al 10% del numero di voci in BaseMap.

Numero di code

Quando si utilizza il programma di esclusione LRU, tutte le voci della cache per una determinata mappa vengono ordinate in base al numero di code LRU che si specifica, migliorando significativamente le prestazioni e evitando la sincronizzazione di tutte le eliminazioni su una coda che contiene l'ordinamento della mappa. Impostare il numero di code intorno al 10% del numero di voci in BaseMap.

Proprietà MaxSize

Quando un programma di esclusione LFU o LRU inizia l'eliminazione delle voci, questo utilizza la proprietà MaxSize per determinare il numero di heap binari o di elementi della coda LRU da eliminare. Ad esempio, si assuma di impostare il numero di heap o di code in modo da avere 10 voci di mappa su ogni coda. Se la proprietà MaxSize è impostata su 7, il programma di esclusione elimina 3 voci da ogni heap o da ogni coda in modo da riportare la dimensione di ogni heap o di ogni coda a 7. Il programma di esclusione elimina le voci della mappa da un heap o da una coda soltanto quanto se l'heap o la coda ha un numero di elementi maggiore di quanto stabilito dal valore della proprietà MaxSize. Impostare il valore della proprietà MaxSize sul 70% della dimensione del valore heap o della coda. Per questo esempio, il valore è impostato su 7. È possibile ottenere una dimensione approssimativa di ogni heap o di ogni coda dividendo il numero di voci BaseMap per il numero di heap o code utilizzati.

Proprietà SleepTime

Un programma di esclusione non rimuove costantemente le voci dalla mappa. Esso rimane invece inattivo per un determinato periodo, riattivandosi ogni *n* secondi, dove *n* fa riferimento alla proprietà SleepTime. Questa proprietà influenza positivamente le prestazioni: l'esecuzione di un programma di esclusione troppe volte riduce le prestazioni a causa delle risorse necessarie all'elaborazione. Tuttavia, se non si utilizza il programma di esclusione è possibile che si abbia un certo numero di voci non necessarie. Una mappa piena di voci non necessarie influenza negativamente sia i requisiti di memoria che le risorse di elaborazione richieste per la mappa. L'impostazione delle esclusioni ogni quindici secondi è un valore per la maggior parte delle mappe. Se la mappa viene scritta frequentemente e viene utilizzata a una frequenza di transazioni elevata, potrebbe risultare utile impostare un valore inferiore. Tuttavia, se si esegue l'accesso alla mappa molto poco frequentemente, è possibile impostare il periodo di tempo su un valore superiore.

Esempio

Il seguente esempio definisce una mappa, crea un nuovo programma di esclusione LFU, imposta le proprietà del programma di esclusione e imposta la mappa in modo da utilizzare il programma di esclusione:

```
//Utilizza ObjectGridManager per creare o richiamare l'ObjectGrid. Fare riferimento
// alla sezione ObjectGridManger
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");
//Imposta le proprietà assumendo 50.000 voci della mappa
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

L'utilizzo del programma di esclusione LRU è del tutto simile a quello del programma di esclusione LFU. Di seguito è riportato un esempio:

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
//Imposta le proprietà assumendo 50.000 voci della mappa
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Come si può notare, solo due righe sono differenti dall'esempio LFUEvictor.

Procedure ottimali per il programma di esclusione predefinito

procedure ottimali per il programma di esclusione *time to live* predefinito.

Oltre ai programmi di esclusione dei plug-in descritti nella sezione "Procedure ottimali delle prestazioni del programma di esclusione dei plug-in" a pagina 332, un programma di esclusione predefinito viene creato con ogni mappa di backup. Il programma di esclusione predefinito rimuove le voci in base al concetto di *durata temporale*. Questo comportamento è definito dall'attributo ttlType. Esistono tre attributi ttlTypes:

- **Nessuno**: specifica che le voci non scadono mai e pertanto non vengono rimosse dalla mappa.
- **Ora di creazione**: specifica che le voci vengono escluse in base all'ora in cui sono state create.
- **Ora dell'ultimo accesso**: specifica che le voci vengono escluse in base all'ora in cui è stato eseguito l'ultimo accesso.

Proprietà del programma di esclusione predefinito e procedure ottimali per prestazioni elevate

Proprietà TimeToLive

Questa proprietà, insieme alla proprietà ttlType, è una proprietà cruciale da un punto di vista delle prestazioni. Se si utilizza CREATION_TIME ttlType, il programma di esclusione elimina una voce quando la sua ora di creazione è uguale al valore dell'attributo TimeToLive. Se si imposta il valore dell'attributo TimeToLive su 10 secondi, tutte le voci nella mappa verranno eliminate dopo dieci secondi. È importante fare attenzione quando si imposta questo valore per CREATION_TIME ttlType. Questo programma di esclusione viene utilizzato al meglio quando un numero elevato di aggiunte alla cache viene utilizzato per un periodo di tempo definito. Con questa strategia, qualsiasi voce che viene creata verrà rimossa dopo il periodo specificato.

Di seguito è riportato un esempio di quando utilizzare un tipo TTL di CREATION_TIME. In questo caso, si utilizza un'applicazione Web che ottiene quote di azioni e il richiamo delle azioni più recenti non è un'operazione critica. In questo caso, le quote vengono memorizzate nella cache in un ObjectGrid per 20 minuti. Dopo 20 minuti, le voci della mappa di ObjectGrid scadono e vengono eliminate. Ogni venti minuti la mappa ObjectGrid utilizza il plug-in del programma di caricamento per aggiornare i dati della mappa con dati aggiornati dal database. Il database viene aggiornato ogni 20 minuti con le quote più recenti. Per questa applicazione pertanto l'utilizzo di un valore di TimeToLive uguale a 20 minuti è l'ideale.

Se si utilizza l'attributo LAST_ACCESSED_TIME ttlType, impostare il valore TimeToLive su un numero inferiore a quello che si utilizza per CREATION_TIME ttlType, in quanto l'attributo TimeToLive viene reimpostato ogni volta che si accede all'attributo. In altre parole, se l'attributo TimeToLive è uguale a 15 e ha una voce che è stata attiva per 14 secondi a cui poi è stato eseguito l'accesso, questa voce non scadrà per altri 15 secondi. Se si imposta il valore di TimeToLive su un numero relativamente alto, molte voci potrebbero non essere mai eliminate. Tuttavia, se si imposta il valore su un numero tipo 15 secondi, le voci a cui non viene effettuato l'accesso di frequente verranno rimosse.

Di seguito è riportato un esempio di quando utilizzare un tipo TTL di LAST_ACCESSED_TIME. Una mappa ObjectGrid viene utilizzata per conservare i dati della sessione da un client. I dati della sessione devono essere eliminati se il client non li utilizza per un determinato periodo di tempo. Ad esempio, i dati della sessione scadono dopo 30 minuti di nessuna attività da parte del client. In questo caso, l'utilizzo di un tipo TTL di LAST_ACCESSED_TIME con l'attributo TimeToLive impostato su 30 minuti è esattamente l'impostazione necessaria per questa applicazione.

Esempio

Nel seguente esempio viene creata una mappa di backup, viene impostato il relativo attributo ttlType del programma di esclusione predefinito e viene impostata la proprietà TimeToLive.

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
bMap.setTtlEvictorType(TTLType.LAST_ACCESSED_TIME);
bMap.setTimeToLive(15);
```

La maggior parte delle impostazioni del programma di esclusione deve essere impostata prima dell'inizializzazione di ObjectGrid. Per maggiori informazioni sui programmi di esclusione, fare riferimento a "Programmi di esclusione" a pagina 187.

Capitolo 12. Distribuzione delle modifiche tra JVM del peer

Gli oggetti LogSequence e LogElement comunicano le modifiche che si sono verificate in una transazione ObjectGrid con un plug-in.

Per ulteriori informazioni su come utilizzare Java Message Service (JMS) per distribuire le modifiche alle transazioni, fare riferimento a “Java Message Service per la distribuzione delle modifiche alle transazioni” a pagina 340.

Un prerequisite è che l'istanza ObjectGrid deve essere memorizzata nella cache da ObjectGridManager. Per ulteriori informazioni, consultare “Metodi createObjectGrid” a pagina 89. Questo valore booleano cacheInstance deve essere impostato su true.

Gli oggetti forniscono un mezzo perché un'applicazione possa facilmente pubblicare le modifiche che si sono verificate in un object grid utilizzando un trasporto di messaggi agli ObjectGrids peer su JVM (Java virtual machine) remote e applicando quindi tali modifiche sulla JVM. La classe LogSequenceTransformer è di fondamentale importanza per abilitare questo supporto. Questa sezione descrive come scrivere un listener mediante un sistema di messaggistica Java Message Service (JMS) per la propagazione dei messaggi.

ObjectGrid supporta la trasmissione di LogSequences che provoca un commit delle transazioni di ObjectGrid sui membri del cluster di WebSphere Application Server con un plug-in non IBM. Per impostazione predefinita, questa funzione non è abilitata ma è possibile configurarla in modo che sia operativa. Tuttavia, se il consumatore o il produttore non è WebSphere Application Server, potrebbe essere necessario utilizzare un sistema di messaggistica JMS.

1. Inizializzazione del plug-in. L'ObjectGrid richiama il metodo `initialize` del plug-in, parte del contratto di interfaccia ObjectGridEventListener, quando viene avviato ObjectGrid. Il metodo `initialize` deve ottenere le risorse JMS, comprese connessioni, sessioni e publisher e avvia il thread che rappresenta il listener JMS. Il metodo initialize ha il seguente aspetto:

```
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();
  // per ricevere i messaggi, è necessario avviare la connessione
  connection.start();
  // la sessione jms non è transazionale (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);
  // avvia il thread del listener.
listenerRunning = true;
  listenerThread = new Thread(this);
  listenerThread.start();
}
 catch(Throwable e)
{
throw new ObjectGridRuntimeException("Cannot initialize", e);
}
}
```

Il codice per avviare il thread utilizza un thread Java 2 Platform, Standard Edition (J2SE). Se si utilizza WebSphere Application Server Versione 6.x o un server WebSphere Application Server Version 5.x Enterprise, utilizzare l'API (application programming interface) per i bean asincroni per avviare il thread del daemon. È possibile utilizzare anche le API comuni. Di seguito è riportato un frammento di sostituzione di esempio che mostra la stessa azione utilizzando un gestore lavori:

```
  // avvia il thread del listener.
listenerRunning = true;
workManager.startWork(this, true);
```

Il plug-in deve implementare anche l'interfaccia Work invece che l'interfaccia Runnable. È inoltre necessario aggiungere un metodo release per impostare la variabile listenerRunning su false. Il plug-in deve essere fornito con una istanza WorkManager nel costruttore o mediante iniezione se si utilizza un contenitore Inversion of Control (IoC).

2. Trasmettere le modifiche. Di seguito è riportato un metodo transactionEnd di esempio per la pubblicazione delle modifiche locali che sono apportate a un ObjectGrid. tale metodo utilizza JMS, ma è possibile utilizzare un qualsiasi trasporto di messaggi capace di pubblicare e registrare i messaggi.

```
// Questo metodo è sincronizzato per garantire che
// i messaggi vengano pubblicati nell'ordine con cui è stato eseguito
// il  commit delle transazioni. Se la pubblicazione dei messaggi è stata avviata
// in parallelo, allora i riceventi possono danneggiare la mappa
// in quanto le istruzioni di eliminazione possono arrivare prima
   di quelle di inserimento e così via.
public synchronized void transactionEnd(String txid,
boolean isWriteThroughEnabled,
boolean committed, Collection changes)
{
try
{
  // must be write through and commited.
  if(isWriteThroughEnabled && committed)
{
   // write the sequences to a byte []
   ByteArrayOutputStream bos = new ByteArrayOutputStream();
   ObjectOutputStream oos = new ObjectOutputStream(bos);
   if (publishMaps.isEmpty()) {
    // serialize the whole collection
    LogSequenceTransformer.serialize(changes, oos, this, mode);
}
else {
    // filter LogSequences based on publishMaps contents
    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);
}
  // crea un messaggio di oggetto per le modifiche
  oos.flush();
  ObjectMessage om = jmsSession.createObjectMessage(
  bos.toByteArray());
// set properties
  om.setStringProperty(PROP_TX, txid);
  om.setStringProperty(PROP_GRIDNAME, myGrid.getName());
  // transmit it.
  publisher.publish(om);
}
}
 catch(Throwable e)
{
throw new ObjectGridRuntimeException("Cannot push changes", e);
}
}
```

Questo metodo utilizza diverse variabili di istanza:

- Variabile **jmsSession**: una sessione JMS utilizzata per pubblicare i messaggi. Essa viene creata quando viene inizializzato il plug-in.
- Variabile **mode**: la modalità di distribuzione.
- Variabile **publishMaps**: una serie che contiene il nome di ciascuna mappa con le modifiche da pubblicare. Se la variabile è vuota, allora vengono pubblicate tutte le mappe.
- Variabile **publisher**: un oggetto TopicPublisher creato durante il metodo initialize del plug-in.

3. Ricevere ed applicare i messaggi di aggiornamento. Di seguito è riportato il metodo run. Questo metodo viene eseguito in un loop fino a che il loop viene arrestato da un'applicazione. Ogni iterazione del loop prova a ricevere un messaggio JMS e lo applica all'ObjectGrid.

```
private synchronized boolean isListenerRunning()
{
 return listenerRunning;
}
 public void run()
{
try
{
System.out.println("Listener starting");
  // richiama una sessione jms per la ricezione dei messaggi.
  // Non transazionale.
  TopicSession myTopicSession;
  myTopicSession = connection.createTopicSession(false,
  javax.jms.Session.AUTO_ACKNOWLEDGE);
  // richiama un utente registrato per l'argomento, true indica che non riceve
  // messaggi trasmessi mediante i publisher
  // su questa connessione. In caso contrario, verrebbero ricevuti
  // i propri aggiornamenti.
  TopicSubscriber subscriber = myTopicSession.createSubscriber(topic,
  null, true);
System.out.println("Listener started");
  while(isListenerRunning())
```

```
{
   ObjectMessage om = (ObjectMessage)subscriber.receive(2000);
   if(om != null)
{
    // Utilizza l'oggetto Session inviato al metodo initialize...
    // è molto importante non scrivere nulla a questo punto
    mySession.beginNoWriteThrough();
    byte[] raw = (byte[])om.getObject();
    ByteArrayInputStream bis = new ByteArrayInputStream(raw);
    ObjectInputStream ois = new ObjectInputStream(bis);
    // esegue l'inflazione della raccolta di LogSequences
     collection = LogSequenceTransformer.inflate(ois,
     myGrid);
    Iterator iter = collection.iterator();
  while (iter.hasNext()) {
     // elabora ogni modifica alla mappa in base alla modalità
     // di quando è stato serializzato LogSequence
     LogSequence seq = (LogSequence)iter.next();
     mySession.processLogSequence(seq);
}
    mySession.commit();
   } // se era presente un messaggio
  } // durante il loop
  // interrompere la connessione
  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");
}
```

La classe LogSequenceTransformer e le API ObjectGridEventListener, LogSequence e LogElement consentono l'utilizzo di una qualsiasi metodologia di pubblicazione e registrazione per distribuire le modifiche e filtrare le mappe che si desidera distribuire. I frammenti di codice in questa attività mostrano come utilizzare queste API con JMS per creare un ObjectGrid peer-to-peer condiviso dalle applicazioni presenti su diverse piattaforme che condividono un trasporto di messaggi comune.

Java Message Service per la distribuzione delle modifiche alle transazioni

Utilizzare Java Message Service (JMS) per le modifiche distribuite tra livelli diversi o in ambienti a piattaforme miste.

JMS è un protocollo ideale per le modifiche distribuite tra livelli differenti o in ambienti su piattaforme miste. Ad esempio, alcune applicazioni che utilizzano ObjectGrid potrebbero essere distribuite su Gluecode o Tomcat, dove altre

applicazioni vengono eseguite su WebSphere Application Server Versione 6.0. JMS è adatto alle modifiche distribuite tra i peer ObjectGrid in questi ambienti differenti. Il trasporto dei messaggi del gestore HA è molto rapido, ma è in grado di distribuire le modifiche soltanto alle JVM che fanno parte di un gruppo principale. JMS è più lento, ma consente a un numero maggiore e vario di client applicativi di condividere un ObjectGrid. JMS è ideale quando si condividono i dati in un ObjectGrid tra un client Swing e un'applicazione distribuita su WebSphere Extended Deployment.

Panoramica

JMS è implementato per la distribuzione delle modifiche alle transazioni mediante un oggetto Java che funziona da listener ObjectGridEventListener. Questo oggetto può propagare lo stato in uno dei seguenti quattro modi:

Invalida
Una voce che viene esclusa, aggiornata o eliminata viene rimossa su tutte le JVM (Java Virtual Machines) del peer quando viene ricevuto il messaggio.

Invalida condizionale
La voce viene esclusa solo se la versione locale è la stessa o precedente alla versione sul publisher.

Push Qualasiasi voce che viene esclusa, aggiornata, eliminata o inserita viene aggiunta o sovrascritta su tutte le JVM del peer quando viene ricevuto il messaggio JMS.

Push condizionale
La voce viene aggiornata o aggiunta solo sul lato di ricezione se la voce locale è meno recente della versione che viene pubblicata.

Ascolto delle notifiche di pubblicazione

Il plug-in implementa l'interfaccia ObjectGridEventListener per intercettare l'evento transactionEnd. Quando ObjectGrid richiama questo metodo, il plug-in prova a convertire l'elenco LogSequence per ogni mappa considerata dall'applicazione in un messaggio JMS e quindi la pubblica. Il plug-in può essere configurato per pubblicare le modifiche per tutte le mappe o per una serie di mappe. Gli oggetti LogSequence vengono elaborati per le mappe che hanno l'opzione di pubblicazione abilitata. La classe LogSequenceTransformer di ObjectGrid serializza un LogSequence filtrato per ogni mappa su un flusso. Dopo aver serializzato tutti i LogSequence sul flusso, allora viene creato un JMS ObjectMessage cjhe viene quindi pubblicato su un argomento noto.

Ascolto dei messaggi JMS e applicazione a un ObjectGrid locale

Lo stesso plug-in avvia anche un thread che viene eseguito in un loop, ricevendo tutti i messaggi che sono pubblicati su un argomento noto. Quando arriva un messaggio, il relativo contenuto viene inviato alla classe LogSequenceTransformer per convertirlo in una serie di oggetti LogSequence. Quindi, viene avviata una transazione senza scrittura. Ogni oggetto LogSequence è fornito al metodo Session.processLogSequence, che aggiorna le mappe locali con le modifiche. Il metodo processLogSequence rileva la modalità di distribuzione. Viene quindi eseguito il commit della transazione e la cache locale riflette le modifiche.

Per ulteriori informazioni sull'utilizzo di JMS per distribuire le modifiche alla transazione, fare riferimento a Capitolo 12, "Distribuzione delle modifiche tra JVM del peer", a pagina 337.

Capitolo 13. Integrazione del contenitore basato sulle introduzioni

È possibile utilizzare la struttura Inversion of Control (IoC) per configurare completamente l'ObjectGrid. Come risultato, non è necessario utilizzare la struttura di configurazione XML di ObjectGrid.

Contenitori basati sulle introduzioni

I contenitori basati sulle introduzioni, detti anche Inversion of Control (IoC), rappresentano un modello comune utilizzato dalle applicazioni sia sul lato client che sul lato server. Esistono diverse implementazioni open source di tali contenitori. Anche la nuova specifica di Enterprise JavaBeans (EJB) Versione 3.0 utilizza alcuni di questi concetti. La maggior parte di queste strutture sono contenitori Enterprise JavaBean e hanno la responsabilità di creare un'istanza di un determinato bean. Tali strutture possono inoltre inizializzare un bean con una serie di proprietà e attivare gli Enterprise JavaBeans richiesti utilizzando coppie getter e setter sugli attributi degli Enterprise JavaBean. Le API (application programming interfaces) di ObjectGrid sono progettate per funzionare con tali contenitori. A cominciare con il metodi ObjectGridManager.createObjectGrid, è possibile configurare tali contenitori per il bootstrap di un ObjectGrid in modo che l'applicazione abbia un riferimento sia a un ObjectGrid operativo oppure possa richiedere al contenitore di fornire una sessione ObjectGrid quando richiesto.

Modelli supportati

Nelle seguenti sezioni vengono descritte le operazioni effettuate per verificare che le API dell'ObjectGrid possano essere utilizzate da un'applicazione che utilizza una struttura IoC.

Utilizzare ObjectGridManager per creare ObjectGrids

Le API di ObjectGrid sono progettate per funzionare con le strutture IoC. Il singleton root utilizzato da una tale struttura è l'interfaccia ObjectGridManager, che ha diversi metodi predefiniti `createObjectGrid` che restituiscono un riferimento a un determinato ObjectGrid. È possibile impostare tale riferimento a ObjectGrid come singleton in una struttura IoC, in modo che le richieste successive per il bean restituiscano la stessa istanza ObjectGrid.

Plug-in di ObjectGrid

I plug-in su ObjectGrid includono:

- TransactionCallback
- ObjectGridEventListener
- SubjectSource
- MapAuthorization
- SubjectValidation

Ognuno di essi è un semplice JavaBeans che implementa un'interfaccia. È possibile utilizzare la struttura IoC per creare tali plug-in e collegarli agli attributi appropriati sull'istanza ObjectGrid. Ognuno di questi plug-in ha un corrispondente metodo impostato sull'interfaccia ObjectGrid per una semplice integrazione con la struttura IoC.

Creazione di mappe

Il metodo predefinito createMap sull'interfaccia ObjectGrid viene utilizzato per definire una mappa appena denominata. Qualsiasi plug-in richiesto da BackingMap (l'oggetto restituito da createMap) viene costruito utilizzando la struttura IoC e quindi viene collegato a BackingMap utilizzano il nome dell'attributo appropriato. Poiché nessun altro oggetto fa riferimento a BackingMap, le strutture IoC non lo creano automaticamente. Come soluzione alternativa, è possibile collegare ogni BackingMap a un attributo fittizio sul bean dell'applicazione principale. Utilizzare il metodo setMaps() per cancellare i BackingMaps che sono stati precedentemente definiti su questo ObjectGrid e sostituirli con l'elenco di BackingMaps fornito.

Backup dei plug-in delle mappe

I plug-in su BackingMap funzionano allo stesso modo su ObjectGrid. Ogni plug-in ha un corrispondente metodo `set` sull'interfaccia BackingMap. I plug-in di BackingMap sono:

- Loader
- ObjectTransformer
- OptimisticCallback
- Evictor
- MapEventListener

Modelli di utilizzo

Una volta impostato il file di configurazione IoC per produrre un ObjectGrid, creare un bean enterprise denominato `gridName_Session` o qualcosa di simile. Definirlo come un Enterprise JavaBean ottenuto dalla chiamata al metodo `getSession` sull'Enterprise JavaBean del singleton ObjectGrid. L'applicazione utilizza quindi la struttura IoC per ottenere un riferimento a un oggetto gridName_Session nel caso in cui un oggetto richieda una nuova sessione.

Riepilogo

L'utilizzo di ObjectGrid in ambienti che hanno già una struttura IoC per la creazione delle istanze e la configurazione dei bean è il metodo più diretto. È possibile utilizzare la struttura IoC per configurare completamente l'ObjectGrid e come risultato non sarà necessario utilizzare la struttura di configurazione XML di ObjectGrid. ObjectGrid funziona in maniera integrata con la struttura IoC esistente.

Capitolo 14. Risoluzione dei problemi

In questa sezione sono riportati gli scenari possibili per la risoluzione dei problemi causati da un errore dell'applicazione o da un problema nella progettazione dell'errore. Se si presume che ObjectGrid abbia un difetto, potrebbe essere necessario abilitare la traccia di ObjectGrid come descritto nella sezione di traccia di ObjectGridManager.

Errori a intermittenza e non spiegati

Un'applicazione prova a migliorare le proprie prestazioni utilizzando la modalità di copia COPY_ON_READ, COPY_ON_WRITE o NO_COPY come descritto nella sezione CopyMode. L'applicazione può rilevare problemi intermittenti quando il sintomo del problema cambia e il problema non viene spiegato o non è previsto. I problemi intermittenti non si verificano quando la modalità di copia è impostata su COPY_ON_READ_AND_COMMIT.

Problema

Il problema potrebbe essere dovuto a dati danneggiati nella mappa di ObjectGrid, che è il risultato della dell'applicazione che viola il contratto di programmazione della modalità di copia utilizzata. Il danneggiamento dei dati può provocare errori imprevisti che si verificano a intermittenza o in un modo non determinabile.

Soluzione

L'applicazione deve essere conforme al contratto di programmazione riportato per la modalità di copia utilizzata. Per le modalità di copia COPY_ON_READ e COPY_ON_WRITE, ciò significa che l'applicazione utilizza un riferimento a un valore all'esterno dell'ambito della transazione in cui il valore è stato invece ottenuto. Per utilizzare queste modalità, l'applicazione deve eliminare ogni riferimento al valore una volta completata la transazione e deve ottenere un nuovo riferimento al valore per ogni transazione che richiede l'accesso a tale valore. Per la modalità di copia NO_COPY, l'applicazione non deve mai modificare il valore. In questo caso, l'applicazione deve essere modificata in modo che non modifichi a sua volta il valore oppure deve utilizzare una modalità di copia differente. Fare riferimento alla sezione CopyMode per maggiori dettagli sull'impostazione della modalità di copia.

Tecnica generale per la gestione delle eccezioni

La conoscenza della causa principale di un oggetto Throwable è utile per isolare l'origine di un problema. Di seguito è riportato un esempio di un metodo di utilità che può essere utilizzato da un handler delle eccezioni per trovare la causa principale dell'eccezione Throwable che è stata emessa.

Esempio

```
static public Throwable findRootCause( Throwable t )
{
// Inizia con Throwable che si è verificato come root.
Throwable root = t;
// Segue la catena della causa fino a che viene individuato
     // l'ultimo Throwable nella catena.
Throwable cause = root.getCause();
while ( cause != null )
{
```

```
root = cause;
cause = root.getCause();
}
// Restituisce l'ultimo Throwable nella catena come causa principale.
return root;
}
```

Tecniche specifiche per la gestione delle eccezioni

Inserimento duplicato

Questo problema si verifica soltanto in un ambiente di propagazione delle transazioni distribuite. Ciò non si verifica troppo spesso.

Messaggio

```
[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 la sequenza di log filtrata viene propagata da una JVM a un'altra, la sequenza di log esterna viene elaborata sulla seconda JVM. La voce per questa chiave deve esistere già oppure i codici delle operazioni della sequenza di log devono essere differenti. Questo problema si verifica solo occasionalmente.

Impatto e soluzione

Quando si verifica questo problema, la voce non viene aggiornata su un'altra JVM il che può provocare un'incongruenza in ObjectGrid. Tuttavia, per evitare il problema, esiste una soluzione alternativa. È infatti possibile utilizzare l'utilità di partizione (WPF) sul richiamo degli oggetti oltre all'inserimento, all'aggiornamento e alla rimozione dell'oggetto. Fare riferimento alla sezione Integrazione di WPF e ObjectGrid per ulteriori informazioni su questa tecnica.

Eccezione di collisione ottimistica

È possibile ricevere un'eccezione OptimisticCollisionException direttamente oppure durante un'eccezione ObjectGridException. Il seguente codice è un esempio di come ottenere l'eccezione e visualizzare il relativo messaggio:

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

Causa dell'eccezione

Un'eccezione OptimisticCollisionException viene creata in una situazione in cui due diversi client provano ad aggiornare la stessa voce della mappa quasi allo stesso momento. La sessione di un client viene salvata e pertanto la voce della mappa viene aggiornata. Tuttavia, l'altro client ha già letto i dati prima del commit e contiene quindi dati obsoleti o non corretti. Quando l'altro client proverà ad eseguire il commit dei dati non corretti, viene emessa l'eccezione.

Richiamo della chiave che ha emesso l'eccezione

Potrebbe essere utile, quando si deve risolvere questa eccezione, richiamare la chiave corrispondente alla voce che ha emesso l'eccezione. Il vantaggio dell'eccezione OptimisticCollisionException sta nel fatto che contiene un metodo integrato getKey che restituisce l'oggetto che rappresenta la chiave. Di seguito è riportato un esempio su come richiamare e stampare la chiave quando viene restituita l'eccezione OptimisticCollisionException:

```
try {
...
} catch (OptimisticCollisionException oce) {
  System.out.println(oce.getKey());
}
```

Una eccezione OptimisticCollisionException potrebbe essere la causa di una eccezione ObjectGridException. In questo caso, è possibile utilizzare il seguente codice per determinare il tipo di eccezione e stampare la relativa chiave. Il codice riportato di seguito utilizza il metodo findRootCause come descritto nella sezione Tecnica generale per la gestione delle eccezioni.

```
try {
...
}
catch (ObjectGridException oe) {
 Throwable Root = findRootCause( oe );
 if (Root instanceof OptimisticCollisionException) {
  OptimisticCollisionException oce = (OptimisticCollisionException)Root;
  System.out.println(oce.getKey());
}
}
```

Eccezione LockTimeoutException

Messaggio

È possibile ottenere l'eccezione LockTimeoutException direttamente o durante un'eccezione ObjectGridException. Il seguente frammento di codice mostra come ottenere l'eccezione e visualizzare il messaggio.

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

Il risultato è:

```
com.ibm.websphere.objectgrid.plugins.LockTimeoutException: %Message
```

%Message rappresenta la stringa inviata come parametro quando viene creata l'eccezione e le proprietà e i metodi dell'eccezione vengono utilizzati per visualizzare un messaggio di errore preciso. Molto probabilmente, questo messaggio descrive il tipo di blocco che è stato richiesto e la voce della mappa su cui ha agito la transazione.

Causa dell'eccezione

Quando una transazione o un client richiede la concessione di un blocco per una determinata voce della mappa, spesso deve attendere che il client corrente rilasci il proprio blocco. Se la richiesta di blocco rimane inattiva per un periodo di tempo elevato e il blocco non viene concesso, allora viene restituita un'eccezione `LockTimeoutException`. Ciò si verifica per evitare una condizione di stallo, descritta in maniera più dettagliata nella seguente sezione. È probabile che questa eccezione venga restituita quando si utilizza una strategia di blocco pessimistico in quanto il blocco non viene mai rilasciato fino al commit della transazione.

Ulteriori informazioni sulla richiesta di blocco e sulla relativa eccezione

L'eccezione `LockTimeoutException` ha un metodo integrato denominato getLockRequestQueueDetails che restituisce una stringa che contiene una descrizione più dettagliata della situazione che ha emesso l'eccezione. Di seguito è riportato un esempio di codice che emette l'eccezione e visualizza un messaggio di errore.

```
try {
...
}
catch (LockTimeoutException lte) {
  System.out.println(lte.getLockRequestQueueDetails());
}
```

L'output del risultato è:

```
lock request queue
->[TX:163C269E-0105-4000-E0D7-5B3B090A571D, state =
 Granted 5348 milli-seconds ago, mode = U]
->[TX:163C2734-0105-4000-E024-5B3B090A571D, state =
 Waiting for 5348 milli-seconds, mode = U]
->[TX:163C328C-0105-4000-E114-5B3B090A571D, state =
 Waiting for 1402 milli-seconds, mode = U]
```

Se l'eccezione viene ottenuta in un blocco catch `ObjectGridException`, il seguente codice determina l'eccezione e stampa i dettagli della coda. Il codice utilizza il metodo del programma di utilità findRootCause descritto nella sezione Tecnica generale per la gestione delle eccezioni.

```
try {
...
}
catch (ObjectGridException oe) {
 Throwable Root = findRootCause( oe );
 if (Root instanceof LockTimeoutException) {
  LockTimeoutException lte = (LockTimeoutException)Root;
  System.out.println(lte.getLockRequestQueueDetails());
}
}
```

Soluzione possibile

L'eccezione `LockTimeoutException` consente di evitare possibili condizioni di stallo nell'applicazione. Un'eccezione di questo tipo viene emessa soltanto quando è necessario attendere una determinata quantità di tempo. La quantità di tempo da attendere, tuttavia, può essere impostata utilizzando il metodo `setLockTimeout(int)` disponibile per il BackingMap. L'utilizzo del metodo setLockTimeout elimina l'eccezione `LockTimeoutException`. Se non si verifica alcuna condizione di stallo per l'applicazione, la definizione di un valore di timeout per il blocco consente di evitare anche l'eccezione LockTimeoutException.

Il seguente codice mostra come creare un ObjectGrid, definire una mappa e impostare il relativo valore LockTimeout su 30 secondi.

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("MapName");
// Imposta la quantità di tempo che una
// richiesta di blocco deve attendere prima che venga emessa una eccezione.
bMap.setLockTimeout(30);
```

Il codice precedente può essere utilizzato per la codifica di ObjectGrid e le proprietà della mappa. Se si crea l'ObjectGrid da un file XML, la proprietà LockTimeout può essere impostata all'interno della tag backingMap. Di seguito è riportato un esempio della tag backingMap che imposta un valore LockTimeout per la mappa su 30 secondi.

```
<backingMap name="MapName" lockStrategy="PESSIMISTIC" lockTimeout="30">
```

LockDeadlockException

Messaggio

È possibile ottenere l'eccezione LockDeadLockException direttamente o durante un'eccezione ObjectGridException. Di seguito è riportato un esempio di codice che mostra l'eccezione e il messaggio visualizzato.

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

Il risultato è:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: %Message
```

%Message rappresenta la stringa inviata come parametro quando viene creata ed emessa l'eccezione.

Causa dell'eccezione

Il tipo più comune di condizione di stallo si verifica quando si utilizza una strategia di blocco pessimistico e due client separati, ognuno con un blocco condiviso su un determinato oggetto. Entrambi questi client provano a passare a un blocco esclusivo sull'oggetto. Di seguito è riportato un diagramma che mostra questa situazione con i blocchi delle transazioni che provocano l'emissione dell'eccezione.

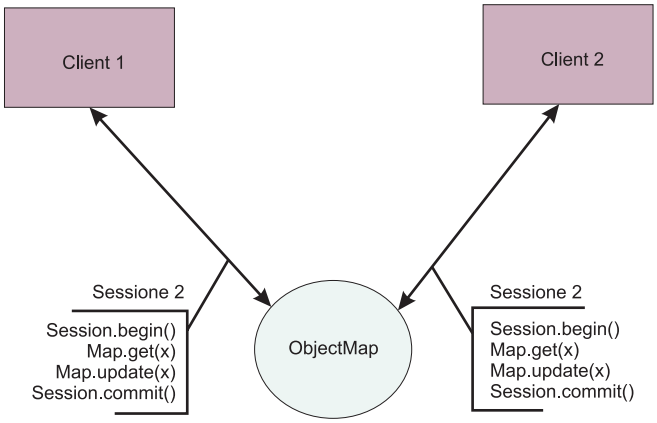


Figura 23. Esempio di una condizione di stallo potenziale

Questa è una vista astratta che si verifica nel programma quando viene emessa l'eccezione. In un'applicazione con molti thread che aggiornano lo stesso ObjectMap, è possibile che si verifichi questa situazione. Di seguito è riportata una procedura passo-passo che descrive due client che eseguono lo stesso blocco di codice della transazione, riportato nella figura precedente.

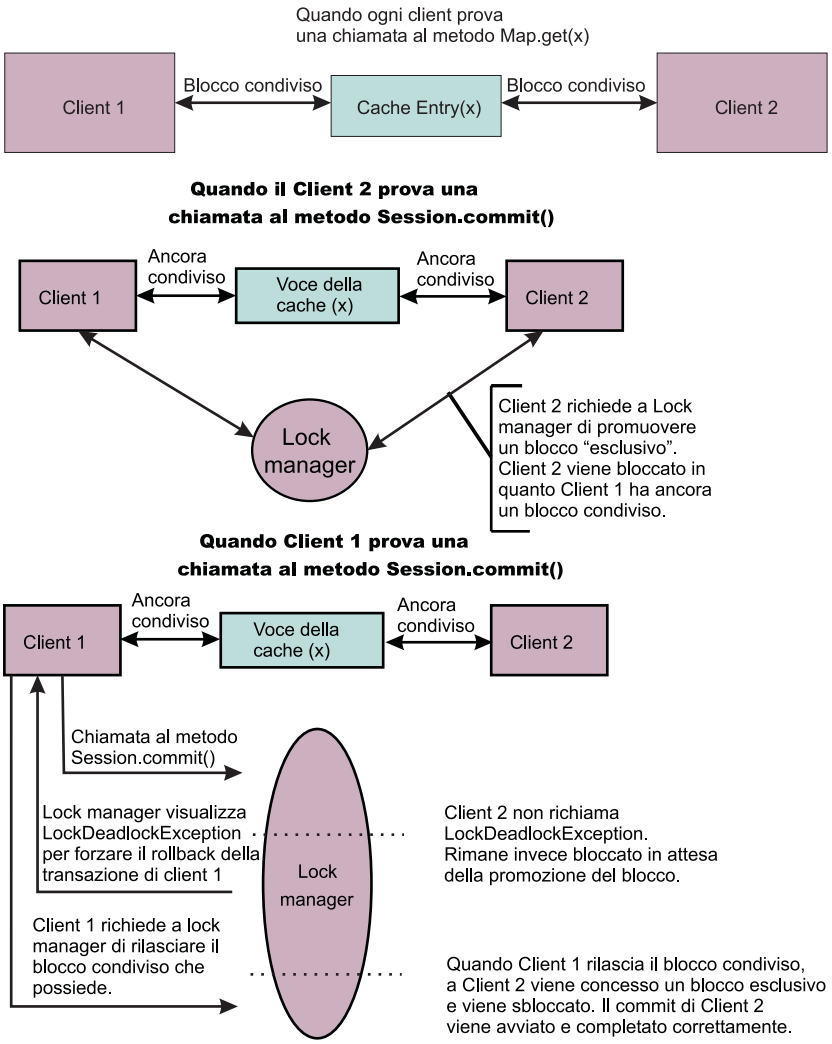


Figura 24. Situazione di condizione di stallo

come mostrato, quando entrambi i client provano a passare ai blocchi esclusivi dai blocchi condivisi, non è possibile che li ricevano. Devono sempre attendere che l'altro client rilasci il proprio blocco condiviso e pertanto si verifica un'eccezione `LockDeadlockException`.

Soluzioni possibili

La ricezione di questa eccezione è un fatto positivo. Quando sono presenti numerosi thread, ognuno dei quali esegue le transazioni su una determinata mappa, è possibile che si verifichi la situazione descritta precedentemente (Figura 1). Questa eccezione viene emessa per evitare che il programma si blocchi. Questa eccezione consente di avvisare gli utenti e di aggiungere un blocco catch in modo da ottenere maggiori dettagli sulla causa dell'eccezione stessa. Poiché questa eccezione viene visualizzata soltanto in una strategia di blocco pessimistico, una semplice soluzione consiste nell'utilizzare una strategia di blocco ottimistico. Se è richiesta una strategia pessimistica, tuttavia, è possibile utilizzare il metodo getForUpdate al posto del metodo get. Ciò elimina il richiamo delle eccezioni per la situazione descritta precedentemente.

Diagnosi di un problema di configurazione XML

Riferimento a una raccolta di plug-in non esistente

Quando si utilizza XML per definire i plug-in BackingMap, l'attributo pluginCollectionRef dell'elemento backingMap deve fare riferimento a backingMapPluginCollection. L'attributo pluginCollectionRef deve corrispondere all'ID di uno degli elementi backingMapPluginCollection.

Messaggio

Se l'attributo pluginCollectionRef non corrisponde ad alcun attributo id degli elementi backingMapPluginConfiguration, nel log viene registrato un messaggio simile a quello riportato di seguito.

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E:
Questo è un messaggio solo in lingua inglese:
File XML non valido. Riga: 14; URI: null;
Messaggio: La chiave 'pluginCollectionRef' con
valore 'bookPlugins' non è stata trovata per il
vincolo di identità per l'elemento 'objectGridConfig'..
```

Il seguente messaggio è un'eccezione dal log con la traccia abilitata:

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E: Questo è un
messaggio di errore solo in lingua inglese:
File XML non valido. Riga: 14; URI: null; Messaggio: La chiave
'pluginCollectionRef' con
valore 'bookPlugins' non è stata trovata per il vincolo di identità
dell'elemento 'objectGridConfig'..
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R com.ibm.websphere.objectgrid.
ObjectGridException:
File XML non valido: 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$2.run(ProcessConfigXML.java:99)
...
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R Caused by: org.xml.sax.
SAXParseException: La chiave 'pluginCollectionRef' con valore 'bookPlugins'
non è stato trovato per
il vincolo di identità per l'elemento 'objectGridConfig'.
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.error(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:011 CDT] 686c060e SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator$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

Il file XML che è stato utilizzato per produrre questo errore è riportato di seguito. Si noti che il registro BackingMap ha l'attributo pluginCollectionRef impostato su bookPlugins e l'unico attributo backingMapPluginCollection ha un ID uguale a 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>
```

Soluzione

Per risolvere il problema, verificare che il valore di ogni pluginCollectionRef corrisponda all'ID di uno degli elementi backingMapPluginCollection. In questo esempio, questo errore si evita modificando semplicemente il nome di pluginCollectionRef in collection1. Tra gli altri modi per risolvere il problema vi sono la modifica dell'ID dell'attributo backingMapPluginCollection esistente in modo che corrisponda a pluginCollectionRef o l'aggiunta di un altro attributo backingMapPluginCollection con un ID che corrisponda a pluginCollectionRef.

Attributo richiesto mancante

Molti degli elementi nel file XML hanno diversi attributi opzionali. È possibile includere o escludere gli attributi opzionali in un file. Il file XML passerà la convalida in ogni caso. Tuttavia, esistono anche degli attributi che invece sono obbligatori. Se gli attributi richiesti non sono presenti quando viene utilizzato il relativo elemento associato, allora la convalida XML non riesce.

Messaggio

Quando un attributo obbligatorio è mancante, nel log viene registrato un messaggio simile a quello riportato di seguito. In questo esempio, l'attributo type manca dall'elemento property.

```
[7/15/05 13:41:41:267 CDT] 6873dcac XmlErrorHandl E CWOBJ9002E:
Questo è un messaggio solo in lingua inglese
Messaggio di errore: file XML non valido.
Riga: 12; URI: null; Messaggio: cvc-complex-type.4:
L'attributo 'type' deve essere presente sull'elemento 'property'..
```

Il seguente messaggio è un'eccezione dal log con la traccia abilitata.

```
[7/15/05 14:08:48:506 CDT] 6873dff9 XmlErrorHandl E CWOBJ9002E: Questo è un
messaggio di errore solo in lingua inglese: file XML non valido.
Riga: 12; URI: null; Messaggio: cvc-complex-type.4: L'attributo 'type'
deve essere presente sull'elemento 'property'..
[7/15/05 14:08:48:526 CDT] 6873dff9 SystemErr R com.ibm.websphere.objectgrid.
ObjectGridException: File XML non valido: 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:
L'attributo 'type' deve essere presente sull'elemento '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

Di seguito è riportato un file XML che è stato utilizzato per riprodurre l'errore precedente. Si noti che la proprietà sul programma di esclusione ha solo due dei tre attributi richiesti. Gli attributi name e value sono presenti, ma manca l'attributo type. Questo attributo mancante provoca la mancata riuscita della convalida XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<backingMap name="book" pluginCollectionRef="collection1" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
<property name="maxSize" value="89" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Soluzione

Per risolvere questo problema, aggiungere l'attributo richiesto al file XML. Nel file XML riportato nell'esempio precedente, è necessario aggiungere l'attributo type e assegnare ad esso un valore intero.

Elemento richiesto mancante

Lo schema richiede degli elementi XML. Se tali elementi non sono presenti, la convalida XML non riesce.

Messaggio

Quando un elemento obbligatorio è mancante, nel log viene registrato un messaggio simile a quello riportato di seguito. In questo caso, è l'elemento objectGrid che risulta mancante.

```
[7/15/05 14:54:23:729 CDT] 6874d511 XmlErrorHandl E CWOBJ9002E:
Questo è un messaggio di errore solo in lingua inglese: file XML non valido.
Riga: 5; URI: null; Messaggio: cvc-complex-type.2.4.b: Il contenuto
dell'elemento 'objectGrids' non è completo.
Uno di '{"http://ibm.com/ws/objectgrid/config":objectGrid}' è previsto.
```

Abilitare la traccia per ulteriori informazioni su questo errore. Nella sezione ObjectGridManager sono riportate le informazioni su come attivare la traccia.

Problema

Di seguito è riportato un file XML che è stato utilizzato per riprodurre il problema. Si tenga presente che l'elemento objectGrids non dispone di elementi secondari objectGrid. In base allo schema XML, l'elemento objectGrid si deve trovare all'interno delle tag objectGrids almeno una volta. Questo elemento mancante provoca la mancata riuscita della convalida XML.

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

Soluzione

Per risolvere questo problema, verificare che tutti gli elementi richiesti si trovino nel file XML. Nell'esempio precedente, almeno un elemento objectGrid deve essere inserito tra le tag objectGrids. Una vola presenti gli elementi richiesti, è possibile convalidare correttamente il file XML.

Il seguente file XML contiene l'elemento richiesto.

```
<?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>
```

Valore XML dell'attributo non valido

Messaggio

Ad alcuni degli attributi nel file XML possono essere assegnati soltanto determinati valori. Tali attributi hanno i valori accettabili numerati in base allo schema. Essi includono:

- l'attributo `authorizationMechanism` sull'elemento objectGrid
- l'attributo `copyMode` sull'elemento `backingMap`
- l'attributo `lockStrategy` sull'elemento `backingMap`
- l'attributo `ttlEvictorType` sull'elemento `backingMap`
- l'attributo `type` sull'elemento `property`

Se a uno di questi attributi viene assegnato un valore non valido, la convalida XML non riesce.

Quando un attributo è impostato su un valore che non è uno dei valori numerati, nel log viene visualizzato il seguente messaggio:

```
[7/19/05 16:45:40:992 CDT] 6870e51b XmlErrorHandl E CWOBJ9002E: Questo è un messaggio
di errore solo in lingua inglese: file XML non valido. Riga: 6; URI: null; Messaggio:
cvc-enumeration-valid: Il valore 'INVALID_COPY_MODE' non è valido
rispetto alla numerazione '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ,
COPY_ON_WRITE, NO_COPY]'. Esso deve essere un valore dalla numerazione.
```

La seguente eccezione viene registrata se la traccia è abilitata.

```
[7/19/05 16:45:40:992 CDT] 6870e51b XmlErrorHandl E CWOBJ9002E: Questo è un
messaggio di errore solo in lingua inglese: file XML non valido. Riga: 6; URI: null; Messaggio:
cvc-enumeration-valid: Il valore 'INVALID_COPY_MODE' non è valido
rispetto alla numerazione '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ,
COPY_ON_WRITE, NO_COPY]'. Esso deve essere un valore dalla numerazione.
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R com.ibm.websphere.objectgrid
.ObjectGridException: File XML non valido: 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$2.run(ProcessConfigXML.java:99)...
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R Caused by:
org.xml.sax.SAXParseException:
cvc-enumeration-valid: Il valore 'INVALID_COPY_MODE' non è valido
with respect
alla numerazione '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE,
NO_COPY]'.
Deve essere un valore dalla numerazione.
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.util
.ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.util
.ErrorHandlerWrapper.error(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.XMLErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.XMLErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
.xs.XMLSchemaValidator$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

Un attributo a cui è assegnato un valore che non fa parte di una serie specifica di valori è stato impostato in maniera non corretta. In questo caso, l'attributo copyMode non viene impostato su uno dei valori numerati. Esso era impostato su INVALID_COPY_MODE. Di seguito è riportato un file XML che è stato utilizzato per riprodurre questo errore.

```
<?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>
```

Soluzione

In questo esempio, copyMode ha un valore non valido. Impostare l'attributo su uno dei seguenti valori validi: COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE o NO_COPY.

Convalida di XML senza il supporto di un'implementazione

IBM Software Development Kit (SDK) versione 1.4.2 contiene l'implementazione di alcune funzioni JAXP da utilizzare per la convalida XML rispetto a uno schema.

Messaggio

Quando si utilizza un SDK che non contiene questa implementazione, la convalida potrebbe non riuscire. Se si desidera convalidare XML utilizzando un SDK che non contiene questa implementazione, scaricare Xerces e includere i relativi file JAR (Java archive) nella variabile classpath.

Quando si prova a convalidare il codice XML con un SDK che non contiene l'implementazione necessaria, nel log viene registrato il seguente errore.

```
[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

L'SDK utilizzato non contiene un'implementazione della funzione JAXP necessaria a convalidate i file XML rispetto a uno schema.

Soluzione

Dopo aver scaricato Apache Xerces e incluso i file JAR nella variabile classpath, è possibile convalidare il file XML correttamente.

Messaggi ObjectGrid

Queste informazioni di riferimento forniscono notizie aggiuntive sui messaggi che potrebbero essere visualizzati durante l'uso di ObjectGrid. I messaggi sono identificati mediante una chiave del messaggio e sono dotati di una spiegazione e della risposta per l'utente. Essi possono rappresentare informazioni, avvertenze o errori e ciò viene indicato dall'ultima lettera (I, W o E) della chiave. La parte relativa alla spiegazione indica il motivo per cui è stato visualizzato il messaggio. La parte della risposta utente descrive l'azione da intraprendere nel caso di un messaggio di avvertenza o di errore.

```
CWOBJ0001E: Il metodo {0} è stato richiamato una volta completata l'inizializzazione.
Spiegazione: Una volta completata l'inizializzazione,
alcuni richiami di metodo non sono più accettati.
Risposta utente: Ristrutturare il codice in
modo che la configurazione venga completata prima dell'inizio dell'uso del runtime.
```

ICWOBJ0002W: Il componente ObjectGrid
sta ignorando un'eccezione imprevista: {0}.
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ0005W: Il thread ha creato una InterruptedException: {0}
Spiegazione: Si è verificata una InterruptedException.
Risposta utente: Controllare il messaggio
dell'eccezione per verificare se l'interruzione è prevista.

CWOBJ0006W: Si è verificata un'eccezione: {0}
Spiegazione: Si è verificata un'eccezione
durante il runtime.
Risposta utente: Controllare il messaggio
dell'eccezione per verificare se si tratta di un'eccezione prevista.

CWOBJ0007W: Il valore nullo è stato specificato per {0} e
viene utilizzato un valore predefinito di {1}.
Spiegazione: È stato specificato un valore
nullo per la variabile. Viene utilizzato un valore predefinito.
Risposta utente: Impostare il valore appropriato.
Fare riferimento alla documentazione di ObjectGrid
per conoscere i valori validi per le variabili o le proprietà.

CWOBJ0008E: Il valore {0} fornito per
la proprietà {1} non è valido.
Spiegazione: È stato specificato un valore non valido
per la variabile.
Risposta utente: Impostare il valore appropriato.
Fare riferimento alla documentazione di ObjectGrid
per conoscere i valori validi per le variabili o le proprietà.

CWOBJ0010E: La chiave del messaggio {0} risulta mancante.
Spiegazione: Una chiave del messaggio risulta mancante nel bundle di risorse del messaggio
Risposta utente: CMSG0002

CWOBJ0011W: Impossibile deserializzare il campo {0} nella classe {1};
viene utilizzato il valore predefinito.
Spiegazione: Durante la deserializzazione di un oggetto,
un campo richiesto non è stato trovato. Tale campo non è stato trovato probabilmente
perché l'oggetto è stato deserializzato da una versione differente della classe
rispetto a quella che l'ha serializzato.
Risposta utente: Questa avvertenza indica un problema potenziale.
Non è necessaria alcuna azione da parte dell'utente a meno che non venga emesso un errore.

CWOBJ0012E: Il codice del tipo LogElement, {0} ({1}), non è
riconosciuto per questa operazione.
Spiegazione: Si è verificato un errore interno nel runtime ObjectGrid.
Risposta utente: CMSG0002

CWOBJ0013E: Si è verificata un'eccezione nel tentativo di eliminare
le voci dalla cache: {0}
Spiegazione: Si è verificato un problema
nel tentativo di applicare
voci di eliminazione alla cache.
Risposta utente: Controllare il messaggio
dell'eccezione per verificare se si tratta di un'eccezione prevista.

CWOBJ0014E: Il runtime di ObjectGrid ha rilevato un tentativo
di nidificare le transazioni.
Spiegazione: La nidificazione delle transazioni non è consentita.
Risposta utente: Modificare il codice per evitare la nidificazione delle transazioni.

CWOBJ0015E: Si è verificata un'eccezione nel tentativo
di elaborare una transazione: {0}
Spiegazione: Si è verificato un problema durante l'elaborazione della transazione.
Risposta utente: Controllare il messaggio
dell'eccezione per verificare se questa eccezione è prevista.

CWOBJ0016E: Non è stata rilevata alcuna transazione attiva per l'operazione corrente.
Spiegazione: È necessario una transazione attiva per eseguire questa operazione.
Risposta utente: Modificare il codice per avviare una transazione prima di eseguire questa operazione.

CWOBJ0017E: È stata rilevata un'eccezione di chiave duplicata durante l'elaborazione dell'operazione ObjectMap: {0}
Spiegazione: La chiave per la voce esiste già nella cache.
Risposta utente: Modificare il codice per evitare di inserire la stessa chiave più volte.

CWOBJ0018E: La chiave non è stata trovata durante l'elaborazione dell'operazione ObjectMap: {0}
Spiegazione: La chiave per la voce non esiste nella cache.
Risposta utente: Modificare il codice per essere certi che la voce esista prima di tentare l'operazione.

CWOBJ0019W: Non sono stati trovati dati nello slot della voce della cache riservata per {0} da utilizzare per il nome ObjectMap {1}.
Spiegazione: Si è verificato un errore interno nel runtime ObjectGrid.
Risposta utente: CMSG0002

CWOBJ0020E: La voce della cache non si trova in BackingMap {0}.
Spiegazione: Errore interno nel runtime di ObjectGrid.
Risposta utente: CMSG0002

CWOBJ0021E: Un'istanza di ObjectTransformer utilizzabile non è stata trovata durante la deserializzazione dell'oggetto LogSequence per ObjectGrid {0} e per ObjectMap {1}.
Spiegazione: Il lato ricevente di un oggetto LogSequence non dispone della configurazione appropriata per supportare l'istanza ObjectTransformer richiesta.
Risposta utente: Verificare la configurazione delle istanze ObjectGrid sia per il lato mittente che per il ricevente dell'oggetto LogSequence.

CWOBJ0022E: Il chiamante non possiede un mutex: {0}.
Spiegazione: Si è verificato un errore interno nel runtime ObjectGrid.
Risposta utente: CMSG0002

CWOBJ0023E: La modalità CopyMode ({0}) non è riconosciuta per questa operazione.
Spiegazione: Si è verificato un errore interno nel runtime ObjectGrid.
Risposta utente: CMSG0002

CWOBJ0024E: Impossibile deserializzare il campo {0} nella classe {1}. La deserializzazione non è riuscita.
Spiegazione: Durante la deserializzazione di un oggetto, un campo richiesto non è stato trovato. Probabilmente questo problema è dovuto a un errore di runtime ObjectGrid.
Risposta utente: CMSG0002

CWOBJ0025E: La serializzazione dell'oggetto LogSequence non è riuscita. Il numero di oggetti LogElement serializzati ({0}) non corrisponde al numero di oggetti LogElement letti ({1}).
Spiegazione: Si è verificato un errore interno nel runtime ObjectGrid.
Risposta utente: CMSG0002

CWOBJ0026E: Il tipo di credenziale JMX non è corretto. Dovrebbe essere di tipo {0}.
Spiegazione: Il tipo di credenziale JMX non è corretto. Se si utilizza l'autenticazione di base, il tipo previsto è String[], in cui il primo elemento è il nome utente e il secondo è la password. Se viene utilizzato il certificato client, il tipo previsto è Certificate[].
Risposta utente: Utilizzare le credenziali corrette.

CWOBJ0027E: Errore di runtime interno. Metodo di clonazione non supportato: {0}
Spiegazione: Si è verificato un errore interno nel runtime ObjectGrid.
CLONE_METHOD_NOT_SUPPORTED_CWOBJ0027.useraction=CMSG0002

CWOBJ0028E: Si è verificato un errore in {0} per la mappa {1}. La chiave, {2}, non è stata trovata nella mappa. Il tipo LogElement è {3}.
Spiegazione: Si è verificato un errore interno nel tentativo di eliminare una voce.
Risposta utente: CMSG0002

CWOBJ0029E: Si è verificato un errore in {0} per la mappa {1}. Nella CacheEntry manca un oggetto {2} per la chiave {3}. Il tipo LogElement è {4}.
Spiegazione: Si è verificato un errore interno nel tentativo di eliminazione di una voce.
Risposta utente: CMSG0002

CWOBJ0900I: Il componente di runtime ObjectGrid viene avviato per il server {0}.
Spiegazione: Il componente ObjectGrid viene avviato.
Risposta utente:nessuna. Voce informativa.

CWOBJ0901E: È richiesta la proprietà di sistema "{0}" per avviare il componente ObjectGrid per il server {1}.
Spiegazione: Il componente di runtime ObjectGrid richiede di specificare "{0}" come proprietà del sistema JVM (Java Virtual Machine).
Risposta utente: Consultare l'Information Center per l'utilizzo della console di gestione WebSphere per fornire le proprietà personalizzate richieste da ObjectGrid.

CWOBJ0902W: Un errore ha impedito l'avvio del componente di runtime ObjectGrid per il server {0}.
Spiegazione: Un errore precedente ha impedito l'avvio del componente ObjectGrid.
Risposta utente: Consultare i precedenti messaggi di errore per determinare la causa che ha impedito l'avvio del componente ObjectGrid.

CWOBJ0910I: Il componente di runtime ObjectGrid viene arrestato per il server {0}.
Spiegazione: Il componente ObjectGrid viene arrestato.
Risposta utente:nessuna. Voce informativa.

CWOBJ0911I: Avvio del componente di runtime ObjectGrid per il server {0}.
Spiegazione: Il componente ObjectGrid è in fase di avvio.
Risposta utente:nessuna. Voce informativa.

CWOBJ1001I: ObjectGrid Server {0} è pronto per elaborare le richieste.
Spiegazione: Il server ObjectGrid è pronto per elaborare le richieste.
Risposta utente:I servizi per questo server ObjectGrid sono disponibili.

CWOBJ1002E: La porta del server {0} è già in uso.
Spiegazione: Il server ObjectGrid non può essere avviato a causa di un conflitto di porte.
Risposta utente:È necessario che gli utenti scelgano un'altra porta.

CWOBJ1003I: Il servizio dell'adattatore DCS è disabilitato dalla configurazione, per abilitarlo modificare la configurazione con un endpoint definito.
Spiegazione:L'adattatore è spento.
Risposta utente: Gli utenti possono accendere l'adattatore DCS modificando la configurazione.

CWOBJ1004E: L'argomento Server è nullo
Spiegazione:L'argomento Server è nullo
Risposta utente: CMSG0002

CWOBJ1005E: La coda di richieste in entrata è nulla.
Spiegazione: Il gestore richieste client non può richiamare le richieste.
Risposta utente: CMSG0002

CWOBJ1006E: La coda di risultati in uscita è nulla.
Spiegazione: Il gestore richieste client non può fornire risultati al client.
Risposta utente: CMSG0002

CWOBJ1007E: La richiesta del client ObjectGrid è nulla.
Spiegazione: Il gestore richieste client non può gestire una richiesta che non contenga informazioni sulla richiesta stessa.
Risposta utente: Verificare la richiesta

CWOBJ1008E: La richiesta del client TxID è nulla.
Spiegazione:TXID viene utilizzato per associare connessioni e per eseguire i lotti; TXID non può essere nullo.
Risposta utente: CMSG0002

CWOBJ1009E: Il client ObjectGrid ha ricevuto una risposta nulla dal server.
Spiegazione: Rilevata una risposta nulla dal server.
Risposta utente: CMSG0002

CWOBJ1010I: La richiesta di spegnimento è in fase di elaborazione.
Spiegazione:I server di cluster stanno elaborando la richiesta di spegnimento.
Risposta utente:nessuna

CWOBJ1011I: La richiesta di spegnimento è in fase di invio.
Spiegazione:I server di cluster stanno elaborando la richiesta di spegnimento
Risposta utente:nessuna

CWOBJ1012I: La richiesta di spegnimento viene eseguita.
Spiegazione:I server di cluster stanno elaborando la richiesta di spegnimento.
Risposta utente:nessuna

CWOBJ1110I: Avvio del trasporto per il cluster ObjectGrid {0} utilizzando indirizzo IP {1}, porta {2}, tipo di trasporto {3}.
Spiegazione: Il trasporto del membro cluster ObjectGrid è in fase di avvio.
Risposta utente:nessuna. Voce informativa.

CWOBJ1111W: La risoluzione degli indirizzi IP per il nome host {0} ha rilevato solo l'indirizzo di loopback. Verrà utilizzato tale indirizzo.
Spiegazione: Potrebbe essersi verificato un problema con il nome host o con la risoluzione DNS.
Per un'implementazione correlata alla produzione, è generalmente previsto un indirizzo non di loopback.
Risposta utente: Modificare il nome host o determinare se esiste un problema DNS.

CWOBJ1112E: È stato rilevato un errore durante la ricerca dell'indirizzo IP per il nome host di un membro cluster ObjectGrid. Il nome host è {0} e il nome server è {1}. Il membro verrà escluso dal cluster.
Spiegazione: Impossibile risolvere l'indirizzo IP per l'host indicato. Il membro cluster ObjectGrid per l'host specificato verrà escluso.
Risposta utente: Correggere il problema di ricerca del nome host e riprovare.

CWOBJ1113E: Il servizio di trasporto cluster ObjectGrid su questo processo non è stato avviato. Questo membro cluster non è definito nella configurazione.
Spiegazione: Questo membro cluster ObjectGrid non è un membro configurato del cluster. Qualora lo fosse, correggere la configurazione.
Risposta utente: Esaminare la configurazione corrente.

CWOBJ1114E: Il servizio di trasporto cluster ObjectGrid su questo processo non ha potuto elaborare il messaggio in entrata. Il messaggio è {0} e l'eccezione è {1}.
Spiegazione:È stato rilevato un errore interno imprevisto.
Risposta utente: Consultare il sito web di supporto internet relativo a IBM ObjectGrid per ricercare un problema simile o contattare l'assistenza IBM.

CWOBJ1115E: È stato ricevuto un evento di modifica vista dal trasporto cluster ObjectGrid. L'identificativo vista è {0} e l'evento è {1}.
Spiegazione: Il tipo di evento non è riconosciuto. HA Manager non ha le informazioni per rispondere all'evento.
Risposta utente: Consultare il sito web di supporto internet relativo a IBM ObjectGrid per ricercare un problema simile o contattare l'assistenza IBM.

CWOBJ1116E: Un tentativo da parte di un altro processo di connettersi al processo corrente tramite il trasporto cluster ObjectGrid è stato rifiutato. Il processo che stava tentando la connessione ha fornito il nome {0}, la destinazione {1}, il nome membro {2} e l'indirizzo IP {3}. Il messaggio di errore è {4}.
Spiegazione: Il trasporto cluster ObjectGrid ha rifiutato il tentativo di connessione.
Risposta utente: Potrebbe trattarsi di un tentativo di connessione non autorizzato.

CWOBJ1117E: Un tentativo di autenticazione connessione ha avuto esito negativo. L'eccezione è {0}.
Spiegazione: Il trasporto cluster ObjectGrid ha rifiutato il tentativo di connessione.
Risposta utente: Potrebbe trattarsi di un tentativo di connessione non autorizzato.

CWOBJ1118I: Inizializzazione del server ObjectGrid [Cluster: {0} Server: {1}].
Spiegazione: Il membro cluster ObjectGrid è in fase di
inizializzazione.
Risposta utente:nessuna. Voce informativa.

CWOBJ1119I: Il client ObjectGrid non è riuscito a connettersi all'host: {0} porta: {1}.
Spiegazione: Il client ObjectGrid non è riuscito a
stabilire una connessione.
Risposta utente:nessuna. Voce informativa.

CWOBJ1120I: Il client ObjectGrid è riuscito a connettersi all'host: {0} porta: {1}.
Spiegazione: Il client ObjectGrid ha stabilito una connessione.
Risposta utente:nessuna. Voce informativa.

CWOBJ1201E: Non viene definito alcun endpoint di accesso client valido.
Spiegazione:=Non viene definito alcun endpoint di accesso client valido.
Risposta utente: Definire un endpoint di accesso client valido.

CWOBJ1202E: inizializzazione socket server SSL non riuscita. Il messaggio di eccezione è {0}
Spiegazione: Inizializzazione del socket server SSL non riuscita.
Le impostazioni SSL potrebbero non essere corrette o il numero porta è già in uso.
Risposta utente: Esaminare l'eccezione per comprendere gli errori.

CWOBJ1203W: Ricevuto un evento di timeout dal server per
la transazione: {0}
Spiegazione: Il client non ha ricevuto il messaggio di risposta previsto
dal server entro il limite di timeout configurato.
Risposta utente: Esaminare i messaggi precedenti che potrebbero spiegare il timeout.
Nel caso non ne sia presente alcuno, aumentare il limite di timeout.

CWOBJ1204W: Ricevuto un messaggio di tipo sconosciuto.
Il messaggio è: {0}
Spiegazione:È stato rilevato un errore interno imprevisto.
Risposta utente: Consultare il sito web di supporto internet
relativo a IBM ObjectGrid per ricercare un problema simile o contattare l'assistenza IBM.

CWOBJ1205E: Inizializzazione SSL non riuscita. Il messaggio di eccezione è {0}
Spiegazione: Inizializzazione SSL non riuscita. Le impostazioni SSL
potrebbero non essere corrette.
Risposta utente: Esaminare l'eccezione per comprendere gli errori.

CWOBJ1206W: Inizializzazione SSL non riuscita. Il messaggio di eccezione è {0}
Spiegazione: Inizializzazione SSL non riuscita.
Le impostazioni SSL potrebbero non essere corrette.
Risposta utente: Esaminare l'eccezione per comprendere gli errori.

CWOBJ1207W: La proprietà {0} sul plug-in {1} sta utilizzando
un tipo di proprietà non supportato.
Spiegazione: Le primitive Java e le relative controparti java.lang
sono gli unici tipi di proprietà supportati. È supportato anche java.lang.String.
Risposta utente: Controllare il tipo di proprietà e modificarlo in
uno dei tipi supportati.

CWOBJ1208W: Il tipo di plug-in specificato, {0}, non è uno dei
tipi di plug-in supportati.
Spiegazione: Questo tipo di plug-in non è supportato.
Risposta utente: Aggiungere uno dei tipi di plug-in
supportati.

CWOBJ1211E: La creazione PMI (Performance Monitoring Infrastructure)
di {0} ha avuto esito negativo. L'eccezione è {1}.
Spiegazione: Il tentativo di creare PMI ObjectGrid non è riuscito.
Risposta utente: Esaminare il messaggio dell'eccezione
e il log FFDC (first failure data capture).

I seguenti messaggi sono utilizzati per raccogliere id utente e password sul pannello
o sull'input standard.
LOGIN_PANEL_TITLE=Accesso al server di destinazione
GENERIC_LOGIN_PROMPT=Immettere le informazioni di accesso
USER_ID=Identità utente
PASSWORD=Password
OK=OK
CANCEL=Annulla

CWOBJ1215I: È in corso l'inizializzazione del listener degli eventi di propagazione delle transazioni ObjectGrid [ObjectGrid {0}].
Spiegazione: Questo messaggio informativo indica che è in corso l'inizializzazione del listener degli eventi di propagazione delle transazioni ObjectGrid.
Risposta utente:nessuna. Voce informativa.

CWOBJ1216I: Il listener degli eventi di propagazione delle transazioni ObjectGrid è inizializzato [ObjectGrid {0}].
Spiegazione: Listener degli eventi di propagazione delle transazioni ObjectGrid inizializzato.
Risposta utente:=Nessuna. Voce informativa.

CWOBJ1217I: Punto di servizio di propagazione delle transazioni ObjectGrid inizializzato [ObjectGrid {0}].
Spiegazione: Questo messaggio informativo indica che il listener degli eventi di propagazione delle transazioni ObjectGrid è inizializzato.
Risposta utente:nessuna. Voce informativa.

CWOBJ1218E: Si è verificato un errore del listener degli eventi di propagazione delle transazioni [ObjectGrid {0} Messaggio dell'eccezione {1}].
Spiegazione: Il runtime ObjectGrid ha rilevato un errore di propagazione transazioni ObjectGrid.
Risposta utente: Esaminare l'eccezione per comprendere gli errori.

CWOBJ1219E: Si è verificato un errore dell'endpoint di servizio di propagazione delle transazioni ObjectGrid [ObjectGrid {0} Messaggio dell'eccezione {1}].
Spiegazione: Il runtime ObjectGrid ha rilevato un errore dell'endpoint di servizio di propagazione transazioni ObjectGrid.
Risposta utente: Esaminare l'eccezione per comprendere gli errori.

CWOBJ1220E: Il servizio di propagazione transazioni ObjectGrid non è supportato in questo ambiente.
Spiegazione:=Il servizio di propagazione transazioni ObjectGrid non è supportato in z/OS o nell'ambiente del server ObjectGrid autonomo.
Risposta utente: Non utilizzare il servizio di propagazione transazioni ObjectGrid in z/OS o nell'ambiente del server ObjectGrid autonomo.

CWOBJ1300I: Inizializzazione di ObjectGrid da parte dell'adattatore riuscita.
Spiegazione: Inizializzazione di ObjectGrid da parte dell'adattatore riuscita.
Risposta utente:nessuna. Voce informativa.

CWOBJ1301E: Inizializzazione di ObjectGrid da parte dell'adattatore non riuscita. Si è verificata un'eccezione [Messaggio di eccezione {0}].
Spiegazione: Tentativo degli adattatori di inizializzazione ObjectGrid non riuscito.
Risposta utente: Esaminare l'eccezione per comprendere gli errori.

CWOBJ1302I: Adattatore arrestato.
Spiegazione: Adattatore arrestato.
Risposta utente:nessuna. Solo a scopo informativo.

CWOBJ1303I: Adattatore avviato.
Spiegazione:PMA_CWOBJ1303.explanation=Adattatore avviato.
Risposta utente:nessuna. Solo a scopo informativo.

CWOBJ1304I: La protezione ObjectGrid è abilitata.
Spiegazione: La protezione ObjectGrid è abilitata.
Risposta utente:nessuna

CWOBJ1305I: La protezione ObjectGrid è disabilitata.
Spiegazione: La protezione ObjectGrid è disabilitata.
Risposta utente:nessuna

CWOBJ1306W: Impossibile richiamare i certificati client dal socket SSL.
Spiegazione: Per qualche ragione, il runtime non è in grado di richiamare i certificati client dal socket SSL.
Risposta utente: Verificare le configurazioni SSL.

CWOBJ1307I: La protezione dell'istanza ObjectGrid {0} è abilitata.
Spiegazione: La protezione dell'istanza ObjectGrid {0} è abilitata.
Risposta utente:nessuna

CWOBJ1308I: La protezione dell'istanza ObjectGrid {0} è disabilitata.
Spiegazione: La protezione dell'istanza ObjectGrid {0} è disabilitata.
Risposta utente:nessuna

CWOBJ1309E: Si è verificato un errore imprevisto
nella creazione del token di connessione: {0}.
Spiegazione: Si è verificato un errore interno
nella creazione del token di connessione.
Risposta utente: controllare la configurazione della protezione.

CWOBJ1310E: Un tentativo da parte di un altro processo di
connettere tale processo tramite il trasporto del gruppo principale è stato rifiutato. Il
processo di connessione ha fornito il nome gruppo principale di origine {0}, la destinazione {1},
il nome membro {2} e l'indirizzo IP {3}. Il messaggio di errore è {4}.
Spiegazione: High Availability Manager ha
rifiutato un tentativo di connessione.
Risposta utente: Potrebbe trattarsi di un tentativo di connessione non autorizzato.

CWOBJ1400W: Rilevati più file JARS di runtime ObjectGrid nella JVM.
L'utilizzo di più file JAR di runtime ObjectGrid potrebbe causare dei problemi.
Spiegazione: Generalmente, in una JVM dovrebbe
essere presente un solo JAR runtime ObjectGrid.
Risposta utente: Utilizzare i file JAR di runtime ObjectGrid appropriati per
la propria configurazione.

CWOBJ1401E: Rilevato un file JAR di runtime ObjectGrid non corretto per
questa configurazione. La configurazione rilevata è {0}. Il file jar previsto è {1}.
Spiegazione: Ogni file JAR del runtime ObjectGrid corrisponde a una specifica
configurazione supportata.
Risposta utente: Utilizzare i file JAR di runtime ObjectGrid appropriati per la configurazione.

CWOBJ1402E: Richiamata del collegamento di connessione ObjectGrid non trovata per l'id: {0}.
Spiegazione: Errore interno nel runtime di ObjectGrid.
Risposta utente: CMSG0002

CWOBJ1500E: Si è verificata un'eccezione nel tentativo di creare
un GroupName per il gruppo HA ({0}): {1}.
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ1501E: Si è verificata un'eccezione quando il membro ({0}) ha tentato
di partecipare al gruppo HA ({1}): {2}.
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ1503E: Impossibile accedere a ObjectGrid ({0}) per applicare gli aggiornamenti
al membro di replica ({1}).
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ1504E: Si è verificata un'eccezione nel tentativo di
elaborare le LogSequences per la replica ({0}): {1}.
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ1505E: Più membri del gruppo di replica
riportati come primari. Può essere attivo un solo gruppo primario. ({0}).
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ1506E: Più membri del gruppo di replica primari
esistono in questo gruppo ({1}). Può essere attivo un solo gruppo primario. ({0}).
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ1507W: Si è verificata un'eccezione nel tentativo di arrestare
il processo di replica per BackingMap ({0}): {1}.
Spiegazione: Nel tentativo di arrestare un membro del gruppo di replica primario,
si è verificata un'eccezione durante l'elaborazione della ripulitura.
Risposta utente: CMSG0002

CWOBJ1508E: Si è verificata un'eccezione nel tentativo di inviare il messaggio ({0})
dal mittente ({1}) al ricevente ({2}): {3}.
Spiegazione: Si è verificato un problema nel tentativo di inviare un messaggio tra
membri del gruppo di replica.
Risposta utente: CMSG0002

CWOBJ1509E: Si è verificata un'eccezione nel tentativo di serializzare
il messaggio ({0}): {1}.
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ1510E: Si è verificata un'eccezione nel tentativo di aumentare
il messaggio ({0}): {1}.
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ1511I: {0} ({1}) è aperto per l'azienda.
Spiegazione: Il membro del gruppo di replica specificato è ora pronto
per accettare le richieste.
Risposta utente:nessuna.

CWOBJ1512W: {0} esiste già nel gruppo di replica {1}.
Spiegazione: Il membro del gruppo di replica specificato è
già attivo in questo gruppo di replica.
Risposta utente:nessuna.

CWOBJ1513E: Replica sincrona non riuscita in {0} ({1}).
Questo membro non è più attivo.
Spiegazione:È stato rilevato un problema che ha impedito
il completamento della replica sincrona.
Risposta utente: Consultare i messaggi precedenti nel log per
diagnosticare il problema. È possibile che sia necessario un arresto o riavvio del server.

CWOBJ1514I: È in corso il downgrade di ({0}) primario in
replica o standby.
Spiegazione: Questa non è un'operazione consueta, ma l'elaborazione ObjectGrid
può continuare.
Risposta utente: CMSG0002

CWOBJ1515I: Requisiti di configurazione minimi non soddisfatti
per il gruppo di replica ({0}).
Spiegazione:I requisiti di configurazione replica e primari necessari
non sono stati soddisfatti con la recente modifica del membro del gruppo di replica.
Risposta utente: Attendere che ulteriori risorse vengano avviate e
riconosciute per questa configurazione.

CWOBJ1516E: Si è verificata un'eccezione nel tentativo
di attivare il processo di replica per ObjectGrid ({0}): {1}.
Spiegazione: Nel tentativo di avviare un membro del gruppo di replica primario,
si è verificata un'eccezione durante l'elaborazione dell'attivazione.
Risposta utente: CMSG0002

CWOBJ1517E: Replica sincrona non riuscita per
la transazione {2} su {0} ({1}). Questo membro non è più attivo.
Spiegazione:È stato rilevato un problema che ha impedito
il completamento della replica sincrona.
Risposta utente: Consultare i messaggi precedenti nel log per
diagnosticare il problema. È possibile che sia necessario un arresto o riavvio del server.

CWOBJ1518E: Si è verificata un'eccezione nel tentativo di
eseguire il commit di una transazione di replica ({0}) per la transazione primaria ({1}) su
Replica ({2}): {3}.
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ1519E: Si è verificata un'eccezione nel tentativo
eseguire il rollback delle LogSequences per la replica ({0}): {1}.
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ1610W: Reimpostare un cluster nullo per {0}.
Spiegazione:I dati del cluster del gruppo di replica non sono disponibili.
Risposta utente:nessuna

CWOBJ1611I: Il cluster del gruppo di replica {0} è aperto per l'azienda.
Spiegazione: Ora il cluster del gruppo di replica può accettare le richieste.
Risposta utente:nessuna

CWOBJ1612I: Il cluster del gruppo di replica {0} è chiuso per l'azienda.
Spiegazione: Ora il cluster del gruppo di replica non può accettare le richieste.
Risposta utente:=nessuna

CWOBJ1620I: Sostituzione della destinazione per richiesta instradata in modo non corretto a causa di modifiche nel server. La nuova destinazione è {0}.
Spiegazione: La vecchia destinazione di instradamento sostituita con la nuova.
Risposta utente: Se il gruppo di replica previsto non è attivo, è necessario riportarlo all'origine.

CWOBJ1630I: Il gruppo di replica non può soddisfare questa richiesta {0}.
Spiegazione:L'instradamento viene rifiutato a causa di un servizio non disponibile come ad esempio un effetto Domino
Risposta utente: Solo scopo informativo.

CWOBJ1632E: La richiesta originale non ha un ID valido; non vi è modo per inoltrare questa richiesta.
Spiegazione: Non vi è modo per inoltrare questa richiesta perché la richiesta originale non ha un ID valido.
Riposta utente: Rivolgersi all'Assistenza IBM

CWOBJ1634I: Il router non può individuare la destinazione di inoltro; utilizzo dell'inoltro senza distinzioni.
Spiegazione: Il router non riesce a rilevare la destinazione di inoltro.
Risposta utente:nessuna

CWOBJ1660I: Il membro del gruppo di replica è stato modificato. Questo server non ospita più gli elementi richiesti. La richiesta originale è {0}.
Spiegazione: Il membro del gruppo di replica è stato modificato.
Risposta utente: Se il gruppo di replica previsto non è attivo, è necessario riportarlo all'origine.

CWOBJ1661I: I dati di cluster vengono aggiornati per il gruppo di replica: {0}
SpiegazioneI dati di cluster vengono aggiornati
Risposta utente:nessuna

CWOBJ1663E: Il router del server non può verificare l'instradamento del server per {0}, perché i dati del cluster per questo gruppo di replica sono nulli nel server.
Spiegazione: Non è disponibile alcun dato del cluster del gruppo di replica da verificare.
Riposta utente: Rivolgersi all'Assistenza IBM

CWOBJ1668W: Una richiesta sta per essere inviata al server che non è ancora stato avviato.
N**Spiegazione:**L'avvio del server impiega 60-120 secondi. La richiesta verrà ritentata automaticamente se è stata configurata in tal modo (per impostazione predefinita la richiesta verrà ritentata automaticamente).
Risposta utente: Modificare la configurazione o avviare i client 60-120 secondi dopo l'avvio dei server.

CWOBJ1680W: Il timeout di connessione TCT configurato è inferiore al retryInterval * max(startupRetries, maxRetries), quindi è possibile che la transazione scada.
Spiegazione: Il timeout di connessione TCT configurato dovrebbe essere superiore al retryInterval * max(startupRetries, maxRetries).
Risposta utente: Modificare la configurazione.

CWOBJ1682W: Il timeout di transazione configurato è inferiore rispetto a maxForwards * al retryInterval * max(startupRetries, maxRetries), quindi è possibile che la transazione scada.
Spiegazione: Il timeout di transazione configurato dovrebbe essere superiore rispetto a maxForwards * retryInterval * max(startupRetries, maxRetries).
Risposta utente: Modificare la configurazione.

CWOBJ1700I: HAManager autonomo inizializzato con coregroup {0}.
Spiegazione:HAManager autonomo è stato inizializzato correttamente.
Risposta utente:nessuna

CWOBJ1701I: HAManager autonomo è già inizializzato.
Spiegazione:HAManager autonomo è è già stato inizializzato correttamente.
Risposta utente:nessuna

CWOBJ1702E: HAManager autonomo non è
inizializzato, quindi non può essere avviato.
Spiegazione:HAManager autonomo non è inizializzato.
Risposta utente: Inizializzarlo
prima di avviarlo.

CWOBJ1710I: HAManager autonomo è
stato avviato correttamente.
Spiegazione:HAManager autonomo è
stato avviato correttamente.
Risposta utente:nessuna

CWOBJ1711I: HAManager autonomo è già stato avviato correttamente.
Spiegazione:HAManager autonomo è
stato avviato correttamente.
Risposta utente:nessuna

Spiegazione:HAManager autonomo non è stato avviato.
Spiegazione:HAManager autonomo non è stato avviato.
Risposta utente: Inizializzarlo e avviarlo prima di utilizzarlo.

CWOBJ1713E: Avvio di HAManager autonomo non riuscito.
Spiegazione: Avvio di HAManager autonomo non riuscito.
Risposta utente: Verificare se le porte
sono già in uso.

CWOBJ1720I: Il controller HAManager ha rilevato che
il server ObjectGrid si trova nell'ambiente WebSphere e utilizza WebSphere HAManager
anziché inizializzare e avviare HAManager autonomo.
Spiegazione: Il server ObjectGrid è in esecuzione nell'ambiente WebSphere.
Risposta utente:nessuna

CWOBJ1730I: Il controller HAManager ha rilevato
che HAManager esterno di WebSphere è nullo.
Spiegazione: Impossibile richiamare HAManager esterno da WebSphere.
Risposta utente:nessuna

CWOBJ1790I: Occorre inizializzare e
avviare HAManager autonomo.
Spiegazione: Impossibile richiamare HAManager esterno da WebSphere.
Occorre inizializzare e avviare HAManager autonomo.
Risposta utente:nessuna

CWOBJ1792I: Il numero massimo di thread è {0} e il numero minimo
di thread è {1}.
Spiegazione: Configurare il lotto di thread.
Risposta utente: Solo scopo informativo.

CWOBJ1800I: L'inoltro è necessario per la richiesta {0} con risposta {1}.
Spiegazione:È richiesto un instradamento di inoltro.
Risposta utente:nessuna. Gestita
automaticamente

CWOBJ1810I: L'inoltro è necessario per la risposta {0}.
Spiegazione:L'inoltro è necessario per la risposta.
Risposta utente:nessuna

CWOBJ1811E: L'inoltro è necessario, ma
la richiesta originale non è stata trovata.
Spiegazione:L'inoltro è necessario, ma
la richiesta originale non è stata trovata.
Risposta utente:nessuna

CWOBJ1820E: La richiesta di inoltro non dispone di
un identificativo del gruppo di replica.
Spiegazione: Non è presente alcun identificativo del gruppo di replica
in questa richiesta di inoltro.
Riposta utente: Rivolgersi all'Assistenza IBM

CWOBJ1870I: Il servizio server non è disponibile per la risposta {0}.
Spiegazione: Il servizio server non è disponibile a causa dell'effetto Domino
o di altri eventi.
Risposta utente: Attivare almeno il numero minimo di server.

CWOBJ1871E: Rilevato servizio non disponibile, ricevuta risposta nulla,
non vi è possibilità di ripetere l'operazione.
Spiegazione: Risposta nulla dal servizio non disponibile
Riposta utente: Rivolgersi all'Assistenza IBM

CWOBJ1872I: Il servizio non è disponibile con la risposta {0}.
Spiegazione: Il servizio non è disponibile
Risposta utente: Attivare almeno il numero minimo di
server o verificare se l'avvio del server è riuscito.

CWOBJ1890I: Reinstradamento della richiesta {0} a causa
di un server che non risponde.
Spiegazione:la richiesta per il server previsto non è stata completata.
La richiesta è stata reinstradata su un altro server.
Risposta utente:nessuna. Gestita automaticamente. Se il gruppo di replica previsto
non è attivo, è necessario riportarlo all'origine.

CWOBJ1891E: Tutti i server del gruppo di replica non sono disponibili {0}.
Spiegazione: Tutti i server non sono stati avviati o hanno avuto
esito negativo. Essi non sono disponibili
Risposta utente: Se il gruppo di replica previsto non è attivo,
è necessario riportarlo all'origine.

CWOBJ1898W: L'inoltro è necessario, ma il router non riesce a rilevare la nuova
destinazione disponibile per la risposta {0}
Spiegazione: Il servizio non è disponibile.
Risposta utente: Rendere il servizio disponibile.

CWOBJ1899W: L'inoltro è necessario, ma il router non riesce a rilevare il
gruppo di replica appropriato per la risposta {0}
Spiegazione:L'ID del gruppo di replica è andato perduto.
Riposta utente: Rivolgersi all'Assistenza IBM

CWOBJ1900I: Il servizio di chiamata della procedura remota del server
client è stato inizializzato.
Spiegazione: Il servizio di chiamata della procedura
remota del server client è stato inizializzato.
Risposta utente:nessuna

CWOBJ1901I: Il servizio di chiamata della procedura remota del server client è stato avviato.
Spiegazione: Il servizio di chiamata della procedura remota del server client è stato avviato.
Risposta utente:nessuna

CWOBJ1902I: I thread del gestore di chiamata della procedura remota del server client
sono stati avviati.
Spiegazione:I thread del gestore di chiamata della procedura remota del server client
sono stati avviati.
Risposta utente:nessuna

CWOBJ1903I: Il servizio di rete della configurazione è stato inizializzato.
Spiegazione: Il servizio di rete della configurazione è stato inizializzato.
Risposta utente:nessuna

CWOBJ1904I: Il servizio di rete della configurazione è stato avviato.
Spiegazione: Il servizio di rete della configurazione è stato avviato.
Risposta utente:nessuna

CWOBJ1905I: Il gestore di configurazione è stato avviato.
Spiegazione: Il gestore di configurazione è stato avviato.
Risposta utente:nessuna

CWOBJ1913I: Il servizio di rete di gestione sistema è stato inizializzato.
Spiegazione: Il servizio di rete di gestione sistema è stato inizializzato.
Risposta utente:nessuna

CWOBJ1914I: Il servizio di rete di gestione sistema è stato avviato.
Spiegazione: Il servizio di rete di gestione sistema è stato avviato.
Risposta utente:nessuna

CWOBJ1915I: Il gestore di gestione sistema è stato avviato.
Spiegazione: Il gestore di gestione sistema è stato avviato.
Risposta utente:nessuna

CWOBJ2000E: Nessun membro in questo gruppo di replica {0}.
Spiegazione: Non è stato rilevato alcun membro in questo gruppo di replica.
Risposta utente: Verificare se i server sono stati avviati o se i dati sono disponibili

CWOBJ2001W: Nessun membro disponibile in questo gruppo di replica {0}.
Spiegazione: Non è stato rilevato alcun membro disponibile in questo gruppo di replica.
Risposta utente: Verificare se il servizio server è disponibile

CWOBJ2002W: Nessuna tabella di instradamento disponibile per questo gruppo di replica {0}.
Spiegazione: Nessuna tabella di instradamento disponibile per questo gruppo di replica {0}.
Risposta utente: Verificare se i client sono stati inseriti nella tabella di instradamento

CWOBJ2003I: Impossibile rilevare la cache di instradamento per la chiave di cache {0}; creazione di una nuova cache di instradamento in corso.
Spiegazione: Instradamento eseguito per la prima volta o modifiche al cluster.
R**Risposta utente:**nessuna

CWOBJ2010E: La destinazione per questa richiesta è nulla.
Spiegazione: La richiesta non è stata fornita con le informazioni di destinazione.
Riposta utente: Contattare l'Assistenza IBM.

WOBJ2060I: Il client ha ricevuto una nuova versione del cluster del gruppo di replica {0}.
Spiegazione: Il client ha ricevuto una nuova versione del cluster del gruppo di replica {0}.
Risposta utente:nessuna

CWOBJ2068I: Il controllo di accessibilità ha rilevato un problema nel membro del gruppo di replica {0}.
Spiegazione: Non è possibile accedere ad alcuni server;
il meccanismo di accessibilità gestirà tale problema.
Risposta utente:nessuna.

CWOBJ2069I: Il timer del controllo accessibilità rilascia il membro del gruppo di replica {0}.
Spiegazione: Questo membro è disponibile per l'instradamento.
Risposta utente:nessuna

CWOBJ2086I: Il controllo del thread di instradamento è stato attivato a causa di un sovraccarico per il gruppo di replica {0}.
Spiegazione: Il controllo thread è attivo.
Risposta utente:nessuna

CWOBJ2088I: Il controllo di accessibilità è stato attivato per regolare la disponibilità del server per il gruppo di replica {0}.
Spiegazione:L'accessibilità è attiva.
Risposta utente:nessuna

CWOBJ2090W: Impossibile rilevare la tabella di instradamento per il gruppo di replica {0}.
Spiegazione: Il cluster del gruppo di replica è nullo.
Risposta utente:nessuna

CWOBJ2091W: La tabella di instradamento non è nulla, ma non contiene alcun server per il gruppo di replica {0}.
Spiegazione: Il cluster del gruppo di replica è vuoto.
Risposta utente:nessuna

CWOBJ2092I: La tabella di instradamento è nulla nel runtime per il gruppo di replica {0}.
Spiegazione: Richiamo della tabella di instradamento dal runtime.
Risposta utente:nessuna

CWOBJ2093I: La tabella di instradamento non è nulla nell'archivio di cluster per il gruppo di replica {0}
Spiegazione: Richiamo della tabella di instradamento dall'archivio di cluster.
Risposta utente:nessuna

CWOBJ2096I: La tabella di instradamento è stata ottenuta dall'archivio di cluster per il gruppo di replica {0}.
Spiegazione: Ottenuto cluster del gruppo di replica dal relativo archivio di cluster.
Risposta utente:nessuna

CWOBJ2097I: L'instradamento si basa sull'algoritmo round robin per il gruppo di replica {0}.
Spiegazione:L'instradamento si basa sull'algoritmo round robin.
Risposta utente:nessuna

CWOBJ2098I: L'instradamento si basa sulla selezione casuale per il gruppo di replica {0}.
Spiegazione:L'instradamento si basa sulla selezione casuale.
Risposta utente:nessuna

CWOBJ2100I: La connessione ({0}) non è aggiornata e non può essere riutilizzata.
Spiegazione: La connessione non è aggiornata.
Risposta utente:nessuna

CWOBJ2101W: Non è possibile acquisire la connessione una volta trascorso
il tempo massimo di attesa.
Spiegazione: Non è rimasta più alcuna connessione nel lotto.
Risposta utente: Aumentare il numero massimo di
connessioni nella configurazione.

CWOBJ1600I: Avviato servizio ManagementGateway sulla porta ({0}).
Spiegazione: Il servizio ManagementGateway è
pronto per elaborare le richieste.
Risposta utente: Il servizio ManagementGateway è disponibile.

CWOBJ1601E: Avvio del servizio ManagementGateway sulla porta ({0}) non riuscito.
Spiegazione: Servizio ManagementGateway non avviato.
Risposta utente: Verificare che la porta
specificata non sia già in uso.

CWOBJ1602E: Il servizio ManagementGateway non è riuscito a connettersi al server in
({0}):({1}).
Spiegazione: Connessione del servizio ManagementGateway
al server non riuscita.
Risposta utente: Verificare che il server sia in esecuzione.

CWOBJ1603E: Mancata risposta del servizio di gestione alla richiesta remota ({0}).
Spiegazione:CMSG0001
Risposta utente: CMSG0002

CWOBJ2400E: Configurazione non valida: la mappa di backup {0} è
membro di più serie di mappe.
Spiegazione: Una backingMap puõ appartenere a una sola serie di mappe.
Risposta utente: Modificare il file XML del cluster in modo che ogni mappa di backup
appartenga a una sola serie di mappe.

CWOBJ2401E: Configurazione non valida: la mappa di backup {0}
nell'ObjectGrid distribuito {1} non appartiene a una serie di mappe.
Spiegazione: Ogni mappa di backup di un ObjectGrid distribuito deve
essere inserita in una serie di mappe.
Risposta utente: Modificare il file XML del cluster in modo che ogni mappa di backup
in un ObjectGrid distribuito appartenga a una sola serie di mappe.

CWOBJ2402E: Configurazione non valida: la serie di mappe ha un riferimento
a una mappa {0}. Tale mappa di backup non esiste nel file XML di ObjectGrid.
Spiegazione: Ogni mappa all'interno di una serie di mappe deve fare riferimento
a una mappa di backup dal file XML di ObjectGrid.
Risposta utente: Modificare il file XML in modo che ogni mappa all'interno
della serie di mappe faccia riferimento a una mappa di backup dal file XML ObjectGrid.

CWOBJ2403E: Il file XML non è valido. È stato rilevato un problema con
{0} alla riga {1}. Il messaggio di errore è {2}.
Spiegazione: Il file XML non è conforme allo schema.
Risposta utente: Modificare il file XML in modo che sia conforme allo schema.

CWOBJ2404W: Il valore specificato per {0} è {1}. Questo valore non è valido.
{0} non verrà impostato.
Spiegazione: Il valore per questo attributo di configurazione non è valido.
Risposta utente: Impostare l'attributo di configurazione su un valore appropriato nel
file XML.

CWOBJ2405E: rif collegamento objectgrid {0} nel file XML del cluster
non fa riferimento a un ObjectGrid valido dal file XML ObjectGrid.
Spiegazione: Ogni collegamento di objectgrid deve fare riferimento
a un ObjectGrid del file XML ObjectGrid.
Risposta utente: Modificare i file XML in modo che il collegamento objectgrid
nel file XML del cluster faccia riferimento a un ObjectGrid valido nel file XML ObjectGrid.

CWOBJ2500E: Avvio del server ObjectGrid {0} non riuscito.
Spiegazione: Avvio del server ObjectGrid non riuscito.
Risposta utente: Consultare il log per ricercare le eccezioni.

CWOBJ2501I: Avvio del server ObjectGrid {0}.
Spiegazione: Il server ObjectGrid è in fase di avvio.
Risposta utente:nessuna

CWOBJ2502I: Avvio del server ObjectGrid utilizzando l'URL del file XML ObjectGrid, "{0}" e l'URL del file XML del cluster "{1}".
Spiegazione: Il server ObjectGrid è in fase di avvio tramite un file XML del cluster e un file xml di ObjectGrid.
Risposta utente:nessuna

CWOBJ2503I: Avvio su un server Objectgrid peer sull'host {0} e sulla porta {1}.
Spiegazione: Questo server ObjectGrid verrà avviato su un server peer per richiamare le informazioni necessarie per l'avvio.
Risposta utente:nessuna

COBJ2504I: Tentativo di avvio su un server ObjectGrid peer utilizzando i seguenti host e porte "{0}".
Spiegazione: Questo server ObjectGrid utilizzerà l'elenco di host e porte fornite in un tentativo di connessione a un server ObjectGrid peer.
Risposta utente:nessuna

CWOBJ2505I: Tentativo di avvio su un server ObjectGrid peer utilizzando l'URL del file XML del cluster "{0}".
Spiegazione: Questo server ObjectGrid utilizzerà l'elenco di server nel file XML del cluster in un tentativo di connessione a un server ObjectGrid peer.
Risposta utente:nessuna

CWOBJ2506I: Registrazione in corso della traccia su {0}.
Spiegazione: Il file di traccia è stato impostato sulla riga comandi.
Risposta utente: Consultare il file di traccia specificato per la traccia di avvio del server ObjectGrid.

CWOBJ2507I: La specifica della traccia è impostata su {0}.
C**Spiegazione:**La specifica della traccia è stata impostata sulla riga comandi.
Risposta utente:nessuna

CWOBJ2508I: Un file di proprietà di protezione "{0}" è stato specificato e verrà utilizzato per avviare il server.
Spiegazione: Un file di proprietà di protezione è stato fornito per avviare un server protetto.
Risposta utente:nessuna

CWOBJ2509E: Timeout dopo avere atteso {0} secondi che il server venisse avviato.
Spiegazione: Il server ObjectGrid non è stato avviato nell'intervallo di timeout.
Risposta utente: Consultare il log per ricercare le eccezioni.

CWOBJ2510I: Arresto del server ObjectGrid {0}.
Spiegazione: Arresto del server ObjectGrid.
Risposta utente:

CWOBJ2511I: In attesa che il server venga arrestato.
Spiegazione: In attesa che il server ObjectGrid venga arrestato.
Risposta utente:nessuna

CWOBJ2512I: Server ObjectGrid {0} arrestato.
Spiegazione: Server ObjectGrid arrestato.
Risposta utente:nessuna

CWOBJ2513E: Time out dopo avere atteso {0} secondi che il server venisse arrestato.
Spiegazione: Il server ObjectGrid non è stato arrestato nell'intervallo di timeout.
Risposta utente: Consultare il log per ricercare le eccezioni.

CWOBJ2514I: In attesa del completamento dell'attivazione del server ObjectGrid.
Spiegazione: Il server ObjectGrid è stato avviato. In attesa della completa attivazione del server.
Risposta utente:nessuna.

CWOBJ2515E: Gli argomenti forniti non sono validi. Di seguito sono riportati gli argomenti validi.{0}{1}
Spiegazione: Gli argomenti forniti per lo script non sono validi.
Risposta utente: Immettere argomenti validi.

CWOBJ2516I: L'attivazione del server ObjectGrid è stata completata.
Spiegazione: Il server ObjectGrid è attivo e pronto per elaborare le richieste.
Risposta utente:nessuna.

CWOBJ2517I: Avvio corretto sul server Objectgrid peer
sull'host {0} e sulla porta {1}.
Spiegazione: Il server ObjectGrid è stato avviato correttamente su un server peer per richiamare le informazioni necessarie per l'avvio di questo server.
Risposta utente:nessuna

CWOBJ2407W: La proprietà {0} nella classe di plug-in {1}
non è stata impostata.
L'eccezione è {2}.
Spiegazione: Non è stato possibile impostare la proprietà per questo plug-in.
Risposta utente: Per ulteriori informazioni, consultare l'eccezione.

Informazioni particolari

I riferimenti disponibili in questa pubblicazione su prodotti, programmi o servizi IBM non implicano che si intenda renderli disponibili in tutti i paesi in cui IBM opera. Qualsiasi riferimento a programmi, prodotti o servizi IBM contenuti in questa pubblicazione non significa che soltanto tali programmi e/o prodotti possano essere usati. In sostituzione a quelli forniti da IBM, possono essere usati prodotti, programmi o servizi funzionalmente equivalenti che non comportino la violazione dei diritti di proprietà intellettuale di IBM. La valutazione e la verifica del corretto funzionamento, in caso di utilizzo di altri prodotti, eccetto quelli espressamente indicati da IBM, è ad esclusiva responsabilità dell'utente.

IBM può avere brevetti o richieste di brevetti in corso relativi a quanto trattato nella presente pubblicazione. La fornitura di questa pubblicazione non implica la concessione di alcuna licenza relativa a questi brevetti. Per ricevere informazioni sulle licenze, scrivere a:

Director of Commercial Relations
IBM Corporation
:NONE.
D-7030 Boeblingen
Deutschland

Per i licenziatari di questo programma che desiderano avere informazioni sul programma stesso per abilitare (i) lo scambio di informazioni tra programmi creati indipendentemente e altri programmi (compreso questo) e (ii) l'uso delle informazioni messe in comune, si consiglia di contattare:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attenzione: richieste di informazioni

Queste informazioni possono essere rese disponibili sulla base di condizioni contrattuali appropriate, in alcuni casi dietro pagamento di un canone.

Marchi e marchi di servizi

I seguenti termini sono marchi di IBM Corporation, negli Stati Uniti, in altri paesi o in entrambi:

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java e tutti i marchi basati su Java sono della Sun Microsystems, Inc. negli Stati Uniti, in altri paesi o in entrambi.

LINUX è un marchio di Linus Torvalds, negli Stati Uniti, in altri paesi o in entrambi.

Microsoft, Windows, Windows NT e il logo Windows sono marchi della Microsoft Corporation negli Stati Uniti, in altri paesi o in entrambi.

UNIX è un marchio registrato di The Open Group negli Stati Uniti e in altri paesi.

Nomi di altri prodotti, società o servizi possono essere marchi di altre società.