WebSphere® eXtreme Scale Versione 7.1
Programming Guide

WebSphere eXtreme Scale Guida alla programmazione

IBM

Questa edizione si riferisce alla versione 7, release 1, di WebSphere eXtreme Scale e a tutte le release e modifiche successive se non diversamente indicato nelle nuove edizioni.

Indice

Figure

Tabelle

Informazioni sul manuale *Guida alla programmazione*

L'insieme della documentazione WebSphere eXtreme Scale comprende tre volumi che forniscono le informazioni necessarie per utilizzare, programmare e gestire il prodotto WebSphere eXtreme Scale.

Libreria WebSphere eXtreme Scale

La libreria WebSphere eXtreme Scale contiene i seguenti manuali:

- Il manuale *Guida alla gestione* contiene informazioni necessarie per gli amministratori di sistema, che comprendono informazioni su come pianificare le distribuzioni dell'applicazione, pianificare la capacità, installare e configurare il prodotto, avviare ed arrestare i server, monitorare l'ambiente e renderlo sicuro.
- Il manuale *Guida alla programmazione* contiene informazioni per gli sviluppatori di applicazioni su come sviluppare le applicazioni per WebSphere eXtreme Scale utilizzando le informazioni API incluse.
- Il manuale *Panoramica sul prodotto* contiene una vista di livello superiore dei concetti WebSphere eXtreme Scale, che comprendono l'impiego di scenari d'utilizzo e di supporti didattici.

Per scaricare i manuali, andare alla WebSphere eXtreme Scale pagina della libreria.

È possibile inoltre accedere alle stesse informazioni presenti in questa libreria nel WebSphere eXtreme Scale centro informazioni.

A chi è rivolto questo manuale

Questo manuale è rivolto principalmente agli sviluppatori di applicazioni.

Struttura di questo manuale

Questo manuale contiene informazioni sui seguenti argomenti principali:

- Il **capitolo 1** contiene informazioni su come iniziare con WebSphere eXtreme Scale.
- Il **capitolo 2** contiene informazioni su come programmare WebSphere eXtreme Scale.
- Il **capitolo 3** contiene informazioni sull'accesso ai dati.
- Il **capitolo 4** contiene informazioni sui plug-in e sulle API di sistema.
- Il **capitolo 5** contiene informazioni sull'integrazione con Spring Framework.
- Il **capitolo 6** contiene informazioni sull'API di sicurezza.
- Il **capitolo 7** contiene informazioni sull'API di gestione.
- Il **capitolo 8** contiene informazioni relative a considerazioni sulle prestazioni.
- Il **capitolo 9** contiene informazioni sulla risoluzione dei problemi.
- Il **capitolo 10** contiene il glossario del prodotto.

Come ottenere aggiornamenti a questo manuale

È possibile ottenere aggiornamenti a questo manuale scaricando la versione più recente dalla WebSphere eXtreme Scale pagina della libreria.

Come inviare i commenti

Contattare il team della documentazione. Sono state trovate le informazioni richieste? Le informazioni erano accurate e complete? Inviare commenti relativi a questa documentazione via e-mail all'indirizzo wasdoc@us.ibm.com.

Capitolo 1. Introduzione a WebSphere eXtreme Scale

Una volta installato WebSphere eXtreme Scale in un ambiente standalone, utilizzare i passi riportati di seguito per un'introduzione alle relative capability come una griglia di dati in memoria.

L'installazione standalone di WebSphere eXtreme Scale include un esempio che è possibile utilizzare per verificare la propria installazione e per visualizzare il modo in cui è possibile utilizzare un client ed una griglia eXtreme Scale semplice. L'esempio iniziale è contenuto nella directory *installRoot*/ObjectGrid/gettingstarted.

L'esempio iniziale fornisce una rapida introduzione al funzionamento di base ed alle funzionalità di eXtreme Scale. L'esempio è composto da script batch e della shell progettati per l'avvio di una griglia semplice che richiede un numero ridotto di operazioni di personalizzazione. Inoltre, viene fornito un programma client, inclusa l'origine, per l'esecuzione di semplici funzioni CRUD (create, read, update, delete) sulla griglia di base.

Script e relative funzioni

Questo esempio fornisce i quattro script riportati di seguito:

Lo script env.sh|bat viene richiamato dagli altri script per impostare le variabili di ambiente necessarie. Generalmente, non è necessario modificare tale script.

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

Lo script runcat.sh|bat avvia il processo del servizio catalogo eXtreme Scale sul sistema locale.

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

Lo script runcontainer.sh|bat avvia un processo del server contenitore. È possibile eseguire questo script più volte specificando nomi server univoci per avviare qualsiasi numero di contenitori. Tali istanze possono funzionare insieme per ospitare nella griglia informazioni ridondanti e partizionate.

- UNIX Linux `./runcontainer.sh` *`nome_server_univoco`*
- Windows `runcontainer.bat` *`nome_server_univoco`*

Lo script runclient.sh|bat esegue il client CRUD semplice ed avvia l'operazione fornita.

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

Per *command*, utilizzare una delle seguenti opzioni:

- Specificare `i` per inserire *value2* nella griglia con la chiave *value1*
- Specificare `u` per aggiornare l'oggetto a cui è stata assegnata la chiave mediante *value1* su *value2*

- Specificare `d` per eliminare l'oggetto a cui è stata assegnata la chiave mediante *value1*
- Specificare `g` per richiamare e visualizzare l'oggetto a cui è stata assegnata la chiave mediante *value1*

Nota: Il file *installRoot*`/ObjectGrid/ gettingstarted/src/Client.java` è il programma client che indica come effettuare la connessione ad un server di catalogo, ottenere un'istanza ObjectGrid ed utilizzare l'API ObjectMap.

Passi di base

Effettuare le operazioni riportate di seguito per avviare la prima griglia ed eseguire un client per interagire con la griglia.

1. Aprire una finestra della riga comandi o una sessione di terminale.
2. Utilizzare il comando riportato di seguito per passare alla directory `gettingstarted`:

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

   Sostituire *installRoot* con il percorso della directory root di installazione di eXtreme Scale o con il percorso file root della versione di prova di eXtreme Scale estratta *installRoot*.
3. Eseguire lo script riportato di seguito per avviare un processo del servizio catalogo su localhost:
   - UNIX Linux `./runcat.sh`
   - Windows `runcat.bat`

   Il processo del servizio catalogo viene eseguito nella finestra di terminale corrente.
4. Aprire un'altra finestra della riga comandi o sessione di terminale ed eseguire il comando riportato di seguito per avviare un'istanza del server contenitore:
   - UNIX Linux `./runcontainer.sh server0`
   - Windows `runcontainer.bat server0`

   Il server contenitore viene eseguito nella finestra di terminale corrente. È possibile ripetere i passi 5 e 6 se si desidera avviare più istanze del server contenitore per il supporto della replica.
5. Aprire un'altra finestra della riga comandi o sessione di terminale per eseguire i comandi del client.
   - Aggiungere i dati alla griglia:
     - UNIX Linux `./runclient.sh i key1 helloWorld`
     - Windows `runclient.bat i key1 helloWorld`
   - Ricercare e visualizzare il valore:
     - UNIX Linux `./runclient.sh g key1`
     - Windows `runclient.bat g key1`
   - Aggiornare il valore:
     - UNIX Linux `./runclient.sh u key1 goodbyeWorld`
     - Windows `runclient.bat u key1 goodbyeWorld`
   - Eliminare il valore:
     - UNIX Linux `./runclient.sh d key1`

– Windows `runclient.bat d key1`

6. Utilizzare i tasti <ctrl+c> per arrestare il processo del servizio catalogo ed i server contenitore nelle rispettive finestre.

Definizione di ObjectGrid

L'esempio utilizza i file `objectgrid.xml` e `deployment.xml` contenuti nella directory *installRoot*`/ObjectGrid/gettingstarted/xml` per avviare un server contenitore. Il file `objectgrid.xml` è il file XML descrittore ObjectGrid ed il file `deployment.xml` è il file XML descrittore della politica di distribuzione ObjectGrid. Tali file, insieme, definiscono una topologia ObjectGrid distribuita.

File XML descrittore ObjectGrid

Un file XML descrittore ObjectGrid viene utilizzato per definire la struttura di ObjectGrid utilizzato dall'applicazione. Tale file include un elenco di configurazioni BackingMap. Tali BackingMap rappresentano la memoria dati reale per i dati memorizzati nella cache. Di seguito è riportato un file `objectgrid.xml` di esempio. Le prime righe del file includono l'intestazione richiesta per ciascun file XML ObjectGrid. Questo file di esempio definisce la griglia ObjectGrid con le BackingMap Map1 e Map2.

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

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

</objectGridConfig>
```

File XML descrittore della politica di distribuzione

Il file XML descrittore della politica di distribuzione viene passato ad un server contenitore ObjectGrid durante l'avvio. È necessario utilizzare una politica di distribuzione con un file XML ObjectGrid e tale politica deve essere compatibile con il file XML ObjectGrid con essa utilizzato. Per ciascun elemento objectgridDeployment nella politica di distribuzione, è necessario un elemento ObjectGrid corrispondente nel file XML ObjectGrid. Gli elementi backingMap definiti nell'elemento objectgridDeployment devono essere congruenti con gli elementi backingMap nel file XML ObjectGrid. È necessario fare riferimento ad ogni backingMap all'interno di un e solo un mapSet.

Il file XML descrittore della politica di distribuzione deve essere accoppiato con il file XML ObjectGrid corrispondente, il file `objectgrid.xml`. Nell'esempio riportato di seguito, le prime righe del file `deployment.xml` includono l'intestazione richiesta per ciascun file XML della politica di distribuzione. Il file definisce l'elemento objectgridDeployment per la griglia ObjectGrid definita nel file `objectgrid.xml`. I BackingMap Map1 e Map2 definiti nella griglia ObjectGrid sono inclusi nella serie di mappe mapSet per cui sono configurati gli attributi numberOfPartitions, minSyncReplicas e maxSyncReplicas.

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

```
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">

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

</deploymentPolicy>
```

L'attributo NumberOfPartitions dell'elemento mapSet specifica il numero di partizioni per mapSet. È un attributo facoltativo il cui valore predefinito è 1. Il numero deve essere appropriato per la capacità prevista della griglia.

L'attributo minSyncReplicas di mapSet consente di specificare il numero minimo di repliche sincrone per ciascuna partizione in mapSet. È un attributo facoltativo il cui valore predefinito è 0. L'elemento principale e la replica non sono posizionati fino a quando il dominio non è in grado di supportare il numero minimo di repliche sincrone. Per supportare il valore minSyncReplicas, è necessario un ulteriore contenitore rispetto al valore di minSyncReplicas. Se il numero di repliche sincrone è inferiore al valore di minSyncReplicas, le transazioni di scrittura non sono più consentite per tale partizione.

L'attributo maxSyncReplicas di mapSet consente di specificare il numero massimo di repliche sincrone per ciascuna partizione in mapSet. È un attributo facoltativo il cui valore predefinito è 0. Dopo che il dominio ha raggiunto questo numero di repliche sincrone per tale partizione specifica non vengono inserite ulteriori repliche sincrone per una partizione. L'aggiunta di contenitori che possono supportare questo ObjectGrid può determinare un numero incrementato di repliche sincrone se il valore maxSyncReplicas non è ancora stato raggiunto. Nell'esempio, maxSyncReplicas è impostato su 1; ciò significa che il dominio posizionerà almeno una replica sincrona. Se vengono avviate più di un'istanza del server contenitore, in una delle istanza del server contenitore verrà inserita una sola replica sincrona.

Utilizzo di ObjectGrid

Il file `Client.java` nella directory *installRoot*`/ObjectGrid/ gettingstarted/src/` è il programma client che illustra come effettuare la connessione al server di catalogo, richiedere l'istanza ObjectGrid ed utilizzare l'API ObjectMap.

Dalla prospettiva di un'applicazione client, l'utilizzo di WebSphere eXtreme Scale può essere suddiviso nelle fasi riportate di seguito.

1. Connessione al servizio di catalogo mediante la richiesta di un'istanza ClientClusterContext.
2. Richiesta di un'istanza ObjectGrid del client.
3. Richiesta di un'istanza Session.
4. Richiesta di un'istanza ObjectMap.
5. Utilizzo dei metodi di ObjectMap.

1. **Connessione al servizio catalogo ottenendo un'istanza ClientClusterContext**

   Per effettuare la connessione al server di catalogo, utilizzare il metodo connect dell'API ObjectGridManager. Il metodo connect che è utilizzato richiede solo l'endpoint del server di catalogo nel formato `hostname:port`, come ad esempio `localhost:2809`. Se la connessione al server di catalogo ha esito positivo, il metodo connect restituisce un'istanza ClientClusterContext. L'istanza

ClientClusterContext è necessaria per ottenere l'ObjectGrid dall'API ObjectGridManager. Il frammento di codice riportato di seguito illustra come effettuare la connessione ad un server di catalogo e richiedere un'istanza ClientClusterContext.

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

2. **Ottenimento di un'istanza ObjectGrid**

   Per ottenere un'istanza ObjectGrid, utilizzare il metodo getObjectGrid dell'API ObjectGridManager. Il metodo getObjectGrid richiede l'istanza ClientClusterContext ed il nome dell'istanza ObjectGrid. L'istanza ClientClusterContext si ottiene durante la connessione al server di catalogo. Il nome di ObjectGrid è `Grid` che è specificato nel file `objectgrid.xml`. Il frammento di codice di seguito riportato dimostra in che modo ottenere ObjectGrid richiamando il metodo getObjectGrid dell'API ObjectGridManager.

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

3. **Richiesta di un'istanza Session**

   È possibile ottenere un'istanza Session dall'istanza ObjectGrid ottenuta. Un'istanza Session è richiesta per ottenere l'istanza ObjectMap ed eseguire la demarcazione della transazione. Il frammento di codice di seguito riportato dimostra in che modo ottenere un'istanza Session richiamando il metodo getSession dell'API ObjectGrid.

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

4. **Richiesta di un'istanza ObjectMap**

   Una volta richiesta una Session, è possibile ottenere un'istanza ObjectMap da un'istanza Session richiamando il metodo getMap dell'API Session. È necessario passare il nome della mappa come parametro al metodo getMap per ottenere l'istanza ObjectMap. Il frammento di codice di seguito riportato dimostra in che modo ottenere ObjectMap richiamando il metodo getMap dell'API Session.

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

5. **Utilizzo dei metodi ObjectMap**

   Una volta ottenuto ObjectMap, è possibile utilizzare l'API ObjectMap. Tener presente che l'interfaccia ObjectMap è una mappa transazionale e richiede la demarcazione della transazione utilizzando i metodi begin e commit dell'API Session. Se non esiste una demarcazione della transazione esplicita nell'applicazione, le operazioni ObjectMap si eseguono con le transazioni auto-commit.

   Il frammento di codice di seguito riportato dimostra in che modo utilizzare l'API ObjectMap con la transazione auto-commit.

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

   Il frammento di codice di seguito riportato dimostra in che modo utilizzare l'API ObjectMap con una demarcazione della transazione esplicita.

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

Ulteriori informazioni

Questo esempio illustra come avviare il server di catalogo ed il server contenitore e l'utilizzo dell'API ObjectMap in un ambiente standalone. È anche possibile utilizzare l'API EntityManager.

In un ambiente WebSphere Application Server in cui è installato o abilitato WebSphere eXtreme Scale, lo scenario più comune è rappresentato da una topologia collegata in rete. In una topologia collegata in rete, il server di catalogo è

contenuto nel processo del gestore distribuzione WebSphere Application Server e ciascuna istanza WebSphere Application Server ospita un server eXtreme Scale automaticamente. Le applicazioniJava™ Platform, Enterprise Edition devono solo includere il file XML descrittore ObjectGrid ed il file XML descrittore della politica di distribuzione ObjectGrid nella directory `META-INF` di ogni modulo ed ObjectGrid diventa automaticamente disponibile. L'applicazione, quindi, può effettuare la connessione ad un server di catalogo disponibile in locale ed ottenere un'istanza ObjectGrid da utilizzare.

Capitolo 2. Preparazione per sviluppare le applicazioni

Impostare l'ambiente di sviluppo e imparare dove reperire i dettagli relativi alle interfacce di programmazione disponibili.

Interfacce di programmazione di WebSphere eXtreme Scale

WebSphere eXtreme Scale fornisce diverse funzioni che sono accedute programmaticamente con il linguaggio di programmazione Java attraverso API (application programming interfaces) ed interfacce di programmazione di sistema.

API di WebSphere eXtreme Scale

Quando si utilizzano le API di eXtreme Scale, è necessario distinguere tra operazioni transazionali e non transazionali. Un'operazione transazionale è un'operazione eseguita all'interno di una transazione. Le API ObjectMap, EntityManager, Query e DataGrid sono API transazionali contenute all'interno di una Session che rappresenta un contenitore transazionale. Le operazioni non transazionali non hanno nulla a che vedere con una transazione, un esempio è rappresentato dalle operazioni di configurazione.

Le API ObjectGrid, BackingMap ed i plug-in sono non transazionali. Le API ObjectGrid, BackingMap ed altre API di configurazione sono catalogate come API ObjectGrid Core. I plug-in servono per la personalizzazione della cache in modo da ottenere le funzioni che si desiderano e sono catalogati come API di Programmazione di sistema. Un plug-in in eXtreme Scale è un componente che fornisce determinati tipi di funzioni ai componenti di eXtreme Scale collegabili, compresi gli ObjectGrid e le BackingMap. Un funzione rappresenta una caratteristica specifica di un componente di eXtreme Scale, come ObjectGrid, Session, BackingMap, ObjectMap e così via. Solitamente, le funzioni sono configurabili con le API di configurazione. I plug-in possono essere incorporati, ma in alcuni casi, potrebbe essere necessario sviluppare i propri plug-in.

È solitamente possibile configurare ObjectGrid e BackingMap in modo che soddisfino i requisiti dell'applicazione. Quando l'applicazione ha requisiti speciali, si può considerare l'utilizzo di plug-in specializzati. WebSphere eXtreme Scale potrebbe contenere plug-in incorporati che soddisfino i requisiti dell'utente. Ad esempio, se si necessita di un modello di replica peer-to-peer tra due istanze ObjectGrid locali o due griglie eXtreme Scale distribuite, è disponibile uno JMSObjectGridEventListener incorporato. Se nessuno dei plug-in incorporati risulta adatto a risolvere i propri problemi di business, si può fare riferimento alle API di Programmazione di sistema per creare i propri plug-in.

ObjectMap è una semplice API basata su mappe. Se gli oggetti memorizzati nella cache sono semplici e senza alcuna relazione, l'API ObjectMap è l'ideale per la propria applicazione. Se sono presenti relazioni tra oggetti, utilizzare l'API EntityManagerm, che supporta relazioni simili a grafici.

La Query è un meccanismo potente per trovare dati nell'ObjectGrid. Sia Session che EntityManager forniscono le capability di query tradizionali.

L'API DataGrid è una capability di calcolo potente in un ambiente eXtreme Scale distribuito che coinvolge diverse macchine, repliche e partizioni. Le applicazioni

possono eseguire le logiche di business in parallelo in tutti i nodi dell'ambiente eXtreme Scale distribuito. Le applicazioni possono ottenere l'API DataGrid attraverso l'API ObjectMap.

7.0.0.0 FIX 2+ Il servizio dati REST di WebSphere eXtreme Scale è un servizio HTTP di Java compatibile con i servizi dati WCF di Microsoft® (precedentemente noti come servizi dati ADO.NET) ed implementa il protocollo OData (Open Data). Il servizio dati REST consente a qualsiasi client HTTP di accedere una griglia eXtreme Scale. È compatibile con il supporto dei servizi di dati WCF fornito con Microsoft .NET Framework 3.5 SP1. Le applicazioni RESTful possono essere sviluppate con la ricca strumentazione fornita da Microsoft Visual Studio 2008 SP1. Per ulteriori dettagli, fare riferimento alla guida eXtreme Scale REST data service user guide.

Programma di caricamento classe e considerazioni sul percorso di classe.

Poiché, per impostazione predefinita, eXtreme Scale memorizza oggetti Java nella cache, è necessario definire le classi nel percorso di classe ovunque venga eseguito l'accesso ai dati.

Specificamente, i processi client e contenitore di eXtreme Scale devono includere le classi o i file JAR nel percorso di classe quando vengono avviati. Quando si progetta un'applicazione per l'utilizzo con eXtreme Scale, separare la logica di business dagli oggetti di dati persistenti.

Per ulteriori informazioni consultare la sezione Caricamento classe nel Centro informazioni di WebSphere Application Server Network Deployment.

Per considerazioni nell'ambito delle impostazioni di Spring Framework, consultare la sezione contenuta nell'argomento relativo all'integrazione con Spring framework in *Guida alla programmazione*.

Per le impostazioni relative all'utilizzo dell'agent di strumentazione di WebSphere eXtreme Scale, consultare l'argomento relativo all'agent di strumentazione in *Guida alla programmazione*.

Impostazione di un ambiente di sviluppo

Configurare un ambiente di sviluppo integrato basato su Eclipse per creare ed eseguire un'applicazione Java SE con eXtreme Scale.

Procedura

- Configurare Eclipse per creare ed eseguire un'applicazione Java SE con eXtreme Scale.
  1. Definire una libreria utente per consentire all'applicazione di fare riferimento alle interfacce di programmazione dell'applicazione eXtreme Scale
     a. In ambiente IBM® Rational Application Developer, fare clic su **Finestra > Preferenze**.
     b. Espandere la sezione Percorso di creazione **Java >** e selezionare **Librerie utente**. Fare clic su **Nuova**.
     c. Selezionare la libreria utente eXtreme Scale. Fare clic su **Aggiungi JAR**.

1) Ricercare e selezionare i file `objectgrid.jar` e `cglib.jar` dalla directory `wxs_root/lib`. Fare clic su **OK**.
2) Per includere Javadoc per le API ObjectGrid, selezionare la location Javadoc per il file `objectgrid.jar` che è stato aggiunto nei passi precedenti. Fare clic su **Modifica**. Nella casella di percorso della location, immettere il seguente indirizzo web:

   ```
   http://www.ibm.com/developerworks/wikis/extremescale/docs/api/
   ```

d. Fare clic su **OK** per attivare le impostazioni e chiudere la finestra Preferenze.

Le librerie eXtreme Scale ora sono nel percorso di creazione per il progetto.

2. Aggiungere la libreria utente al proprio progetto Java.
   a. Da Esplora package, fare clic con il tasto destro del mouse sul progetto e selezionare **Proprietà**.
   b. Selezionare la scheda **Librerie**.
   c. Fare clic su **Aggiungi libreria**.
   d. Selezionare **Libreria utente** . Fare clic su **Avanti**.
   e. Selezionare la libreria utente eXtreme Scale che è stata configurata precedentemente.
   f. Fare clic su **OK** per applicare le modifiche e chiudere la finestra Proprietà.

- Eseguire un'applicazione Java SE con eXtreme Scale con Eclipse. Creare una configurazione di esecuzione per eseguire la propria applicazione.
  1. Configurare Eclipse per creare ed eseguire un'applicazione Java SE con eXtreme Scale. Dal menu **Esegui** selezionare **Configurazioni di esecuzione** .
  2. Fare clic con il tasto destro del mouse su Categoria dell'applicazione Java e selezionare **Nuova**.
  3. Selezionare la nuova configurazione di esecuzione denominata *Nuova_Configurazione*.
  4. Configurare il profilo.
     – **Progetto** (sulla pagina principale con scheda): *nome_del proprio_progetto*
     – **Classe principale** (sulla pagina principale con scheda): *propria_classe_principale*
     – **Argomenti VM** (sulla pagina degli argomenti con scheda): -Djava.endorsed.dirs=`wxs_root`/lib/endorsed

     Spesso si verificano problemi con gli **Argomenti VM** perché il percorso `java.endorsed.dirs` deve essere un percorso assoluto senza alcuna variabile o tasti di scelta rapida.

     Altri problemi comuni di impostazione riguardano ORB (Object Request Broker). Potrebbe essere visualizzato il seguente errore. Fare riferimento a Configurazione di un Object Request Broker personalizzato per ulteriori informazioni:

     ```
     Caused by: java.lang.RuntimeException: The ORB that comes
     with the Sun Java implementation does not work with
     ObjectGrid at this time.
     ```

     Se non si dispone di `objectGrid.xml` o `deployment.xml` accessibile all'applicazione, potrebbe essere visualizzato il seguente errore:

     ```
     Exception in thread "P=211046:O=0:CT" com.ibm.websphere.objectgrid.
         ObjectGridRuntimeException: Cannot start OG container at
         Client.startTestServer(Client.java:161) at Client.
         main(Client.java:82) Caused by: java.lang.IllegalArgumentException:
     ```

```
The objectGridXML must not be null at com.ibm.websphere.objectgrid.
deployment.DeploymentPolicyFactory.createDeploymentPolicy
 (DeploymentPolicyFactory.java:55) at Client.startTestServer(Client.
java:154) .. 1 more
```

5. Fare clic su **Applica** e chiudere la finestra o fare clic su **Esegui**.

Esecuzione di un'applicazione client o server WebSphere eXtreme Scale con Tomcat Apache in Rational Application Developer

Se si ha un'applicazione client o server, utilizzare le stesse operazioni di base per eseguire l'applicazione Tomcat Apache in Rational Application Developer. Per un'applicazione client, si desidera configurare ed eseguire un'applicazione web per utilizzare un client WebSphere eXtreme Scale in Rational Application Developer. Seguire queste istruzioni per creare un progetto web per l'esecuzione di un servizio catalogo WebSphere eXtreme Scale o di un contenitore. Per un'applicazione server, si desidera abilitare un'applicazione Java EE nell'interfaccia Rational Application Developer con un'installazione autonoma di WebSphere eXtreme Scale. Seguire queste istruzioni per configurare un progetto dell'applicazione Java EE per l'utilizzo della libreria client WebSphere eXtreme Scale.

Prima di iniziare

Installare la versione di prova WebSphere eXtreme Scale oppure il prodotto completo.

- Installare la versione autonoma del prodotto WebSphere eXtreme Scale Versione 7.1.
- Scaricare ed estrarre la versione di prova di WebSphere eXtreme Scale.
- Installare Tomcat Apache 6.0 o successivo.
- Installare Rational Application Developer e creare un'applicazione web Java EE.

Procedura

1. Aggiungere la libreria di runtime di WebSphere eXtreme Scale al percorso di creazione Java EE.

   Applicazione client In questo scenario, si desidera configurare ed eseguire un'applicazione web per utilizzare un client WebSphere eXtreme Scale in Rational Application Developer.

   a. **Finestra** → **Preferenze** → **Java** → **Percorso di creazione** → **Librerie utente**. Fare clic su **Nuova**.

   b. Immettere un **Nome libreria utente** di `eXtremeScaleClient`, e fare clic su **OK**.

   c. Fare clic su **Aggiungi Jar...** e selezionare il file `wxs_home/lib/ogclient.jar`. Fare clic su **Apri**.

   d. Opzionale: (Facoltativo) per aggiungere Javadoc, selezionare location Javadoc e fare clic su **Modifica....** Nel percorso della location Javadoc, è possibile immettere l'URL della documentazione API oppure scaricare la documentazione API.

      - Per utilizzare la documentazione API in linea, immettere `http://www.ibm.com/developerworks/wikis/extremescale/docs/api/` nel percorso della location Javadoc.
      - Per scaricare la documentazione API, andare alla pagina WebSphere eXtreme Scale Scarica documentazione API. Per il percorso della location Javadoc, immettere la location in cui scaricare in locale.

   e. Fare clic su **OK**.

f. Fare clic su **OK** per chiudere la finestra di dialogo Librerie utente.
g. Fare clic su **Progetto** → **Proprietà**.
h. Fare clic su **Percorso di creazione Java**.
i. Fare clic su **Aggiungi libreria** .
j. Selezionare **Libreria utente**. Fare clic su **Avanti**.
k. Selezionare la libreria **eXtremeScaleClient** e fare clic su **Fine**.
l. Fare clic su **OK** per chiudere la finestra di dialogo **Proprietà del progetto**.

Applicazione server In questo scenario, si desidera configurare ed eseguire un'applicazione web per eseguire un server integrato WebSphere eXtreme Scale in Rational Application Developer.

a. Fare clic su **Finestra** → **Preferenze** → **Java** → **Percorso di creazione** → **Librerie utente**. Fare clic su **Nuova**.
b. Immettere un **Nome libreria utente** di `eXtremeScale`, e fare clic su **OK**.
c. Fare clic su **Aggiungi Jar...** e selezionare wxs_home/lib/objectgrid.jar. Fare clic su Apri.
d. (Facoltativo) Per aggiungere Javadoc, selezionare la location Javadoc e fare clic su **Modifica....** Nel percorso della location Javadoc, immettere `http://www.ibm.com/developerworks/wikis/extremescale/docs/api/`.
e. Fare clic su **OK**.
f. Fare clic su **OK** per chiudere la finestra di dialogo Librerie utente.
g. Fare clic su **Progetto** → **Proprietà**.
h. Fare clic su **Percorso di creazione Java**.
i. Fare clic su **Aggiungi libreria** .
j. Selezionare **Libreria utente**. Fare clic su **Avanti**.
k. Selezionare la libreria **eXtremeScaleClient** e fare clic su **Fine**.
l. Fare clic su **OK** per chiudere la finestra di dialogo **Proprietà del progetto**.

2. Definire il server Tomcat per il progetto.
   a. Assicurarsi di essere nella prospettiva J2EE e fare clic sulla scheda **Server** nella parte inferiore del riquadro. È possibile anche fare clic su **Finestra** → **Mostra vista** → **Server**.
   b. Fare clic con il tasto destro del mouse nel riquadro Server e scegliere **Nuovo** → **Server**.
   c. Scegliere il server **Apache, Tomcat v6.0**. Fare clic su **Avanti**.
   d. Fare clic su **Sfoglia..** e selezionare root_tomcat. Fare clic su **OK**.
   e. Fare clic su **Avanti**.
   f. Selezionare l'applicazione Java EE nel riquadro a sinistra Disponibile e fare clic su **Aggiungi>** per spostarla nel riquadro di destra Configurata sul server e fare clic su **Fine**.
3. Risolvere qualsiasi errore ancora esistente per il progetto. Tutti gli errori per il progetto dovrebbero essere visualizzati nel riquadro Problemi. Se non lo sono, eseguire i passi di seguito riportati.
   a. Fare clic su **Progetto** → **Ripulisci** → *nome_progetto*. Fare clic su **OK**. Creare il progetto.
   b. Fare clic con il tasto destro del mouse sul progetto Java EE che si desidera e scegliere **Percorso di creazione** → **Configura percorso di creazione**.
   c. Fare clic sulla scheda **Librerie**. Assicurarsi che il percorso sia configurato correttamente.

- **Per le applicazioni client:** Assicurarsi che Tomcat Apache, eXtremeScaleClient e Java 1.5 JRE si trovino nel percorso.
- **Per le applicazioni server:** Assicurarsi che Tomcat Apache, eXtremeScale e Java 1.5 JRE si trovino nel percorso.

4. Creare una configurazione di esecuzione per eseguire l'applicazione.
   a. Dal menu **Esegui** selezionare **Configurazioni di esecuzione**.
   b. Fare clic con il tasto destro del mouse sulla categoria dell'applicazione Java e selezionare **Nuova**.
   c. Selezionare la nuova configurazione di esecuzione denominata *Nuova_Configurazione*.
   d. Configurare il profilo.
      - **Progetto** (sulla pagina principale con scheda): *nome_del proprio_progetto*
      - **Classe principale** (sulla pagina principale con scheda): *propria_classe_principale*
      - **Argomenti VM** (sulla pagina con scheda Argomenti): `-Djava.endorsed.dirs=wxs_root/lib/endorsed`

      Spesso si verificano problemi con gli **Argomenti VM** perché il percorso `java.endorsed.dirs` deve essere un percorso assoluto senza alcuna variabile o tasti di scelta rapida.

      Altri problemi comuni di impostazione riguardano ORB (Object Request Broker). Potrebbe essere visualizzato il seguente errore. Fare riferimento a Configurazione di un Object Request Broker personalizzato per ulteriori informazioni:

      ```
      Caused by: java.lang.RuntimeException: The ORB that comes with the Sun
      Java implementation does not work with ObjectGrid at this time.
      ```

      Se non si dispone dei file `objectGrid.xml` o `deployment.xml` accessibili per l'applicazione, potrebbe essere visualizzato il seguente errore:

      ```
      Exception in thread "P=211046:O=0:CT" com.ibm.websphere.objectgrid.ObjectGridRuntimeException
      Cannot start OG container
       at Client.startTestServer(Client.java:161)
       at Client.main(Client.java:82)
      Caused by: java.lang.IllegalArgumentException: The objectGridXML must not be null
       at com.ibm.websphere.objectgrid.deployment.DeploymentPolicyFactory.createDeploymentPolicy
        (DeploymentPolicyFactory.java:55)
       at Client.startTestServer(Client.java:154)
       ... 1 more
      ```

5. Fare clic su **Applica** e chiudere la finestra o fare clic su **Esegui**.

Operazioni successive

Se si è configurata ed eseguita un'applicazione web per utilizzare un client WebSphere eXtreme Scale in Rational Application Developer, ora sviluppare una servlet che utilizzi le API WebSphere eXtreme Scale per memorizzare e recuperare i dati da una griglia WebSphere eXtreme Scale remota.

Se si è abilitata un'applicazione Java EE nell'interfaccia Rational Application Developer con un'installazione autonoma di WebSphere eXtreme Scale, ora sviluppare una servlet che utilizzi le API di sistema per avviare ed arrestare i servizi catalogo WebSphere eXtreme Scale.

Esecuzione di un'applicazione client o server integrata con WebSphere Application Server in Rational Application Developer

Configurare ed eseguire un'applicazione Java EE con un client o server WebSphere eXtreme Scale con il runtime WebSphere Application Server integrato in Rational Application Developer. Se si sta configurando un server, avviando WebSphere Application Server si avvia automaticamente WebSphere eXtreme Scale.

Prima di iniziare

I passi che seguono si riferiscono a WebSphere Application Server Versione 7.0 con Rational Application Developer Versione 7.5. I passi che seguono potrebbero variare se si sta utilizzando una differente versione di questi prodotti.

Installare Rational Application Developer con il client o server WebSphere Application Server con le estensioni per l'ambiente di test.

Installare il client o server WebSphere eXtreme Scale nell'ambiente di test di WebSphere Application Server, Versione 7.0 nella directory *rad_home*\runtimes\base_v7.

Procedura

1. Definire il server eXtreme Scale che è integrato con WebSphere Application Server per il proprio progetto.
   a. Nella prospettiva J2EE, fare clic su **Finestra > Mostra vista > Server.**
   b. Fare clic con il tasto destro del mouse nel riquadro **Server**. Scegliere **Nuovo > Server**.
   c. Scegliere **IBM WebSphere Application Server v7.0**. Fare clic su Avanti.
   d. Selezionare un profilo da utilizzare.Quello predefinito è was70profile1.
   e. Immettere il nome del server. Quello predefinito è server1.
   f. Fare clic su **Avanti**.
   g. Selezionare l'applicazione Java EE nel riquadro **Disponibile**. Fare clic su **Aggiungi>** per spostarla nel riquadro **Configurata** sul server. Fare clic su **Fine**.
2. Per eseguire l'applicazione Java EE, avviare il server delle applicazioni. Fare clic con il tasto destro del mouse su **WebSphere Application Server v7.0** e selezionare **Avvia**.

Capitolo 3. Accesso ai dati con le applicazioni client

Dopo la configurazione dell'ambiente di sviluppo è possibile procedere con lo sviluppo delle applicazioni che creano, accedono e gestiscono i dati nella griglia dei dati.

Informazioni su questa attività

Dalla prospettiva di un'applicazione client, l'utilizzo di WebSphere eXtreme Scale prevede le seguenti fasi principali:

- Connessione al servizio di catalogo mediante la richiesta di un'istanza ClientClusterContext.
- Richiesta di un'istanza ObjectGrid del client.
- Richiesta di un'istanza Session.
- Richiesta di un'istanza ObjectMap.
- Utilizzo dei metodi di ObjectMap.

Interfaccia ObjectGrid

I seguenti metodi consentono di interagire con un'istanza ObjectGrid.

Creazione ed inizializzazione

Consultare l'argomento relativo all'interfaccia ObjectGridManager per i passi necessari alla creazione di un'istanza ObjectGrid. Esistono due metodi distinti per creare un'istanza ObjectGrid: programmaticamente o con i file di configurazione XML. Consultare la documentazione API per ulteriori informazioni.

Metodi get o set e factory

Prima di inizializzare l'istanza ObjectGrid è è necessario chiamare un qualsiasi metodo set. Se lo si chiama dopo il metodo initialize, ne risulterà un'eccezione java.lang.IllegalStateException. Ciascuno dei metodi getSession dell'interfaccia ObjectGrid chiama implicitamente anche il metodo initialize. Per cui, si dovranno chiamare i metodi set prima di chiamare qualsiasi metodo getSession. L'unica eccezione a questa regola è rappresentata dall'impostazione, dall'aggiunta o dalla rimozione di oggetti ObjectGridEventListener. Questi oggetti possono essere elaborati dopo il completamento dell'elaborazione dell'inizializzazione.

L'interfaccia ObjectGrid contiene i seguenti metodi principali.

Tabella 1. Interfaccia ObjectGrid. Metodi principali di ObjectGrid.

Metodo	Descrizione
BackingMap defineMap(String name);	defineMap: è un metodo factory per definire BackingMap denominate in modo univoco. Per ulteriori informazioni riguardo le mappe di backup, consultare Interfaccia BackingMap.
BackingMap getMap(String name);	getMap: restituisce una BackingMap precedentemente definita chiamando defineMap. Utilizzando questo metodo, è possibile configurare la BackingMap, se non è stata già configurata attraverso la configurazione XML.
BackingMap createMap(String name);	createMap: crea una BackingMap, ma non la memorizza nella cache in modo da farla utilizzare da questa ObjectGrid. Utilizzare questo metodo con il metodo setMaps(List) dell'interfaccia ObjectGrid, che memorizza nella cache le BackingMap per l'utilizzo con questa ObjectGrid. Utilizzare questi metodi quando si configura un ObjectGrid con Spring Framework.

Tabella 1. Interfaccia ObjectGrid (Continua). Metodi principali di ObjectGrid.

Metodo	Descrizione
void setMaps(List mapList);	setMaps: cancella qualsiasi BackingMap precedentemente definita su questa ObjectGrid e la sostituisce con l'elenco di BackingMap fornito.
public Session getSession() throws ObjectGridException, TransactionCallbackException;	getSession: restituisce Session, che fornisce le funzionalità begin, commit e rollback per un'unità di lavoro. Per ulteriori informazioni sugli oggetti Session, consultare l'interfaccia Session.
Session getSession(CredentialGenerator cg);	getSession(CredentialGenerator cg): esegue il get di Session con un oggetto CredentialGenerator. Questo metodo può essere chiamato solo dal client ObjectGrid in un ambiente client server.
Session getSession(Subject subject);	getSession(Subject subject): consente l'uso di uno specifico oggetto Subject piuttosto che quello configurato sull'ObjectGrid per effettuare il get di una Session.
void initialize() throws ObjectGridException;	initialize: ObjectGrid viene inizializzata e resa disponibile per l'utilizzo generale. Questo metodo viene chiamato implicitamente quando si chiama il metodo getSession, se ObjectGrid non si trova nello stato inizializzato.
void destroy();	destroy: il framework viene disassemblato e non può essere utilizzato dopo che questo metodo è stato chiamato.
void setTxTimeout(int timeout);	setTxTimeout: utilizzare questo metodo per impostare la quantità di tempo, in secondi, concessa ad una transazione avviata da una sessione, creata da questa istanza ObjectGrid, per giungere al completamento. Se la transazione non giunge al completamento nel tempo definito, la sessione che ha avviato la transazione viene contrassegnata come "timed out". Ciò comporta che, al metodo ObjectMap richiamato dalla sessione in timeout, venga restituita un'eccezione . La sessione viene contrassegnata come "rollback only", ciò comporterà il rollback della transazione, anche se l'applicazione chiama il metodo commit invece di quello rollback dopo aver ricevuto l'eccezione TransactionTimeoutException. Un valore di timeout pari a 0 indica che la transazione ha a disposizione un tempo illimitato per giungere al completamento. La transazione non andrà mai in timeout se si utilizza un valore di timeout pari a 0. Se questo metodo non viene chiamato, qualsiasi sessione restituita dal metodo getSession di questa interfaccia, per impostazione predefinita, avrà un valore di timeout della transazione pari a 0. Un'applicazione potrà sostituire l'impostazione di timeout della transazione per la singola sessione utilizzando il metodo setTransactionTimeout dell'interfaccia com.ibm.websphere.objectgrid.Session. In uno scenario distribuito il timeout della transazione può anche essere configurato con il file `objectGrid.xml`.
int getTxTimeout();	getTxTimeout: restituisce il valore di timeout della transazione in secondi. Questo metodo restituisce lo stesso valore passato come parametro di timeout nel metodo setTxTimeout. Se il metodo setTxTimeout non è stato chiamato, questo metodo restituirà 0 per indicare che alla transazione è stato concesso un tempo illimitato per giungere a completamento.
public int getObjectGridType();	Restituisce il tipo di ObjectGrid. Il valore restituito è equivalente ad una delle costanti dichiarate su questa interfaccia: LOCAL, SERVER, o CLIENT.
public void registerEntities(Class[] entities);	Registra una o più entità in base ai metadati della classe. La registrazione dell'entità è richiesta prima dell'inizializzazione di ObjectGrid per effettuare il bind di un entità con una BackingMap e qualsiasi indice definito. Questo metodo può essere chiamato più volte.
public void registerEntities(URL entityXML);	Registra una o più entità da un file XML di entità. La registrazione dell'entità è richiesta prima dell'inizializzazione di ObjectGrid per effettuare il bind di un entità con una BackingMap e qualsiasi indice definito. Questo metodo può essere chiamato più volte.
void setQueryConfig(QueryConfig queryConfig);	Imposta l'oggetto QueryConfig per questa ObjectGrid. Un oggetto QueryConfig fornisce la configurazione della query per consentire l'esecuzione delle query di oggetto sulle mappe in questa ObjectGrid.
//Parole chiave	
void associateKeyword(Serializable parent, Serializable child);	Obsoleta: usa la funzione query o Index per effettuare il get di oggetti con attributi specifici. Il metodo associateKeyword fornisce un meccanismo di invalidazione basato su parole chiave. Questo metodo collega tra loro le due parole chiave in una relazione direzionale. Se la parent viene invalidata, anche la child lo sarà. L'invalidazione della child non avrà alcun impatto sulla parent.
//Sicurezza	
void setSecurityEnabled()	setSecurityEnabled: abilita la sicurezza. Per impostazione predefinita la sicurezza è disabilitata.

Tabella 1. Interfaccia ObjectGrid (Continua). Metodi principali di ObjectGrid.

Metodo	Descrizione
void setPermissionCheckPeriod(long period);	setPermissionCheckPeriod: questo metodo richiede un unico parametro che indica la frequenza con cui si dovranno verificare le autorizzazioni utilizzate per concedere un accesso al client. Se il parametro è 0, tutti i metodi richiedono il metodo di autorizzazione, autorizzazione JAAS o personalizzata, per verificare se il soggetto corrente ha le autorizzazioni. Questa strategia potrebbe causare dei problemi di prestazioni a seconda dell'autorizzazione implementata. Tuttavia, questo tipo di autorizzazione è disponibile se necessaria. In alternativa, un valore del parametro minore di 0 indica il numero di millisecondi per cui una serie di autorizzazioni deve essere memorizzata in cache prima di ritornare al meccanismo di autorizzazione per l'aggiornamento. Questo parametro fornisce prestazioni molto migliori, ma se, in questo periodo di tempo, si modificano le autorizzazioni di backend l'ObjectGrid potrebbe consentire o negare l'accesso anche se il provider di sicurezza del backend è stato modificato.
void setAuthorizationMechanism(int authMechanism);	setAuthorizationMechanism: imposta il meccanismo di autorizzazione. Quello predefinito è SecurityConstants.JAAS_AUTHORIZATION.
setMapAuthorization(MapAuthorization ma);	Obsoleto: utilizzare il metodo setObjectGridAuthorization (ObjectGridAuthorization) invece di collegare le autorizzazioni personalizzate. setMapAuthorization: imposta il plug-in MapAuthorization per questa istanza di ObjectGrid. Questo plug-in può essere utilizzato per autorizzare gli accessi a ObjectMap o JavaMap ai principal contenuti nell'oggetto Subject. Un'implementazione tipica di questo plug-in è quella che consente di recuperare, da un oggetto Subject, i Principal e quindi di verificare se gli sono state concesse le autorizzazioni.
setSubjectSource(SubjectSource ss);	setSubjectSource: imposta il plug-in SubjectSource. Questo plug-in può essere utilizzato per ottenere un oggetto Subject che rappresenta il client ObjectGrid. Questo Subject viene utilizzato per le autorizzazioni ObjectGrid. Il metodo SubjectSource.getSubject viene chiamato dal runtime di ObjectGrid quando si utilizza il metodo ObjectGrid.getSession per ottenere una sessione e l'abilitazione della sicurezza. Questo plug-in è utile per un client già autenticato: esso potrà recuperare l'oggetto Subject autenticato e quindi passarlo all'istanza ObjectGrid. Non sarà necessaria un'altra autenticazione.
setSubjectValidation(SubjectValidation sv);	setSubjectValidation: imposta il plug-in SubjectValidation per questa istanza ObjectGrid. Questo plug-in può essere utilizzato per confermare se un soggetto javax.security.auth.Subject passato all'ObjectGrid è un soggetto valido, che non è stato manomesso. Un'implementazione di questo plug-in necessita del supporto del creatore dell'oggetto Subject, perché egli è l'unico a sapere se l'oggetto è stato manomesso. Tuttavia, il creatore del Subject potrebbe non sapere se l'oggetto è stato manomesso. In questo caso è bene non utilizzare questo plug-in.
void setObjectGridAuthorization (ObjectGridAuthorization ogAuthorization);	Imposta ObjectGridAuthorization per questa istanza di ObjectGrid. Passando null a questo metodo si rimuoverà un oggetto ObjectGridAuthorization precedentemente impostato da un'invocazione di questo metodo e si indica che questo <code>ObjectGrid</code> non è associato ad un oggetto ObjectGridAuthorization. Questo metodo deve essere utilizzato solo quando la sicurezza di ObjectGrid è abilitata. Se la sicurezza è disabilitata, l'oggetto ObjectGridAuthorization non sarà utilizzato. Per autorizzare l'accesso ad un ObjectGrid ed alle mappe si potrà utilizzare il plug-in ObjectGridAuthorization. Per quanto riguarda XD 6.1, la setMapAuthorization è obsoleta, si consiglia l'uso di setObjectGridAuthorization. Tuttavia, se si utilizzano entrambi i plug-in MapAuthorization e ObjectGridAuthorization, ObjectGrid utilizzerà il plug-in MapAuthorization fornito per autorizzare l'accesso alle mappe, anche se è obsoleto.

Interfaccia ObjectGrid: plug-in

L'interfaccia ObjectGrid ha diversi punti di collegamento facoltativi per interazioni estendibili.

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

- ObjectGridEventListener: un'interfaccia ObjectGridEventListener viene utilizzata per ricevere notifiche quando si verificano eventi significativi sull'ObjectGrid. Questi eventi includono l'inizializzazione di ObjectGrid, l'avvio di una

transazione, il termine di una transazione e la distruzione di ObjectGrid. Per ascoltare questi eventi, è necessario creare una classe che implementi l'interfaccia ObjectGridEventListener ed aggiungerla all'ObjectGrid. I listener sono associati a ciascuna sessione. Consultare le interfacce Listener e Session per ulteriori informazioni.

- TransactionCallback: un'interfaccia del listener TransactionCallback consente agli eventi transazionali quali begin, commit e rollback di inviare segnali a questa interfaccia. Solitamente, un'interfaccia listener TransactionCallback viene utilizzata con un programma di caricamento. Per ulteriori informazioni, consultare plug-in TransactionCallback e programmi di caricamento. Questi eventi potranno quindi essere utilizzati per coordinare le transazioni con una risorsa esterna oppure all'interno di più programmi di caricamento.
- reserveSlot: consente ai plug-in su questo ObjectGrid di riservare slot da utilizzare in istanze di oggetti che hanno slot quali TxID.
- SubjectValidation. Se è abilitata la sicurezza, questo plug-in può essere utilizzato per convalidare una classe javax.security.auth.Subject passata all'ObjectGrid.
- MapAuthorization. Se è abilitata la sicurezza, questo plug-in può essere utilizzato per autorizzare gli accessi a ObjectMap ai principal rappresentati dall'oggetto Subject.
- SubjectSource: se è abilitata la sicurezza, questo plug-in può essere utilizzato per ottenere un oggetto Subject che rappresenti il client ObjectGrid. Questo soggetto viene quindi utilizzato per l'autorizzazione ObjectGrid.

Per ulteriori informazioni sui plug-in, consultare l'introduzione ai plug-in in *Guida alla programmazione*.

Interfaccia BackingMap

Ciascuna istanza ObjectGrid contiene una raccolta di oggetti BackingMap. Utilizzare il metodo defineMap oppure il metodo createMap dell'interfaccia ObjectGrid per denominare ed aggiungere ciascuna BackingMap ad un'istanza ObjectGrid. Questi metodi restituiscono un'istanza BackingMap che viene quindi utilizzata per definire il comportamento di una Mappa individuale. Una BackingMap può essere considerata come una cache in memoria dei dati di cui si è eseguito il commit per una mappa individuale.

Interfaccia Session

L'interfaccia Session viene utilizzata per avviare una transazione e per ottenere l'ObjectMap o il JavaMap richiesto per eseguire le interazioni transazionali tra un'applicazione e l'oggetto BackingMap. Tuttavia, le modifiche della transazione non vengono applicate finché non si esegue il commit della transazione. Una BackingMap può essere considerata come una cache in memoria dei dati di cui si è eseguito il commit per una mappa individuale. Per ulteriori informazioni, consultare le informazioni relative all'utilizzo di sessioni per accedere i dati in *Guida alla programmazione*.

L'interfaccia BackingMap fornisce metodi per l'impostazione degli attributi di BackingMap. Alcuni dei metodi impostati consentono l'estendibilità di una BackingMap attraverso diversi plug-in personalizzati. Consultare il seguente elenco dei metodi impostati per l'impostazione degli attributi e per fornire il supporto dei plug-in personalizzati:

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

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

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

Esiste un metodo get corrispondente per ciascun metodo impostato elencato.

Attributi BackingMap

Ciascuna BackingMap ha i seguenti attributi che è possibile impostare per modificare o controllare il comportamento della BackingMap:

- ReadOnly: quest'attributo indica se la mappa è in sola lettura oppure in lettura e scrittura. Se l'attributo non viene mai impostato la mappa per impostazione predefinita sarà in lettura e scrittura. Quando una BackingMap è in sola lettura, ObjectGrid, quando possibile, ottimizza le prestazioni per la sola lettura.
- NullValuesSupported: questo attributo indica se è possibile inserire un valore 'null' nella mappa. Se questo attributo non viene impostato la mappa non supporta i valori nulli. Se i valori nulli sono supportati dalla mappa, un'operazione di get che restituisca null potrebbe significare che il valore è nullo o che la mappa non contiene la chiave specificata dall'operazione get.
- LockStrategy: questo attributo determina se BackingMap utilizza un gestore dei blocchi. Se si utilizza un gestore del blocco, l'attributo LockStrategy viene utilizzato per indicare se per il blocco delle voci della mappa si usa un approccio a blocco ottimistico o pessimistico. Se questo attributo non è impostato, viene utilizzata la LockStrategy ottimistica. Consultare l'argomento relativo ai Blocchi per dettagli sulle strategie di blocco supportate.
- CopyMode: questo attributo determina se la BackingMap effettua una copia dell'oggetto valore al momento della lettura di un valore dalla mappa o se esso viene inserito nella BackingMap al momento del ciclo di commit di una transazione. Sono supportate diverse modalità di copia per consentire all'applicazione di ottenere il giusto compromesso tra prestazioni ed integrità dei dati. Se questo attributo non viene impostato, verrà utilizzata la modalità di copia COPY_ON_READ_AND_COMMIT. Questa modalità di copia non ha le prestazioni migliori, ma assicura la migliore protezione per quanto riguarda l'integrità dei dati. Se la BackingMap viene associata ad un'entità API EntityManager, le impostazioni CopyMode non avranno alcun effetto, a meno che il valore non sia impostato su COPY_TO_BYTES. Se viene impostata qualsiasi altra modalità di copia, esso verrà sempre impostato su NO_COPY. Per

sostituire il CopyMode della mappa di entità, utilizzare il metodo ObjectMap.setCopyMode. Per ulteriori informazioni sulle modalità di copia, consultare le informazioni relative alle migliori pratiche per il metodo CopyMode in *Guida alla programmazione*.

- CopyKey: questo attributo determina se la BackingMap effettua una copia dell'oggetto chiave quando una voce viene creata per la prima volta nella mappa. L'azione predefinita è quella di non effettuare la copia degli oggetti chiave, poiché le chiavi sono solitamente oggetti non modificabili.
- NumberOfBuckets: questo attributo indica il numero di bucket di hash che la BackingMap deve utilizzare. L'implementazione BackingMap utilizza una mappa di hash per la sua implementazione. Se esistono molte voci nella BackingMap, un maggior numero di bucket significherà migliori prestazioni. Il numero di chiavi che hanno lo stesso bucket diminuisce al crescere del numero di bucket. Un maggior numero di bucket significa anche più contemporaneità. Questo attributo è utile per la ottimizzazione delle prestazioni. Se l'applicazione non imposta l'attributo NumberOfBuckets verrà utilizzato un valore non definito.
- NumberOfLockBuckets: questo attributo indica il numero di bucket di blocco utilizzati dal gestore del blocco per la BackingMap. Quando LockStrategy è impostato su OPTIMISTIC o PESSIMISTIC, viene creato un gestore del blocco per la BackingMap. Il gestore del blocco utilizza una mappa di hash per tenere traccia delle voci bloccate da una o più transazioni. Se sono presenti molte voci nella mappa di hash, un maggior numero di bucket assicura migliori prestazioni poiché il numero di chiavi che confluisce nello stesso bucket è minore al crescere del numero di bucket. Un maggior numero di bucket di blocco significa anche maggiore contemporaneità. Quando l'attributo LockStrategy è impostato su NONE, dalla BackingMap non verrà utilizzato alcun gestore del blocco. In questo caso l'impostazione di numberOfLockBuckets non avrà alcun effetto. Se questo attributo non è impostato, verrà utilizzato un valore non definito.
- LockTimeout: questo attributo viene utilizzato quando la BackingMap utilizza un gestore del blocco. La BackingMap utilizza un gestore del blocco quando l'attributo LockStrategy è impostato su OPTIMISTIC o PESSIMISTIC. Il valore dell'attributo è riportato in secondi e determina quanto tempo il gestore del blocco attenderà per la concessione del blocco. Se questo attributo non viene impostato, si utilizzerà un valore di 15 secondi come LockTimeout. Consultare il Blocco pessimistico per i dettagli relativi alle eccezioni del timeout di attesa del blocco che possono verificarsi.
- TtlEvictorType: ogni BackingMap ha il proprio programma di eliminazione TTL (time to live) incorporato che utilizza un algoritmo basato sull'orario per determinare quali voci della mappa eliminare. Per impostazione predefinita, il programma di eliminazione TTL (time to live) non è attivo. Lo si può attivare chiamando il metodo setTtlEvictorType con uno dei seguenti valori: CREATION_TIME, LAST_ACCESS_TIME o NONE. Un valore CREATION_TIME indica che il programma di eliminazione aggiunge un attributo TimeToLive all'ora in cui la voce della mappa è stata creata nella BackingMap per determinare quando esso dovrà eliminare la voce della mappa dalla BackingMap. Un valore LAST_ACCESS_TIME (o LAST_UPDATE_TIME) indica che il programma di eliminazione aggiunge un attributo TimeToLive all'ora in cui la voce della mappa è stata acceduta per l'ultima volta (o acceduta *ed* aggiornata) da una transazione che l'applicazione sta eseguendo, per determinare quando il programma di eliminazione dovrà eliminare la voce della mappa. La voce della mappa viene eliminata solo se essa non viene mai acceduta da alcuna transazione per il periodo di tempo specificato dall'attributo TimeToLive. Un valore NONE indica che il programma di eliminazione deve restare inattivo e che non deve eliminare alcuna voce dalla mappa. Se questo attributo non viene impostato, come valore predefinito verrà utilizzato NONE ed

il programma di eliminazione TTL (time to live) non sarà attivato. Consultare programmi di eliminazione per i dettagli relativi al programma di eliminazione TTL (time to live) incorporato.

- TimeToLive: questo attributo viene utilizzato per specificare il numero di secondi che il programma di eliminazione TTL (time to live) dovrà aggiungere all'ora di creazione o di ultimo accesso di ciascuna voce, come descritto per l'attributo TtlEvictorType. Se questo attributo non viene impostato, verrà utilizzato il valore speciale zero per indicare che il TTL (time to live) è infinito. Se questo attributo è impostato su infinito, il programma di eliminazione non eliminerà mai le voci della mappa.

L'esempio che segue dimostra come definire la BackingMap someMap nella ObjectGrid someGrid ed impostare i vari attributi della BackingMap utilizzando i metodi di impostazione 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 ); // override default of read/write
bm.setNullValuesSupported(false); // override default of allowing Null values
bm.setLockStrategy( LockStrategy.PESSIMISTIC ); // override default of OPTIMISTIC
bm.setLockTimeout( 60 ); // override default of 15 seconds.
bm.setNumberOfBuckets(251); // override default (prime numbers work best)
bm.setNumberOfLockBuckets(251); // override default (prime numbers work best)
```

Plug-in BackingMap

L'interfaccia BackingMap ha diversi punti di collegamento facoltativi per un'interazione estendibile con la ObjectGrid:

- **ObjectTransformer plug-in**: per alcune operazioni della mappa, una BackingMap potrebbe necessitare di serializzare, deserializzare o copiare una chiave o un valore di una voce nella BackingMap. La BackingMap può eseguire queste azioni fornendo un'implementazione predefinita dell'interfaccia ObjectTransformer. Un applicazione può migliorare le prestazioni fornendo un plug-in ObjectTransformer personalizzato utilizzato dalla BackingMap per serializzare, deserializzare o copiare una chiave o un valore di una voce nella BackingMap. Consultare le informazioni relative al plug-in ObjectTransformer in *Guida alla programmazione* per maggiori informazioni.
- **Plug-in Evictor**: il programma di eliminazione TTL (Time To Live) incorporato utilizza un algoritmo basato sull'orario per decidere quando un voce della BackingMap deve essere eliminata. Alcune applicazioni potrebbero dover utilizzare algoritmi diversi per decidere quando deve essere eliminata una voce della BackingMap. Il plug-in Evictor rende disponibile alla BackingMap un programma di eliminazione personalizzato. Il plug-in Evictor si aggiunge al programma di eliminazione TTL (Time To Live), non lo sostituisce. ObjectGrid fornisce un plug-in Evictor personalizzato che implementa algoritmi noti quali "meno utilizzato di recente" o "meno frequentemente utilizzato". Le applicazioni potranno collegare uno dei plug-in Evictor forniti o potranno fornire il proprio plug-in Evictor. Consultare Le informazioni relative all'eliminazione in *Guida alla programmazione*.

- **MapEventListener plug-in**: un'applicazione potrebbe aver bisogno di informazioni relative agli eventi della BackingMap quali le eliminazioni di voci dalla mappa o l'avvenuto precaricamento della BackingMap. Una BackingMap chiama i metodi su un plug-in MapEventListener per notificare un'applicazione degli eventi della BackingMap. Un'applicazione potrà ricevere notifiche di diversi eventi della BackingMap utilizzando il metodo setMapEventListener in modo che fornisca uno o più plug-in MapEventListener personalizzati alla BackingMap. L'applicazione può modificare gli oggetti MapEventListener elencati utilizzando il metodo addMapEventListener oppure il metodo removeMapEventListener. Consultare le informazioni relative al plug-in MapEventListener in *Guida alla programmazione* per maggiori informazioni.
- **Plug-in Loader**: una BackingMap rappresenta una cache in memoria di una mappa. Un plug-in Loader è un'opzione utilizzata dalla BackingMap per spostare i dati tra la memoria e l'archivio persistente. Ad esempio un Loader JDBC (Java database connectivity) può essere utilizzato per spostare i dati all'interno ed all'esterno di una BackingMap ed una o più tabelle relazionali di un database relazionale. Non è necessario utilizzare un database relazionale come archivio persistente per una BackingMap. Il Loader può anche essere utilizzato per spostare dati tra una BackingMap ed un file, tra una BackingMap ed una mappa Hibernate, tra una BackingMap ed un bean di entity Java 2 Platform, Enterprise Edition (JEE), tra una BackingMap ed un altro server delle applicazioni e così via. L'applicazione deve fornire un plug-in Loader personalizzato per spostare i dati tra una BackingMap e l'archivio persistente per qualsiasi tecnologia utilizzata. Se non viene fornito un plug-in Loader, la BackingMap diventa una semplice cache in memoria. Consultare le informazioni relative all'utilizzo di un programma di caricamento in *Guida alla programmazione* per ulteriori informazioni.
- **OptimisticCallback plug-in**: quando l'attributo LockStrategy per una BackingMap è impostato su OPTIMISTIC, la BackingMap o un plug-in Loader devono effettuare operazioni di confronto per i valori della mappa. Il plug-in OptimisticCallback viene utilizzato dalla BackingMap e dal programma di caricamento per eseguire le operazioni di confronto del controllo versioni ottimistico. Consultare le informazioni relative al plug-in OptimisticCallback in *Guida alla programmazione* per maggiori informazioni.
- **MapIndexPlugin plug-in**: un plug-in MapIndexPlugin oppure in breve un Index, è un'opzione utilizzata dalla BackingMap per eseguire il build di un indice in base all'attributo specificato dell'oggetto archiviato. L'indice consente all'applicazione di trovare oggetti in base ad un valore specificato o ad un intervallo di valori. Esistono due tipi di indice: statico e dinamico. Fare riferimento all'indicizzazione per informazioni dettagliate.

Per ulteriori informazioni sui plug-in, consultare l'introduzione ai plug-in in *Guida alla programmazione*.

Connessione a un ObjectGrid distribuito

È possibile connettersi a un ObjectGrid distribuito con un endpoint di connessione per il servizio catalogo. È necessario disporre del nome host e della port di endpoint del server catalogo a cui si desidera connettersi.

Per poter eseguire il collegamento a una griglia distribuita, è necessario aver configurato il proprio ambiente lato server con un servizio catalogo e i server contenitore.

Il metodo getObjectGrid(ClientClusterContext ccc, String objectGridName) si collega al servizio catalogo specificato e restituisce un'istanza ObjectGrid client corrispondente a un'istanza ObjectGrid lato server.

Il seguente frammento di codice è un esempio su come connettersi a una griglia distribuita.

```
// Create an ObjectGridManager instance.

ObjectGridManager ogm = ObjectGridManagerFactory.getObjectGridManager();

// Obtain a ClientClusterContext by connecting to a catalog
// server based distributed ObjectGrid.  You have to provide
// a connection end point for your catalog server in the format
// of hostName:endPointPort.  The hostName is the machine
// where the catalog server resides, and the endPointPort is
// the catalog server's listening port, whose default is 2809.

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

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

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

Interazione con una ObjectGrid utilizzando ObjectGridManager

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

Modello di programmazione

Prima di utilizzare la funzionalità di eXtreme Scale come una griglia di dati in memoria, è necessario creare ed interagire con istanze ObjectGrid con metodi quali i seguenti.

- Metodi createObjectGrid
- Metodi getObjectGrid
- Metodi removeObjectGrid
- Controllo del ciclo di vita di un ObjectGrid

Metodi createObjectGrid

Questo argomento descrive i sette metodi createObjectGrid nell'interfaccia ObjectGridManager. Ciascuno di questi metodi crea un'istanza locale di un ObjectGrid.

Istanza in memoria locale

Il seguente frammento di testo illustra come ottenere e configurare un'istanza ObjectGrid locale con eXtreme Scale.

```
// Obtain a local ObjectGrid reference
        // you can create a new ObjectGrid, or get configured ObjectGrid
        // defined in ObjectGrid xml file
```

```
   ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
   ObjectGrid ivObjectGrid =
objectGridManager.createObjectGrid("objectgridName");

   // Add a TransactionCallback into ObjectGrid
   HeapTransactionCallback tcb = new HeapTransactionCallback();
   ivObjectGrid.setTransactionCallback(tcb);

   // Define a BackingMap
   // if the BackingMap is configured in ObjectGrid xml
   // file, you can just get it.
   BackingMap ivBackingMap = ivObjectGrid.defineMap("myMap");

   // Add a Loader into BackingMap
   Loader ivLoader = new HeapCacheLoader();
   ivBackingMap.setLoader(ivLoader);

   // initialize ObjectGrid
   ivObjectGrid.initialize();

   // Obtain a session to be used by the current thread.
   // Session can not be shared by multiple threads
   Session ivSession = ivObjectGrid.getSession();

   // Obtaining ObjectMap from ObjectGrid Session
   ObjectMap objectMap = ivSession.getMap("myMap");
```

Configurazione condivisa predefinita

Il codice seguente rappresenta un semplice caso di creazione di un ObjectGrid da condividere tra più 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 precedente di Java crea e memorizza in memoria l'ObjectGrid Employees. L'ObjectGrid Employees viene inizializzata con la configurazione predefinita ed è pronta per essere utilizzata. Il secondo parametro nel metodo createObjectGrid è impostato su true, ciò richiede a ObjectGridManager la memorizzazione nella cache dell'istanza ObjectGrid che esso crea. Se questo parametro viene impostato su false, l'istanza non viene memorizzata nella cache. Ogni istanza ObjectGrid ha un nome ed, in base ad esso, l'istanza viene condivisa tra più client ed utenti.

Se l'istanza objectGrid viene utilizzata in una condivisione peer-to-peer, la memorizzazione nella cache dovrà essere impostata su true. Per ulteriori informazioni sulla condivisione peer-to-peer, consultare Distribuzione delle modifiche tra JVM (Java Virtual Machine) peer.

Configurazione XML

WebSphere eXtreme Scale è altamente configurabile. L'esempio precedente dimostra come creare un ObjectGrid semplice senza alcuna configurazione. Questo

esempio mostra invece come creare un'istanza ObjectdGrid preconfigurata avendo come base un file di configurazione XML. È possibile configurare un'istanza ObjectGrid programmaticamente oppure utilizzando un file di configurazione basato su XML. È anche possibile configurare un ObjectGrid utilizzando una combinazione di entrambi i metodi. L'interfaccia ObjectGridManager consente di creare un'istanza ObjectGrid basata sulla configurazione XML. L'interfaccia ObjectGridManager ha diversi metodi che accettano un URL come argomento. Ogni file XML passato nel ObjectGridManager deve essere convalidato rispetto allo schema. La convalida XML può essere disabilitata solo quando il file è stato precedentemente convalidato e ad esso non sono state apportate modifiche dall'ultima convalida. La disabilitazione della convalida consente di risparmiare una minima quantità di sovraccarico, ma rischia di utilizzare un file XML non valido. IBM Java Developer Kit (JDK) 1.4.2 supporta la convalida XML. Quando si utilizza un JDK che non ha questo supporto, potrebbe essere necessario richiedere ad Apache Xerces di convalidare il file XML.

Il seguente frammento di codice Java dimostra come passare 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; // turn XML validation on
boolean cacheInstance = true; // Cache the instance
String objectGridName="Employees"; // Name of Object Grid URL
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
 ObjectGridManagerFactory.getObjectGridManager();
 ObjectGrid employees =
 oGridManager.createObjectGrid(objectGridName, allObjectGrids,
  bvalidateXML, cacheInstance);
```

Il file XML può contenere le informazioni di configurazione di diversi ObjectGrid. Il precedente frammento di codice restituisce ObjectGrid Employees, supponendo che la configurazione di Employees sia definita nel file. Per la sintassi XML, consultare configurazione ObjectGrid. Esistono sette metodi createObjectGrid e sono documentati nel seguente blocco di codice.

```
/**
 * A simple factory method to return an instance of an
 * Object Grid. A unique name is assigned.
 * The instance of ObjectGrid is not cached.
 * Users can then use {@link ObjectGrid#setName(String)} to change the
 * ObjectGrid name.
 *
 * @return ObjectGrid an instance of ObjectGrid with a unique name assigned
 * @throws ObjectGridException any error encountered during the
 * ObjectGrid creation
 */
public ObjectGrid createObjectGrid() throws ObjectGridException;

/**
 * A simple factory method to return an instance of an ObjectGrid with the
 * specified name. The instances of ObjectGrid can be cached. If an ObjectGrid
 * with the this name has already been cached, an ObjectGridException
 * will be thrown.
 *
 * @param objectGridName the name of the ObjectGrid to be created.
 * @param cacheInstance true, if the ObjectGrid instance should be cached
 * @return an ObjectGrid instance
 * @this name has already been cached or
```

```
 * any error during the ObjectGrid creation.
 */
public ObjectGrid createObjectGrid(String objectGridName, boolean cacheInstance)
  throws ObjectGridException;

/**
 * Create an ObjectGrid instance with the specified ObjectGrid name. The
 * ObjectGrid instance created will be cached.
 * @param objectGridName the Name of the ObjectGrid instance to be created.
 * @return an ObjectGrid instance
 * @throws ObjectGridException if an ObjectGrid with this name has already
 * been cached, or any error encountered during the ObjectGrid creation
 */
public ObjectGrid createObjectGrid(String objectGridName)
  throws ObjectGridException;

/**
 * Create an ObjectGrid instance based on the specified ObjectGrid name and the
 * XML file. The ObjectGrid instance defined in the XML file with the specified
 * ObjectGrid name will be created and returned. If such an ObjectGrid
 * cannot be found in the xml file, an exception will be thrown.
 *
 * This ObjecGrid instance can be cached.
 *
 * If the URL is null, it will be simply ignored. In this case, this method behaves
 * the same as {@link #createObjectGrid(String, boolean)}.
 *
 * @param objectGridName the Name of the ObjectGrid instance to be returned. It
 * must not be null.
 * @param xmlFile a URL to a wellformed xml file based on the ObjectGrid schema.
 * @param enableXmlValidation if true the XML is validated
 * @param cacheInstance a boolean value indicating whether the ObjectGrid
 * instance(s)
 * defined in the XML will be cached or not. If true, the instance(s) will
 * be cached.
 *
 * @throws ObjectGridException if an ObjectGrid with the same name
 * has been previously cached, no ObjectGrid name can be found in the xml file,
 * or any other error during the ObjectGrid creation.
 * @return an ObjectGrid instance
 * @see ObjectGrid
 */
public ObjectGrid createObjectGrid(String objectGridName, final URL xmlFile,
final boolean enableXmlValidation, boolean cacheInstance)
 throws ObjectGridException;

/**
 * Process an XML file and create a List of ObjectGrid objects based
 * upon the file.
 * These ObjecGrid instances can be cached.
 * An ObjectGridException will be thrown when attempting to cache a
 * newly created ObjectGrid
 * that has the same name as an ObjectGrid that has already been cached.
 *
 * @param xmlFile the file that defines an ObjectGrid or multiple
 * ObjectGrids
 * @param enableXmlValidation setting to true will validate the XML
 * file against the schema
 * @param cacheInstances set to true to cache all ObjectGrid instances
 * created based on the file
 * @return an ObjectGrid instance
 * @throws ObjectGridException if attempting to create and cache an
 * ObjectGrid with the same name as
 * an ObjectGrid that has already been cached, or any other error
 * occurred during the
 * ObjectGrid creation
 */
```

```
public List createObjectGrids(final URL xmlFile, final boolean enableXmlValidation,
boolean cacheInstances) throws ObjectGridException;

/** Create all ObjectGrids that are found in the XML file. The XML file will be
 * validated against the schema. Each ObjectGrid instance that is created will
 * be cached. An ObjectGridException will be thrown when attempting to cache a
 * newly created ObjectGrid that has the same name as an ObjectGrid that has
 * already been cached.
 * @param xmlFile The XML file to process. ObjectGrids will be created based
 * on what is in the file.
 * @return A List of ObjectGrid instances that have been created.
 * @throws ObjectGridException if an ObjectGrid with the same name as any of
 * those found in the XML has already been cached, or
 * any other error encounterred during ObjectGrid creation.
 */
public List createObjectGrids(final URL xmlFile) throws ObjectGridException;

/**
 * Process the XML file and create a single ObjectGrid instance with the
 * objectGridName specified only if an ObjectGrid with that name is found in
 * the file. If there is no ObjectGrid with this name defined in the XML file,
 * an ObjectGridException
 * will be thrown. The ObjectGrid instance created will be cached.
 * @param objectGridName name of the ObjectGrid to create. This ObjectGrid
 * should be defined in the XML file.
 * @param xmlFile the XML file to process
 * @return A newly created ObjectGrid
 * @throws ObjectGridException if an ObjectGrid with the same name has been
 * previously cached, no ObjectGrid name can be found in the xml file,
 * or any other error during the ObjectGrid creation.
 */
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
  throws ObjectGridException;
```

Blocco del client durante la chiamata al metodo getObjectGrid

Un client potrebbe sembrare bloccarsi durante la chiamata del metodo getObjectGrid su ObjectGridManager oppure potrebbe generare un'eccezione: com.ibm.websphere.projector.MetadataException. Il repository EntityMetadata non è disponibile e si raggiunge la soglia di timeout. Ciò è dovuto al fatto che il client attende che i metadati dell'entity su ObjectGrid diventino disponibili. Questo errore si potrà verificare quando si è avviato un contenitore e non si è raggiunto il numero iniziale di contenitori o il numero minimo di repliche sincronizzate. Esaminare la politica di distribuzione per ObjectGrid e verificare che il numero di contenitori attivi sia maggiore o uguale ad entrambi gli attributi numInitialContainers e minSyncReplicas nel file di descrizione della politica di distribuzione.

metodi getObjectGrid

Utilizzare i metodi ObjectGridManager.getObjectGrid per recuperare le istanze memorizzate nella cache.

Recupero di un'istanza memorizzata nella cache

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

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

Di seguito sono riportati i due metodi getObjectGrid che restituiscono le istanze ObjectGrid memorizzate nella cache:

- **Recupero di tutte le istanze memorizzate nella cache**

  Per ottenere tutte le istanze ObjectGrid precedentemente memorizzate nella cache, utilizzare il metodo getObjectGrids, che restituisce un elenco di tutte le istanze. Se non esiste alcuna istanza memorizzata nella cache, il metodo restituisce il valore null.
- **Recupero, mediante il nome, di un'istanza memorizzata nella cache**

  Per ottenere una singola istanza ObjectGrid memorizzata nella cache, utilizzare getObjectGrid(String objectGridName), passando al metodo il nome dell'istanza memorizzata nella cache. Il metodo restituisce l'istanza ObjectGrid con il nome specificato oppure restituisce il valore null se non esiste alcuna istanza ObjectGrid con tale nome.

Nota: È inoltre possibile utilizzare il metodo getObjectGrid per la connessione ad una griglia distribuita. Per ulteriori informazioni, consultare la sezione "Connessione a un ObjectGrid distribuito" a pagina 22.

Metodi removeObjectGrid

È possibile utilizzare due metodi removeObjectGrid diversi per rimuovere le istanze ObjectGrid dalla cache.

Rimozione di un'istanza ObjectGrid

Per rimuovere le istanze ObjectGrid dalla cache, utilizzare uno dei metodi removeObjectGrid. ObjectGridManager non mantiene un riferimento delle istanze rimosse. Esistono due metodi di rimozione. Un metodo utilizza un parametro booleano. Se il parametro booleano è impostato su `true`, il metodo destroy viene chiamato sull'ObjectGrid. La chiamata al metodo destroy sull'ObjectGrid chiude l'ObjectGrid e libera tutte le risorse da esso utilizzate. Di seguito è riportata una descrizione di come utilizzare i due metodi removeObjectGrid:

```
/**
 * Remove an ObjectGrid from the cache of ObjectGrid instances
 *
 * @param objectGridName the name of the ObjectGrid instance to remove
 * from the cache
 *
 * @throws ObjectGridException if an ObjectGrid with the objectGridName
 * was not found in the cache
 */
public void removeObjectGrid(String objectGridName) throws ObjectGridException;

/**
 * Remove an ObjectGrid from the cache of ObjectGrid instances and
 * destroy its associated resources
 *
 * @param objectGridName the name of the ObjectGrid instance to remove
 * from the cache
 *
 * @param destroy destroy the objectgrid instance and its associated
 * resources
 *
 * @throws ObjectGridException if an ObjectGrid with the objectGridName
 * was not found in the cache
 */
public void removeObjectGrid(String objectGridName, boolean destroy)
 throws ObjectGridException;
```

Controllo del ciclo di vita di un ObjectGrid

È possibile utilizzare l'interfaccia ObjectGridManager per controllare il ciclo di vita di un'istanza ObjectGrid utilizzando un bean di avvio o un servlet.

Gestione del ciclo di vita con un bean di avvio

Un bean di avvio viene utilizzato per controllare il ciclo di vita di un'istanza ObjectGrid. Un bean di avvio viene caricato quando si avvia un'applicazione. Con un bean di avvio, il codice può essere eseguito ogni volta che un'applicazione viene avviata o arrestata come previsto. Per creare un bean di avvio, utilizzare l'interfaccia com.ibm.websphere.startupservice.AppStartUpHome home e l'interfaccia com.ibm.websphere.startupservice.AppStartUp remota. Implementare i metodi start e stop sul bean. Il metodo start viene richiamato ogni volta che l'applicazione viene avviata. Il metodo stop viene richiamato quando l'applicazione viene chiusa. Il metodo start viene utilizzato per creare le istanze ObjectGrid. Il metodo stop viene utilizzato per rimuovere le istanze ObjectGrid. Di seguito è riportato un frammento di codice che dimostra la gestione del ciclo di vita di questo ObjectGrid in un bean di avvio:

```
public class MyStartupBean implements javax.ejb.SessionBean {
    private ObjectGridManager objectGridManager;

    /* The methods on the SessionBean interface have been
     * left out of this example for the sake of brevity */

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

        return true;
    }

    public void stop(){
        // Stopping the startup bean
        // This method is called when the application is stopped
        try {
            // remove the cached ObjectGrids and destroy them
            objectGridManager.removeObjectGrid("bookstore", true);
            objectGridManager.removeObjectGrid("videostore", true);
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }
}
```

Dopo aver chiamato il metodo start, le istanze ObjectGrid create di recente vengono richiamate dall'interfaccia ObjectGridManager. Ad esempio, se un servlet è incluso nell'applicazione, esso accede a eXtreme Scale utilizzando il seguente frammento di codice:

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

Gestione del ciclo di vita con un servlet

Per gestire il ciclo di vita di un'istanza ObjectGrid in un servlet, è possibile utilizzare il metodo init per creare un'istanza ObjectGrid e il metodo destroy per rimuoverla. Se l'istanza ObjectGrid viene memorizzata nella cache, essa viene richiamata e modificata nel codice servlet. Di seguito è riportato un codice di esempio che descrive la creazione, la modifica e l'eliminazione di ObjectGrid 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 {
            // create and cache an ObjectGrid named bookstore
            ObjectGrid bookstoreGrid =
      objectGridManager.createObjectGrid("bookstore", true);
            bookstoreGrid.defineMap("book");
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }

    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
        Session session = bookstoreGrid.getSession();
        ObjectMap bookMap = session.getMap("book");
        // perform operations on the cached ObjectGrid
        // ...
    }

    public void destroy() {
        super.destroy();
        try {
            // remove and destroy the cached bookstore ObjectGrid
            objectGridManager.removeObjectGrid("bookstore", true);
        } catch (ObjectGridException e) {
            e.printStackTrace();
        }
    }
}
```

Accesso al frammento ObjectGrid

WebSphere eXtreme Scale ottiene valutazioni di elaborazione elevate spostando la logica dove si trovano i dati e restituendo al client solo i i risultati.

La logica dell'applicazione in una JVM (Java Virtual Machine) client deve estrapolare i dati dalla JVM server che contiene i dati e reinserirli al commit della transazione. Questo processo rallenta la velocità con cui vengono elaborati i dati. Se la logica dell'applicazione era sulla stessa JVM del frammento che contiene i dati, la latenza della rete e il costo del marshalling vengono eliminati e questo può fornire un notevole miglioramento delle prestazioni.

Riferimento locale ai dati del frammento

Le API ObjectGrid forniscono un oggetto Session al metodo del lato server. Questa sessione è un riferimento diretto ai dati di quel frammento. Non è presente una logica di instradamento su quel percorso. La logica dell'applicazione può utilizzare i dati di quel frammento direttamente. La sessione non può essere utilizzata per accedere ai dati in un'altra partizione perché non esiste una logica di instradamento.

Un plug-in Loader, inoltre, fornisce un modo per ricevere un evento quando un frammento diviene una partizione primaria. Un'applicazione può implementare un Loader ed implementare l'interfaccia ReplicaPreloadController. Il metodo di verifica dello stato di precaricamento viene richiamato solo quando un frammento diviene primario. La sessione fornita a quel metodo è un riferimento locale ai dati del frammento. Questo approccio viene in genere utilizzato se l'elemento primario della partizione deve avviare qualche thread o si deve sottoscrivere ad un fabric di messaggi per il traffico correlato alla partizione. Potrebbe avviare un thread per ascoltare i messaggi in una mappa locale utilizzando l'API getNextKey.

Ottimizzazione dei client-server collocati

Se un'applicazione utilizza le API del client per accedere ad una partizione che si trova nella JVM che contiene il client, si evita la rete ma viene comunque eseguito il marshalling a causa dei problemi di implementazione correnti. Se viene utilizzata una griglia con partizioni, non vi è impatto sulle prestazioni dell'applicazione in quanto il numero (N-1)/N di chiamate viene instradato su una JVM diversa. Se si necessita sempre di un accesso locale con un frammento, utilizzare le API del Loader o dell'ObjectGrid per richiamare quella logica.

Accesso ai dati in WebSphere eXtreme Scale

Una volta che un'applicazione presenta un riferimento ad un'istanza ObjectGrid o una connessione client ad una griglia remota, è possibile accedere ai dati e interagire con essi nella configurazione WebSphere eXtreme Scale. Con l'API ObjectGridManager utilizzare uno dei metodi createObjectGrid per creare un'istanza logica o il metodo getObjectGrid per un'istanza client con una griglia distribuita.

Per un thread di un'applicazione è necessaria la propria Session. Quando un'applicazione desidera utilizzare ObjectGrid su un thread, deve solo richiamare uno dei metodi getSession per ottenere un thread. Questa operazione è facile--non è necessario raggruppare queste operazioni nella maggior parte dei casi. Se l'applicazione sta utilizzando un framework di inserimento di dipendenza come Spring, è possibile inserire una Session in un bean di applicazione, quando necessario.

Una volta ottenuta una Session, l'applicazione può accedere ai dati memorizzati nelle mappe in ObjectGrid. Se ObjectGrid utilizza le entità, è possibile utilizzare l'API EntityManager, che è possibile richiamare con il metodo

Session.getEntityManager. Poiché è più vicina alle specifiche Java, l'interfaccia EntityManager è più semplice dell'API basata sulla mappa. Tuttavia, l'API EntityManager ha un sovraccarico di prestazioni poiché tiene traccia delle modifiche negli oggetti. L'API basata su mappa si ottiene con l'utilizzo del metodo Session.getMap.

WebSphere eXtreme Scale utilizza le transazioni. Quando un'applicazione interagisce con una Session, deve essere nell'ambito di una transazione. Una transazione viene avviata e ne viene eseguito il commit o roll back mediante i metodi Session.begin, Session.commit e Session.rollback sull'oggetto Session. Le applicazioni possono anche funzionare in modalità di commit automatico, in cui la Session viene avviata e esegue il commit di una transazione automaticamente ogni volta che l'applicazione interagisce con le mappe. Tuttavia, la modalità di commit automatico è più lenta.

La logica di utilizzo delle transazioni

È possibile che le transazioni sembrino più lente, ma eXtreme Scale utilizza le transazioni per tre motivi:

1. Per consentire il rollback delle modifiche se si verifica un'eccezione o la logica di business deve annullare le modifiche di stato.
2. Per mantenere i blocchi sui dati e rilasciare i blocchi entro la durata di una transazione, consentendo di apportare automaticamente una serie di modifiche, cioè, tutte le modifiche o nessuna modifica ai dati.
3. Per produrre un'unità atomica di replica.

WebSphere eXtreme Scale consente ad una Session di personalizzare il numero di transazioni realmente necessarie. Un'applicazione può disattivare il supporto e il blocco di rollback, ma a discapito dell'applicazione. L'applicazione deve gestire la mancanza di queste funzioni da sola.

Ad esempio, un'applicazione può disattivare il blocco configurando la strategia di blocco BackingMap su NONE. Questa strategia è veloce, ma le transazioni contemporanee possono adesso modificare gli stessi dati senza proteggersi le une dalle altre. L'applicazione è responsabile per tutto il blocco e la coerenza dei dati quando viene utilizzato NONE.

Un'applicazione può anche modificare il modo in cui gli oggetti vengono copiati quando vengono acceduti dalla transazione. L'applicazione può specificare come gli oggetti vengono copiati con il metodo ObjectMap.setCopyMode. Con questo metodo è possibile disattivare CopyMode. La disattivazione di CopyMode viene normalmente utilizzata per transazioni di sola lettura se è possibile restituire valori differenti per lo stesso oggetto all'interno di una transazione. È possibile restituire valori differenti per lo stesso oggetto all'interno di una transazione.

Ad esempio, se la transazione ha richiamato il metodo ObjectMap.get per l'oggetto nel momento T1, riceve il valore di quel momento. Se richiama di nuovo il metodo get nell'ambito di quella transazione in un momento successivo T2, è possibile che un altro thread abbia modificato il valore. Poiché il valore è stato modificato da un altro thread, l'applicazione vede un valore differente. Se l'applicazione modifica un oggetto richiamato utilizzando un valore CopyMode NONE, sta modificando direttamente la copia con commit di quell'oggetto. Il roll-back della transazione non ha alcun senso in questa modalità. Si sta modificando l'unica copia in ObjectGrid. Sebbene l'utilizzo di CopyMode NONE sia veloce, occorre tenerne presente le conseguenze. Un'applicazione che utilizza un CopyMode NONE non

deve mai eseguire il roll-back della transazione. Se l'applicazione esegue il roll-back della transazione, gli indici non vengono aggiornati con le modifiche *e* le modifiche non vengono replicate, se la replica è attiva. I valori predefiniti sono facili da utilizzare e poco soggetti a errori. Se si preferiscono migliori prestazioni con dati meno affidabili, l'applicazione deve essere consapevole di ciò che sta facendo per evitare problemi indesiderati.

Avvertenza:
Prestare attenzione quando si stanno modificando i valori di blocco o CopyMode. Se si modificano i valori, l'applicazione avrà un comportamento imprevedibile.

Interazione con dati memorizzati

Una volta ottenuta una sessione, è possibile utilizzare il seguente frammento di codice per utilizzare l'API delle mappe per l'inserimento dei dati.

```
Session session = ...;
ObjectMap personMap = session.getMap("PERSON");
session.begin();
Person p = new Person();
p.name = "John Doe";
personMap.insert(p.name, p);
session.commit();
```

Di seguito è riportato lo stesso esempio che utilizza l'API EntityManager. Questo esempio di codice presuppone che l'oggetto Person sia associato ad un Entity.

```
Session session = ...;
EntityManager em = session.getEntityManager();
session.begin();
Person p = new Person();
p.name = "John Doe";
em.persist(p);
session.commit();
```

Il pattern è progettato per ottenere i riferimenti a ObjectMaps per le mappe utilizzate dal thread, per avviare una transazione, utilizzare i dati e quindi per eseguire il commit della transazione.

L'interfaccia ObjectMap dispone di operazioni di mappa tipiche come put, get e remove. Tuttavia, utilizzare nomi di operazioni più specifiche come: get, getForUpdate, insert, update e remove. Questi nomi di metodo indicano più precisamente lo scopo rispetto alle API delle mappe.

Inoltre, è possibile utilizzare il supporto di indicizzazione, che è flessibile.

Di seguito viene riportato un esempio di aggiornamento di un Object:

```
session.begin();
Person p = (Person)personMap.getForUpdate("John Doe");
p.name = "John Doe";
p.age = 30;
personMap.update(p.name, p);
session.commit();
```

L'applicazione in genere utilizza il metodo getForUpdate invece di una semplice operazione get per bloccare il record. Si deve richiamare il metodo di aggiornamento per fornire realmente il valore aggiornato alla mappa. Se non si richiama l'aggiornamento, la mappa non viene modificata. Di seguito viene riportato lo stesso frammento utilizzando l'API EntityManager:

```
session.begin();
Person p = (Person)em.findForUpdate(Person.class, "John Doe");
p.age = 30;
session.commit();
```

L'API EntityManager è più semplice dell'approccio di mappa. In questo caso, eXtreme Scale individua l'Entity e restituisce un oggetto gestito all'applicazione. L'applicazione modifica l'oggetto e esegue il commit della transazione, eXtreme Scale tiene traccia delle modifiche apportate agli oggetti gestiti automaticamente al momento del commit e esegue gli aggiornamenti necessari.

Transazioni e partizioni

Le transazioni WebSphere eXtreme Scale possono solo aggiornare una singola partizione. Le transazioni provenienti da un client possono effettuare la lettura da più partizioni, ma possono aggiornare solo una partizione. Se un'applicazione cerca di aggiornare due partizioni, la transazione non riesce correttamente e ne viene eseguito il roll-back. Una transazione che sta utilizzando un ObjectGrid incorporato (logica della griglia) non ha capability di instradamento e può solo visualizzare i dati nella partizione locale. Questa logica di business può sempre acquisire una seconda sessione che è una sessione client vera per accedere ad altre partizioni. Tuttavia, questa transazione sarebbe una transazione indipendente.

Query e partizioni

Se una transazione ha già ricercato un Entity, la transazione viene associata alla partizione per quell'Entity. Tutte le query in esecuzione su una transazione associata a un Entity vengono instradate alla partizione associata.

Se una query viene eseguita su una transazione prima che venga associata ad una partizione, si deve impostare l'ID partizione da utilizzare per la query. L'ID partizione è un valore intero. La query viene quindi instradata a quella partizione.

Le query possono eseguire ricerche solo all'interno di una singola partizione. Tuttavia, è possibile utilizzare le API DataGrid per eseguire la stessa query in parallelo su tutte le partizioni o su una serie secondaria di partizioni. Utilizzare le API DataGrid per trovare una voce che potrebbe trovarsi in qualsiasi partizione.

7.0.0.0 FIX 2+ Il servizio dati REST consente ad ogni client HTTP di accedere alla griglia di WebSphere eXtreme Scale ed è compatibile con WCF Data Services in Microsoft .NET Framework 3.5 SP1. Per ulteriori informazioni consultare la guida per l'utente relativa al servizio dati REST di eXtreme Scale

.

Migliori pratiche per CopyMode

WebSphere eXtreme Scale esegue una copia del valore sulla base delle sei impostazioni CopyMode disponibili. Determinare quale impostazione meglio si adatta alle proprie esigenze di distribuzione.

È possibile utilizzare il metodo setCopyMode(CopyMode, valueInterfaceClass) dell'API BackingMap per impostare la modalità di copia con uno dei seguenti campi statici finali definiti nella classe com.ibm.websphere.objectgrid.CopyMode.

Quando l'applicazione utilizza l'interfaccia ObjectMap per ottenere un riferimento ad una voce della mappa, esso deve essere utilizzato solo all'interno della

transazione WebSphere eXtreme Scale che lo ha ottenuto. L'utilizzo del riferimento in una transazione diversa potrebbe generare errori. Ad esempio, se si utilizza una strategia di blocco pessimistico per la BackingMap, una chiamata del metodo get o getForUpdate acquisisce un blocco S (condiviso) o U (aggiorna), a seconda della transazione. Il metodo get restituisce un riferimento al valore ed il blocco ottenuto viene rilasciato al completamento della transazione. La transazione deve chiamare il metodo get o getForUpdate per bloccare la voce della mappa in una transazione diversa. Ciascuna transazione deve ottenere il proprio riferimento ad un valore chiamando il metodo get o getForUpdate invece di riutilizzare lo stesso riferimento al valore in più transazioni.

CopyMode per le mappe di entità

Quando si utilizza una mappa associata con un'entità API EntityManager, la mappa restituisce sempre l'oggetto Tupla dell'entità in modo diretto, senza effettuare una copia, a meno che non si stia utilizzando la modalità di copia COPY_TO_BYTES. È importante che il CopyMode sia aggiornato o che la Tupla venga adeguatamente copiata quando si effettuano modifiche.

COPY_ON_READ_AND_COMMIT

La modalità COPY_ON_READ_AND_COMMIT è quella predefinita. Quando si utilizza questa modalità l'argomento valueInterfaceClass verrà ignorato. Questa modalità assicura che un'applicazione non contenga un riferimento ad un oggetto valore presente nella BackingMap. Al contrario, l'applicazione lavora sempre con una copia del valore presente nella BackingMap. La modalità COPY_ON_READ_AND_COMMIT assicura che l'applicazione non danneggi inavvertitamente i dati memorizzati nella cache nella BackingMap. Quando una transazione dell'applicazione chiama un metodo ObjectMap.get per una determinata chiave e si tratta del primo accesso alla voce dell'ObjectMap, viene restituita una copia del valore. Quando viene eseguito il commit della transazione, tutte le modifiche di cui l'applicazione ha eseguito il commit vengono copiate nella BackingMap per assicurare che l'applicazione non abbia un riferimento al valore di cui si è eseguito il commit nella BackingMap.

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 della transazione. Quando si utilizza questa modalità l'argomento valueInterfaceClass verrà ignorato. Per preservare l'integrità dei dati della BackingMap, l'applicazione assicura che qualsiasi riferimento in essa contenuto ad una voce venga distrutto dopo l'esecuzione del commit della transazione. Con questa modalità, il metodo ObjectMap.get restituisce una copia del valore, invece di un riferimento ad esso, in modo da assicurare che le modifiche effettuate al valore dall'applicazione non influenzino il valore nella BackingMap fino al commit della transazione. Tuttavia, dopo il commit della transazione, non viene eseguita una copia delle modifiche. Nella BackingMap verrà, invece, memorizzato il riferimento alla copia restituito dal metodo ObjectMap.get. L'applicazione distruggerà i riferimenti alla voce della mappa dopo il commit della transazione. Se l'applicazione non distrugge i riferimenti alla voce della mappa, si potrà verificare il danneggiamento dei dati memorizzati nella cache BackingMap. Se un'applicazione utilizza questa modalità e si manifestano dei problemi, passare alla modalità COPY_ON_READ_AND_COMMIT per verificare se i problemi persistono. Se i problemi scompaiono, significa che l'applicazione non riesce a distruggere tutti i suoi riferimenti dopo il commit della transazione.

COPY_ON_WRITE

La modalità COPY_ON_WRITE migliora le prestazioni rispetto alla modalità COPY_ON_READ_AND_COMMIT eliminando la copia che si verifica quando viene chiamato il metodo ObjectMap.get per la prima volta da una transazione per una determinata chiave. Il metodo ObjectMap.get restituisce un proxy al valore invece di un riferimento diretto all'oggetto valore. Il proxy assicura che non venga effettuata una copia del valore a meno che l'applicazione non chiami un metodo set sull'interfaccia del valore specificato dall'argomento valueInterfaceClass. Il proxy fornisce una copia sull'implementazione write. Quando avviene il commit della transazione, la BackingMap esamina il proxy per determinare se è stata effettuata una copia causata da una chiamata al metodo set. Se si è effettuata una copia, nella BackingMap verrà memorizzato il riferimento a tale copia. Il grande vantaggio di questo metodo è che il valore non viene mai copiato durante una lettura o un commit quando la transazione non effettua una chiamata ad un metodo set per cambiare il valore.

Le modalità COPY_ON_READ_AND_COMMIT e COPY_ON_READ effettuano sempre una copia completa quando un valore viene recuperato dall'ObjectMap. Questa modalità non è quella ottimale quando un'applicazione aggiorna solo alcuni dei valori recuperati in una transazione. La modalità COPY_ON_WRITE supporta questo comportamento in maniera efficiente, ma richiede che l'applicazione utilizzi un pattern semplice. Per supportare un'interfaccia, sono richiesti gli oggetti valore. L'applicazione deve utilizzare i metodi su questa interfaccia quando interagisce con il valore in una sessione eXtreme Scale. In questo caso eXtreme Scale crea delle deleghe per i valori che sono restituite all'applicazione. Il proxy ha un riferimento al valore reale. Se l'applicazione esegue solo operazioni di lettura, esse vengono sempre eseguite sulla copia reale. Se l'applicazione modifica un attributo sull'oggetto, il proxy effettua una copia dell'oggetto reale e quindi esegue la modifica sulla copia. Da quel momento in poi il proxy utilizzerà la copia. L'utilizzo della copia consente di evitare completamente l'operazione di copia per gli oggetti che vengono solo letti dall'applicazione. Tutte le operazioni di modifica devono cominciare con il prefisso impostato. Gli Enterprise JavaBeans™ sono solitamente codificati in modo da utilizzare questo stile di denominazione del metodo per i metodi che modificano gli attributi degli oggetti. Bisogna seguire questa convenzione. Qualsiasi oggetto modificato dall'applicazione viene copiato in quel momento. Questo scenario di lettura e scrittura è quello più efficiente supportato da eXtreme Scale. Per configurare una mappa in modo che utilizzi la modalità COPY_ON_WRITE, utilizzare l'esempio che segue. In questo esempio, l'applicazione memorizza gli oggetti Person a cui è stata assegnata una chiave utilizzando il nome nella mappa. L'oggetto Person viene rappresentato nel 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 recuperati da un'ObjectMap. Modificare l'oggetto in modo che utilizzi un'interfaccia come nell'esempio seguente.

```
interface IPerson
{
    void setName(String n);
    String getName();
    void setAge(int a);
    int getAge();
}
// Modify Person to implement IPerson interface
class Person implements IPerson {
    ...
}
```

L'applicazione quindi dovrà configurare la BackingMap in modo che utilizzi la modalità COPY_ON_WRITE, come nel seguente esempio:

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// use COPY_ON_WRITE for this Map with
// IPerson as the valueProxyInfo Class
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// The application should then use the following
// pattern when using the PERSON Map.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// the application casts the returned value to IPerson and not Person
IPerson p = (IPerson)person.get("Billy");
p.setAge(p.getAge()+1);
...
// make a new Person and add to Map
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// the following snippet WON'T WORK. Will result in ClassCastException
sess.begin();
// the mistake here is that Person is used rather than
// IPerson
Person a = (Person)person.get("Bobby");
sess.commit();
```

La prima sezione mostra l'applicazione che recupera un valore denominato Billy nella mappa. L'applicazione esegue il cast dei valori restituiti in un oggetto IPerson, non ad un oggetto Person poiché il proxy restituito implementa due interfacce:

- L'interfaccia specificata nella chiamata di metodo BackingMap.setCopyMode
- L'interfaccia com.ibm.websphere.objectgrid.ValueProxyInfo

È possibile eseguire il cast del proxy in due tipi. La parte finale del precedente frammento di codice dimostra cosa non è consentito nella modalità COPY_ON_WRITE. L'applicazione recupera il record Bobby e cerca di eseguire il cast del record in un oggetto Person. Questa azione fallisce con un'eccezione class

cast poiché il proxy restituito non è un oggetto Person. Il proxy restituito implementa un oggetto IPerson e ValueProxyInfo.

Interfaccia ValueProxyInfo e supporto aggiornamento parziale: quest'interfaccia consente ad un'applicazione il recupo dei valori in sola lettura di cui si è effettuato il commit ed a cui il proxy fa riferimento o la serie di attributi modificati durante la transazione.

```
public interface ValueProxyInfo {
    List /**/ ibmGetDirtyAttributes();
    Object ibmGetRealValue();
}
```

Il metodo ibmGetRealValue restituisce una copia dell'oggetto in sola lettura. L'applicazione non deve modificare questo valore. Il metodo ibmGetDirtyAttributes restituisce un elenco di stringhe che rappresentano gli attributi modificati dall'applicazione durante questa transazione. Lo scenario di utilizzo dell'elemento primario di ibmGetDirtyAttributes è un JDBC Java database connectivity) oppure un programma di caricamento basato su CMP. Solo gli attributi presenti nell'elenco devono essere aggiornati nell'istruzione SQL o nell'oggetto associato alla tabella, il che consente al programma di caricamento di generare SQL più efficienti. Quando si esegue il commit di una copia durante una transazione write e se è collegato un programma di caricamento, per ottenere questa informazione, il programma di caricamento potrà effettuare il cast dei valori degli oggetti modificati sull'interfaccia ValueProxyInfo.

Gestione del metodo equals quando si utilizza COPY_ON_WRITE o le deleghe: ad esempio, il codice seguente costruisce un oggetto Person e lo inserisce in un ObjectMap. Successivamente, esso recupera lo stesso oggetto utilizzando il metodo ObjectMap.get. Viene eseguito il cast del valore sull'interfaccia. Se si esegue il cast del valore sull'interfaccia Person, ne risulterà un'eccezione ClassCastException, poiché il valore restituito rappresenta un proxy che implementa l'interfaccia IPerson e non l'oggetto Person. La verifica di uguaglianza non riesce quando si utilizza l'operazione == poiché non si tratta di oggetti uguali.

```
session.begin();
// new the Person object
Person p = new Person(...);
personMap.insert(p.getName, p);
// retrieve it again, remember to use the interface for the cast
IPerson p2 = personMap.get(p.getName());
if(p2 == p) {
    // they are the same
} else {
    // they are not
}
```

Un'altra considerazione riguarda la necessità di sostituire 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 ed eseguire il cast dell'argomento in modo che sia un IPerson. Poiché l'argomento potrebbe essere un proxy che implementa l'interfaccia IPerson, quando si confrontano delle istanza per verificarne l'uguaglianza è necessario utilizzare i metodi getAge e getName.

```
{
    if ( obj == null ) return false;
    if ( obj instanceof IPerson ) {
        IPerson x = (IPerson) obj;
        return ( age.equals( x.getAge() ) && name.equals( x.getName() ) )
    }
    return false;
}
```

Requisiti delle configurazioni ObjectQuery e HashIndex: quando si utilizza COPY_ON_WRITE con un ObjectQuery o con un plug-in HashIndex, è importante configurare lo schema ObjectQuery ed il plug-in HashIndex in modo che accedano agli oggetti utilizzando i metodi delle proprietà, che rappresenta l'impostazione predefinita. Se sono invece configurati in modo da utilizzare l'accesso al campo, l'indice ed il motore di query cercheranno di accedere ai campi nell'oggetto proxy, operazione che restituirà sempre un valore nullo oppure 0, poiché l'istanza dell'oggetto sarà un proxy.

NO_COPY

La modalità NO_COPY consente di assicurarsi che un'applicazione non modifichi mai un oggetto valore ottenuto utilizzando il metodo ObjectMap.get in cambio di un miglioramento delle prestazioni. Quando si utilizza questa modalità l'argomento valueInterfaceClass verrà ignorato. Se si utilizza questo metodo non verrà effettuata mai alcuna copia del valore. Se l'applicazione modifica dei valori, i dati presenti nella BackingMap saranno danneggiati. La modalità NO_COPY è utile principalmente per le mappe in sola lettura, dove i dati non sono mai modificati dall'applicazione. Se l'applicazione utilizza questa modalità e si manifestano dei problemi, passare alla modalità COPY_ON_READ_AND_COMMIT per vedere se i problemi persistono. Se i problemi scompaiono, significa che l'applicazione sta modificando il valore restituito dal metodo ObjectMap.set durante la transazione o dopo che ne sia stato effettuato il commit. Tutte le mappe associate con le entità API EntityManager utilizzano automaticamente questa modalità indipendentemente da ciò che è stato specificato nella configurazione di eXtreme Scale.

Tutte le mappe associate con le entità API EntityManager utilizzano automaticamente questa modalità indipendentemente da ciò che è stato specificato nella configurazione di eXtreme Scale.

COPY_TO_BYTES

È possibile memorizzare gli oggetti in un formato serializzato invece che in un formato POJO. Utilizzando le impostazioni COPY_TO_BYTES, è possibile diminuire lo spazio occupato in memoria da un grafico di Oggetti di grandi dimensioni. Consultare “Mappe array di byte” a pagina 40 per ulteriori informazioni.

Uso errato di CopyMode

Quando un'applicazione tenta di migliorare le prestazioni utilizzando le modalità di copia COPY_ON_READ, COPY_ON_WRITE o NO_COPY si verificano errori, come descritto in precedenza. Gli errori intermittenti non si verificano quando si cambia la modalità di copia in COPY_ON_READ_AND_COMMIT.

Problema

Il problema potrebbe essere causato da dati danneggiati nella mappa ObjectGrid, il che è dovuto alla violazione da parte dell'applicazione del contratto di programmazione della modalità di copia in uso. Il danneggiamento dei dati può causare errori imprevedibili che si presentano in modo intermittente o in maniera inspiegabile o inaspettata.

Soluzione

L'applicazione deve essere conforme al contratto di programmazione definito per la modalità di copia in uso. Per le modalità di copia COPY_ON_READ e COPY_ON_WRITE, l'applicazione utilizza un riferimento ad un oggetto valore esterno all'ambito della transazione da cui è stato ottenuto il riferimento al valore. Per utilizzare queste modalità, l'applicazione deve cancellare il riferimento all'oggetto valore dopo il completamento della transazione ed ottenere un nuovo riferimento in ciascuna transazione che accede l'oggetto valore. Per la modalità NO_COPY, l'applicazione non deve mai modificare l'oggetto valore. In questo caso, scrivere l'applicazione in modo che non modifichi l'oggetto valore oppure impostarla in modo che utilizzi una modalità di copia differente.

Mappe array di byte

Le coppie chiave-valore possono essere memorizzate nelle proprie mappe in un array di byte invece che nel modulo POJO, riducendo lo spazio occupato in memoria utilizzabile da un grande grafico di oggetti.

Vantaggi

Il quantitativo di memoria che viene consumato aumenta con il numero di oggetti in un grafico di oggetti. Riducendo un grafico complicato di oggetti in un array di byte, viene gestito solo un oggetto nell'heap invece di svariati oggetti. Con questa riduzione del numero di oggetti nell'heap, il runtime Java ha meno oggetti da ricercare durante la raccolta dati obsoleti.

Il meccanismo predefinito di copia utilizzato da WebSphere eXtreme Scale è la serializzazione, che è dispendiosa. Ad esempio, utilizzando la modalità predefinita di copia `COPY_ON_READ_AND_COMMIT`, viene fatta una copia sia al momento della lettura che al momento del richiamo. Invece di fare una copia al momento della lettura, con array di byte, il valore viene deserializzato dai byte, ed invece di fare una copia al momento del commit, il valore viene serializzato in byte. L'utilizzo di array di byte produce come risultato una equivalente coerenza di dati nelle impostazioni predefinite con una riduzione dell'utilizzo della memoria.

Quando si utilizzano array di byte, tener presente che disporre di un meccanismo di serializzazione ottimizzato è cruciale per poter osservare una riduzione nel consumo della memoria. Per ulteriori informazioni, consultare “Prestazione della serializzazione” a pagina 232.

Configurazione di mappe array di byte

È possibile abilitare mappe array di byte con il file XML ObjectGrid modificando l'attributo CopyMode utilizzato da una mappa sull'impostazione COPY_TO_BYTES, mostrato nel seguente esempio:

```
<backingMap name="byteMap" copyMode="COPY_TO_BYTES" />
```

Per ulteriori informazioni,

Per ulteriori informazioni, consultare l'argomento nel file descrittore XML ObjectGrid nella *Guida alla gestione*.

Considerazioni

Bisogna ponderare se utilizzare o meno le Mappe array di byte in un dato scenario. Sebbene si possa ridurre l'utilizzo della memoria, quando si utilizzano array di byte può aumentare l'uso del processore.

L'elenco di seguito riportato sottolinea diversi fattori da tenere in considerazione prima di scegliere di utilizzare la funzione della mappa array di byte.

Tipo di oggetto

Comparativamente, la riduzione della memoria può non essere possibile quando si utilizzano mappe array di byte per gli stessi tipi di oggetto. Di conseguenza, esistono diversi tipi di oggetti per i quali non bisognerebbe usare le mappe array di byte. Se come valori si stanno utilizzando alcuni dei wrapper originali Java o un POJO che non contiene riferimenti ad altri oggetti (solo campi primitivi di memorizzazione), il numero di Oggetti Java è già quanto più basso possibile – ce n'è solo uno. Poiché il quantitativo di memoria utilizzato dall'oggetto è già ottimizzato, non si consiglia l'utilizzo di una mappa array di byte per questi tipi di oggetti. Le mappe array di byte si adattano maggiormente a tipi di oggetti che contengono altri oggetti o raccolte di oggetti dove il numero totale di oggetti POJO è maggiore di uno.

Ad esempio, se si ha un oggetto Customer con un indirizzo dell'ufficio ed un indirizzo di casa nonché una raccolta di ordini, il numero degli oggetti nell'heap ed il numero di byte utilizzato da quegli oggetti può essere ridotto con le mappe array di byte.

Accesso locale

Quando si utilizzano altre modalità di copia, le applicazioni possono essere ottimizzate quando vengono fatte le copie se gli oggetti sono duplicabili con l'ObjectTransformer predefinito oppure quando un ObjectTransformer personalizzato viene fornito di un metodo copyValue ottimizzato. Confrontato con altri metodi di copia, le operazioni di copia in lettura, scrittura o di commit avranno costi aggiuntivi quando si accede agli oggetti in locale. Ad esempio, se si ha una cache locale in una topologia distribuita o se si accede direttamente ad un'istanza ObjectGrid server locale, il tempo di accesso e di commit aumenterà quando si utilizzano mappe array di byte a causa dei costi di serializzazione. Un costo simile sarà riscontrabile in una topologia distribuita se si usano agenti di griglie di dati o se si accede al server primario quando si utilizza il plug-in ObjectGridEventGroup.ShardEvents.

Interazioni di plug-in

Con le mappe array di byte, gli oggetti non vengono deserializzati quando comunicano da un client verso un server a meno che il server non abbia bisogno del modulo POJO. I Plugin che interagiscono con il valore mappa sperimenteranno una riduzione delle prestazioni a causa del requisito di deserializzazione del valore.

Eventuali plug-in che dovessero utilizzare LogElement.getCacheEntry o LogElement.getCurrentValue constateranno questo costo aggiuntivo. Se si vuole richiamare la chiave, è possibile utilizzare LogElement.getKey, che evita il sovraccarico aggiuntivo associato al metodo LogElement.getCacheEntry().getKey. Nelle seguenti sezioni vengono trattati i plug-in alla luce dell'utilizzo degli array di byte.

Indici e query

Quando gli oggetti vengono memorizzati in formato POJO, il costo dell'indicizzazione e delle query è minimo in quanto l'oggetto non ha bisogno di

essere deserializzato. Quando si utilizza una mappa array di byte, si avrà un costo aggiuntivo di deserializzazione dell'oggetto. In generale, se la propria applicazione utilizza indici o query, non si consiglia l'utilizzo di mappe array di byte a meno che non vengano eseguite solo query su attributi chiave.

Blocco ottimistico

Quando si utilizza la strategia di blocco ottimistico, si avranno costi aggiuntivi durante gli aggiornamenti e le operazioni di invalidazione. Questo deriva dall'aver deserializzato il valore sul server per richiamare il valore della versione per eseguire il controllo ottimistico di collisione. Se si sta eseguendo il blocco ottimistico per garantire le operazioni fetch e non si ha bisogno del controllo ottimistico di collisione, è possibile utilizzare com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback per disabilitare il controllo della versione.

Programma di caricamento

Con un programma di caricamento, si avrà anche il costo nel runtime eXtreme Scale derivante dalla serializzazione e riserializzazione del valore quando viene utilizzato dal programma di caricamento. È sempre possibile utilizzare le mappe array di byte con i programmi di caricamento, ma occorre considerare il costo derivante dall'apportare modifiche al valore in un tale scenario. Ad esempio, è possibile utilizzare la funzione di array di byte nel contesto di una cache prevalentemente in lettura. In questo caso, il beneficio derivante dall'avere meno oggetti nell'heap e meno memoria utilizzata supererà il costo nel quale si incorre dall'utilizzo degli array di byte nelle operazioni di inserimento e di aggiornamento.

ObjectGridEventListener

Quando si utilizza il metodo transactionEnd nel plug-in ObjectGridEventListener, si avrà un costo aggiuntivo dal lato server per le richieste remote durante l'accesso alla CacheEntry di LogElement o al valore corrente. Se l'implementazione del metodo non accede a questi campi, non si avranno costi aggiuntivi.

Utilizzo di sessioni per accedere ai dati della griglia

Le applicazioni possono iniziare e terminare le transazioni tramite l'interfaccia Session. L'interfaccia Session fornisce anche l'accesso alle interfacce ObjectMap e JavaMap basate sulle applicazioni.

Ogni istanza ObjectMap o JavaMap è collegata direttamente ad uno specifico oggetto Session. Ogni thread che desidera accedere ad un eXtreme Scale deve prima ottenere una sessione dall'oggetto ObjectGrid. Un'istanza Session non può essere condivisa contemporaneamente tra i thread. WebSphere eXtreme Scale non utilizza uno storage locale di thread ma le restrizioni della piattaforma potrebbero limitare l'opportunità di passare una sessione da un thread ad un altro.

Metodi

Con l'interfaccia Session sono disponibili i metodi riportati di seguito. Per ulteriori informazioni sui seguenti metodi, consultare la documentazione dell'API:

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

    void begin() throws TransactionAlreadyActiveException, TransactionException;

    void beginNoWriteThrough() throws TransactionAlreadyActiveException,
```

```
   TransactionException;

    public void commit() throws NoActiveTransactionException, TransactionException;

    public void rollback() throws NoActiveTransactionException, TransactionException;

    public void flush() throws TransactionException;

    TxID getTxID() throws NoActiveTransactionException;

    boolean isWriteThroughEnabled();

    void setTransactionType(String tranType);

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

    ObjectGrid getObjectGrid();

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

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

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

Metodo Get

Un'applicazione ottiene un'istanza Session da un oggetto ObjectGrid tramite il metodo ObjectGrid.getSession. Il seguente esempio mostra come ottenere un'istanza Session:

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

Una volta ottenuta una sessione il thread conserva un riferimento alla sessione per proprio utilizzo. Richiamare il metodo getSession più volte restituisce un uovo oggetto Session ogni volta.

Transazioni e metodi Session

Un metodo Session può essere utilizzato per iniziare, eseguire il commit o eseguire il rollback delle transazioni. Le operazioni con BackingMap mediante ObjectMap e JavaMap vengono eseguite in modo più efficiente all'interno di una transazione Session. Dopo l'avvio di una transazione, qualsiasi modifica a una o più BackingMap in quell'ambito di transazione viene memorizzata in una speciale cache di transazione fino al commit della transazione. Quando viene eseguito il commit di una transazione, le modifiche in sospeso vengono applicate alle BackingMap e ai Loader, e divengono visibili a qualsiasi altro client di quell'ObjectGrid.

WebSphere eXtreme Scale supporta anche la possibilità di eseguire automaticamente il commit delle transazioni, noto anche come commit automatico. Se vengono seguite operazioni ObjectMap all'esterno del contesto di una transazione attiva, viene avviata una transazione implicita prima dell'operazione e il commit della transazione viene eseguito automaticamente 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 ha senso solo quando un Loader viene associato ad una BackingMap. Il metodo flush richiama il Loader con la serie corrente di modifiche nella cache della transazione. Il Loader applica le modifiche al backend. Il commit di queste modifiche non viene eseguito quando viene richiamato il flush. Se il commit di una transazione Session viene eseguito dopo la chiamata di un flush, solo gli aggiornamenti verificatisi dopo la chiamata del flush vengono applicati a Loader. Se dopo una chiamata di flush viene eseguito il rollback di una transazione Session, le modifiche sottoposte a flush vengono eliminate insieme a tutte le altre modifiche in sospeso della transazione. Utilizzare il metodo Flush saltuariamente poiché limita l'opportunità di operazioni batch con un Loader. Di seguito è riportato un esempio di utilizzo del metodo Session.flush:

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

Metodo NoWriteThrough

Alcune mappe eXtreme Scale vengono sottoposte a backup da un Loader, che fornisce uno storage persistente per i dati della mappa. A volte è utile eseguire il commit dei dati solo per la mappa eXtreme Scale e non inviare i dati al Loader. L'interfaccia Session a questo scopo fornisce il metodo beginNoWriteThough. Il metodo beginNoWriteThrough avvia una transazione come il metodo begin. Con il metodo beginNoWriteThrough, quando viene eseguito il commit della transazione, viene eseguito solo il commit dei dati per la mappa eXtreme Scale in memoria e non per lo storage persistente fornito dal Loader. Questo metodo è molto utile quando si esegue il precaricamento dei dati sulla mappa.

Quando si utilizza un'istanza ObjectGrid distribuita, il metodo beginNoWriteThrough è utile per apportare modifiche solo alla cache locale, senza modificare la cache remota sul server. Se è noto che i dati della cache locale sono obsoleti, l'utilizzo del metodo beginNoWriteThrough può consentire la convalida delle voci sulla cache locale senza convalidarle anche sul server.

L'interfaccia Session fornisce anche il metodo isWriteThroughEnabled per determinare quale tipo di transazione è attiva al momento.

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

Ottenimento del metodo dell'oggetto TxID

L'oggetto TxID è un oggetto opaco che identifica la transazione attiva. Utilizzare l'oggetto TxID per i seguenti scopi:

- Per il confronto quando si cerca una particolare transazione.
- Per memorizzare i dati condivisi tra gli oggetti TransactionCallback e Loader.

Per ulteriori informazioni sulla funzione slot oggetto, vedere i plug-in TransactionCallback e Loader.

Metodo di monitoraggio delle prestazioni

Se si utilizza eXtreme Scale all'interno di WebSphere Application Server, potrebbe essere necessario reimpostare il tipo di transazione per il monitoraggio delle prestazioni. È possibile impostare il tipo di transazione con il metodo setTransactionType. Per ulteriori informazioni sul metodo setTransactionType, consultare Monitoraggio delle prestazioni di ObjectGrid con PMI (Performance Monitoring Infrastructure) di WebSphere Application Server.

Elaborazione di un metodo LogSequence completo

WebSphere eXtreme Scale può propagare serie di modifiche di mappa sui listener ObjectGrid come mezzo di distribuzione delle mappe da una Java virtual machine ad un'altra. Per facilitare al listener l'elaborazione delle LogSequence ricevute, l'interfaccia Session fornisce il metodo processLogSequence. Questo metodo esamina ogni LogElement all'interno della LogSequence ed esegue l'operazione appropriata, ad esempio, inserimento, aggiornamento, invalidazione e così via, rispetto alla BackingMap identificata dal MapName della LogSequence. Una sessione ObjectGrid deve essere disponibile prima che venga richiamato il metodo processLogSequence. L'applicazione è inoltre responsabile dell'invio delle chiamate di commit o rollback appropriate per completare la sessione. L'elaborazione del commit automatico non è disponibile per questa chiamata di metodo. Una normale elaborazione dell'ObjectGridEventListener ricevente sulla JVM remota, sarebbe avviare una sessione utilizzando il metodo beginNoWriteThrough, che impedisce la propagazione delle modifiche, seguito da una chiamata a questo metodo processLogSequence ed infine eseguendo il commit o il rollback della transazione.

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

Metodo markRollbackOnly

Questo metodo viene utilizzato per contrassegnare la transazione corrente come "rollback only" (solo rollback). Contrassegnando una transazione come "rollback only" si assicura che anche se l'applicazione richiama il metodo di commit, viene eseguito il rollback della transazione. Questo metodo in genere viene utilizzato dallo stesso ObjectGrid o dall'applicazione quando sa che potrebbe verificarsi un danneggiamento dei dati se viene consentito il commit della transazione. Dopo il richiamo di questo metodo, l'oggetto Throwable passato al metodo viene concatenato all'eccezione com.ibm.websphere.objectgrid.TransactionException che comporta il metodo commit, se viene richiamato su una sessione precedentemente contrassegnata come "rollback only". Tutte le successive chiamate a questo metodo per una transazione che è già contrassegnata come "rollback only" vengono ignorate. Vale a dire, viene utilizzata solo la prima chiamata che passa un riferimento Throwable non nullo. Una volta completata la transazione contrassegnata, il contrassegno "rollback only" viene rimosso in modo che possa essere eseguito il commit della successiva transazione avviata da Session.

Metodo isMarkedRollbackOnly

Restituisce un valore se Session è attualmente contrassegnata come "rollback only". Questo metodo restituisce il valore booleano true se, e solo se, precedentemente è stato chiamato il metodo markRollbackOnly su questa Session e la transazione avviata da Session è ancora attiva.

Metodo setTransactionTimeout

Imposta su un numero di secondi specificato il timeout di transazione per la successiva transazione avviata da questa Session. Questo metodo non influisce sul timeout di transazione delle transazioni avviate in precedenza da questa Session. Interessa solo le transazioni avviate dopo la chiamata a questo metodo. Se questo metodo non viene mai chiamato, viene utilizzato il valore di timeout trasmesso al metodo setTxTimeout del metodo com.ibm.websphere.objectgrid.ObjectGrid.

Metodo getTransactionTimeout

Questo metodo restituisce il valore di timeout della transazione espresso in secondi. Questo metodo restituisce l'ultimo valore trasmesso come valore di timeout al metodo setTransactionTimeout. Se il metodo setTransactionTimeout non viene mai chiamato, viene utilizzato il valore di timeout trasmesso al metodo setTxTimeout del metodo com.ibm.websphere.objectgrid.ObjectGrid.

Metodo transactionTimedOut

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

Metodo isFlushing

Questo metodo restituisce il valore booleano true se, e solo se, tutte le modifiche della transazione vengono svuotate nel plug-in Loader come risultato del metodo flush dell'interfaccia Session richiamata. Un plug-in Loader può trovare utile questo metodo quando deve sapere perché 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 della transazione come risultato del metodo commit dell'interfaccia Session richiamata. Un plug-in Loader potrebbe trovare utile questo metodo quando deve sapere perché è stato richiamato il metodo batchUpdate.

Metodo setRequestRetryTimeout

Questo metodo imposta il valore di timeout dei tentativi di richiesta per la sessione espresso in millisecondi. Se il client ha impostato un timeout dei tentativi di richiesta, l'impostazione della sessione sostituisce il valore del client.

Metodo getRequestRetryTimeout

Questo metodo ottiene l'impostazione corrente del timeout dei tentativi di richiesta sulla sessione. Il valore `-1` indica che il timeout non è impostato. Il valore `0` indica che è in modalità fail-fast. Un valore maggiore di `0` indica l'impostazione del timeout in millisecondi.

SessionHandle per l'instradamento

Quando si utilizza una politica di posizionamento di partizione per contenitore, è possibile utilizzare un SessionHandle. Un'istanza SessionHandle contiene informazioni sulla partizione relative alla sessione corrente e possono essere riutilizzate per una nuova sessione.

Un SessionHandle contiene le informazioni per la partizione a cui viene collegata la sessione corrente. Il SessionHandle è estremamente utile per la politica di posizionamento di partizione per contenitore e può essere serializzato con la serializzazione Java standard.

Se si dispone di un'istanza SessionHandle, è possibile applicare quell'handle ad una sessione con il metodo setSessionHandle(SessionHandle target), fornendo l'handle come destinazione. È possibile richiamare il SessionHandle con il metodo Session.getSessionHandle.

Poiché è applicabile solo in uno scenario di posizionamento per contenitore, il richiamo del SessionHandle genera un IllegalStateException se un determinato ObjectGrid ha più serie di mappe per contenitore o non ne ha nessuna. Se non si richiama il metodo setSessionHandle prima di richiamare il metodo getSessionHandle l'appropriato SessionHandle verrà selezionato in base alla configurazione ClientProperties.

È possibile anche utilizzare la classe helper SessionHandleTransformer per convertire l'handle in formati diversi. I metodi di questa classe possono modificare la rappresentazione di un handle da array di byte ad istanza, da stringa ad istanza e viceversa per entrambi i casi, e possono anche scrivere il contenuto dell'handle nel flusso di output.

Per un esempio di come è possibile utilizzare un SessionHandle, consultare l'argomento relativo all'instradamento a zone preferite nella *Panoramica del prodotto*.

Integrazione SessionHandle

Un oggetto SessionHandle include le informazioni sulla partizione per la Session alla quale è legato e facilita l'instradamento delle richieste. Gli oggetti SessionHandle si riferiscono solo allo scenario di posizionamento delle partizioni per contenitore.

Oggetto SessionHandle per l'instradamento di richieste

È possibile eseguire il bind di un oggetto SessionHandle ad una Session nei seguenti modi:

Suggerimento: In ognuna delle seguenti chiamate di metodo, una volta che un SessionHandle è associato ad una Session, SessionHandle può essere ottenuto dal metodo Session.getSessionHandle che viene utilizzato con il metodo Session.setSessionHandle.

- Richiamare il metodo Session.getSessionHandle: nel momento in cui viene richiamato questo metodo, se non vi è alcun SessionHandle associato alla Session, un SessionHandle verrà selezionato in modo casuale e associato alla Session.
- Richiamare operazioni create, read, update, delete (CRUD) transazionali: nel momento in cui vengono richiamati questi metodi o al momento del commit, se

non vi è alcun SessionHandle associato a Session, SessionHandle verrà selezionato in modo casuale e associato alla Session.

- Richiamare il metodo ObjectMap.getNextKey: nel momento in cui questo metodo viene richiamato, se non vi è alcun SessionHandle associato alla Session, la richiesta dell'operazione verrà istradata in modo casuale a singole partizioni finché non viene ottenuta una chiave. Se viene restituita una chiave da una partizione, un SessionHandle che corrisponde a quella partizione verrà associato alla Session. Se non viene trovata alcuna chiave, non verrà associato alcun SessionHandle alla Session.
- Richiamare il metodo QueryQueue.getNextEntity o QueryQueue.getNextEntities: nel momento in cui viene richiamato questo metodo, se non vi è alcun SessionHandle associato alla Session, la richiesta di operazione verrà instradata in modo casuale alle singole partizioni finché non viene ottenuto un oggetto. Se viene restituito un oggetto da una partizione, un SessionHandle che corrisponde a quella partizione verrà associato alla Session. Se non viene trovato alcun oggetto, non verrà associato alcun SessionHandle alla Session.
- Impostare un SessionHandle con il metodo Session.setSessionHandle(SessionHandle sh): se si ottiene un SessionHandle dal metodo Session.getSessionHandle, è possibile associare un SessionHandle ad una Session. Impostando un SessionHandle si influenza l'instradamento delle richieste nell'ambito della Session alla quale è associato.

Il metodo Session.getSessionHandle restituirà sempre un SessionHandle e assocerà automaticamente un SessionHandle nella Session se non vi è alcun SessionHandle associato alla Session. Se si desidera solo verificare se una Session presenta un SessionHandle, richiamare il metodo Session.isSessionHandleSet. Se questo metodo restituisce un valore pari a false, non viene attualmente associato alcun SessionHandle alla Session.

```
new method added to Session API
   /**
     * Determines if a SessionHandle is currently set on this Session.
     *
     * @return true if a SessionHandle is currently set on this session.
     *
     * @since 7.1
     */

    public boolean isSessionHandleSet();
```

Principali tipi di operazione nello scenario di posizionamento per contenitore

Di seguito viene fornito un riepilogo del comportamento di instradamento dei principali tipi di operazione nello scenario di posizionamento delle partizioni per contenitore rispetto ad oggetti SessionHandle.

- **Oggetto Session con oggetto SessionHandle associato**
  - Indice - API MapIndex e MapRangeIndex: SessionHandle
  - Query e ObjectQuery: SessionHandle
  - Agent - API MapGridAgent e ReduceGridAgent: SessionHandle
  - ObjectMap.Clear: SessionHandle
  - ObjectMap.getNextKey: SessionHandle
  - QueryQueue.getNextEntity, QueryQueue.getNextEntities: SessionHandle
  - Operazioni CRUD transazionali (API ObjectMap API e EntityManager): SessionHandle

- **Oggetto Session senza oggetto SessionHandle associato**
  - Indice - API MapIndex e MapRangeIndex: tutte le partizioni attualmente attive
  - Query e ObjectQuery: partizione specificata tramite il metodo setPartition di Query e ObjectQuery
  - Agent - MapGridAgent e ReduceGridAgent
    - Non supportato: metodo ReduceGridAgent.reduce(Session s, ObjectMap map, Collection key) e MapGridAgent.process(Session s, ObjectMap map, Object key).
    - Tutte le partizioni attualmente attive: metodo ReduceGridAgent.reduce(Session s, ObjectMap map) e MapGridAgent.processAllEntries(Session s, ObjectMap map).
  - ObjectMap.clear: tutte le partizioni attualmente attive.
  - ObjectMap.getNextKey: associa un SessionHandle alla Session se viene restituita una chiave da una delle partizioni selezionate in modo casuale. Se non viene restituita alcuna chiave, la Session non viene associata a SessionHandle.
  - QueryQueue: specifica una partizione con il metodo QueryQueue.setPartition. Se non viene impostata alcuna partizione, il metodo selezionerà in modo casuale la partizione da restituire. Se viene restituito un oggetto, la Session corrente viene associata al SessionHandle associato alla partizione che restituisce l'oggetto. Se non viene restituito alcun oggetto, la Session non viene associata ad un SessionHandle.
  - Operazioni CRUD transazionali (API ObjectMap e API EntityManager): selezionare in modo casuale una partizione.

  Nella maggior parte dei casi, si deve utilizzare SessionHandle per controllare l'instradamento ad una partizione particolare. È possibile richiamare e memorizzare nella cache SessionHandle dalla Session che inserisce i dati. Dopo la memorizzazione nella cache di SessionHandle, è possibile impostarlo su un'altra Session così da poter instradare le richieste ad una partizione specificata dal SessionHandle nella cache. Per eseguire operazioni come ObjectMap.clear senza SessionHandle, è possibile impostare temporaneamente SessionHandle su null richiamando Session.setSessionHandle(null). Senza un SessionHandle specificato, le operazioni verranno eseguite su tutte le partizioni attualmente attive.
- **Comportamento di instradamento di QueryQueue**

  Nello scenario di posizionamento per contenitore, è possibile utilizzare SessionHandle per controllare l'instradamento dei metodi getNextEntity e getNextEntities dell'API QueryQueue. Se la Session è associata a SessionHandle, le richieste vengono instradate alla partizione alla quale è associato SessionHandle. Se la Session non è associata a SessionHandle, le richieste vengono instradate alla partizione impostata con il metodo QueryQueue.setPartition se una partizione è stata impostata in questo modo. Se la Session non ha alcun SessionHandle associato o partizione, verrà restituita una partizione selezionata in modo casuale. Se non viene trovata alcuna partizione, il processo si arresta e non viene associata alcun SessionHandle alla Session corrente.

Il seguente frammento di codice mostra in che modo utilizzare SessionHandle.

Programming example for the SessionHandle object

```
Session ogSession = objectGrid.getSession();
```

```
// binding the SessionHandle
SessionHandle sessionHandle = ogSession.getSessionHandle();

ogSession.begin();
ObjectMap map = ogSession.getMap("planet");
map.insert("planet1", "mercury");

// transaction is routed to partition specified by SessionHandle
ogSession.commit();

// cache the SessionHandle that inserts data
SessionHandle cachedSessionHandle = ogSession.getSessionHandle();

// verify if SessionHandle is set on the Session
boolean isSessionHandleSet = ogSession.isSessionHandleSet();

// temporarily unbind the SessionHandle from the Session
if(isSessionHandleSet) {
    ogSession.setSessionHandle(null);
}

// if the Session has no SessionHandle bound,
// the clear operation will run on all current active partitions
// and thus remove all data from the map in the entire grid
map.clear();

// after clear is done, reset the SessionHandle back,
// if the Session needs to use previous SessionHandle.
// Optionally, calling getSessionHandle can get a new SessionHandle
ogSession.setSessionHandle(cachedSessionHandle);
```

Considerazioni sulla progettazione delle applicazioni

Nello scenario delle strategie di posizionamento per contenitore si deve utilizzare SessionHandle per ulteriori operazioni. SessionHandle controlla l'instradamento alle partizioni. Per richiamare i dati, il SessionHandle associato alla Session deve essere lo stesso SessionHandle proveniente da un'eventuale transazione di dati di inserimento,.

Quando si desidera eseguire un'operazione senza che un SessionHandle sia impostato sulla sessione, è possibile annullare l'associazione a un SessionHandle da una Session effettuando una chiamata di metodo Session.setSessionHandle(null).

Quando una Session viene associata ad un SessionHandle, tutte le richieste di operazioni verranno instradate alla partizione specificata da SessionHandle. Se SessionHandle non è impostato, le operazioni vengono instradate verso tutte le partizioni o verso una partizione selezionata in modo casuale.

Oggetti memorizzati in cache senza relazioni coinvolte (API ObjectMap)

ObjectMaps sono come le mappe Java che consentono di memorizzare i dati come coppie di chiave-valore. ObjectMaps fornisce un semplice ed intuitivo approccio per l'applicazione per memorizzare i dati. Un ObjectMap è adatto per oggetti memorizzati nella cache che non implicano relazioni. Se sono implicate relazioni di un oggetto, allora si dovrebbe utilizzare l'API EntityManager.

Per ulteriori informazioni relative all'API EntityManager, consultare “Oggetti memorizzati nella cache e loro relazioni (API EntityManager)” a pagina 61.

Generalmente le applicazioni assumono un riferimento WebSphere eXtreme Scale e successivamente ottengono un oggetto Session nel riferimento per ciascun thread. Le sessioni non possono essere condivise tra thread. Il metodo getMap della sessione restituisce un riferimento ad un ObjectMap da utilizzare per questo thread.

Introduzione a ObjectMap

L'interfaccia ObjectMap viene utilizzata per l'interazione transazionale tra le applicazioni e i BackingMap.

Scopo

Un'istanza ObjectMap viene ottenuta da un oggetto Session corrispondente al thread corrente. L'interfaccia ObjectMap è lo strumento principale utilizzato dalle applicazioni per apportare modifiche alle voci in un BackingMap.

Come ottenere un'istanza ObjectMap

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

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

Ogni istanza ObjectMap corrisponde a un particolare oggetto Session. Più chiamate del metodo getMap su un particolare oggetto Session con lo stesso nome BackingMap restituisce sempre la stessa istanza ObjectMap.

Transazioni sottoposte al commit automatico

Le operazioni a fronte di BackingMaps che utilizzano ObjectMaps e JavaMaps vengono eseguite in modo più efficiente all'interno di una transazione Session. WebSphere eXtreme Scale fornisce il supporto di autocommit quando i metodi nelle interfacce ObjectMap e JavaMap vengono richiamati al di fuori di una transazione Session. I metodi avviano una transazione implicita, effettuano l'operazione richiesta ed eseguono il commit della transazione implicita.

Semantiche del metodo

Di seguito è riportata una descrizione delle semantiche presenti dietro a ciascun metodo nelle interfacce ObjectMap e JavaMap. Il metodo setDefaultKeyword, il metodo invalidateUsingKeyword e i metodi che hanno un argomento Serializable vengono trattati nell'argomento Parole chiave. Il metodo setTimeToLive viene trattato nell'argomento Programmi di eliminazione. Per ulteriori informazioni su questi metodi, vedere la documentazione API.

Metodo containsKey
: Il metodo containsKey determina se una chiave possiede un valore in BackingMap o Loader. Se un'applicazione supporta valori nulli, questo metodo può essere utilizzato per determinare se un riferimento nullo restituito da un'operazione get si riferisce a un valore nullo o indica che BackingMap e Loader non contengono la chiave.

Metodo flush
: Le semantiche del metodo flush sono simili a quelle del metodo flush

nell'interfaccia Session. La differenza principale è che il flush di Session viene applicato alle modifiche correnti in sospeso per tutte le mappe modificate nella sessione corrente. Con questo metodo, nel programma di caricamento viene eseguito solo il flush delle modifiche di questa istanza ObjectMap.

Metodo get
Il metodo get esegue il fetch della voce dall'istanza BackingMap. Se la voce non viene trovata nell'istanza BackingMap ma un programma di caricamento (Loader) viene associato all'istanza BackingMap, tale istanza tenta di eseguire il fetch della voce dal programma di caricamento. Viene fornito il metodo getAll per consentire l'elaborazione fetch nel batch.

Metodo getForUpdate
Il metodo getForUpdate è uguale al metodo get, ma l'utilizzo del metodo getForUpdate comunica a BackingMap e Loader l'intenzione di aggiornare la voce. Un Loader può utilizzare questo suggerimento (hint) per emettere una query SELECT for UPDATE su un backend del database. Se viene definita una strategia di blocco pessimistico per la BackingMap, la voce viene bloccata dal gestore blocco. Il metodo getAllForUpdate viene fornito per consentire l'elaborazione fetch del batch.

Metodo insert
Il metodo insert inserisce una voce in BackingMap e Loader. L'utilizzo di questo metodo comunica a BackingMap e Loader che si desidera inserire una voce non esistente prima. Quando si richiama questo metodo su una voce esistente, si verifica un'eccezione quando il metodo viene richiamato o quando viene eseguito il commit della transazione corrente.

Metodo invalidate
Le semantiche del metodo invalidate dipendono dal valore del parametro **isGlobal** passato al metodo. Il metodo invalidateAll viene fornito per consentire l'elaborazione di invalidazione del batch.

L'invalidazione locale viene specificata quando il valore false viene passato come parametro **isGlobal** del metodo invalidate. L'invalidazione locale elimina tutte le modifiche apportate alla voce nella cache di transazione. Se l'applicazione immette un comando get, viene eseguito il fetch della voce dall'ultimo valore con commit in BackingMap. Se nessuna voce è presente nella BackingMap, viene eseguito il fetch della voce dall'ultimo valore sottoposto a flush o commit nel Loader. Quando viene eseguito il commit di una transazione, tutte le voci contrassegnate come invalidate a livello locale non hanno alcun impatto sulla BackingMap. Viene ancora eseguito il commit di eventuali modifiche sottoposte a flush nel Loader anche se la voce è stata invalidata.

L'invalidazione globale viene specificata quando il valore true viene passato come parametro **isGlobal** del metodo invalidate. L'invalidazione globale elimina tutte le modifiche in sospeso della voce nella cache della transazione e ignora il valore BackingMap in successive operazioni eseguite sulla voce. Quando viene eseguito il commit di una transazione, tutte le voci contrassegnate come invalidate a livello globale vengono eliminate dalla BackingMap. Considerare come esempio il seguente scenario di utilizzo per l'invalidazione: viene eseguito il backup della BackingMap da una tabella di database che dispone di una colonna con incremento automatico. Le colonne con incremento sono utili per assegnare numeri univoci ai record. L'applicazione inserisce una voce. Dopo l'inserimento, l'applicazione deve acquisire il numero di sequenza della riga inserita. Poiché l'applicazione identifica la propria copia dell'oggetto

come obsoleta, essa utilizza l'invalidazione globale per ottenere il valore dal Loader. Il seguente codice descrive questo scenario 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 esempio di codice aggiunge una voce per `Billy`. L'attributo della versione di Person viene impostato utilizzando una colonna con incremento automatico nel database. L'applicazione esegue prima un comando di inserimento. Quindi, emette un flush, che provoca l'invio dell'inserimento al Loader e database. Il database imposta la colonna della versione sul successivo numero nella sequenza, rendendo obsoleto l'oggetto Person nella transazione. Per aggiornare l'oggetto, l'applicazione viene invalidata globalmente. Il successivo metodo get emesso ottiene la voce dal programma di caricamento, ignorando il valore di transazione. La voce viene recuperata dal database con il valore di versione aggiornata.

Metodo put

Le semantiche del metodo put sono dipendenti dal richiamo o meno di un precedente metodo get nella transazione per la chiave. Se l'applicazione esegue un'operazione get che restituisce una voce esistente in BackingMap o Loader, il richiamo del metodo put viene interpretato come aggiornamento e restituisce il valore precedente nella transazione. Se un richiamo del metodo put è stato eseguito senza un precedente richiamo del metodo get o se un precedente richiamo del metodo get non ha trovato una voce, l'operazione viene interpretata come inserimento. Le semantiche dei metodi insert e update vengono applicate quando viene eseguito il commit dell'operazione put. Il metodo putAll viene fornito per abilitare l'elaborazione di inserimento e aggiornamento del batch.

Metodo remove

Il metodo remove rimuove la voce da BackingMap e Loader, se viene eseguito il plug-in di un programma di caricamento. Il valore dell'oggetto rimosso viene restituito da questo metodo. Se l'oggetto non esiste, questo metodo restituisce un valore nullo. Il metodo removeAll viene fornito per abilitare l'elaborazione di eliminazione del batch senza i valori di ritorno.

Metodo setCopyMode

Il metodo setCopyMode specifica un valore CopyMode per questo ObjectMap. Con questo metodo, un'applicazione può sostituire il valore CopyMode specificato su BackingMap. Il valore CopyMode specificato è valido fino a quando viene richiamato il metodo clearCopyMode. Entrambi i metodi sono richiamati all'esterno dei limiti transazionali. Non è possibile modificare il valore CopyMode durante una transazione.

Metodo touch

Il metodo touch aggiorna l'ultimo orario di accesso per una voce. Questo metodo non richiama il valore da BackingMap. Utilizzare questo metodo nella sua propria transazione. Se la chiave fornita non esiste nella BackingMap a causa di una invalidazione o rimozione, si verifica un'eccezione durante l'elaborazione del commit.

Metodo update

Il metodo update aggiorna in modo esplicito una voce in BackingMap e

Loader. L'utilizzo di questo metodo indica a BackingMap e Loader che si desidera aggiornare una voce esistente. Si verifica un'eccezione se si richiama questo metodo su una voce non esistente quando il metodo è richiamato o durante l'elaborazione del commit.

Metodo getIndex
Il metodo getIndex tenta di ottenere un indice con nome integrato sulla BackingMap. L'indice non può essere condiviso tra thread ed opera con le stesse regole di Session. Eseguire il cast dell'oggetto indice restituito alla corretta interfaccia indice dell'applicazione, come ad esempio l'interfaccia MapIndex, l'interfaccia MapRangeIndex o un'interfaccia dell'indice personalizzata.

Metodo clear
Il metodo clear rimuove tutte le voci di cache di una mappa da tutte le partizioni. Questa operazione è una funzione di autocommit, pertanto non devono essere presenti transazioni attive durante il richiamo del metodo clear.

Nota: Il metodo clear cancella soltanto il contenuto della mappa per cui viene richiamato, lasciando inalterate eventuali mappe di entità associate. Questo metodo non richiama il plug-in Loader.

Mappe dinamiche

Con la funzione delle mappe dinamiche è possibile creare mappe dopo l'inizializzazione della griglia.

Nelle versioni precedenti, eXtreme Scale richiedeva di definire le mappe prima di inizializzare l'ObjectGrid. Come risultato, bisognava creare tutte le mappe da utilizzare prima dell'esecuzione delle transazioni rispetto a ciascuna mappa.

Vantaggi delle mappe dinamiche

L'introduzione di mappe dinamiche riduce la restrizione derivante dal dover definire tutte le mappe prima dell'inizializzazione. Attraverso l'uso di mappe del template, le mappe ora possono essere create dopo che ObjectGrid è stato inizializzato.

Le mappe del template sono definite nel file XML ObjectGrid. I confronti tra template vengono eseguiti quando una Sessione richiede una mappa che non è stata definita precedentemente. Se il nome della nuova mappa corrisponde all'espressione regolare di una mappa del template, la mappa viene creata dinamicamente e viene assegnato il nome della mappa richiesta. Questa mappa appena creata eredita tutte le impostazioni della mappa del template così come è stata definita nel file XML ObjectGrid.

Creazione di mappe dinamiche

La creazione della mappa dinamica è legata al metodo Session.getMap(String). Le chiamate a questo metodo restituiscono un ObjectMap basato su BackingMap configurato dal file XML ObjectGrid.

Passando in una stringa (String) che corrisponde all'espressione regolare di una mappa del template ne deriverà la creazione di un ObjectMap e di una BackingMap associata.

Per ulteriori informazioni sul metodo Session.getMap(String cacheName), consultare la documentazione API.

La definizione di una mappa di template in XML è tanto semplice quanto l'impostazione di un attributo booleano del template sull'elemento backingMap. Quando il template è impostato su true, il nome della backingMap viene interpretato come un'espressione regolare.

WebSphere eXtreme Scale utilizza la corrispondenza del pattern di espressione regolare Java. Per ulteriori informazioni sul motore dell'espressione regolare in Java, consultare la documentazione API per il pacchetto e le classi java.util.regex.

Segue un file XML ObjectGrid di esempio con una mappa del template definito.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
 <objectGrids>
  <objectGrid name="accounting">
   <backingMap name="payroll" readOnly="false" />
   <backingMap name="templateMap.*" template="true"
    plug-inCollectionRef="templatePlugins" lockStrategy="PESSIMISTIC" />
  </objectGrid>
 </objectGrids>

 <backingMapPluginCollections>
  <backingMapPluginCollection id="templatePlugins">
   <bean id="Evictor"
    className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor" />
  </backingMapPluginCollection>
 </backingMapPluginCollections>
</objectGridConfig>
```

IL file XML precedente definisce una mappa del template ed una del non-template. Il nome della mappa del template è un'espressione regolare: `templateMap.*`. Quando viene chiamato il metodo Session.getMap(String) con un nome della mappa che corrisponde a questa espressione regolare, l'applicazione crea una nuova mappa.

Nota: Se è stata definita più di una mappa del template, assicurarsi che il nome di qualunque argomento per il metodo Session.getMap(String) non corrisponda a più di una mappa del template.

Esempio

La configurazione di una mappa del template è richiesta per creare una mappa dinamica. Aggiungere il booleano del template ad una backingMap nel file XML ObjectGrid.

```
<backingMap name="templateMap.*" template="true" />
```

Il nome della mappa del template viene trattato come un'espressione regolare.

La chiamata al metodo Session.getMap(String cacheName) con un cacheName che è una corrispondenza per l'espressione regolare, comporta la creazione della mappa dinamica. Da questa chiamata del metodo viene restituito un oggetto ObjectMap e viene creato l'oggetto BackingMap associato.

```
Session session = og.getSession();
    ObjectMap map = session.getMap("templateMap1");
```

La mappa appena creata viene configurata con tutti gli attributi e i plug-in che erano stati definiti nella definizione della mappa del template. Considerare di nuovo il precedente file XML ObjectGrid.

Una mappa dinamica creata in base alla mappa del template in questo file XML avrebbe un programma di eliminazione configurato e la sua strategia di blocco sarebbe pessimistica.

Nota: Un template non è una BackingMap effettiva. Cioè, l'ObjectGrid "accounting" non contiene una mappa "templateMap.*" reale. Il template viene utilizzato solo come base per la creazione dinamica della mappa. Tuttavia, bisogna includere la mappa dinamica nell'elemento mapRef del file XML della politica di distribuzione denominato esattamente come nell'XML di ObjectGrid. Questo identifica in quale mapSet si troveranno le mappe dinamiche.

Quando si utilizzano le mappe del template, considerare i cambi di comportamento del metodo Session.getMap(String cacheName). Prima della WebSphere eXtreme Scale Versione 7.0, tutte le chiamate al metodo Session.getMap(String cacheName) davano come risultato un'eccezione UndefinedMapException se la mappa richiesta non esisteva. Con le mappe dinamiche, ogni nome corrispondente all'espressione regolare per una mappa del template comporta la creazione della mappa. Assicurarsi di annotare il numero delle mappe create nella propria applicazione, specialmente se la propria espressione regolare è generica.

Inoltre, ObjectGridPermission.DYNAMIC_MAP è richiesto per la creazione della mappa dinamica quando è abilitata la sicurezza eXtreme Scale. Questa autorizzazione viene bloccata quando viene chiamato il metodo Session.getMap(String). Per ulteriori informazioni, consultare le informazioni sull'autorizzazione del client dell'applicazione in *Panoramica sul prodotto*.

Ulteriori esempi

objectGrid.xml

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

   <objectGrid name="session.partition.info">
    <backingMap name="partition.info" readOnly="false" lockStrategy="PESSIMISTIC"
       ttlEvictorType="NONE" copyMode="NO_COPY" numberOfBuckets="107"/>
    <backingMap name="clone.info" readOnly="false" lockStrategy="PESSIMISTIC"
      ttlEvictorType="NONE" copyMode="NO_COPY" numberOfBuckets="107"
      lockTimeout="300"/>
   </objectGrid>

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

```
     lockStrategy="PESSIMISTIC" ttlEvictorType="NONE" copyMode="NO_COPY"/>
   </objectGrid>

  </objectGrids>
</objectGridConfig>
```

objectGridDeployment.xml

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

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

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

</deploymentPolicy>
```

Limitazioni e considerazioni:

Limitazioni:

- Non è possibile utilizzare mappe dinamiche con Query.
- QuerySchema non supporta il template per mapName.
- Non è possibile utilizzare entità con mappe dinamiche.
- Una BackingMap dell'entità è definita implicitamente, associata all'entità attraverso il nome della classe.

Considerazioni:

- Molti plug-in non hanno modo di determinare la mappa con cui è associato ciascun plug-in.
- Altri plug-in si differenziano attraverso l'uso di un nome di mappa o del nome di BackingMap come un argomento.

ObjectMap e JavaMap

Un'istanza JavaMap viene ottenuta da un oggetto ObjectMap. L'interfaccia JavaMap possiede le stesse firme del metodo di ObjectMap, ma con una diversa gestione delle eccezioni. JavaMap estende l'interfaccia java.util.Map, quindi tutte le eccezioni sono istanze della classe java.lang.RuntimeException. Poiché JavaMap estende l'interfaccia java.util.Map, è facile utilizzare rapidamente WebSphere eXtreme Scale con un'applicazione esistente che utilizza un'interfaccia java.util.Map per la memorizzazione nella cache dell'oggetto.

Come ottenere un'istanza JavaMap

Un'applicazione ottiene un'istanza JavaMap da un oggetto ObjectMap utilizzando il metodo ObjectMap.getJavaMap. Il seguente frammento di codice descrive il modo in cui ottenere un'istanza 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;
```

Il backup di JavaMap viene eseguito dall'ObjectMap da cui è stato ottenuto. Più tentativi di richiamo del metodo getJavaMap utilizzando un particolare oggetto ObjectMap restituisce sempre la stessa istanza JavaMap.

Metodi

L'interfaccia JavaMap supporta solo un sottoinsieme di metodi nell'interfaccia java.util.Map. L'interfaccia java.util.Map supporta i seguenti metodi:

Metodo containsKey(java.lang.Object)

Metodo get(java.lang.Object)

Metodo put(java.lang.Object, java.lang.Object)

Metodo putAll(java.util.Map)

Metodo remove(java.lang.Object)

clear()

Tutti gli altri metodi ereditati dall'interfaccia java.util.Map determinano un'eccezione java.lang.UnsupportedOperationException.

Mappe come code FIFO

Con WebSphere eXtreme Scale, è possibile fornire una funzionalità tipo coda FIFO (First-In First-Out) per tutte le mappe. WebSphere eXtreme Scale tiene traccia dell'ordine di inserimento di tutte le mappe. Un client può richiedere a una mappa la successiva voce non bloccata di una mappa nell'ordine di inserimento e bloccare la voce. Questo processo consente a più client di sfruttare efficacemente le voci della mappa.

Esempio FIFO

Il seguente frammento di codice mostra l'immissione di un loop da parte del client per elaborare le voci della mappa fino all'esaurimento della mappa stessa. Il loop avvia una transazione, quindi chiama il metodo ObjectMap.getNextKey(5000). Questo metodo restituisce la chiave della successiva voce non bloccata disponibile e la blocca. Se la transazione viene bloccata per più di 5000 millisecondi, il metodo restituisce un valore nullo.

```
Session session = ...;
ObjectMap map = session.getMap("xxx");
// this needs to be set somewhere to stop this loop
boolean timeToStop = false;

while(!timeToStop)
{
  session.begin();
```

```
  Object msgKey = map.getNextKey(5000);
  if(msgKey == null)
  {
    // current partition is exhausted, call it again in
    // a new transaction to move to next partition
    session.rollback();
    continue;
  }
  Message m = (Message)map.get(msgKey);
  // now consume the message
  ...
  // need to remove it
  map.remove(msgKey);
  session.commit();
}
```

Modalità locale e modalità client

Se l'applicazione utilizza un core locale, ovvero non è un client, allora il meccanismo funziona come descritto in precedenza.

Per la modalità client, se la JVM (Java Virtual Machine) è un client, quest'ultimo si connette inizialmente a un elemento primario di partizione casuale. Se non vi sono attività da eseguire in quella partizione, il client si sposta nella successiva partizione alla ricerca di attività. Il client potrebbe trovare una partizione con voci o andare in loop fino alla partizione casuale iniziale. Se il client esegue un loop fino alla partizione iniziale, esso restituisce un valore nullo all'applicazione. Se il client trova una partizione con una mappa che contiene voci, esso utilizza le voci da quella fonte fino a quando si esauriscono le voci per il periodo di timeout. Alla scadenza del timeout, viene restituito il valore nullo. Questa azione indica che alla restituzione del valore nullo e all'utilizzo della mappa suddivisa in partizioni, è necessario avviare una nuova transazione e ripristinare l'esecuzione del listener. Il precedente frammento di esempio del codice ha questo comportamento.

Esempio

Quando si è in esecuzione come client e viene restituita una chiave, quella transazione viene ora associata alla partizione con la voce per quella chiave. Se non si desidera aggiornare altre mappe durante quella transazione, il problema non esiste. Se si desidera eseguire l'aggiornamento, è possibile aggiornare unicamente le mappe dalla stessa partizione della mappa dalla quale è stata ricevuta la chiave. La voce che viene restituita dal metodo getNextKey deve fornire all'applicazione la modalità di rilevare dati importanti in quella partizione. Come esempio, se si dispone di due mappe; una per eventi ed un'altra per lavori impattati dagli eventi. Le due mappe vengono definite con le seguenti entità:

Job.java

```
package tutorial.fifo;

import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;

@Entity
public class Job
{
 @Id String jobId;

 int jobState;
}
```

JobEvent.java

```
package tutorial.fifo;

import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;
import com.ibm.websphere.projector.annotations.OneToOne;

@Entity
public class JobEvent
{
 @Id String eventId;
 @OneToOne Job job;
}
```

Il lavoro dispone di un ID e stato, il quale è un numero intero. Si supponga di voler incrementare lo stato ogni volta che sia arrivato un evento. Gli eventi vengono memorizzati nella mappa JobEvent. Ogni voce ha un riferimento al lavoro indicato dall'evento. A tal fine, il codice richiesto per il listener è simile all'esempio seguente:

JobEventListener.java

```
package tutorial.fifo;

import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.em.EntityManager;

public class JobEventListener
{
 boolean stopListening;

 public synchronized void stopListening()
 {
  stopListening = true;
 }

 synchronized boolean isStopped()
 {
  return stopListening;
 }

 public void processJobEvents(Session session)
  throws ObjectGridException
 {
  EntityManager em = session.getEntityManager();
  ObjectMap jobEvents = session.getMap("JobEvent");
  while(!isStopped())
  {
   em.getTransaction().begin();

   Object jobEventKey = jobEvents.getNextKey(5000);
   if(jobEventKey == null)
   {
    em.getTransaction().rollback();
    continue;
   }
   JobEvent event = (JobEvent)em.find(JobEvent.class, jobEventKey);
   // process the event, here we just increment the
   // job state
   event.job.jobState++;
   em.getTransaction().commit();
  }
 }
}
```

Il listener viene avviato in un thread dall'applicazione. Il listener è in esecuzione fino a quando viene chiamato il metodo stopListening. Il metodo processJobEvents viene eseguito nel thread fino a quando viene chiamato il metodo stopListening. Il loop blocca l'attesa di un elemento eventKey della mappa JobEvent e quindi utilizza l'EntityManager per accedere all'oggetto dell'evento, rimuove il riferimento al lavoro e incrementa lo stato.

L'API EntityManager non dispone di un metodo getNextKey, contrariamente a ObjectMap. Pertanto, il codice utilizza l'ObjectMap affinché JobEvent ottenga la chiave. Se una mappa viene utilizzata con le entità, essa non memorizza più gli oggetti. Invece, la mappa memorizza oggetti Tuple; un oggetto Tuple per la chiave e un oggetto Tuple per il valore. Il metodo EntityManager.find accetta un oggetto Tuple per la chiave.

Il codice richiesto per creare un evento è simile al codice riportato nel seguente esempio:

```
em.getTransaction().begin();
Job job = em.find(Job.class, "Job Key");
JobEvent event = new JobEvent();
event.id = Random.toString();
event.job = job;
em.persist(event); // insert it
em.getTransaction().commit();
```

È necessario trovare il lavoro per l'evento, costruire un evento, indicarlo in modo che punti al lavoro, inserirlo nella Map JobEvent ed eseguire il commit della transazione.

Programmi di caricamento e mappe FIFO

È necessario effettuare delle attività aggiuntive se si desidera eseguire il backup di una mappa utilizzata come coda FIFO con un programma di caricamento (Loader). Se l'ordine delle voci nella mappa non è un problema, non sono richieste attività aggiuntive. Se l'ordine è importante, è necessario aggiungere un numero di sequenza a tutti i record inseriti quando si esegue la persistenza dei record nel backend. Per inserire i record all'avvio utilizzando questo ordine, è necessario scrivere un meccanismo di precaricamento.

Oggetti memorizzati nella cache e loro relazioni (API EntityManager)

La maggior parte dei prodotti cache utilizzano API basate su mappe per memorizzare dati come coppie chiave-valore. L'API ObjectMap e la cache dinamica in WebSphere Application Server, tra altri, utilizzano questo approccio. Tuttavia, le API basate su mappe presentano dei limiti. L'API EntityManager semplifica l'interazione con la cache di eXtreme Scale fornendo un modo semplice per dichiarare ed interagire con un grafico complesso di oggetti correlati.

Limitazioni di API basate su mappe

Se si sta utilizzando un'API basata su mappa, come la cache dinamica in WebSphere Application Server o l'API ObjectMap, bisogna tener presenti le seguenti limitazioni.

- La cache deve utilizzare una reflection per estrarre dati dagli oggetti nella cache, il che ha implicazioni sulle prestazioni.
- Due applicazioni non possono condividere una cache se utilizzano oggetti differenti per gli stessi dati.

- Non è possibile utilizzare un'evoluzione dei dati poiché non è possibile aggiungere facilmente un attributo ad un oggetto Java inserito nella cache.
- È difficile lavorare con grafici ed oggetti. L'applicazione deve memorizzare riferimenti artificiali tra gli oggetti ed unirli insieme manualmente.

Uso di EntityManager

L'API EntityManager utilizza l'infrastruttura esistente basata sulla mappa ma converte gli oggetti entità in tuple e viceversa, prima di memorizzarle o di leggerle dalla mappa. Un oggetto entità viene trasformato in una tupla chiave e in una tupla valore che vengono memorizzate come coppie chiave-valore. Una tupla è un array di attributi primitivi.

Questa serie di API semplifica significativamente l'uso di eXtreme Scale seguendo lo stile di programmazione POJO (Plain Old Java Object) adotatto dalla maggior parte dei framework.

Definizione di uno schema di entità

Un ObjectGrid può disporre di un qualsiasi numero di schemi di entità logiche. Le entità vengono definite utilizzando classi Java annotate, XML o una combinazione di XML e classi Java. Le entità definite vengono quindi registrate con un server eXtreme Scale e associate a più BackingMap, indici e altri plug-in.

Durante la designazione di uno schema di entità, è necessario completare le seguenti attività:

1. Definire le entità e le relative relazioni.
2. Configurare eXtreme Scale.
3. Registrare le entità.
4. Creare applicazioni basate su entità che interagiscono con le API EntityManager di eXtreme Scale.

Configurazione dello schema di entità

Uno schema di entità consiste in una serie di entità e relazioni tra entità. In un'applicazione eXtreme Scale con più partizioni, vengono applicate le seguenti limitazioni e opzioni agli schemi di entità:

- Ogni schema di entità deve avere un singolo root definito. Ciò è noto come root di schema.
- Tutte le entità per un determinato schema devono trovarsi nella stessa serie di mappe, il che indica che tutte le entità raggiungibili da un root di schema con relazioni chiave o non chiave devono essere definite nella stessa serie di mappe del root di schema.
- Ogni entità può appartenere a un solo schema di entità.
- Ogni applicazione eXtreme Scale può avere più schemi.

Le entità vengono registrate con un'istanza ObjectGrid prima che sia inizializzata. Ogni entità definita deve essere denominata in modo univoco e associata automaticamente a un ObjectGrid BackingMap dello stesso nome. Il metodo di inizializzazione varia a seconda della configurazione utilizzata:

Configurazione eXtreme Scale locale

Se si utilizza un ObjectGrid locale, è possibile configurare in modo programmatico lo schema di entità. In questo modo, è possibile utilizzare i metodi ObjectGrid.registerEntities per registrare le classi entità annotate o un file descrittore di metadati entità.

Configurazione eXtreme Scale distribuita

Se si utilizza una configurazione eXtreme Scale distribuita, è necessario fornire un file descrittore di metadati entità con lo schema di entità.

Per ulteriori dettagli, consultare “EntityManager in un ambiente distribuito” a pagina 72.

Requisiti di entità

I metadati di entità vengono configurati utilizzando i file di classe Java, un file XML descrittore di entità o entrambi. Di base, si richiede il file XML descrittore di entità per identificare quali BackingMap di eXtreme Scale devono essere associate alle entità. Gli attributi persistenti dell'entità e le relative relazioni ad altre entità sono descritti in una classe Java annotata (classe di metadati entità) o nel file XML descrittore di entità. La classe metadati di entità, quando specificata, viene utilizzata anche dall'API EntityManager per interagire con i dati nella griglia.

7.0.0.0 FIX 2+ È possibile definire una griglia eXtreme Scale senza fornire classi entità. Ciò può essere molto utile quando il server e il client interagiscono direttamente con i dati tupla memorizzati nelle mappe sottostanti. Tali entità vengono definite completamente nel file XML descrittore di entità e vengono indicate come entità senza classe.

Entità senza classe

Le entità senza classe sono utili quando non è possibile includere le classi dell'applicazione nel percorso classi del server o del client. Tali entità vengono definite nel file XML descrittore metadati di entità, in cui il nome classe dell'entità viene specificato utilizzando un identificativo entità senza classe nel formato: @<identificativo entità>. Il simbolo @ identifica l'entità come senza classe ed è utilizzato per la mappatura di associazioni tra le entità. Vedere la figura "Metadati di entità senza classe" come esempio di un file XML descrittore metadati di entità con due entità senza classe definite.

Se un client o server eXtreme Scale non dispone dell'accesso alle classi, può ancora utilizzare l'API di EntityManager utilizzando le entità senza classe. Casi di utilizzo comuni includono quanto segue:

- Il contenitore eXtreme Scale viene ospitato su un server che non consente l'inserimento delle classi dell'applicazione nel percorso classi. In questo caso, i client possono anche accedere alla griglia utilizzando l'API di EntityManager di un client, dove le classi sono consentite.
- Il client eXtreme Scale non richiede l'accesso alle classi entità perché sta utilizzando un client non Java, come il servizio dati REST di eXtreme Scale, o sta accedendo ai dati tupla nella griglia utilizzando l'API di ObjectMap.

Se vi è compatibilità dei metadati di entità tra client e server, è possibile creare tali metadati utilizzando le classi metadati di entità, un file XML o entrambi.

Ad esempio, la "Classe entità programmatica" nella seguente figura è compatibile con il codice metadati senza classe della sezione successiva.

Classe entità programmatica

```
@Entity
public class Employee {
    @Id long serialNumber;
    @Basic byte[] picture;
    @Version int ver;
    @ManyToOne(fetch=FetchType.EAGER, cascade=CascadeType.PERSIST)
    Department department;
}

@Entity
public static class Department {
    @Id int number;
    @Basic String name;
    @OneToMany(fetch=FetchType.LAZY, cascade=CascadeType.ALL, mappedBy="department")
    Collection<Employee> employees;
}
```

Versioni, chiavi e campi senza classe

Come indicato in precedenza, le entità senza classe vengono configurate completamente nel file descrittore XML di entità. Le entità basate su classi definiscono i propri attributi utilizzando le annotazioni, le proprietà ed i campi Java. Pertanto le entità senza classe devono definire la struttura attributo e chiave nel descrittore XML di entità con i tag <basic> e <id>.

Metadati di entità senza classe

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

<entity class-name="@Employee" name="Employee">
    <attributes>
        <id name="serialNumber" type="long"/>
        <basic name="firstName" type="java.lang.String"/>
        <basic name="picture" type="[B"/>
        <version name="ver" type="int"/>
        <many-to-one
            name="department"
            target-entity="@Department"
            fetch="EAGER"">
                <cascade><cascade-persist/></cascade>
        </many-to-one>
    </attributes>
</entity>

<entity class-name="@Department" name="Department" >
    <attributes>
        <id name="number" type="int"/>
        <basic name="name" type="java.lang.String"/>
        <version name="ver" type="int"/>
        <one-to-many
            name="employees"
            target-entity="@Employee"
            fetch="LAZY"
            mapped-by="department">
            <cascade><cascade-all/></cascade>
        </one-to-many>
    </attributes>
</entity>
```

Si noti che ogni entità sopra riportata dispone di un elemento <id>. Un'entità senza classe deve avere uno o più elementi <id> definiti oppure un'associazione con valore singolo che rappresenta la chiave per l'entità. I campi dell'entità sono rappresentati da elementi ù <basic>. Gli elementi <id>, <version> e <basic> richiedono un nome e tipo nelle entità senza classi. Per dettagli sui tipi supportati, consultare la seguente sezione sui tipi di attributo supportati.

Requisiti della classe entità

Le entità basate su classi vengono identificate associando i diversi metadati con una classe Java. I metadati possono essere specificati utilizzando le annotazioni Java Platform, Standard Edition 5, un file descrittore metadati di entità o una combinazione di annotazioni e il file descrittore. Le classi entità devono soddisfare i seguenti criteri:

- L'annotazione `@Entity` viene specificata nel file descrittore XML di entità.
- La classe ha un costruttore pubblico o protetto senza argomento.
- Deve essere una classe di livello principale. Le interfacce ed i tipi enumerati non sono classi entità valide.
- Non può utilizzare la parola chiave final.
- Non può utilizzare eredità.
- Deve avere un tipo e nome univoco per ogni istanza ObjectGrid.

Tutte le entità hanno un tipo e nome univoco. Per impostazione predefinita, il nome è quello semplice (breve) della classe se utilizza annotazioni, ma può essere sovrascritto utilizzando l'attributo name dell'annotazione `@Entity`.

Attributi persistenti

I client e il gestore entità accedono allo stato persistente di un'entità utilizzando campi (variabili di istanze) o accessor di proprietà stile Enterprise JavaBeans. Ogni entità deve definire un accesso al campo o basato sulla proprietà. Le entità annotate hanno l'accesso al campo se i campi della classe sono annotati, mentre hanno l'accesso alla proprietà se il metodo getter della proprietà è annotato. Non è consentita una combinazione di accessi al campo e alla proprietà. Se non è possibile determinare il tipo in modo automatico, l'attributo **accessType** nell'annotazione `@Entity` o nell'XML equivalente può essere utilizzato per identificare il tipo di accesso.

Campi persistenti
Le variabili di istanze dell'entità di accesso al campo vengono accedute direttamente dal gestore entità e dai client. I campi che vengono contrassegnati con il modificatore transitorio o annotazione transitoria vengono ignorati. I campi persistenti non devono avere modificatori final o static.

Proprietà persistenti
Le entità con accesso alla proprietà devono essere conformi alle convenzioni di firma JavaBeans per leggere e scrivere le proprietà. I metodi che non si attengono alle convenzioni JavaBeans o hanno l'annotazione Transient sul metodo getter vengono ignorati. Per una proprietà di tipo `T`, è necessario che vi sia un metodo getter getProperty che restituisca un valore di tipo T e un metodo setter vuoto setProperty(T). Per tipi booleani, è possibile esprimere il metodo getter come isProperty, con restituzione del valore true o false. Le proprietà persistenti non possono avere il modificatore statico.

Tipi di attributi supportati

Sono supportati i seguenti tipi di proprietà e campi persistenti:

- Tipi di primitive Java che includono wrapper
- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- byte[]
- java.lang.Byte[]
- char[]
- java.lang.Character[]
- enum

Sono supportati tipi di attributi serializzabili dell'utente, ma presentano limitazioni su prestazioni, query e rilevamento di modifiche. Se modificati, i dati persistenti che non possono essere sottoposti al proxy, quali gli array e gli oggetti serializzabili dell'utente, devono essere riassegnati all'entità.

Gli attributi serializzabili sono rappresentati nel file XML descrittore di entità utilizzando il nome classe dell'oggetto. Se l'oggetto è un array, il tipo di dati viene rappresentato utilizzando il formato interno Java. Ad esempio, se un tipo di dati dell'attributo è java.lang.Byte[][], la rappresentazione della stringa è [[Ljava.lang.Byte;

I tipi serializzabili dell'utente devono essere conformi alle seguenti migliori pratiche:

- Implementare i metodi di serializzazione ad alte prestazioni. Implementare l'interfaccia java.lang.Cloneable e il metodo clone pubblico.
- Implementare l'interfaccia java.io.Externalizable.
- Implementare equals e hashCode

Associazioni di entità

È possibile definire le associazioni di entità bidirezionali e unidirezionali o le relazioni tra entità come uno-a-uno, molti-a-uno, uno-a-molti e molti-a-molti. Il gestore entità risolve automaticamente le relazioni entità con riferimenti chiave appropriati quando le entità vengono memorizzate.

La griglia eXtreme Scale è una cache di dati e non applica l'integrità referenziale come un database. Sebbene le relazioni consentono operazioni di rimozione e persistenza in cascata per entità child, non vengono rilevati o rafforzati i collegamenti interrotti con gli oggetti. Quando si rimuove un oggetto child, il riferimento a quell'oggetto deve essere rimosso dall'elemento parent.

Se si definisce un'associazione bidirezionale tra le due entità, è necessario identificare il proprietario della relazione. In un'associazione a-molti, il lato molti della relazione è sempre il lato di appartenenza. Se non è possibile determinare in modo automatico l'appartenenza, è necessario specificare l'attributo **mappedBy**

dell'annotazione o l'XML equivalente. L'attributo **mappedBy** identifica il campo nell'entità di destinazione proprietaria della relazione. Questo attributo consente di identificare anche i campi associati quando esistono più attributi dello stesso tipo e cardinalità.

Associazioni con valore singolo

Le associazioni uno-a-uno e molti-a-uno sono indicate utilizzando le annotazioni @OneToOne e @ManyToOne o gli attributi XML equivalenti. Il tipo di entità di destinazione è determinato dal tipo di attributo. Il seguente esempio definisce un'associazione unidirezionale tra Person e Address. L'entità Customer dispone di un riferimento a un'entità Address. In questo caso, l'associazione può anche essere molti-a-uno poiché non vi è alcuna relazione inversa.

```
@Entity
public class Customer {
  @Id id;
  @OneToOne Address homeAddress;
}

@Entity
public class Address{
  @Id id
  @Basic String city;
}
```

Per specificare una relazione bidirezionale tra le classi Customer e Address, aggiungere un riferimento alla classe Customer dalla classe Address e aggiungere l'annotazione appropriata per contrassegnare il lato inverso della relazione. Poiché questa associazione è di tipo uno-a-uno, è necessario specificare un proprietario della relazione utilizzando l'attributo mappedBy nell'annotazione @OneToOne.

```
@Entity
public class Address{
  @Id id
  @Basic String city;
  @OneToOne(mappedBy="homeAddress") Customer customer;
}
```

Associazioni con valori di raccolta

Le associazioni uno-a-molti e molti-a-molti sono indicate utilizzando le annotazioni `@OneToMany` e `@ManyToMany` o gli attributi XML equivalenti. Tutte le relazioni di tipo molti sono rappresentate utilizzando i seguenti tipi: java.util.Collection, java.util.List o java.util.Set. Il tipo di entità di destinazione viene determinato dal tipo generico di Collection, List o Set oppure utilizzando esplicitamente l'attributo **targetEntity** nell'annotazione `@OneToMany` o `@ManyToMany` (o XML equivalente).

Nell'esempio precedente, non è pratico avere un oggetto indirizzo per cliente poiché molti clienti potrebbero condividere un indirizzo o avere più indirizzi. Questa situazione viene risolta in modo migliore utilizzando un'associazione molti (many):

```
@Entity
public class Customer {
  @Id id;
  @ManyToOne Address homeAddress;
  @ManyToOne Address workAddress;
}

@Entity
public class Address{
```

```
  @Id id
  @Basic String city;
  @OneToMany(mappedBy="homeAddress") Collection<Customer> homeCustomers;


  @OneToMany(mappedBy="workAddress", targetEntity=Customer.class)
  Collection workCustomers;
}
```

In questo esempio, esistono due diverse relazioni tra le stesse entità: una relazione di indirizzi Home e Work. Una raccolta (Collection) non generica viene utilizzata per l'attributo **workCustomers** per descrivere in che modo utilizzare l'attributo **targetEntity** quando i valori generici non sono disponibili.

Associazioni senza classe

Le associazioni di entità senza classi vengono definite nel file XML descrittore metadati di entità in modo analogo a come vengono definite le associazioni basate sulla classe. L'unica differenza è che l'entità di destinazione anziché puntare a una classe effettiva, punta all'identificativo entità senza classe utilizzato per il nome classe dell'entità.

Di seguito è riportato un esempio:

```
<many-to-one name="department" target-entity="@Department" fetch="EAGER">
      <cascade><cascade-all/></cascade>
</many-to-one>
<one-to-many name="employees" target-entity="@Employee" fetch="LAZY">
      <cascade><cascade-all/></cascade>
</one-to-many>
```

Chiavi primarie

Tutte le entità devono avere una chiave primaria, che può essere una chiave semplice (singolo attributo) o una chiave composta (più attributi). Gli attributi chiave sono indicati utilizzando l'annotazione Id o definiti nel file descrittore XML di entità. Gli attributi chiave hanno i seguenti requisiti:

- Il valore di una chiave primaria non può cambiare.
- Un attributo di chiave primaria deve essere uno dei seguenti tipi: wrapper e tipo di primitiva Java, java.lang.String, java.util.Date o java.sql.Date.
- Una chiave primaria può contenere un qualsiasi numero di associazioni con valore singolo. L'entità di destinazione dell'associazione chiave primaria non deve avere un'associazione inversa diretta o indiretta all'entità di origine.

Le chiavi primarie composte possono definire facoltativamente una classe di chiave primaria. Un'entità è associata a una classe di chiave primaria utilizzando l'annotazione `@IdClass` o il file descrittore XML di entità. Un'annotazione `@IdClass` è utile se associata al metodo EntityManager.find.

Le classi di chiavi primarie hanno i seguenti requisiti:

- Deve essere pubblica con un costruttore senza argomento.
- Il tipo di accesso della classe di chiave primaria è determinato dall'entità che dichiara la classe di chiave primaria.
- Se l'accesso è alla proprietà, le proprietà della classe di chiave primaria devono essere pubbliche o protette.
- Le proprietà o i campi della chiave primaria devono corrispondere ai tipi e nomi dell'attributo chiave definiti nell'entità di riferimento.

- Le classi di chiavi primarie devono implementare i metodi equals e hashCode.

Di seguito è riportato un esempio:

```
@Entity
@IdClass(CustomerKey.class)
public class Customer {
    @Id @ManyToOne Zone zone;
    @Id int custId;
    String name;
    ...
}

@Entity
public class Zone{
    @Id String zoneCode;
    String name;
}

public class CustomerKey {
    Zone zone;
    int custId;

    public int hashCode() {...}
    public boolean equals(Object o) {...}
}
```

Chiavi primarie senza classe

Le entità senza classe sono richieste per avere almeno un elemento <id> o un'associazione nel file XML con l'attributo `id=true`. Di seguito è riportato un esempio di entrambi i casi:

```
<id name="serialNumber" type="int"/>
<many-to-one name="department" target-entity="@Department" id="true">
<cascade><cascade-all/></cascade>
</many-to-one>
```

Attenzione:
Il tag XML <id-class> non è supportato per le entità senza classe.

Proxy di entità e intercettazione di campo

Le classi entità ed i tipi di attributi supportati modificabili vengono estesi da classi proxy per entità di accesso alla proprietà, e con codice byte avanzato per entità di accesso al campo JDK (Java Development Kit) 5. Tutti gli accessi all'entità, anche tramite metodi di business interni e metodi equals, devono utilizzare il campo appropriato o i metodi di accesso alla proprietà.

I proxy e gli intercettatori di campo vengono utilizzati per consentire al gestore entità di tenere traccia dello stato dell'entità, determinare se l'entità è cambiata e migliorare le prestazioni. Gli intercettatori di campo sono disponibili solo su piattaforme Java SE 5 quando viene configurato l'agent della strumentazione entità.

Attenzione: Quando si utilizzano le entità di accesso alla proprietà, il metodo equals deve utilizzare l'operatore instanceof per confrontare l'istanza corrente con l'oggetto input. Tutta l'attività di introspezione dell'oggetto di destinazione deve essere eseguita attraverso le proprietà dell'oggetto e non dai campi stessi, poiché l'istanza oggetto sarà il proxy.

File emd.xsd

Utilizzare la definizione di schema XML dei metadati entità per creare un file XML descrittore e definire uno schema entità per WebSphere eXtreme Scale.

Consultare le informazioni sul file descrittore metadati di entità in *Guida alla gestione* per descrizioni su ogni elemento e attributo del file `emd.xsd`.

File emd.xsd

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

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

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

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

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

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

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

```
    <xsd:attribute name="alias" type="xsd:string" use="optional" />
</xsd:complexType>

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

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

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

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

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

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

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

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

```
    <xsd:complexType name="emptyType" />

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

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

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

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

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

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

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

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

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

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

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

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

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

</xsd:schema>
```

EntityManager in un ambiente distribuito

È possibile utilizzare EntityManager con un ObjectGrid locale oppure in un ambiente eXtreme Scale distribuito. La differenza principale è rappresentata dalla modalità di connessione a tale ambiente remoto. Una volta stabilita una connessione, non esiste alcuna differenza tra l'utilizzo di un oggetto Session o l'utilizzo dell'API EntityManager.

File di configurazione richiesti

Sono richiesti i file di configurazione XML riportati di seguito:

- File XML descrittore ObjectGrid
- File XML descrittore Entity
- File XML descrittore della griglia o di distribuzione

Tali file specificano le entità e le BackingMap che verranno ospitate da un server.

Il file descrittore dei metadati di entità contiene una descrizione delle entità utilizzate. È necessario specificare almeno il nome e la classe entità. Se si sta utilizzando un ambiente Java Platform, Standard Edition 5, eXtreme Scale legge automaticamente la classe entità e le relative annotazioni. È possibile definire ulteriori attributi XML se la classe entità non dispone di annotazioni oppure se è richiesto di sovrascrivere gli attributi della classe. Se si sta eseguendo la registrazione delle entità senza classe, fornire tutte le informazioni relative all'entità solo nel file XML.

È possibile utilizzare il frammento di configurazione XML riportato di seguito per definire una griglia data con entità. In tale frammento di codice, il server crea un ObjectGrid denominato `bookstore` ed una mappa di backup associata denominata `order`. Si noti che il frammento del file `objectgrid.xml` fa riferimento al file `entity.xml`. In questo caso, il file `entity.xml` contiene una entità, l'entità Order.

objectgrid.xml

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

 <objectGrids>
    <objectGrid name="bookstore" entityMetadataXMLFile="entity.xml">
     <backingMap name="Order"/>
    </objectGrid>
</objectGrids>

</objectGridConfig>
```

Tale file `objectgrid.xml` fa riferimento a `entity.xml` con l'attributo **entityMetadataXMLFile**. Il percorso di tale file è relativo al percorso del file `objectgrid.xml`. Di seguito è riportato un esempio di file `entity.xml`:

entity.xml

```
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">
 <entity class-name="com.ibm.websphere.tutorials.objectgrid.em.
    distributed.step1.Order" name="Order"/>
</entity-mappings>
```

In questo esempio, si suppone che i campi orderNumber e desc della classe Order siano annotati in modo simile.

Di seguito è riportato un file `entity.xml` senza classe equivalente

classless entity.xml

```
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">
 <entity class-name="@Order " name="Order">
     <description>"Entity named: Order"</description>
        <attributes>
            <id name="orderNumber" type="int"/>
```

```
            <basic name="desc" type="java.lang.String"/>
        </attributes>
 </entity>
</entity-mappings>
```

Per informazioni relative all'avvio di un server eXtreme Scale, consultare *Avvio dei processi server WebSphere eXtreme Scale* in the *Guida alla gestione*, in cui vengono utilizzati i file `deployment.xml` ed `objectgrid.xml` per avviare il server di catalogo.

Connessione ad un server eXtreme Scale distribuito

Il codice riportato di seguito abilita il meccanismo di connessione per un client ed un server sullo stesso computer.

```
String catalogEndpoints="localhost:2809";
URL clientOverrideURL= new URL("file:etc/emtutorial/distributed/step1/objectgrid.xml");
ClientClusterContext clusterCtx = ogMgr.connect(catalogEndpoints, null, clientOverrideURL);
ObjectGrid objectGrid=ogMgr.getObjectGrid(clusterCtx, "bookstore");
```

Nel frammento di codice precedente, notare il riferimento al server eXtreme Scale remoto. Una volta stabilita una connessione, è possibile richiamare i metodi dell'API EntityManager come persist, update, remove e find.

Attenzione: quando vengono utilizzate le entità, passare il file XML descrittore ObjectGrid di sostituzione client al metodo connect. Se viene passato un valore null alla proprietà clientOverrideURL ed il client ha una struttura directory diversa da quella del server, il client potrebbe non essere in grado di individuare i file XML descrittore entità oppure ObjectGrid. È possibile almeno copiare sul client i file XML di entità ed ObjectGrid per il server.

In precedenza, per l'utilizzo delle entità su un client ObjectGrid era richiesto che i file XML di entità ed XML ObjectGrid fossero resi disponibili per il client in uno dei due modi riportati di seguito:

1. Passando un file XML ObjectGrid di sostituzione al metodo ObjectGridManager.connect(String catalogServerAddresses, ClientSecurityConfiguration securityProps, URL overRideObjectGridXml).

   ```
   String catalogEndpoints="myHost:2809";
   URL clientOverrideURL= new URL("file:etc/emtutorial/distributed/step1/objectgrid.xml");
   ClientClusterContext clusterCtx = ogMgr.connect(catalogEndpoints, null, clientOverrideURL);
   ObjectGrid objectGrid=ogMgr.getObjectGrid(clusterCtx, "bookstore");
   ```

2. Passando un valore null per il file di sostituzione e verificando che il file XML ObjectGrid ed il file XML di entità di riferimento fossero disponibili sul client nello stesso percorso utilizzato dal server.

   ```
   String catalogEndpoints="myHost:2809";
   ClientClusterContext clusterCtx = ogMgr.connect(catalogEndpoints, null, null);
   ObjectGrid objectGrid=ogMgr.getObjectGrid(clusterCtx, "bookstore");
   ```

7.0.0.0 FIX 2+ I file XML erano richiesti indipendentemente dal fatto che si desiderasse utilizzare le entità dell'insieme secondario sul lato client. Non è più richiesto che i file utilizzino le entità come definito dal server. Invece, passare il valore null per il parametro overRideObjectGridXml come nell'opzione 2 della sezione precedente. Se il file XML non viene trovato nello stesso percorso impostato sul server, il client utilizza la configurazione dell'entità sul server.

Tuttavia, se vengono utilizzate le entità dell'insieme secondario sul client, è necessario fornire un file XML ObjectGrid di sostituzione, come nell'opzione 1.

Schema lato client e server

Lo schema lato server definisce il tipo di dati memorizzati nelle mappe su un server. Lo schema lato client è un'associazione agli oggetti dell'applicazione dallo schema sul server. Ad esempio, potrebbe essere disponibile lo schema lato server riportato di seguito:

```
@Entity
class ServerPerson
{
  @Id String ssn;
  String firstName;
  String surname;
  int age;
  int salary;
}
```

Un client potrebbe avere un oggetto annotato come nell'esempio riportato di seguito:

```
@Entity(name="ServerPerson")
class ClientPerson
{
  @Id @Basic(alias="ssn") String socialSecurityNumber;
  String surname;
}
```

Tale client utilizza un'entità lato server e proietta il sottoinsieme dell'entità nell'oggetto client. Tale proiezione determina un risparmio di memoria e larghezza di banda sul client, in quando il client ha solo le informazioni necessarie invece di tutte le informazioni presenti nell'entità lato server. Applicazioni differenti possono utilizzare i propri oggetti invece di obbligare tutte le applicazioni a condividere una serie di classi per l'accesso ai dati.

Il file XML descrittore di entità lato client è richiesto nei seguenti casi: se il server è in esecuzione con entità basate sulle classi mentre il lato client è in esecuzione senza classe; se il server è senza classi ed il client utilizza entità basate sulle classi. La modalità client senza classi consente al client di eseguire query di entità senza disporre di accesso alle classi fisiche. Supponendo che il server abbia registrato l'entità ServerPerson sopra riportata, il client esegue la sostituzione della griglia con un `entity.xml` come riportato di seguito:

```
<entity-mappings xmlns="http://ibm.com/ws/projector/config/emd"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/projector/config/emd ./emd.xsd">
 <entity class-name="@ServerPerson " name="Order">
     <description>"Entity named: Order"</description>
        <attributes>
            <id name="socialSecurityNumber" type="java.lang.String"/>
            <basic name="surname" type="java.lang.String"/>
        </attributes>
 </entity>
</entity-mappings>
```

Questo file ottiene un'entità del sottoinsieme equivalente sul client, senza richiedere che il client fornisca la reale classe annotata. Se il server è senza classe, a differenza del client, il client fornisce un file XML descrittore di entità di sostituzione. Tale file XML descrittore di entità contiene una sostituzione per il riferimento al file di classe.

Riferimento allo schema

Se la propria applicazione è in esecuzione in Java SE 5, è possibile aggiungere l'applicazione agli oggetti utilizzando le annotazioni. EntityManager è in grado di leggere lo schema dalle annotazioni su tali oggetti. L'applicazione fornisce il runtime eXtreme Scale con riferimenti a tali oggetti utilizzando il file `entity.xml`, indicato dal file `objectgrid.xml`. Il file `entity.xml` elenca tutte le entità, ciascuna delle quali è associata ad una classe o ad uno schema. Se viene specificato un

nome classe appropriato, l'applicazione prova a leggere le annotazioni Java SE 5 da tali classi per determinare lo schema. Se il file di classe non viene annotato o viene specificato un identificativo senza classe come nome della classe, lo schema viene rilevato dal file XML. Il file XML è utilizzato per specificare tutti gli attributi, le chiavi e le relazioni per ciascuna entità.

Una griglia locale non richiede file XML. Il programma può ottenere un riferimento ObjectGrid e richiamare il metodo ObjectGrid.registerEntities per specificare l'elenco di classi annotate Java SE 5 oppure un file XML.

Il runtime utilizza il file XML oppure un elenco di classi annotate per individuare nomi di entità, nomi e tipi di attributi, campi e tipi delle chiavi e relazioni tra le entità. Se eXtreme Scale è in esecuzione su un server o in modalità standalone, crea automaticamente una mappa indicata dopo ciascuna entità. Tali mappe possono essere ulteriormente personalizzate utilizzando il file `objectgrid.xml` o le API impostate dall'applicazione o dai framework di inserimento, come Spring.

File descrittore dei metadati di entità

Consultare “File `emd.xsd`” a pagina 70 per ulteriori informazioni relative al file descrittore dei metadati.

Interazione con EntityManager

Le applicazioni di solito ottengono prima un riferimento ObjectGrid e poi una sessione da quel riferimento per ogni thread. Le sessioni non possono essere condivise tra thread. È disponibile un metodo extra sulla sessione, il metodo getEntityManager. Questo metodo restituisce un riferimento ad un gestore entità da utilizzare per questo thread. L'interfaccia di EntityManager può sostituire le interfacce Session e ObjectMap per tutte le applicazioni. È possibile utilizzare queste API EntityManager se il client ha accesso alle classi di entità definite.

Come ottenere un'istanza EntityManager da una sessione

Il metodo getEntityManager è disponibile su un oggetto Session. Il seguente esempio di codice illustra il modo in cui creare un'istanza ObjectGrid in locale e come accedere a EntityManager. Consultare l'interfaccia EntityManager nella documentazione API per i dettagli su tutti i metodi supportati.

```
ObjectGrid og =
ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("intro-grid");
Session s = og.getSession();
EntityManager em = s.getEntityManager();
```

Esiste una relazione uno-a-uno tra l'oggetto Session e l'oggetto EntityManager. È possibile utilizzare l'oggetto EntityManager più di una volta.

Persistenza di un'entità

La persistenza di un'entità significa salvare lo stato di una nuova entità in una cache di ObjectGrid. Dopo aver chiamato il metodo persist, l'entità si trova nello stato gestito. Persist è un'operazione transazionale e la nuova entità viene memorizzata nella cache di ObjectGrid dopo i commit della transazione.

Ogni entità ha una BackingMap corrispondente in cui vengono memorizzate le tuple. BackingMap ha lo stesso nome dell'entità e viene creata quando viene registrata la classe. Il seguente esempio di codice dimostra come creare un oggetto Order utilizzando l'operazione persist.

```
Order order = new Order(123);
em.persist(order);
order.setX();
...
```

L'oggetto Order viene creato con la chiave 123, e l'oggetto viene passato al metodo persist. È possibile continuare a modificare lo stato dell'oggetto prima del commit sulla transazione.

Importante: Il precedente esempio non include limite transazionale obbligatorio, come begin e commit. Per ulteriori informazioni, consultare il supporto didattico di il supporto didattico di entity manager in *Panoramica sul prodotto*.

Ricerca di un'entità

È possibile individuare l'entità nella cache ObjectGrid con il metodo find fornendo una chiave dopo l'entità memorizzata nella cache. Questo metodo non richiede alcun boundary transazionale che è utile per le semantiche di sola lettura. L'esempio di seguito riportato illustra che per individuare l'entità è necessaria una sola riga di codice.

```
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
```

Rimozione di un'entità

Il metodo remove, così come il metodo persist, è un'operazione transazionale. L'esempio di seguito riportato illustra il boundary transazionale mediante chiamata dei motedi begin e commit.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
em.remove(foundOrder );
em.getTransaction().commit();
```

Prima che l'entità possa essere rimossa, deve essere prima gestita e ciò è realizzabile chiamando il metodo find all'interno del boundary transazionale. Quindi, chiamare il metodo remove nell'interfaccia EntityManager.

Invalidazione di un'entità

Il metodo invalidate si comporta molto come il metodo remove ma non richiama alcun programma di caricamento di plug-in. Utilizzare questo metodo per rimuovere le entità da ObjectGrid ma per preservarle nell'archivio dati di backend.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
em.invalidate(foundOrder );
em.getTransaction().commit();
```

Prima che l'entità possa essere invalidata, deve essere prima gestita e ciò è realizzabile chiamando il metodo find all'interno del boundary transazionale. Dopo aver chiamato il metodo find, è possibile chiamare il metodo invalidate nell'interfaccia EntityManager.

Aggiornamento di un'entità

Il metodo update è anche un'operazione transazionale. L'entità deve essere gestita prima che possano essere applicati gli aggiornamenti.

```
em.getTransaction().begin();
Order foundOrder = (Order)em.find(Order.class, new Integer(123));
foundOrder.date = new Date(); // update the date of the order
em.getTransaction().commit();
```

Nel precedente esempio, il metodo persist non viene chiamato dopo l'aggiornamento dell'entità. L'entità viene aggiornata nella cache ObjectGrid quando viene eseguito il commit della transazione.

Query e code di query

Con il motore di query flessibile è possibile richiamare le entità utilizzando l'API EntityManager. Creare una query di tipo SELECT su un'entità o su uno schema Object-based utilizzando il linguaggio di query ObjectGrid. L'interfaccia della query spiega in dettaglio come eseguire le query utilizzando l'API EntityManager. Per ulteriori informazioni sull'utilizzo delle query, consultare l'API Query.

Un'entità QueryQueue è una struttura di dati simile alla coda, associata ad una query di entity. Seleziona tutte le entità che corrispondono alla condizione WHERE nel filtro di query e mette le entità del risultato in una coda. I client possono iterativamente richiamare le entità da questa coda. Per ulteriori informazioni, consultare “Code della query di entità” a pagina 89.

Listener di entità e metodi di callback

Le applicazioni possono essere notificate quando lo stato di un'entità transita da uno stato all'altro. Esistono due meccanismi di callback per gli eventi di modifica dello stato: metodi di callback con ciclo di vita definiti su una classe entità e richiamati ogni volta che lo stato dell'entità cambia, e i listener entità, che sono meccanismi più generali in quanto possono essere registrati su svariate entità.

Ciclo di vita di un'istanza dell'entità

Un'istanza dell'entità ha i seguenti stati:

- **Nuovo**: un'istanza dell'entità creata di recente che non è presente nella cache eXtreme Scale.
- **Gestito**: l'istanza dell'entità è presente nella cache eXtreme Scale ed è richiamata o è persistente utilizzando il gestore entità. Per trovarsi nello stato gestito, un'entità deve essere associata a una transazione attiva.
- **Scollegato**: l'istanza dell'entità è presente nella cache eXtreme Scale, ma non è più associata a una transazione attiva.
- **Rimosso**: l'istanza dell'entità viene rimossa o è programmata per essere rimossa dalla cache eXtreme Scale quando viene eseguito il commit o il flush della transazione.
- **Invalidato**: l'istanza dell'entità viene invalidata o è programmata per essere invalidata dalla cache eXtreme Scale quando viene eseguito il commit o il flush della transazione.

Quando le entità cambiano da stato a stato, è possibile richiamare i metodi di call-back con ciclo di vita.

Le seguenti sezioni descrivono ulteriormente il significato degli stati Nuovo, Gestito, Scollegato, Rimosso e Invalidato quando gli stati vengono applicati a un'entità.

Metodi di callback con ciclo di vità dell'entità

I metodi di callback con ciclo di vita dell'entità possono essere definiti sulla classe entità e sono richiamati quando lo stato dell'entità cambia. Tali metodi sono utili per convalidare i campi entità e per aggiornare lo stato transitorio che non è generalmente persistente con l'entità. I metodi di callback con ciclo di vita dell'entità possono anche essere definiti su classi che non utilizzano entità. Tali classi sono classi di listener entità, che possono essere associate a più tipi di entità. I metodi callback con ciclo di vita possono essere definiti utilizzando sia le annotazioni metadati che un file descrittore XML metadati di entità:

- **Annotazioni**: i metodi callback con ciclo di vita possono essere indicati utilizzando le annotazioni PrePersist, PostPersist, PreRemove, PostRemove, PreUpdate, PostUpdate e PostLoad in una classe di entità.
- **Descrittore XML di entità** : i metodi di callback con ciclo di vità possono essere descritti utilizzando XML quando le annotazioni non sono disponibili.

Listener entità

Una classe di listener entità è una classe che non utilizza le entità che definiscono uno o più metodi di callback con ciclo di vita dell'entità. I listener entità sono utili per applicazioni di registrazione o di controllo con finalità generali. I listener entità possono essere definiti utilizzando sia le annotazioni metadati che un file descrittore XML metadati di entità:

- **Annotazione**: l'annotazione EntityListeners può essere utilizzata per indicare una o più classi di listener entità su una classe entità. Se vengono definiti più listener entità, l'ordine in cui vengono richiamati è determinato dall'ordine in cui sono specificati nell'annotazione EntityListeners.
- **Descrittore XML di entità**: il descrittore XML può essere utilizzato come soluzione alternativa per specificare l'ordine di richiamo dei listener entità o per sostituire l'ordine specificato nelle annotazioni metadati.

Requisiti del metodo di callback

È possibile specificare un qualsiasi sottoinsieme o combinazione di annotazioni su una classe di entità o classe listener. Una classe singola non può avere più di un metodo di callback con ciclo di vita per lo stesso evento con ciclo di vita. Tuttavia, lo stesso metodo può essere utilizzato per più eventi di callback. La classe listener di entità deve avere un costruttore no-arg pubblico. I listener entità sono privi di stato. Il ciclo di vita di un listener entità non è specificato. eXtreme Scale non supporta la funzione di eredità dell'entità, pertanto i metodi di callback possono solo essere definiti nella classe entità, ma non nella superclasse.

Firma del metodo di callback

I metodi di callback con ciclo di vita dell'entità possono essere definiti su una classe listener di entità, direttamente su una classe entità o su entrambe. I metodi di callback con ciclo di vita dell'entità possono essere definiti utilizzando sia le annotazioni metadati che il descrittore XML di entità. Le annotazioni utilizzate per i metodi di callback sulla classe entità e sulla classe di listener entità sono uguali. Le firme dei metodi di callback sono diverse quando definite su una classe entità rispetto a una classe listener di entità. I metodi callback definiti su una classe entità o superclasse associata hanno la seguente firma:

```
void <METHOD>()
```

I metodi di callback definiti su una classe listener di entità hanno la seguente firma:

```
void <METHOD>(Object)
```

L'argomento Object è l'istanza dell'entità per cui viene richiamato il metodo callback. L'argomento Object può essere dichiarato come oggetto java.lang.Object object o tipo di entità reale.

I metodi callback possono avere accessi di livello pubblico, privato, protetto o package, ma non possono essere statici o finali.

Le seguenti annotazioni vengono definite per progettare i metodi callback di eventi con ciclo di vita dei tipi corrispondenti:

- com.ibm.websphere.projector.annotations.PrePersist
- com.ibm.websphere.projector.annotations.PostPersist
- com.ibm.websphere.projector.annotations.PreRemove
- com.ibm.websphere.projector.annotations.PostRemove
- com.ibm.websphere.projector.annotations.PreUpdate
- com.ibm.websphere.projector.annotations.PostUpdate
- com.ibm.websphere.projector.annotations.PostLoad

Per ulteriori dettagli, consultare la documentazione API. Ogni annotazione ha un attributo XML equivalente definito nel file descrittore XML di metadati entità.

Semantiche del metodo callback con ciclo di vita

Ogni dei diversi metodi callback con ciclo di vita ha una finalità differente ed è chiamato in diverse fasi del ciclo di vita dell'entità:

PrePersist
: Richiamato per un'entità prima che sia stata resa persistente nell'archivio, che include entità rese persistenti a causa di un'operazione in cascata. Questo metodo è richiamato nel thread dell'operazione EntityManager.persist.

PostPersist
: Richiamato per un'entità dopo che è stata resa persistente nell'archivio, che include entità rese persistenti a causa di un'operazione in cascata. Questo metodo è richiamato nel thread dell'operazione EntityManager.persist. Viene richiamato dopo il richiamo di EntityManager.flush o EntityManager.commit.

PreRemove
: Richiamato per un'entità prima che sia stata rimossa, che include entità che sono state rimosse a causa di un'operazione in cascata. Questo metodo è richiamato nel thread dell'operazione EntityManager.remove.

PostRemove
: Richiamato per un'entità dopo che è stata rimossa, che include entità che sono state rimosse a causa di un'operazione in cascata. Questo metodo è richiamato nel thread dell'operazione EntityManager.remove. Viene richiamato dopo il richiamo di EntityManager.flush o EntityManager.commit.

PreUpdate
Richiamato per un'entità prima che sia stata aggiornata nell'archivio. Questo metodo viene richiamato nel thread dell'operazione di commit o flush della transazione.

PostUpdate
Richiamato per un'entità dopo che è stata aggiornata nell'archivio. Questo metodo viene richiamato nel thread dell'operazione di commit o flush della transazione.

PostLoad
Richiamato per un'entità dopo che è stata caricata dall'archivio che include tutte le entità caricate attraverso un'associazione. Questo metodo viene richiamato nel thread dell'operazione di caricamento, come EntityManager.find o una query.

Metodi callback con ciclo di vita duplicati

Se vengono definiti più metodi callback per un evento con ciclo di vita dell'entità, l'ordine di richiamo di questi metodi è riportato di seguito:

1. **Metodi callback con ciclo di vita definiti nei listener entità**: i metodi callback con ciclo di vita definiti nelle classi listener di entità per una classe entità vengono richiamati con lo stesso ordine di specifica delle classi listener di entità nell'annotazione EntityListeners o descrittore XML.
2. **Superclasse listener**: i metodi callback definiti nella superclasse del listener entità vengono richiamati prima degli elementi child.
3. **Metodi con ciclo di vita dell'entità**: WebSphere eXtreme Scale non supporta la funzione di eredità dell'entità, pertanto i metodi con ciclo di vita dell'entità possono essere solo definiti nella classe entità.

Eccezioni

I metodi callback con ciclo di vita potrebbero determinare eccezioni runtime. Se a un metodo callback con ciclo di vita ne consegue un'eccezione runtime in una transazione, viene eseguito il rollback della transazione. Non vengono richiamati ulteriori metodi callback con ciclo di vita dopo che si è verificata un'eccezione runtime.

Esempi di listener entità

È possibile scrivere gli EntityListener in base alle proprie esigenze. Di seguito sono riportati diversi script di esempio.

Esempio di EntityListeners con utilizzo di annotazioni

Il seguente esempio mostra i richiami dei metodi di callback con ciclo di vita e l'ordine dei richiami. Supporre che esistano una classe entità Employee e due listener entità: EmployeeListener e EmployeeListener2.

```
@Entity
@EntityListeners(EmployeeListener.class, EmployeeListener2.class)
public class Employee {
    @PrePersist
    public void checkEmployeeID() {
        ....
    }
}

public class EmployeeListener {
    @PrePersist
```

```
    public void onEmployeePrePersist(Employee e) {
        ....
    }
}

public class PersonListener {
    @PrePersist
    public void onPersonPrePersist(Object person) {
        ....
    }
}

public class EmployeeListener2 {
    @PrePersist
    public void onEmployeePrePersist2(Object employee) {
        ....
    }
}
```

Se un evento PrePersist si verifica su un'istanza Employee, nell'ordine vengono richiamati i seguenti metodi:

1. Metodo onEmployeePrePersist
2. Metodo onPersonPrePersist
3. Metodo onEmployeePrePersist2
4. Metodo checkEmployeeID

Esempio di listener entità che utilizzano XML

Il seguente esempio descrive in che modo impostare un listener entità su un'entità utilizzando il file XML descrittore di entità:

```
<entity
    class-name="com.ibm.websphere.objectgrid.sample.Employee"
    name="Employee" access="FIELD">
    <attributes>
        <id name="id" />
        <basic name="value" />
    </attributes>
    <entity-listeners>
        <entity-listener
            class-name="com.ibm.websphere.objectgrid.sample.EmployeeListener">
            <pre-persist method-name="onListenerPrePersist" />
            <post-persist method-name="onListenerPostPersist" />
        </entity-listener>
    </entity-listeners>
    <pre-persist method-name="checkEmployeeID" />
</entity>
```

L'entità Employee viene configurata con una classe listener di entità com.ibm.websphere.objectgrid.sample.EmployeeListener, che dispone di due metodi callback con ciclo di vita definiti. Il metodo onListenerPrePersist è concepito per l'evento PrePersist e il metodo onListenerPostPersist è concepito per l'evento PostPersist. Inoltre, il metodo checkEmployeeID nella classe Employee viene configurato per ascoltare l'evento PrePersist.

Supporto del piano fetch EntityManager

Un FetchPlan è la strategia che viene utilizzata dal gestore entità per richiamare gli oggetti associati se l'applicazione richiede l'accesso alle relazioni.

Esempio

Presupporre, ad esempio, che l'applicazione abbia due entità: Department e Employee. La relazione tra l'entità Department e quella Employee è una relazione una-a-molti bidirezionale: un reparto ha molti impiegati e un impiegato appartiene solo ad un reparto. Poiché la maggior parte del tempo, quando si esegue il fetch dell'entità Department, è probabile che venga eseguito il fetch dei relativi impiegati, il tipo fetch di questa relazione uno-a-molti è impostato su EAGER.

Qui viene riportato un frammento della classe Department.

```
 @Entity
public class Department {

    @Id
    private String deptId;

    @Basic
    String deptName;

  @OneToMany(fetch = FetchType.EAGER, mappedBy="department", cascade = {CascadeType.PERSIST})
  public Collection<Employee> employees;

}
```

In un ambiente distribuito, quando un'applicazione richiama `em.find(Department.class, "dept1")` per individuare un'entità Department con la chiave "dept1", questa l'operazione di ricerca richiamerà l'entity Department e tutte le relazioni sottoposte a fetch di tipo eager. Nel caso del frammento precedente, sono tutti impiegati del reparto "dept1".

Prima di WebSphere eXtreme Scale 6.1.0.5, il richiamo di un'entità Department e di N entità Employee provocava flussi di richieste tra client e server N+1 poiché il client richiamava un'unica entità per i flussi di richieste tra client e server. È possibile migliorare le prestazioni se si richiamano queste entità N+1 in un unico flusso di richieste.

Piano fetch

È possibile utilizzare un piano fetch per personalizzare le modalità di fetch delle relazioni eager personalizzando la profondità massima delle relazioni. La profondità fetch sovrascrive le relazioni maggiori della profondità specificata per relazioni lazy. Per impostazione predefinita, la profondità fetch è quella massima. Ciò significa che verrà eseguito il fetch delle relazioni eager di tutti i livelli che sono navigabili in modalità eager da un'entità root. Una relazione EAGER è navigabile in modalità eager da un'entità root se, e solo se, tutte le relazioni che iniziano da un'entità root sono configurate con fetch di tipo eager.

Nell'esempio precedente, l'entità Employee è navigabile in modalità eager dall'entità Department poiché la relazione Department-Employee è configurata con fetch di tipo eager.

Se, ad esempio, l'entità Employee ha un'altra relazione eager con un'entità Address, l'entità Address è anche navigabile in modalità eager dall'entità Department. Tuttavia, se le relazioni Department-Employee sono state configurate con fetch di tipo lazy, l'entità Address non è navigabile in modalità eager dall'entità Department poiché la relazione Department-Employee interrompe la catena fetch di tipo eager.

Un oggetto FetchPlan può essere richiamato da un'istanza EntityManager. L'applicazione può utilizzare il metodo setMaxFetchDepth per modificare la profondità fetch massima.

Un piano fetch viene associato ad un'istanza EntityManager. Il piano fetch si applica ad ogni operazione fetch, più specificamente nel modo seguente.

- Operazioni EntityManager `find(Class class, Object key)` e `findForUpdate(Class class, Object key)`
- Operazioni Query
- Operazioni QueryQueue

L'oggetto FetchPlan è mutabile. Una volta modificato, il valore modificato verrà applicato alle operazioni fetch eseguite in seguito.

Un piano fetch è importante per una distribuzione distribuita poiché decide se le entità di relazione sottoposte a fetch di tipo eager vengono richiamate con l'entità root in uno o più flussi di richieste tra client e server.

Proseguendo con il precedente esempio, si consideri inoltre che il piano fetch ha la profondità massima impostata su infinito. In tal caso, quando un'applicazione richiama `em.find(Department.class, "dept1")` per trovare un Department, questa operazione di ricerca richiamerà una entità Department e N entità Employee in un flusso di richieste tra client e server. Tuttavia, per un piano fetch con la profondità massima impostata a zero, verrà richiamato solo l'oggetto Department dal server, mentre le entità Employee vengono richiamate dal server solo quando si accede alla raccolta di Employee dell'oggetto Department.

Piani fetch differenti

Vi sono differenti piani fetch che si basano sui requisiti, illustrati nelle seguenti sezioni.

Impatto su una griglia distribuita

- *Piano fetch con profondità infinita:* un piano fetch con profondità infinita ha la profondità massima impostata su `FetchPlan.DEPTH_INFINITE`.

  In un ambiente client-server, se viene utilizzato un piano fetch con profondità infinita, verranno richiamate tutte le relazioni che sono navigabili in modalità eager dall'entità root in un unico flusso di richieste tra client e server.

  Esempio: se l'applicazione è interessata a tutte le entità Address di tutti gli Employee di un particolare Department, utilizza un piano ferch con profondità infinita per richiamare tutte le entità Address associate. Il seguente codice determina solo un unico flusso di richieste tra client e server.

  ```
  em.getFetchPlan().setMaxFetchDepth(FetchPlan.DEPTH_INFINITE);

  tran.begin();
  Department dept = (Department) em.find(Department.class, "dept1");
  // do something with Address object.
  for (Employee e: dept.employees) {
      for (Address addr: e.addresses) {
      // do something with addresses.
      }
  }
  tran.commit();
  ```

- *Piano fetch con profondità zero:* un piano fetch con profondità zero ha la profondità massima impostata su 0.

In un ambiente client-server, se viene utilizzato un piano fetch con profondità zero, verrà richiamata solo l'entità root nel primo flusso di richieste tra client e server. Tutte le relazioni eager vengono trattate come se fossero di tipo lazy.

Esempio: in questo esempio, l'applicazione è interessata solo all'attributo dell'entità Department. Non è necessario accedere ai relativi Employee, pertanto l'applicazione imposta la profondità del piano fetch su 0.

```
Session session = objectGrid.getSession();
EntityManager em = session.getEntityManager();
EntityTransaction tran = em.getTransaction();
em.getFetchPlan().setMaxFetchDepth(0);

tran.begin();
Department dept = (Department) em.find(Department.class, "dept1");
// do something with dept object.
tran.commit();
```

- *Piano fetch con profondità k* :

  Un piano fetch con profondità *k*- ha la profondità massima impostata su *k*.

  In un ambiente client-server eXtreme Scale, se viene utilizzato un piano fetch con profondità *k*, verranno richiamate tutte le relazioni navigabili in modalità eager dall'entità root all'interno di *k* passi nel primo flusso di richieste tra client e server.

  Il piano fetch con profondità infinita (*k* = infinito) e con profondità zero (*k* = 0) sono solo due esempi di piano fetch con profondità *k*.

  Se continuiamo con l'esempio precedente e lo estendiamo, si immagini che vi sia un'altra relazione eager tra l'entità Employee e quella Address. Se il piano fetch ha la profondità massima impostata su 1, l'operazione `em.find(Department.class, "dept1")` richiamerà l'entità Department e tutte le relative entità Employee in un unico flusso di richieste tra client e server. Tuttavia, le entità Address non verranno richiamate poiché non sono navigabili in modalità eager dall'entità Department nel passo 1, ma nel passo 2.

  Se si utilizza un piano fetch con una profondità impostata su 2, l'operazione `em.find(Department.class, "dept1")` richiamerà l'entità Department, tutte le relative entità Employee e tutte le entità Address associate alle entità Employee in un unico flusso di richieste tra client e server.

  Suggerimento: Il piano fetch predefinito ha la profondità massima impostata su infinito, pertanto il comportamento predefinito di un'operazione fetch può variare. Vengono richiamate tutte le relazioni navigabili in modalità eager dall'entità root. Invece di più flussi di richieste, ora l'operazione fetch con il piano fetch predefinito determina solo un unico flusso di richieste tra client e server. Per mantenere nel prodotto impostazioni della versione precedente, impostare la profondità fetch su 0.

- *Piano fetch utilizzato su query:*

  se si esegue una query di entità, è possibile anche utilizzare un piano fetch per personalizzare il richiamo della relazione.

  Ad esempio, il risultato della query `SELECT d FROM Department d WHERE "d.deptName='Department'"` ha una relazione con l'entità Department. Si noti che la profondità del piano fetch inizia dall'associazione dei risultati della query; in tal caso, l'entità Department e non il risultato della query stessa. Cioè, l'entità Department è sul livello 0 di profondità fetch. Pertanto, un piano fetch con una profondità massima 1 richiamerà l'entità Department e le relative entità Employee in un unico flusso di richieste tra client e server.

  Esempio: in questo esempio, la profondità del piano fetch è impostata su 1, pertanto l'entità Department e le relative entità Employee vengono richiamate in un unico flusso di richieste tra client e server, ma le entità Address non verranno richiamate nello stesso flusso di richieste.

Importante: Se una relazione è ordinata, utilizzando l'annotazione o la configurazione OrderBy, viene considerata una relazione eager anche se viene configurata con un fetch di tipo lazy.

Considerazioni sulle prestazioni in un ambiente distribuito

Per impostazione predefinita, tutte le relazioni che sono navigabili in modalità eager dall'entità root verranno richiamate in un unico flusso di richieste tra client e server. Ciò può migliorare le prestazioni se tutte le relazioni verranno utilizzate. Tuttavia, in determinati scenari di utilizzo, non vengono utilizzate tutte le relazioni navigabili in modalità eager dall'entità root, pertanto possono determinare un sovraccarico del runtime e della larghezza di banda richiamando queste entità non utilizzate.

In tali casi, l'applicazione può impostare la profondità fetch massima su un numero piccolo per diminuire la profondità delle entità da richiamare rendendo tutte le relazioni, dopo quella determinata profondità, di tipo lazy. Questa impostazione può migliorare le prestazioni.

Proseguendo ancora con il precedente esempio Department-Employee-Address, per impostazione predefinita, tutte le entità Address associate agli Employee del Department "dept1" verranno richiamate quando viene richiamato `em.find(Department.class, "dept1")`. Se l'applicazione non utilizza le entità Address, è possibile impostare la profondità fetch massima su 1, pertanto le entità Address non verranno richiamate con l'entità Department.

Impatto sulle prestazioni dell'interfaccia EntityManager

È di poca praticità avere un ambiente che richiede per ogni applicazione di utilizzare gli stessi oggetti di accesso ai dati per un determinato archivio dati. Invece, l'interfaccia EntityManager fornita con WebSphere eXtreme Scale separa le applicazioni dallo stato conservato nel relativo archivio dati della griglia server.

Il peso di utilizzo dell'interfaccia EntityManager non è eccessivo e dipende dal tipo di attività che si sta eseguendo. Utilizzare sempre l'interfaccia EntityManager e ottimizzare la logica di business fondamentale al completamento dell'applicazione. È possibile riutilizzare un qualsiasi codice che adopera le interfacce EntityManager per utilizzare mappe e tuple. In generale, il riutilizzo di codice potrebbe essere necessario per il dieci percento del codice.

Se si utilizzano relazioni tra oggetti, l'impatto sulle prestazioni è minore perché un'applicazione che utilizza mappe deve gestire quelle relazioni in modo simile all'interfaccia EntityManager.

Le applicazioni che utilizzano l'interfaccia EntityManager non devono fornire un ObjectTransformer perché è ottimizzata automaticamente.

Riutilizzo di codice EntityManager per mappe

Di seguito è riportata un'entità di esempio:

```
@Entity
public class Person
{
 @Id
 String ssn;
 String firstName;
```

```
 @Index
 String middleName;
 String surname;
}
```

Di seguito è riportato del codice richiesto per individuare e aggiornare l'entità:

```
Person p = null;
s.begin();
p = (Person)em.find(Person.class, "1234567890");
p.middleName = String.valueOf(inner);
s.commit();
```

Di seguito è riportato lo stesso codice che utilizza mappe e tuple:

```
Tuple key = null;
key = map.getEntityMetadata().getKeyMetadata().createTuple();
key.setAttribute(0, "1234567890");

// The Copy Mode is always NO_COPY for entity maps if not using COPY_TO_BYTES.
// Either we need to copy the tuple or we can ask the ObjectGrid to do it for us:
map.setCopyMode(CopyMode.COPY_ON_READ);
s.begin();
Tuple value = (Tuple)map.get(key);
value.setAttribute(1, String.valueOf(inner));
map.update(key, value);
value = null;
s.commit();
```

Entrambi questi frammenti di codice forniscono lo stesso risultato, e un'applicazione può utilizzare uno o entrambi i frammenti.

Il secondo frammento di codice mostra in che modo utilizzare direttamente le mappe e gestire le tuple (le coppie chiave e valore). La tupla del valore ha tre attributi: firstName, middlename e surname, indicizzati rispettivamente con 0, 1 e 2. Se la tupla di chiavi ha un singolo attributo, il numero ID viene indicizzato su zero. È possibile verificare il modo in cui vengono create le tuple utilizzando i metodi EntityMetadata#getKeyMetaData o EntityMetadata#getValueMetaData. È necessario utilizzare questi metodi per creare tuple di un'entità. Non è possibile implementare l'interfaccia Tuple e passare un'istanza della propria implementazione.

Agent di strumentazione

È possibile migliorare le prestazioni delle entità di accesso al campo abilitando l'agent di strumentazione WebSphere eXtreme Scale quando si utilizza JDK (Java Development Kit) Versione 1.5 o successiva.

Abilitazione dell'agent eXtreme Scale su JDK Versione 1.5 o superiore

L'agent ObjectGrid può essere abilitato con un'opzione della riga comandi Java con la seguente sintassi:

```
-javaagent:jarpath[=options]
```

Il valore *jarpath* indica il percorso di un file JAR (Java archive) di runtime eXtreme Scale che contiene la classe agent eXtreme Scale e le classi di supporto come i file `objectgrid.jar`, `wsobjectgrid.jar`, `ogclient.jar`, `wsogclient.jar` e `ogagent.jar`. In genere, in un programma Java autonomo o in un ambiente Java Platform, Enterprise Edition su cui non viene eseguito WebSphere Application Server, utilizzare il file `objectgrid.jar` o `ogclient.jar`. In WebSphere Application Server o in un ambiente con più programmi di caricamento classi, è necessario utilizzare il file `ogagent.jar` nell'opzione agent della riga comandi Java. Per specificare ulteriori

informazioni, fornire il file `ogagent.config` nel percorso classi o utilizzare le opzioni dell'agent.

Opzioni dell'agent eXtreme Scale

config
Sostituisce il nome file di configurazione.

include
Specifica o sostituisce la definizione del dominio di trasformazione che è la prima parte del file di configurazione.

exclude
Specifica o sostituisce la definizione @Exclude.

fieldAccessEntity
Specifica o sostituisce la definizione @FieldAccessEntity.

trace Specifica un livello di traccia. I livelli possono essere ALL, CONFIG, FINE, FINER, FINEST, SEVERE, WARNING, INFO e OFF.

trace.file
Specifica la posizione del file di traccia.

Il punto e virgola (**;**) viene utilizzato come delimitatore per separare ciascuna opzione. La virgola (**,**) viene utilizzata come delimitatore per separare ciascun elemento all'interno di un'opzione. Il seguente esempio descrive l'opzione dell'agent eXtreme Scale per un programma Java:

```
-javaagent:objectgridRoot/lib/objectgrid.jar=config=myConfigFile;
include=includedPackage;exclude=excludedPackage;
fieldAccessEntity=package1,package2
```

File `ogagent.config`

Il file `ogagent.config` è il nome file di configurazione dell'agent eXtreme Scale progettato. Se il nome file si trova nel percorso classi, l'agent eXtreme Scale individua e analizza il file. È possibile sostituire il nome file designato attraverso l'opzione di configurazione dell'agent eXtreme Scale. Il seguente esempio illustra il modo in cui specificare il file di configurazione:

```
-javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
```

Un file di configurazione dell'agent eXtreme Scale contiene le seguenti parti:

- **Dominio di trasformazione:** il dominio di trasformazione è la prima parte nel file di configurazione. Il dominio di trasformazione consiste in un elenco di package e classi inclusi nel processo di trasformazione della classe. Il dominio di trasformazione deve includere tutte le classi entità di accesso al campo e altre classi che fanno riferimento a tali classi entità. Le classi entità di accesso al campo e quelle classi che fanno riferimento alle classi entità di accesso al campo costruiscono il dominio di trasformazione. Se si intende specificare le classi entità di accesso al campo nella parte @FieldAccessEntity, non è necessario includere tali classi in questo punto. Il dominio di trasformazione deve essere completato. In caso contrario, si potrebbe rilevare un'eccezione FieldAccessEntityNotInstrumentedException.
- **@Exclude:** il token @Exclude indica che le classi e i package elencati dopo questo token sono esclusi dal dominio di trasformazione.
- **@FieldAccessEntity:** il token @FieldAccessEntity indica che le classi e i package elencati dopo questo token sono classi e package Entity di accesso al campo. Se non è presente alcuna riga dopo il token @FieldAccessEntity, il dato equivalente

è "No @FieldAccessEntity specified". L'agent eXtreme Scale determina che non vi sono classi e package Entity di accesso al campo definiti. Se sono presenti delle righe dopo il token @FieldAccessEntity, esse rappresentano le classi ed i package Entity di accesso al campo specificati dall'utente. Ad esempio, "dominio entità di accesso al campo". Il dominio entità di accesso al campo è un dominio secondario del dominio di trasformazione. Le classi ed i package elencati nel dominio entità di accesso al campo sono una parte del dominio di trasformazione, anche quando non sono elencati nel dominio di trasformazione. Il token @Exclude, che elenca le classi ed i package esclusi dalla trasformazione, non ha alcun impatto sul dominio Entity di accesso al campo. Quando viene specificato un token @FieldAccessEntity, tutte le entità di accesso al campo devono trovarsi in questo dominio Entity di accesso al campo. In caso contrario, si potrebbe verificare un'eccezione FieldAccessEntityNotInstrumentedException.

File di configurazione agent di esempio (ogagent.config)

```
##############################
# The # indicates comment line
##############################
# This is an ObjectGrid agent config file (the designated file name is ogagent.config) that can be found and parsed by the ObjectGrid agent
# if it is in classpath.
# If the file name is "ogagent.config" and in classpath, Java program runs with -javaagent:objectgridRoot/ogagent.jar will have
# ObjectGrid agent enabled.
# If the file name is not "ogagent.config" but in classpath, you can specify the file name in config option of ObjectGrid agent
#     -javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
# See comments below for more info regarding instrumentation setting override.

# The first part of the configuration is the list of packages and classes that should be included in transformation domain.
# The includes (packages/classes, construct the instrumentation doamin) should be in the beginning of the file.
com.testpackage
com.testClass

# Transformation domain: The above lines are packages/classes that construct the transformation domain.
# The system will process classes with name starting with above packages/classes for transformation.
#
# @Exclude token : Exclude from transformation domain.
# The @Exclude token indicates packages/classes after that line should be excluded from transformation domain.
# It is used when user want to exclude some packages/classes from above specified included packages
#
# @FieldAccessEntity token: Field-access Entity domain.
# The @FieldAccessEntity token indicates packages/classes after that line are field-access Entity packages/classes.
# If there is no lilne after the @FieldAccessEntity token, it is equivalent to "No @FieldAccessEntity specified".
# The runtime will consider the user does not specify any field-access Entity packages/classes.
# The "field-acces Entity domain" is a sub-domain of transformation domain.
#
# Packages/classes listed in the "field-access Entity domain" will always be part of transformation domain,
# even they are not listed in transformation domain.
# The @Exclude, which lists packages/classes excluded from transformation, has no impact on the "field-acces Entity domain".
# Note: When @FieldAccessEntity is specified, all field-access entities must be in this field-acces Entity domain,
#       otherwise, FieldAccessEntityNotInstrumentedException may occur.
#
# The default ObjectGrid agent config file name is ogagent.config
# The runtime will look for this file as a resource in classpath and process it.
# Users can override this designated ObjectGrid agent config file name via config option of agent.
#
# e.g.
# javaagent:objectgridRoot/lib/objectgrid.jar=config=myOverrideConfigFile
#
# The instrumentation definition, including transformation domain, @Exclude, and  @FieldAccessEntity can be overriden individually
# by corresponding designated agent options.
# Designated agent options include:
#    include              -> used to override instrumentation domain definition that is the first part of the config file
#    exclude              -> used to override @Exclude definition
#    fieldAccessEntity    -> used to override @FieldAccessEntity definition
#
# Each agent option should be separated by ";"
# Within the agent option, the package or class should be seperated by ","
#
# The following is an example that does not override the config file name:
#    -javaagent:objectgridRoot/lib/objectgrid.jar=include=includedPackage;exclude=excludedPackage;fieldAccessEntity=package1,package2
#
##############################

@Exclude
com.excludedPackage
com.excludedClass

@FieldAccessEntity
```

Considerazioni sulle prestazioni

Per prestazioni migliori, specificare il dominio di trasformazione e il dominio entità di accesso al campo.

Code della query di entità

Le code della query consentono alle applicazioni di creare una coda qualificata da una query sul lato server o eXtreme Scale locale su un'entità. Le entità che provengono dal risultato della query sono memorizzate in questa coda. Attualmente, la coda della query è supportata solo in una mappa che utilizza una strategia di blocco pessimistico.

Una coda della query è condivisa da più transazioni e client. Quando la coda della query diventa vuota, la query di entità associata a questa coda viene eseguita nuovamente ed i nuovi risultati vengono aggiunti alla coda. Una coda della query è identificata in modo univoco dai parametri e dalla stringa della query di entità. In un'istanza ObjectGrid è presente una sola istanza per ciascuna coda della query univoca. Per ulteriori informazioni, consultare la documentazione relativa all'API EntityManager.

Esempio di coda della query

L'esempio riportato di seguito illustra come è possibile utilizzare le code della query.

```
/**
 * Get a unassigned question type task
 */
private void getUnassignedQuestionTask() throws Exception {
    EntityManager em = og.getSession().getEntityManager();
    EntityTransaction tran = em.getTransaction();

    QueryQueue queue = em.createQueryQueue("SELECT t FROM Task t
    WHERE t.type=?1 AND t.status=?2", Task.class);
    queue.setParameter(1, new Integer(Task.TYPE_QUESTION));
    queue.setParameter(2, new Integer(Task.STATUS_UNASSIGNED));

    tran.begin();
    Task nextTask = (Task) queue.getNextEntity(10000);
    System.out.println("next task is " + nextTask);
    if (nextTask != null) {
        assignTask(em, nextTask);
    }
    tran.commit();
}
```

Nell'esempio precedente, viene creato un oggetto QueryQueue con una stringa della query di entità, "SELECT t FROM Task t WHERE t.type=?1 AND t.status=?2". Quindi, vengono impostati i parametri per l'oggetto QueryQueue. Questa coda della query rappresenta tutte le attività "unassigned" del tipo "question". L'oggetto QueryQueue è molto simile ad un oggetto Query dell'entità.

Una volta creato l'oggetto QueryQueue, viene avviata una transazione dell'entità e viene richiamato il metodo getNextEntity, che richiama la successiva entità disponibile con un valore di timeout impostato su 10 secondi. Una volta richiamata, l'entità viene elaborata nel metodo assignTask. Il metodo assignTask modifica l'istanza dell'entità Task e ne modifica lo stato in "assigned", rimuovendola dalla coda perché non corrisponde più al filtro di QueryQueue. Una volta assegnata la transazione, ne viene eseguito il commit.

In questo semplice esempio, è possibile notare che una coda della query è simile ad una query di entità. Tuttavia, esistono alcune differenze:

1. Le entità nella coda della query possono essere richiamate in modo iterativo. L'applicazione utente decide il numero di entità da richiamare. Ad esempio, se viene utilizzato QueryQueue.getNextEntity(timeout), viene richiamata una sola entità e se viene utilizzato QueryQueue.getNextEntities(5, timeout), vengono richiamate 5 entità. In un ambiente distribuito, il numero di entità decide direttamente il numero di byte da trasferire dal server al client.
2. Quando viene richiamata un'entità dalla coda della query, sull'entità viene posizionato un blocco U, in modo che nessun'altra transazione possa accedere all'entità.

Richiamo delle entità in un loop

È possibile richiamare le entità in un loop. Di seguito è riportato un esempio che illustra come richiamare tutte le attività unassigned di tipo question completate.

```
/**
 * Get all unassigned question type tasks
 */
private void getAllUnassignedQuestionTask() throws Exception {
    EntityManager em = og.getSession().getEntityManager();
    EntityTransaction tran = em.getTransaction();

    QueryQueue queue = em.createQueryQueue("SELECT t FROM Task t WHERE
   t.type=?1 AND t.status=?2", Task.class);
    queue.setParameter(1, new Integer(Task.TYPE_QUESTION));
    queue.setParameter(2, new Integer(Task.STATUS_UNASSIGNED));

    Task nextTask = null;

    do {
        tran.begin();
        nextTask = (Task) queue.getNextEntity(10000);
        if (nextTask != null) {
            System.out.println("next task is " + nextTask);
        }
        tran.commit();
    } while (nextTask != null);
}
```

Se nella mappa di entità sono presenti 10 attività di tipo question unassigned, si potrebbe pensare che nella console vengano visualizzate 10 entità. Tuttavia, se questo esempio viene eseguito, il programma non termina mai, contrariamente a quanto previsto.

Quando viene creata una coda della query e viene richiamato getNextEntity, la query di entità associata alla coda viene eseguita e nella coda vengono inseriti i 10 risultati. Quando viene richiamato getNextEntity, un'entità viene estratta dalla coda. Una volta eseguite 10 chiamate getNextEntity, la coda è vuota. La query di entità viene automaticamente nuovamente eseguita. Poiché tali 10 entità sono ancora presenti e corrispondono ai criteri di filtro della coda della query, vengono nuovamente inserite nella coda.

Se dopo l'istruzione println() viene aggiunta la riga riportata di seguito, vengono visualizzate solo 10 entità.

```
em.remove(nextTask);
```

Per informazioni relative all'utilizzo di SessionHandle con QueryQueue in una distribuzione di posizionamento per contenitore, consultare Integrazione SessionHandle.

Code della query distribuite a tutte le partizioni

In un ambiente eXtreme Scale distribuito, è possibile creare una coda della query per una o per tutte le partizioni. Se una coda della query viene creata per tutte le partizioni, in ciascuna partizione sarà presente un'istanza della coda della query.

Quando un client prova ad ottenere l'entità successiva utilizzando il metodo QueryQueue.getNextEntity o QueryQueue.getNextEntities, il client invia una richiesta ad una delle partizioni. Il client invia al server richieste peek e pin:

- Con una richiesta peek, il client invia una richiesta ad una partizione ed il server risponde immediatamente. Se nella coda è presente un'entità, il server invia una risposta con l'entità; in caso contrario, il server invia una risposta senza entità. In entrambi i casi, il server risponde immediatamente.
- Con una richiesta pin, il client invia una richiesta ad una partizione ed il server attende fino a quando è disponibile un'entità. Se nella coda è presente un'entità, il server invia immediatamente una risposta con l'entità; in caso contrario, il server attende sulla coda fino a quando è disponibile un'entità o fino allo scadere della richiesta.

Di seguito è riportato un esempio del modo in cui un'entità viene richiamata da una coda della query distribuita a tutte le partizioni (n):

1. Quando viene richiamato un metodo QueryQueue.getNextEntity o QueryQueue.getNextEntities, il client sceglie un numero di partizione a caso da 0 a n-1.
2. Il client invia la richiesta peek alla partizione casuale.
   - Se è disponibile un'entità, il metodo QueryQueue.getNextEntity o QueryQueue.getNextEntities termina restituendo l'entità.
   - Se non è disponibile alcuna entità e la partizione non è l'ultima partizione non visitata, il client invia una richiesta peek alla partizione successiva.
   - Se non è disponibile un'entità e la partizione è l'ultima partizione non visitata, il client invia una richiesta pin.
   - Se la richiesta pin all'ultima partizione scade e ancora non sono disponibili dati, il client invia ancora una volta una richiesta peek a tutte le partizioni in modo seriale. Quindi, se nelle partizioni precedenti è disponibile un'entità, il client sarà in grado di ottenerla.

Entità del sottoinsieme e supporto per nessuna entità

Di seguito è riportato il metodo per la creazione di un oggetto QueryQueue nel gestore entità:

```
public QueryQueue createQueryQueue(String qlString, Class entityClass);
```

Il risultato nella coda della query deve essere proiettato sull'oggetto definito dal secondo parametro del metodo, Class entityClass.

Se questo parametro viene specificato, la classe deve avere lo stesso nome dell'entità specificato nella stringa della query. Ciò è utile se si desidera proiettare un'entità in un'entità del sottoinsieme. Se come classe entità viene utilizzato un valore null, il risultato non verrà proiettato. Il valore memorizzato nella mappa sarà in formato tuple di entità.

Conflitto di chiavi lato client

In un ambiente eXtreme Scale distribuito, la coda della query è supportata solo per mappe eXtreme Scale con modalità di blocco pessimistico. Quindi, sul lato client non è presente alcuna cache locale. Tuttavia, un client può disporre di dati (chiave e valore) nella mappa transazionale. Ciò potrebbe determinare un conflitto di chiavi quando un'entità richiamata dal server condivide la stessa chiave di un'entità già presente nella mappa transazionale.

Quando si verifica un conflitto di chiavi, il runtime del client eXtreme Scale utilizza la seguente regola per generare un'eccezione o sovrascrivere i dati.

1. Se la chiave in conflitto è la chiave dell'entità specificata nella query di entità associata alla coda della query, viene generata un'eccezione. In questo caso, viene eseguito il rollback della transazione ed il blocco U su questa chiave dell'entità verrà rilasciato sul lato server.
2. In caso contrario, se la chiave in conflitto è la chiave dell'associazione di entità, i dati nella mappa transazionale verranno sovrascritti senza preavviso.

Il conflitto di chiavi si verifica solo quando nella mappa transazionale sono presenti dei dati. In altre parole, si verifica solo quando viene richiamata la chiamata getNextEntity o getNextEntities in una transazione che è già stata modificata (sono stati inseriti nuovi dati o i dati presenti sono stati aggiornati). Se in un'applicazione non si desidera che si verifichi un conflitto di chiavi, le chiamate a getNextEntity o getNextEntities devono essere sempre eseguite in una transazione non modificata.

Errori del client

Una volta inviata una richiesta getNextEntity o getNextEntities al server, si possono verificare i seguenti errori del client:

1. Il client invia una richiesta al server e si disattiva.
2. Il client riceve una o più entità dal server e si disattiva.

Nel primo caso, il server rileva che il client si sta disattivando quando prova ad inviare la risposta al client. Nel secondo caso, quando il client riceve una o più entità dal server, su tali entità viene posizionato un blocco X. Se il client si disattiva, la transazione scade ed il blocco X viene rilasciato.

Query con clausola ORDER BY

Generalmente, le code delle query non onorano la clausola ORDER BY. Se dalla coda della query vengono richiamati getNextEntity o getNextEntities, non vi è alcuna garanzia che le entità vengano restituite in base all'ordine. Il motivo è che le entità non possono essere ordinate tra le partizioni. Nel caso in cui la coda della query viene distribuita a tutte le partizioni, quando viene eseguita una chiamata getNextEntity o getNextEntities, viene scelta una partizione a caso per elaborare la richiesta. Per questo motivo, l'ordine non è garantito.

La clausola ORDER BY viene onorata se una coda della query viene distribuita ad una partizione singola.

Per ulteriori informazioni, consultare “API della query di EntityManager” a pagina 105.

Una chiamata per transazione

Ogni chiamata QueryQueue.getNextEntity o QueryQueue.getNextEntities richiama le entità corrispondenti da una partizione casuale. Le applicazioni devono chiamare esattamente un'entità QueryQueue.getNextEntity o QueryQueue.getNextEntities su una transazione. In caso contrario, eXtreme Scale potrebbe concludere coinvolgendo le entità di più partizioni, generando un'eccezione in fase di commit.

Interfaccia EntityTransaction

È possibile utilizzare l'interfaccia EntityTransaction per delineare le transazioni.

Scopo

Per delineare una transazione è possibile utilizzare l'interfaccia EntityTransaction, la quale è associata a un'istanza del gestore entità. Utilizzare il metodo EntityManager.getTransaction per richiamare l'istanza EntityTransaction del gestore entità. Ogni istanza EntityManager e EntityTransaction è associata a Session. È possibile delineare le transazioni con EntityTransaction o Session. I metodi nell'interfaccia EntityTransaction non dispongono di eccezioni controllate. Risultano disponibili solo eccezioni runtime di tipo PersistenceException o relative sottoclassi.

Per ulteriori informazioni sull'interfaccia EntityTransaction, consultare la documentazione APIl'interfaccia EntityTransaction nella documentazione API.

Supporto didattico gestore entità: panoramica

Il Supporto didattico per il gestore entità mostra come utilizzare WebSphere eXtreme Scale per memorizzare informazioni sull'ordine in un sito Web. È possibile creare una semplice applicazione Java Platform, Standard Edition 5 che utilizza eXtreme Scale in memoria, locale. Le entità utilizzano annotazioni e generics di Java SE 5.

Prima di iniziare

Assicurarsi di aver soddisfatto i seguenti requisiti prima di iniziare il supporto didattico:

- È necessario avere Java SE 5.
- È necessario avere il file `objectgrid.jar` nel proprio percorso di classe.

Richiamo di entità e oggetti (API di query)

WebSphere eXtreme Scale fornisce un motore di query flessibile per il richiamo delle entità utilizzando l'API EntityManager e degli oggetti Java utilizzando l'API ObjectQuery.

Capability della query WebSphere eXtreme Scale

Con il motore di query eXtreme Scale, è possibile eseguire query di tipo SELECT su un'entità o uno schema basato su oggetti utilizzando il linguaggio di query eXtreme Scale.

Tale linguaggio di query fornisce le capability riportate di seguito:

- Risultati a singoli e più valori
- Funzioni di aggregazione
- Ordinamento e raggruppamento
- Unioni
- Espressioni condizionali con query secondarie
- Parametri denominati e posizionali
- utilizzo dell'indice eXtreme Scale
- Sintassi dell'espressione di percorso per la navigazione negli oggetti
- Paginazione

Interfaccia della query

Utilizzare l'interfaccia della query per controllare l'esecuzione della query di entità.

Utilizzare il metodo EntityManager.createQuery(String) per creare una query. È possibile utilizzare ciascuna istanza di query più volte con l'istanza EntityManager in cui è stata richiamata.

Ciascun risultato della query produce un'entità, in cui la chiave dell'entità è l'ID riga (di tipo long) ed il valore dell'entità contiene i risultati del campo della clausola SELECT. È possibile utilizzare ciascun risultato della query nelle query successive.

Per l'interfaccia com.ibm.websphere.objectgrid.em.Query sono disponibili i metodi riportati di seguito.

public ObjectMap getResultMap()

Il metodo getResultMap esegue una query SELECT e restituisce i risultati in un oggetto ObjectMap con i risultati nell'ordine specificato dalla query. L'oggetto ObjectMap risultante è valido solo per la transazione corrente.

La chiave della mappa è il numero del risultato, di tipo long, che inizia da 1. Il valore della mappa è di tipo com.ibm.websphere.projector.Tuple in cui ciascun attributo ed associazione è denominato in base alla propria posizione ordinale all'interno della clausola select della query. Utilizzare il metodo per richiamare l'EntityMetadata per l'oggetto Tuple memorizzato all'interno della mappa.

Il metodo getResultMap è il metodo più rapido per il richiamo dei dati dei risultati della query quando possono esistere più risultati. È possibile richiamare il nome dell'entità risultante utilizzando i metodi ObjectMap.getEntityMetadata() ed EntityMetadata.getName().

Esempio: la query riportata di seguito restituisce due righe.

```
String ql = SELECT e.name, e.id, d from Employee e join e.dept d WHERE d.number=5
 Query q = em.createQuery(ql);
 ObjectMap resultMap = q.getResultMap();
 long rowID = 1; // starts with index 1
 Tuple tResult = (Tuple) resultMap.get(new Long(rowID));
 while(tResult != null) {
     // The first attribute is name and has an attribute name of 1
     // But has an ordinal position of 0.
     String name = (String)tResult.getAttribute(0);
     Integer id = (String)tResult.getAttribute(1);

     // Dept is an association with a name of 3, but
     // an ordinal position of 0 since it's the first association.
     // The association is always a OneToOne relationship,
     // so there is only one key.
     Tuple deptKey = tResult.getAssociation(0,0);
     ...
     ++rowID;
     tResult = (Tuple) resultMap.get(new Long(rowID));

 }
```

public Iterator getResultIterator

Il metodo getResultIterator esegue una query SELECT e restituisce i risultati della query utilizzando un Iterator in cui ciascun risultato è un Object per una query con valore singolo oppure un array di Object per una query con più valori. I valori nel risultato Object[] sono memorizzati nell'ordine della query. L'Iterator del risultato è valido solo per la transazione corrente.

Questo metodo è il metodo preferito per il richiamo dei risultati della query nel contesto EntityManager. È possibile utilizzare il metodo setResultEntityName(String) facoltativo per assegnare un nome all'entità risultante in modo che sia possibile utilizzarla nelle query successive.

Esempio: la query riportata di seguito restituisce due righe.

```
String ql = SELECT e.name, e.id, e.dept from Employee e WHERE e.dept.number=5
 Query q = em.createQuery(ql);
 Iterator results = q.getResultIterator();
 while(results.hasNext()) {
     Object[] curEmp = (Object[]) results.next();
     String name = (String) curEmp[0];
     Integer id = (Integer) curEmp[1];
     Dept d = (Dept) curEmp[2];
     ...
 }
```

public Iterator getResultIterator(Class resultType)

Il metodo getResultIterator(Class resultType) esegue una query SELECT e restituisce i risultati della query utilizzando un Iterator di entità. Il tipo dell'entità è determinato dal parametro resultType. L'Iterator del risultato è valido solo per la transazione corrente.

Utilizzare questo metodo quando di desidera utilizzare le API EntityManager per accedere alle entità risultanti.

Esempio: la query riportata di seguito restituisce tutti gli impiegati ed il relativo reparto di appartenenza per una divisione, ordinati in base al salario. Per visualizzare i cinque impiegati con i salari più alti e lavorare con impiegati provenienti da un solo reparto nella stessa unità lavorativa, utilizzare il codice riportato di seguito.

```
String string_ql = "SELECT e.name, e.id, e.dept from Employee e WHERE
  e.dept.division='Manufacturing' ORDER BY e.salary DESC";
Query query1 = em.createQuery(string_ql);
query1.setResultEntityName("AllEmployees");
Iterator results1 = query1.getResultIterator(EmployeeResult.class);
int curEmployee = 0;
System.out.println("Highest paid employees");
while (results1.hasNext() && curEmployee++ < 5) {
 EmployeeResult curEmp = (EmployeeResult) results1.next();
 System.out.println(curEmp);
 // Remove the employee from the resultset.
 em.remove(curEmp);
}

// Flush the changes to the result map.
em.flush();

// Run a query against the local working set without the employees we
// removed
String string_q2 = "SELECT e.name, e.id, e.dept from AllEmployees e
  WHERE e.dept.name='Hardware'";
Query query2 = em.createQuery(string_q2);
Iterator results2 = query2.getResultIterator(EmployeeResult.class);
System.out.println("Subset list of Employees");
while (results2.hasNext()) {
 EmployeeResult curEmp = (EmployeeResult) results2.next();
 System.out.println(curEmp);
}
```

public Object getSingleResult

Il metodo getSingleResult esegue una query SELECT che restituisce un singolo risultato.

Se per la clausola SELECT è definito più di un campo, il risultato è un array di oggetti, in cui ciascun elemento dell'array è basato sulla propria posizione ordinale all'interno della clausola SELECT della query.

```
String ql = SELECT e from Employee e WHERE e.id=100"
 Employee e = em.createQuery(ql).getSingleResult();

 String ql = SELECT e.name, e.dept from Employee e WHERE e.id=100"
 Object[] empData = em.createQuery(ql).getSingleResult();
 String empName= (String) empData[0];
 Department empDept = (Department) empData[1];
```

public Query setResultEntityName(String entityName)

Il metodo setResultEntityName(String entityName) specifica il nome dell'entità del risultato della query.

Ogni volta che vengono richiamati i metodi getResultIterator o getResultMap, viene dinamicamente creata un'entità con un ObjectMap per contenere i risultati della query. Se l'entità non è specificata o è null, il nome dell'entità e di ObjectMap vengono generati automaticamente.

Poiché tutti i risultati della query sono disponibili per la durata di una transazione, non è possibile utilizzare nuovamente il nome della query in una transazione singola.

public Query setPartition(int partitionId)

Imposta la partizione verso cui è indirizzata la query.

Questo metodo è richiesto se le mappe nella query sono suddivise in partizioni e se il gestore entità non ha affinità con una singola partizione dell'entità root dello schema.

Utilizzare l'interfaccia PartitionManager per determinare il numero di partizioni per la mappa di backup di una determinata entità.

La tabella riportata di seguito fornisce le descrizioni degli altri metodi disponibili nell'interfaccia della query.

Tabella 2. Altri metodi

Metodo	Risultato
public Query setMaxResults(int maxResult)	Imposta il numero massimo di risultati da richiamare.
public Query setFirstResult(int startPosition)	Imposta la posizione del primo risultato da richiamare.
public Query setParameter(String name, Object value)	Esegue il bind di un argomento ad un parametro denominato.
public Query setParameter(int position, Object value)	Esegue il bind di un argomento ad un parametro posizionale.

Tabella 2. Altri metodi (Continua)

Metodo	Risultato
public Query setFlushMode(FlushModeType flushMode)	Imposta il tipo di modalità di scaricamento da utilizzare quando viene eseguita la query, sostituendo il tipo di modalità di scaricamento impostato su EntityManager.

Elementi della query eXtreme Scale

Con il motore di query eXtreme Scale, è possibile utilizzare un linguaggio di query singolo per eseguire ricerche nella cache eXtreme Scale. Tale linguaggio di query è in grado di eseguire query di oggetti Java memorizzati negli oggetti ObjectMap e negli oggetti Entity. Utilizzare la sintassi riportata di seguito per creare una stringa di query.

Una query eXtreme Scale è una stringa che contiene gli elementi riportati di seguito:

- Una clausola SELECT che specifica gli oggetti o i valori da restituire.
- Una clausola FROM che indica le raccolte di oggetti.
- Una clausola WHERE facoltativa che contiene predicati di ricerca nelle raccolte.
- Una clausola GROUP BY e HAVING facoltativa (consultare le funzioni di aggregazione della query eXtreme Scale).
- Una clausola ORDER BY facoltativa che specifica l'ordinamento della raccolta di risultati.

Le raccolte di oggetti Java sono identificate nelle query mediante l'utilizzo del rispettivo nome della clausola FROM della query.

Gli elementi del linguaggio di query sono illustrati in dettaglio negli argomenti correlati riportati di seguito:

- Sintassi "BNF (Backus-Naur Form) query ObjectGrid" a pagina 117
- "Guida di riferimento per le query di eXtreme Scale" a pagina 109

Gli argomenti riportati di seguito descrivono le modalità di utilizzo dell'API Query:

- "API della query di EntityManager" a pagina 105
- "Utilizzo dell'API ObjectQuery" a pagina 100

Query dei dati in più fasi orari

In un ambiente distribuito, le query vengono di fatto eseguite sui server. Quando si eseguono query di dati con predicati di tipo calendar, java.util.Date e timestamp, il valore data e ora specificato si basa sul fuso orario locale del server.

In un sistema con un singolo fuso orario tutti i client ed i server sono in esecuzione nello stesso fuso orario, non sarà quindi necessario prendere in considerazione tutte le problematiche legate ai tipi di predicato con calendar, java.util.Date e timestamp. Tuttavia quando i client ed i server si trovano in fusi orari diversi, il valore data/ora specificato nelle query si basa sul fuso orario del server e potrebbe restituire dati indesiderati al client. Se non si conosce il fuso orario del server, il valore data/ora specificato non ha alcun significato. Quindi il valore data/ora specificato deve considerare la differenza di fuso orario tra il fuso orario di destinazione e quello del server.

Differenza di fuso orario

Supporre, ad esempio, che un client sia nel fuso orario [GMT-0] ed il server in quello [GMT-6]. Il fuso orario del server è 6 ore indietro rispetto a quello del client. Il client intenderebbe eseguire la seguente query:

```
SELECT e FROM Employee e WHERE e.birthDate='1999-12-31 06:00:00'
```

Supponendo che l'entità Employee abbia un attributo birthDate di tipo java.util.Date, il client, che si trova nel fuso orario [GMT-0], desidera recuperare gli impiegati con un valore di birthDate pari a '1999-12-31 06:00:00 [GMT-0]' basato sul proprio fuso orario.

La query verrà eseguita sul server ed il valore birthDate utilizzato dal motore di query sarà '1999-12-31 06:00:00 [GMT-6]' che equivale a '1999-12-31 12:00:00 [GMT-0]'. Al client verranno restituiti gli impiegati che hanno un valore birthDate uguale a '1999-12-31 12:00:00 [GMT-0]'. Il client, quindi, non riceverà gli impiegati con valore birthDate '1999-12-31 06:00:00 [GMT-0]' che aveva richiesto.

Il problema descritto si verifica a causa della differenza di fuso orario tra client e server. Un approccio per risolvere questo problema consiste nel calcolare la differenza di fuso orario tra client e server ed applicarla al valore data/ora di destinazione nella query. Nell'esempio di query precedente, la differenza di fuso orario è -6 ed il predicato birthDate corretto dovrebbe essere "birthDate='1999-12-31 00:00:00'" se il client intende recuperare gli impiegati con un valore birthDate '12-31 06:00:00 [GMT-0]'. Con il valore birthDate modificato, il server utilizzerà '1999-12-31 00:00:00 [GMT-6]' che equivale ad un valore di destinazione di '12-31 06:00:00 [GMT-0]', in questo modo saranno restituiti al client gli impiegati effettivamente richiesti.

Distribuzione su più fusi orari

Se la distribuzione della griglia di eXtreme Scale avviene su più server ObjectGrid in fusi orari diversi, l'approccio di modifica con la differenza del fuso orario non funzionerà poiché il client non sa quale server eseguirà la query e non può quindi determinare la differenza di fuso orario. L'unica soluzione è rappresentata dall'utilizzo del suffisso 'Z' (non sensibile alle maiuscole/minuscole) nel formato di escape data/ora di JDBC per indicare l'utilizzo del valore data/ora basato sul fuso orario GMT. Il suffisso 'Z' (non sensibile alle maiuscole/minuscole) indica di utilizzare il valore data/ora basato sul fuso orario GMT. Senza il suffisso 'Z', nel processo che esegue la query verrà utilizzato il valore data/ora basato sul fuso orario locale.

La seguente query è equivalente a quella dell'esempio precedente, ma utilizza il suffisso ‘Z':

```
SELECT e FROM Employee e WHERE e.birthDate='1999-12-31 06:00:00Z'
```

La query deve trovare gli impiegati con un valore birthDate pari a '1999-12-31 06:00:00'. Il suffisso 'Z' indica che il valore birthDate specificato è basato sul fuso orario GMT, in questo modo, il motore di query utilizzerà come criterio di corrispondenza il valore birthDate '1999-12-31 06:00:00 [GMT-0]' basato sul fuso orario GMT. Nei risultati della query verranno quindi inseriti gli impiegati con il valore dell'attributo birthDate pari al birthDate basato sul fuso orario GMT '1999-12-31 06:00:00 [GMT-0]'. L'utilizzo in qualsiasi query del suffisso 'Z' nel formato di escape data/ora di JDBC è fondamentale per mettere le applicazioni al riparo dai problemi derivanti dal fuso orario. Senza questo approccio, il valore

data/ora si baserà sul fuso orario del server e non sarà significativo dal punto di vista del client quando il client ed il server si trovano in fusi orari diversi.

Per ulteriori informazioni, consultare l'argomento relativo all'inserimento di dati per fusi orari diversi nella *Panoramica sul prodotto.*

Inserimento di dati per fusi orari diversi

Quando si inseriscono dati con gli attributi calendar, java.util.Date e timestamp in un ObjectGrid, è necessario assicurarsi che questi attributi di data/ora vengano creati in base allo stesso fuso orario, specialmente quando vengono distribuiti in più server in fusi orari diversi. L'utilizzo di oggetti data/ora basati sullo stesso fuso orario può assicurare che l'applicazione sia affidabile da un punto di vista di fuso orario e che sia possibile eseguire query dei dati tramite i predicati calendar, java.util.Date e timestamp.

Senza la specifica esplicita di un fuso orario durante la creazione di un oggetto data/ora, Java utilizzerà il fuso orario locale e questo potrebbe causare un'incongruenza nei valori di data e ora nei client e nei server.

Prendere in considerazione un esempio in una distribuzione distribuita in cui client1 è nel fuso orario [GMT-0] e client2 è in [GMT-6] ed entrambi desiderano creare un oggetto java.util.Date con valore '1999-12-31 06:00:00'. Client1 creerà l'oggetto java.util.Date con valore '1999-12-31 06:00:00 [GMT-0]' e client2 lo crerà con valore '1999-12-31 06:00:00 [GMT-6]'. I due oggetti java.util.Date non sono uguali perché il fuso orario è diverso. Un problema simile si verifica quando vengono precaricati i dati in partizioni che risiedono su server di fusi orari diversi e viene utilizzato il fuso orario locale per creare gli oggetti data/ora.

Per evitare il problema descritto, l'applicazione può scegliere un fuso orario, ad esempio [GMT-0], come fuso orario base per la creazione di oggetti calendar, java.util.Date e timestamp.

Per ulteriori informazioni, consultare l'argomento relativo alla query dei dati in più fusi orari nella *Guida alla programmazione*.

Utilizzo dell'API ObjectQuery

L'API ObjectQuery fornisce i metodi per eseguire la query sui dati in ObjectGrid che sono memorizzati utilizzando 'API ObjectMap. Quando viene definito uno schema nell'istanza ObjectGrid, può essere utilizzata l'API ObjectQuery per creare ed eseguire le query su oggetti eterogenei memorizzati nelle mappe dell'oggetto.

Mappe della query e dell'oggetto

È possibile utilizzare una capability avanzata della query per gli oggetti che sono memorizzati utilizzando l'API ObjectMap. Queste query consentono il recupero degli oggetti utilizzando gli attributi non-chiave ed eseguono una semplice aggregazione come una somma, una media una sottrazione e un totale rispetto a tutti i dati che trovano corrispondenza nella query. Le applicazioni possono costruire una query utilizzando il metodo Session.createObjectQuery. Questo metodo restituisce un oggetto ObjectQuery che può essere interrogato per ottenere i risultati della query. L'oggetto query consente anche di personalizzare la query prima che questa venga eseguita. La query viene automaticamente eseguita quando viene richiamato qualsiasi metodo che restituisce il risultato.

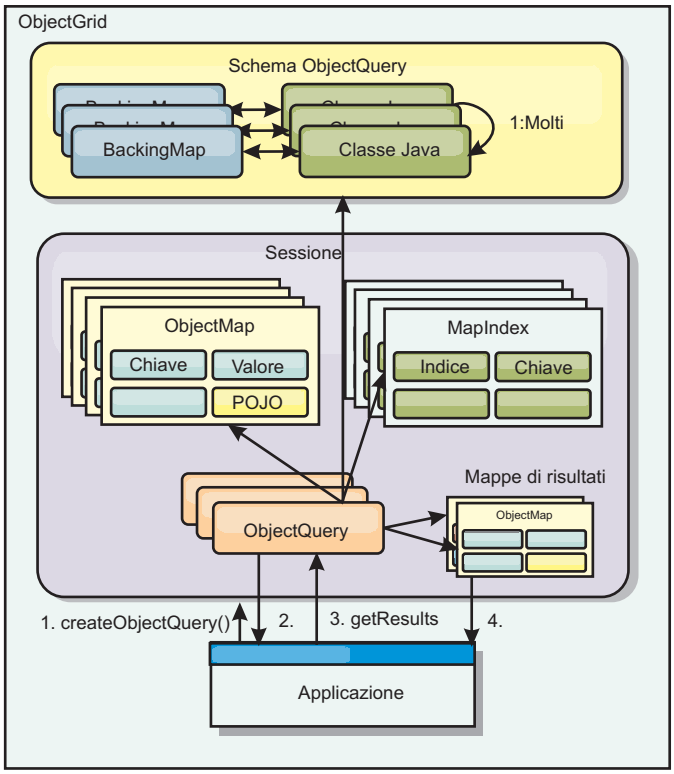


Figura 1. L'interazione della query con le mappe dell'oggetto ObjectGrid e il modo in cui uno schema è definito per le classi con una mappa ObjectGrid

Definizione di uno schema ObjectMap

Le mappe oggetto vengono utilizzate per memorizzare gli oggetti in varie forme e sono in gran parte ignare del formato. Uno schema deve essere definito in ObjectGrid che definisce il formato dei dati. Uno schema è costituito dalle seguenti parti:

- il tipo di oggetto memorizzato in ObjectMap
- le relazioni tra ObjectMaps
- il metodo secondo cui ciascuna query dovrebbe accedere gli attributi dei dati negli oggetti (campi o metodi proprietà)
- il nome dell'attributo chiave primaria nell'oggetto.

Per i dettagli, consultare Configurazione di un o schema ObjectQuery.

Ad esempio, per la creazione di uno schema in modo programmatico oppure utilizzando un file XML del descrittore ObjectGrid, consultare il supporto didattico per ObjectQuery nella sezione *Panoramica sul prodotto*.

Esecuzione della query degli oggetti utilizzando l'API ObjectQuery

L'interfaccia ObjectQuery consente l'esecuzione della query per oggetti non-entità che sono eterogenei e sono memorizzati direttamente nelle OBjectmaps di ObjectGrid L'API ObjectQuery fornisce un modo semplice per trovare gli oggetti di ObjectMap direttamente senza l'utilizzo della parola chiave e i meccanismi dell'indice.

Esistono due metodi per il recupero dei risultati da un ObjectQuery: getResultIterator e getResultMap.

Il recuopero dei risultati della query utilizzando getResultIterator

I risultati della query sono fondamentalmente un elenco di attributi. Supponiamo che la query selezionò a,b,c da X dove y=z. Questa query restituisce un elenco di righe contenenti a, b e c. Questo elenco viene effettivamente memorizzato in una mappa nell'ambito della transazione, che indica che è necessario associare una chiave fittizia a ciascuna riga ed utilizzare un numero intero che incrementi di una riga. Questa mappa è ottenuta utilizzando il metodo ObjectQuery.getResultMap(). È possibile accedere agli elementi di ciascuna riga utilizzando un codice simile a quello di seguito riportato.

```
ObjectQuery q = session.createQuery(
    "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");

  q.setParameter(1, "Claus");

  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id "
      + row[objectgrid: 0 ] + ", firstName: "
      + row[objectgrid: 1 ] + ", surname: "
      + row[objectgrid: 2 ]);
  }
```

Il recupero dei risultati della query utilizzando getResultMap

I risultati della query possono anche essere recuperati utilizzando la mappa del risultato direttamente. L'esempio di seguito riportato mostra una query che recupera parti specifiche delle corrispondenze Customer e dimostra in che modo accedere alle righe risultanti. Considerare che se si utilizza l'oggetto ObjectQuery per accedere ai dati, allora viene nascosto l'identificativo della lunga fila generato. La lunga file è visibile solo quando si utilizza ObjectMap per accedere al risultato.

Quando la transazione è completata questa mappa scompare. La mappa è anche visibile solo alla sessione utilizzata cioè normalmente proprio al thread che l'ha creata. La mappa utilizza una chiave di tipo Long che rappresenta l'ID della riga. I valori memorizzati nella mappa sono sia di tipo Object che Object[], laddove ciascun elemento corrisponde al tipo di elemento nella clausola select della query.

```
ObjectQuery q = em.createQuery(
      "select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
  q.setParameter(1, "Claus");
  ObjectMap qmap = q.getResultMap();
  for(long rowId = 0; true; ++rowId)
  {
    Object[] row = (Object[]) qmap.get(new Long(rowId));
    if(row == null) break;
    System.out.println(" I Found a Claus with id " + row[0]
      + ", firstName: " + row[1]
      + ", surname: " + row[2]);
  }
```

Per gli esempi sull'utilizzo di ObjectQuery, consultare il supporto didattico nell'API ObjectQuery nella sezione *Panoramica sul prodotto*.

Configurazione di uno schema ObjectQuery

ObjectQueryfa fa affidamento sullo schema o sulle informazioni di modello per eseguire il controllo semantico e valutare le espressioni del percorso. Questa sezione descrive in che modo definire lo schema in XML o in modo programmatico.

Definizione dello schema

Lo schema ObjectMap viene definito nell' XML del descrittore di distribuzione di ObjectGrid o in modo programmatico utilizzando le normali tecniche di configurazione di eXtreme Scale. Ad esempio per creare uno schema, consultare "Configurazione di uno schema ObjectQuery"

Le informazioni di schema descrivono POJO (Java plain old objects): da quali attributi essi sono costituiti e quali tipi di attributi potrebbero esistere, se gli attributi sono campi di chiave primaria, relazioni con valore singolo o con più valori oppure relazioni bidirezionali. Le informazioni dello schema instradano ObjectQuery ad utilizzare l'accesso al campo o l'accesso alla proprietà.

Attributi per cui è possibile eseguire la query

Quando lo schema è definito nell'ObjectGrid, gli oggetti nello schema sono esaminati utilizzando la tecnica 'reflection' per determinare quali attributi sono disponibili per la query. È possibile eseguire la query sui seguenti tipi di attributi:

- tipi primitivi Java che includono i wrapper
- java.lang.String
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- java.util.Calendar
- byte[]
- java.lang.Byte[]
- char[]
- java.lang.Character[]
- J2SE enum

In un risultato di query possono anche essere inclusi tipi serializzabili incorporati oltre quelli precedentemente dichiarati, ma non possono essere inclusi nelle clausole WHERE o FROM della query. Gli attributi serializzabili non sono selezionabili.

I tipi di attributi possono essere esclusi dallo schema se il tipo non è serializzabile, il campo o la proprietà sono statici o il campo è transitorio. Poiché tutti gli oggetti della mappa devono essere serializzabili, solo ObjectGrid comprende gli attributi che possono essere resi persistenti nell'oggetto. Gli altri oggetti sono ignorati.

Attributi del campo

Quando lo schema viene configurato per accedere l'oggetto utilizzando i campi, tutti i campi serializzabili non transitori vengono automaticamente incorporati nello schema. Per selezionare un attributo del campo, utilizzare il nome dell'identificativo del campo poiché esso esiste nella definizione della classe.

Tutti i campi pubblici, privati, protetti e i campi protetti del package sono inclusi nello schema.

Attributi delle proprietà

Quandolo schema viene configurato per accedere l'oggetto utilizzando le proprietà, tutti i metodi serializzabili che seguono le convenzione di denominazione della proprietà di JavaBeans saranno automaticamente incorporati nello schema. Per selezionare un attributo della proprietà per la query, utilizzare le convenzioni di denominazione della proprietà dello stile JavaBeans

Tutte le proprietà pubbliche, private, protette e le proprietà protette del package sono incluse nello schema.

Nella classe di seguito riportata, i seguenti attributi vengono aggiunti allo schema: name, birthday, valid.

```
public class Person {
  public String getName(){}
  private java.util.Date getBirthday(){}
  boolean isValid(){}
  public NonSerializableObject getData(){}
}
```

Quando si utilizza una CopyMode di COPY_ON_WRITE, lo schema della query deve sempre utilizzare l'accesso basato sulla proprietà. COPY_ON_WRITE crea oggetti proxy se gli oggetti vengono recuperati dalla mappa e possono accedere solo quegli oggetti che utilizzano i metodi proprietà. Così facendo, risulterà un errore in ciascun risultato di query impostato a null.

Relazioni

Ciascuna relazione deve essere esplicitamente definita nella configurazione dello schema. La cardinalità della relazione viene automaticamente determinata dal tipo dell'attributo. se l'attributo implementa l'interfaccia java.util.Collection, allora la relazione è sia di uno-a-molti che di molti-a-molti.

Diversamente dalle query dell'entità, gli attributi che fanno riferimento ad altri oggetti memorizzati nella cache non devono memorizzare i riferimenti diretti agli oggetti. I riferimenti ad altri oggetti vengono serializzati come parte contenente i dati dell'oggetto. Invece, memorizzare la chiave per l'oggetto il relativo oggetto.

Ad esempio, se esiste una relazione molti-a-uno tra Customer e Order:

Incorrect. Storing an object reference.

```
public class Customer {
  String customerId;
  Collection<Order> orders;
}

public class Order {
  String orderId;
  Customer customer;
}
```

```
Correct. The key to the related object.

public class Customer {
  String customerId;
  Collection<String> orders;
}

public class Order {
  String orderId;
  String customer;
}
```

Quando una query in esecuzione unisce due oggetti della mappa insiemi, la chiave sarà automaticamente deserializzato. Ad esempio, la query che segue restituirebbe gli oggetti Customers:

```
SELECT c FROM Order o JOIN Customer c WHERE orderId=5
```

Utilizzo degli indici

ObjectGrid utilizza i plug-in dell'indice per aggiungere gli indici alle mappe. Il motore di query incorpora automaticamente qualsiasi indice che è definito su un elemento della mappa di schema del tipo: com.ibm.websphere.objectgrid.plugins.index.HashIndex e la proprietà rangeIndex è impostata su true. Se il tipo di indice non è HashIndex e la proprietà rangeIndex non è impostata su true, allora l'indice viene ignorato dalla query. consultare il supporto didittico della query di oggetto il supporto didattico ObjectQuery in *Panoramica sul prodotto* per un esempio sul modo in cui aggiungere un indice allo schema.

API della query di EntityManager

L'API EntityManager fornisce metodi per eseguire query di dati nell'ObjectGrid memorizzata utilizzando l'API EntityManager. L'API della Query di EntityManager viene utilizzata per creare ed eseguire query su una o più entità definite in eXtreme Scale.

Query e ObjectMaps per entità

WebSphere Extended Deployment v6.1 ha introdotto una capability avanzata di query per entità memorizzate in eXtreme Scale. Queste query consentono agli oggetti di essere richiamati utilizzando attributi non chiave e di eseguire semplici aggregazioni come la somma, la media, il minimo ed il massimo considerando tutti i dati che corrispondono alla query. Le applicazioni costruiscono una query utilizzando l'API EntityManager.createQuery. Ciò restituisce un oggetto Query che può essere quindi interrogato per ottenere i risultati della query. L'oggetto query consente inoltre di essere personalizzato prima di eseguire la query. La query viene eseguita automaticamente quando viene chiamato un qualunque metodo che restituisce il risultato.

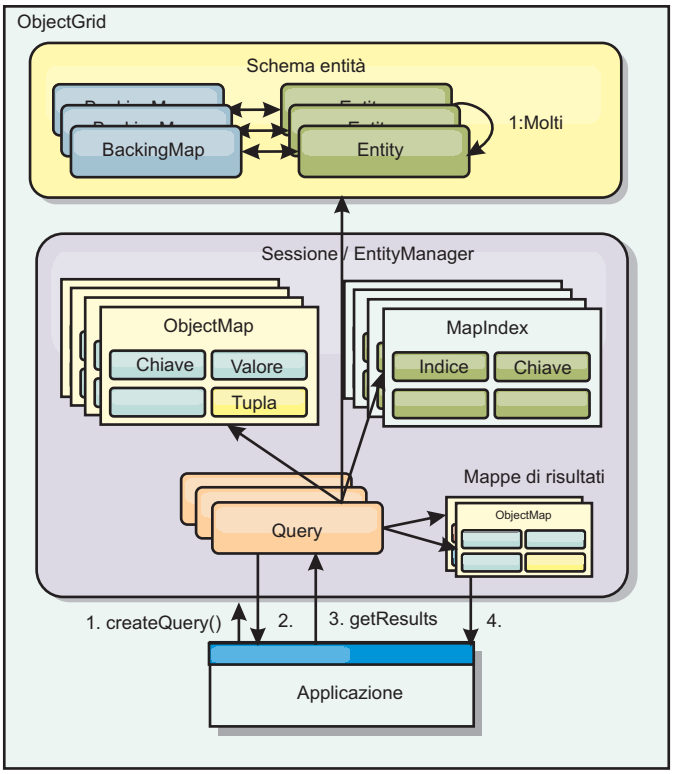


Figura 2. L'interazione della query con le mappe di oggetti ObjectGrid ed il modo in cui lo schema dell'entità viene definito ed associato con una mappa ObjectGrid.

Il richiamo della query comporta l'utilizzo del metodo getResultIterator

I risultati della query sono un elenco di attributi. Se la query era la selezione di a,b,c da X dove y=z, viene restituito un elenco di righe contenente a, b e c. Questo elenco viene memorizzato in una Mappa di ambito della transazione che vuol dire che bisogna associare una chiave artificiale ad ogni riga ed utilizzare un valore intero che incrementi con ogni riga. Questa mappa viene ottenuta utilizzando il metodo Query.getResultMap. La mappa ha EntityMetaData, che descrive ogni riga nella Mappa ad essa associata. È possibile accedere agli elementi di ogni riga utilizzando codice simile al seguente:

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");

  q.setParameter(1, "Claus");

  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Object[] row = (Object[])iter.next();
    System.out.println("Found a Claus with id "  + row[objectgrid: 0 ]
      + ", firstName: " + row[objectgrid: 1 ]
      + ", surname: " + row[objectgrid: 2 ]);
  }
```

Richiamo dei risultati della query utilizzando getResultMap

Il seguente codice mostra il richiamo di parti specifiche di clienti corrispondenti e mostra come accedere alle righe di risultato. Se si utilizza l'oggetto Query per accedere ai dati, viene nascosto l'identificativo lungo di riga generato. Il lungo è visibile solo quando si utilizza l'ObjectMap per accedere ai risultati. Una volta completata la transazione, questa Mappa sparisce. La mappa è visibile solo alla Sessione utilizzata che è, di solito, proprio il thread che l'ha creata. La Mappa

utilizza una tupla per la chiave con un attributo singolo, un lungo con l'ID riga. Il valore è un'altra tupla con un attributo per ogni colonna nella serie di risultato.

Il seguente codice di esempio dimostra ciò:

```
Query q = em.createQuery("select c.id, c.firstName, c.surname from
Customer c where c.surname=?1");
q.setParameter(1, "Claus");
ObjectMap qmap = q.getResultMap();
Tuple keyTuple = qmap.getEntityMetadata().getKeyMetadata().createTuple();
for(long i = 0; true; ++i)
{
  keyTuple.setAttribute(0, new Long(i));
  Tuple row = (Tuple)qmap.get(keyTuple);
  if(row == null) break;
  System.out.println(" I Found a Claus with id "  + row.getAttribute(0)
    + ", firstName: " + row.getAttribute(1)
    + ", surname: " + row.getAttribute(2));
}
```

Richiamo dei risultati della query utilizzando un iteratore del risultato dell'entità

Il codice di seguito riportato mostra la query ed il loop che richiama ciascuna riga di risultato utilizzando le normali API della mappa. La chiave della mappa è una tupla. Per cui, costruire una dei tipi corretti utilizzando il risultato del metodo createTuple in keyTuple. Provare a richiamare tutte le righe con rowIds da 0 in avanti. Il loop termina quando viene restituito un valore nullo (che indica il non ritrovamento della chiave). Impostare il primo attributo di keyTuple in modo che sia il lungo che si desidera trovare. Anche il valore che viene restituito da get è una tupla con un attributo per ogni colonna nel risultato della query. Quindi, estrarre ogni attributo dal valore Tupla utilizzando getAttribute.

Quello che segue è il frammento di codice successivo:

```
Query q2 = em.createQuery("select c.id, c.firstName, c.surname from Customer c where c.surname=?1");
q2.setResultEntityName("CustomerQueryResult");
q2.setParameter(1, "Claus");

Iterator iter2 = q2.getResultIterator(CustomerQueryResult.class);
while(iter2.hasNext())
{
  CustomerQueryResult row = (CustomerQueryResult)iter2.next();
  // firstName is the id not the firstName.
  System.out.println("Found a Claus with id " + row.id
    + ", firstName: " + row.firstName
    + ", surname: " + row.surname);
}

em.getTransaction().commit();
```

Specified è un valore ResultEntityName nella query. Questo valore indica al motore di query che si desidera progettare ogni riga su un oggetto specifico, in questo caso CustomerQueryResult. Segue la classe:

```
@Entity
public class CustomerQueryResult {
 @Id long rowId;
 String id;
 String firstName;
 String surname;
};
```

Nel primo frammento, notare che ogni riga di query viene restituita come un oggetto CustomerQueryResult piuttosto che come un Object[]. Le colonne del risultato della query sono progettate sull'oggetto CustomerQueryResult: La progettazione del risultato è leggermente più lenta al runtime ma più leggibile. Il

risultato della query di Entities non deve essere registrata con eXtreme Scale all'avvio. Se le entità sono registrate, viene creata una Mappa globale con lo stesso nome e la query fallisce con un errore che indica il nome duplicato della mappa.

Query semplici con EntityManager

WebSphere eXtreme Scale viene fornito con l'API della query EntityManager.

L'API della query EntityManager è molto simile ad altri motori di query SQL che eseguono query sugli oggetti. Viene definita una query e poi il risultato viene recuperato dalla query utilizzando vari metodi getResult.

I seguenti esempi si riferiscono alle entità utilizzate nel supporto didattico di EntityManager nella Panoramica sul prodotto.

Esecuzione di una query semplice

In questo esempio, viene eseguita la query sui clienti con cognome Claus:

```
em.getTransaction().begin();

  Query q = em.createQuery("select c from Customer c where c.surname='Claus'");

  Iterator iter = q.getResultIterator();
  while(iter.hasNext())
  {
    Customer c = (Customer)iter.next();
    System.out.println("Found a claus with id " + c.id);
  }

  em.getTransaction().commit();
```

Utilizzo dei parametri

Poiché si desidera trovare tutti i clienti con cognome Claus, viene utilizzato un parametro per specificare il cognome, dal momento che l'utente potrebbe voler utilizzare tale query più di una volta.

Esempio di parametro posizionale

```
Query q = em.createQuery("select c from Customer c where c.surname=?1");
  q.setParameter(1, "Claus");
```

L'utilizzo dei parametri è molto importante quando la query viene utilizzata più di una volta. EntityManager deve esaminare la stringa di query e creare un piano per la query e tale operazione è dispendiosa. Utilizzando un parametro, EntityManager memorizza nella cache il piano per la query, riducendo quindi il tempo che viene impiegato per eseguirla.

Vengono utilizzati sia i parametri posizionali sia quelli denominati:

Esempio di parametro denominato

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
  q.setParameter("name", "Claus");
```

Utilizzo di un indice per migliorare le prestazioni

Se esistono milioni di clienti, è necessario che la query precedente esegua la scansione su tutte le righe nella mappa Customer. Tale operazione non è molto

efficiente. Ma eXtreme Scale fornisce un meccanismo per la definizione di indici su singoli attributi in un'entità. La query utilizza tale indice quando opportuno, il che velocizza le query in modo significativo.

È possibile specificare gli attributi da indicizzare in modo semplice utilizzando l'annotazione @Index sull'attributo dell'entità:

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

EntityManager crea un indice ObjectGrid appropriato per l'attributo del cognome (surname) nell'entità Customer e il motore di query utilizza l'indice in modo automatico, il che riduce in modo considerevole il tempo di esecuzione della query.

Utilizzo della paginazione per migliorare le prestazioni

Se esistono milioni di clienti con cognome Claus, è improbabile che l'utente desideri visualizzare una pagina che mostra un milione di clienti. È più probabile che si desideri visualizzare 10 o 15 clienti alla volta.

I metodi setFirstResult e setMaxResults della query forniscono un aiuto in tal senso restituendo una serie secondaria dei risultati.

Esempio di paginazione

```
Query q = em.createQuery("select c from Customer c where c.surname=:name");
  q.setParameter("name", "Claus");
  // Display the first page
  q.setFirstResult=1;
  q.setMaxResults=25;
  displayPage(q.getResultIterator());

  // Display the second page
  q.setFirstResult=26;
  displayPage(q.getResultIterator());
```

Guida di riferimento per le query di eXtreme Scale

WebSphere eXtreme Scale ha il suo proprio linguaggio tramite il quale l'utente può eseguire la query dei dati.

Query ObjectGrid e clausola FROM

La clausola FROM specifica le raccolte di oggetti sui quali applicare la query. Ogni raccolta viene identificato tramite un nome di schema astratto e una variabile di identificazione, denominata variabile di intervallo o tramite una dichiarazione di membro di raccolta che identifica una relazione multivalore o singola e una variabile di identificazione.

Concettualmente, la semantica della query è di formare prima una raccolta temporanea di tuple, cui si fa riferimento come se R. Tuples fossero composti di elementi delle raccolte che vengono identificati nella clausola FROM. Ogni tupla contiene un elemento di ciascuno delle raccolte nella clausola FROM. Tutte le possibili combinazioni formate sono soggette ai vincoli imposti dalle dichiarazioni

dei membri della raccolta. Se un qualsiasi nome di schema identifica una raccolta per la quale non ci sono record nell'archivio permanente, la raccolta temporanea R è vuota.

Esempi utilizzando FROM

L'oggetto DeptBean contiene i record 10, 20 e 30. L'oggetto EmpBean contiene i record 1, 2 e 3 relativi al reparto 10 e i record 4 e 5 relativi al reparto 20. Il reparto 30 non ha i relativi dipendenti.

```
FROM DeptBean d, EmpBean e
```

Questa clausola forma una raccolta temporanea R che contiene 15 tuple.

```
FROM  DeptBean d,  DeptBean d1
```

Questa clausola forma una raccolta temporanea R che contiene 9 tuple.

```
FROM  DeptBean d,  IN (d.emps) AS e
```

Questa clasuola forma un insieme temporaneo R che contiene 5 tuple. Il reparto 30 non è incluso nella raccolta temporanea R in quanto non contiene dipendenti. Il reparto 10 è contenuto nella raccolta temporanea R tre volte e il reparto 20 è contenuto in R due volte.

Anziché utilizzare IN(d.emps) come e, è possibile utilizzare un predicato JOIN:

```
FROM DeptBean d JOIN d.emps as e
```

Dopo aver formato la raccolta temporanea, le condizioni di ricerca della clausola WHERE vengono applicata alla raccolta temporanea R, producendo una raccolta temporanea R1. Le clausole ORDER BY e SELECT vengono applicate su R1 per produrre la serie di risultati finali.

Una variabile di identificazione è una variabile che viene dichiarata nella clausola FROM utilizzando l'operatore IN o l'operatore facoltatvo AS.

```
FROM DeptBean AS d,  IN (d.emps) AS e
```

è equivalente a:

```
FROM DeptBean d, IN (d.emps) e
```

Una variabile di identificazione che viene dichiarata come nome schema astratto viene chiamata variabile intervallo. Nella query precedente, "d" è una variabile intervallo. Una variabile di identificazione che viene dichiarata come espressione di percorso con più valori viene chiamata dichiarazione del membro della raccolta. I valori "d" e "e" nel precedente esempio sono dichiarazioni del membro della raccolta.

Di seguito viene riportato un esempio di utilizzo di un'espressione di percorso con unico valore nella clausola FROM:

```
FROM EmpBean e, IN(e.dept.mgr) as m
```

Query ObjectGrid clausola SELECT

La sintassi della clausola SELECT viene illustrata nel seguente esempio:

```
SELECT { ALL | DISTINCT }  [ selection , ]*  selection
selection  ::= {single_valued_path_expression |
                identification_variable |
                OBJECT ( identification_variable) |
        aggregate_functions } [[ AS ] id ]
```

La clausola SELECT consiste di una o più dei seguenti elementi: un'unica variabile di identificazione definita nella clausola FROM, un'espressione di percorso con unico valore su riferimenti oggetto o valori e una funzione aggregata. È possibile utilizzare la parola chiave DISTINCT per eliminare i riferimenti duplicati.

Una selezione secondaria scalare è una selezione secondaria che restituisce un unico valore.

Esempi di utilizzo di SELECT

Trova tutti i dipendenti che guadagnano più del dipendente John:

```
SELECT OBJECT(e) FROM EmpBean ej, EmpBean eWHERE  ej.name = 'John'  and
e.salary > ej.salary
```

Trovare tutti i reparti che hanno uno o più dipendenti che guadagnano meno di 20000:

```
SELECT DISTINCT e.dept  FROM EmpBean e where e.salary < 20000
```

Una query può avere un'espressione di percorso che valuta su un valore arbitrario:

```
SELECT e.dept.name  FROM EmpBean e where e.salary < 20000
```

La query precedente restituisce una raccolta di valori di nomi per i reparti che hanno impiegati che guadagnano meno di 20000.

Una query può restituire un valore aggregato:

```
SELECT  avg(e.salary)  FROM EmpBean e
```

Di seguito viene riportata una query che richiama i nomi e i riferimenti oggetto per i dipendenti sottopagati:

```
SELECT e.name as name, object(e) as emp from EmpBean e where e.salary <
50000
```

Query ObjectGrid clausola WHERE

La clausola WHERE contiene condizioni di ricerca che vengono composte dagli elementi presentati di seguito. Quando una condizione di ricerca valuta su TRUE, il tuple viene aggiunto alla serie di risultati.

ObjectGrid esegue la query dei valori letterali

Un valore letterale della stringa viene racchiuso tra apici. Se ricorre un unico apice all'interno del valore letterale della stringa viene rappresentato da due apici, ad esempio: 'Tom''s'.

Un valore letterale numerico può essere uno dei seguenti valori:
- Un valore esatto come 57, -957 o +66
- Qualsiasi valore supportato dal tipo lungo Java
- Un valore letterale decimale come 57.5 o -47.02
- Un valore numerico approssimativo come 7E3 o -57.4E-2
- I tipi float devono includere il qualificatore "F", ad esempio 1.0F
- I tipi lunghi devono includere il qualificatore "L", ad esempio 123L

I valori letterali booleani sono TRUE e FALSE.

I valori letterali temporali seguono la sintassi escape JDBC basata sul tipo dell'attributo:
- java.util.Date: aaaa-mm-ss
- java.sql.Date: aaaa-mm-ss
- java.sql.Time: hh-mm-ss
- java.sql.Timestamp: aaaa-mm-gg hh:mm:ss.f...
- java.util.Calendar: aaaa-mm-gg hh:mm:ss.f...

I valori letterali enum sono espressi utilizzando la sintassi dei valori letterali enum di Java che utilizza il nome classe enum completo.

Parametri di input per la query di ObjectGrid

È possibile specificare i parametri di input utilizzando una posizione ordinale o un nome di variabile. La scrittura delle query che utilizzano i parametri di input è fortemente consigliata in quanto migliora la prestazione, consentendo all'ObjectGrid di rilevare il piano della query tra le azioni in esecuzione.

Un parametro di input può essere uno dei seguenti tipi: Byte, Short, Integer, Long, Float, Double, BigDecimal, BigInteger, String, Boolean, Char, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar, un enum Java SE 5, un oggetto POJO o un'entità o una stringa di dati binari nel modulo di byte Java [].

Un parametro di input non deve avere un valore NULL. Per la ricerca della ricorrenza di un valore NULL, utilizzare il predicato NULL.

Parametri di posizione

I parametri di input di posizione vengono definiti utilizzando il punto interrogativo seguito da un numero positivo:

```
?[positive integer].
```

I parametri di input di posizione vengono numerati a partire da 1 e corrispondono agli argomenti della query; pertanto, una query non deve contenere un parametro di input che superi il numero degli argomenti di input.

Esempio: `SELECT e FROM Employee e WHERE e.city = ?1 and e.salary >= ?2`

Parametri denominati

I parametri di input denominati vengono definiti utilizzando un nome di variabile nel formato: :[nome parametro].

Esempio: `SELECT e FROM Employee e WHERE e.city = :city and e.salary >= :salary`

Query ObjectGrid predicato BETWEEN

Il predicato BETWEEN determina se un dato valore si trova tra due altri valori specifici.

```
expression [NOT]  BETWEEN expression-2  AND expression-3
```

Esempio 1

```
e.salary BETWEEN 50000 AND 60000
```

è equivalente a:

```
e.salary >= 50000  AND e.salary <= 60000
```

Esempio 2

```
e.name NOT BETWEEN 'A' AND 'B'
```

è equivalente a:

```
e.name < 'A'  OR e.name > 'B'
```

Query ObjectGrid predicato IN

Il predicato IN confronta un valore a una serie di valori. È possibile utilizzare il predicato IN in uno dei due moduli:

```
expression [NOT] IN ( subselect )expression [NOT] IN  ( value1, value2, .... )
```

Il valore ValueN può essere un valore letterale o un parametro di input. L'espressione non può valutare su un tipo di riferimento.

Esempio 1

```
e.salary IN ( 10000, 15000 )
```

è equivalente a

```
( e.salary = 10000 OR  e.salary = 15000 )
```

Esempio 2

```
e.salary IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno = 10)
```

è equivalente a

```
e.salary = ANY   ( select  e1.salary from EmpBean e1 where e1.dept.deptno =
10)
```

Esempio 3

```
e.salary NOT IN ( select  e1.salary from EmpBean e1 where e1.dept.deptno =
10)
```

è equivalente a

```
e.salary <> ALL    ( select  e1.salary from EmpBean e1 where e1.dept.deptno
= 10)
```

Query ObjectGrid predicato LIKE

Il predicato LIKE ricerca un valore di stringa per un certo pattern.

```
string-expression   [NOT] LIKE  pattern    [ ESCAPE escape-character ]
```

Il valore del pattern è il valore letterale di una stringa o un indicatore di parametro di tipo stringa in cui il segno di sottolineatura (_) sta per qualsiasi carattere singolo e il simbolo di percentuale (%) sta per qualsiasi sequenza di caratteri, inclusa una sequenza vuota. Tutti gli altri caratteri rappresentano se stessi. Il carattere escape può essere utilizzato per la ricerca del carattere _ e %. Il carattere escape può essere specificato come valore letterale di una stringa o come un parametro di input.

Se l'espressione stringa è null, il risultato è sconosciuto.

Se sia l'espressione stringa che il pattern sono vuoti, il risultato è true.

Esempio

```
'' LIKE '' is true
'' LIKE '%' is true
e.name LIKE '12%3' is true for '123' '12993' and false for '1234'
e.name LIKE 's_me' is true for 'some' and 'same', false for 'soome'
e.name LIKE '/_foo' escape '/' is true for '_foo', false for 'afoo'
e.name LIKE '//_foo' escape '/' is true for '/afoo' and for '/bfoo'
e.name LIKE '///_foo' escape '/' is true for '/_foo' but false for '/afoo'
```

Query ObjectGrid predicato NULL

Il predicato NULL verifica i valori null.

```
{single-valued-path-expression | input_parameter} IS [NOT] NULL
```

Esempio

```
e.name IS NULL
e.dept.name IS NOT NULL
e.dept IS NOT NULL
```

Query ObjectGrid predicato raccolta EMPTY

Utilizzare il predicato d'insieme EMPTY per eseguire il test di un insieme vuoto.

Per eseguire il test se una relazione con più valori è vuota, utilizzare la seguente sintassi:

```
collection-valued-path-expression IS [NOT]  EMPTY
```

Esempio

Predicato insieme vuoto per trovare tutti i reparti che non hanno impiegati:

```
SELECT OBJECT(d) FROM DeptBean d  WHERE  d.emps IS EMPTY
```

Query ObjectGrid predicato MEMBER OF

La seguente espressione esegue il test se il riferimento oggetto specificato dall'espressione di percorso con unico valore o parametro di input è un membro dell'insieme designato. Se l'espressione di percorso con valore di insieme designa un insieme vuoto, il valore dell'espressione MEMBER OF è FALSE.

```
{ single-valued-path-expression | input_parameter } [ NOT ] MEMBER [ OF ]
collection-valued-path-expression
```

Esempio

Trova i dipendenti che non sono membri di un numero di reparto specificato:

```
SELECT OBJECT(e) FROM EmpBean e , DeptBean d
WHERE e NOT MEMBER OF d.emps AND d.deptno = ?1
```

Trova i dipendenti il cui manager è membro di un numero reparto specifico:

```
SELECT OBJECT(e) FROM EmpBean e, DeptBean d
WHERE e.dept.mgr MEMBER  OF d.emps  and d.deptno=?1
```

Query ObjectGrid predicato EXISTS

Il predicato EXISTS esegue il test per la presenza o assenza di una condizione specificata da una selezione secondaria.

```
EXISTS ( subselect )
```

Il risultato di EXISTS è true se la selezione secondaria restituisce almeno un valore, altrimenti il risultato è false.

Per negare un predicato EXISTS, far precedere il predicato con l'operatore logico NOT.

Esempio

Restituisci i reparti che hanno almeno un dipendente che guadagna più di 1000000:

```
SELECT  OBJECT(d) FROM  DeptBean d
WHERE EXISTS ( SELECT  e  FROM IN (d.emps) e WHERE  e.salary > 1000000 )
```

Restituisci reparti senza dipendenti:

```
SELECT OBJECT(d) FROM DeptBean d
WHERE NOT EXISTS  ( SELECT e FROM IN (d.emps) e)
```

È possibile inoltre riscrivere la query precedente come nel seguente esempio:

```
SELECT OBJECT(d) FROM DeptBean d WHERE SIZE(d.emps)=0
```

Query ObjectGrid clausola ORDER BY

La clausola ORDER BY specifica un ordine degli oggetti nell'insieme che ne deriva. Di seguito viene riportato un esempio:

ORDER BY [order_element ,]* order_element order_element ::={ path-expression }[ASC | DESC]

L'espressione di percorso deve specificare un campo con unico valore che è un tipo di primitiva di tipo byte, short, int, long, float, double, char o un tipo wrapper di Byte Short, Integer, Long, Float, Double, BigDecimal, String, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp e java.util.Calendar. L'elemento di ordine ASC specifica che i risultati vengono visualizzati in ordine ascendente, che è l'impostazione predefinita. Un elemento di ordine DESC specifica che i risultati vengono visualizzati in ordine discendente.

Esempio

Restituire gli oggetti del reparto. Visualizzare i numeri del reparto in ordine decrescente:

```
SELECT OBJECT(d) FROM DeptBean d ORDER BY d.deptno DESC
```

Restituisce gli oggetti dipendenti, ordinati per nome e numero reparto:

```
SELECT OBJECT(e) FROM EmpBean e ORDER BY e.dept.deptno ASC, e.name DESC
```

Funzioni di aggregazione query ObjectGrid

Le funzioni di aggregazione agiscono su una serie di valori per restituire un singolo valore scalare. È possibile utilizzare queste funzioni nei metodi select e subselect. Il seguente esempio illustra un'aggregazione:

```
SELECT SUM (e.salary) FROM EmpBean e WHERE e.dept.deptno =20
```

Questa aggregazione computa il salario totale per il reparto 20.

Le funzioni di aggregazione sono: AVG, COUNT, MAX, MIN e SUM. La sintassi di una funzione di aggregazione viene mostrata nel seguente esempio:

```
aggregation-function ( [ ALL | DISTINCT ] expression )
```

o:

```
COUNT( [ ALL | DISTINCT ] identification-variable )
```

l'opzione DISTINCT elimina i valori duplicati prima di applicare la funzione. L'opzione ALL è l'opzione predefinita e non elimina i valori duplicati. I valori null vengono ignorati nel computo della funzione di aggregazione tranne quando si utilizza la funzione COUNT(variabile di identificazione), che restituisce un conteggio di tutti gli elementi nella serie.

Definizione tipo di ritorno

Le funzioni MAX e MIN possono essere applicate a qualsiasi stringa numerica, o tipo di dati data-ora e restituire il tipo di dati corrispondente. Le funzioni SUM e AVG assumono come input un tipo numerico. La funzione AVG restituisce un tipo

doppio. La funzione SUM restituisce un tipo lungo se il tipo di input è un tipo intero, tranne quando l'input è un tipo Java BigInteger, in tal caso la funzione restituisce un tipo Java BigInteger. La funzione SUM restituisce un tipo doppio se il tipo di input non è un tipo intero, tranne quando l'input è un tipo Java BigDecimal, in tal caso la funzione restituisce un tipo Java BigDecimal. La funzione COUNT può prendere qualsiasi tipo di dati tranne gli insiemi e restituisce un tipo lungo.

Quando applicato a una serie vuota, le funzioni SUM, AVG, MAX e MIN possono restituire un valore null. La funzione COUNT restituisce zero (0) quando viene applicata a una serie vuota.

Utilizzo delle clausole GROUP BY e HAVING

La serie di valori che viene utilizzata per la funzione di aggregazione viene determinata dall'insieme che risulta dalla clausola FROM e WHERE della query. È possibile dividere la serie in gruppi e applicare la funzione di aggregazione ad ogni gruppo. Per eseguire questa azione, utilizzare una clausola GROUP BY nella query. La clausola GROUP BY definisce i membri del raggruppamento, che comprende un elenco di espressioni di percorso. Ogni espressione di percorso specifica un campo che è un tipo di primitiva di byte, short, int, long, float, double, boolean, char o un tipo wrapper di Byte, Short, Integer, Long, Float, Double, BigDecimal, String, Boolean, Character, java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar o un enum Java SE 5.

Il seguente esempio illustra l'utilizzo della clausola GROUP BY in una query che computa il salario medio per ogni reparto:

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e GROUP BY e.dept.deptno
```

Nella divisione di una serie in gruppi, un valore NULL viene considerato uguale a un altro valore NULL.

I gruppi possono essere filtrati utilizzando una clausola HAVING che esegua il test delle proprietà del gruppo prima di coinvolgere le funzioni di aggregazione o i membri del raggruppamento. Il filtro è simile al modo in cui la clausola WHERE esegue la tupla (ossia, record dei valori di insieme restituiti) dalla clausola FROM. Di seguito viene riportato un esempio della clausola HAVING:

```
SELECT e.dept.deptno, AVG ( e.salary) FROM EmpBean e
GROUP BY e.dept.deptno
HAVING COUNT(e) > 3 AND e.dept.deptno > 5
```

Questa query restituisce il salario medio per i reparti che hanno più di tre dipendenti e il numero di reparto è maggiore di cinque.

È possibile utilizzare una clausola HAVING senza una clausola GROUP BY. In tal caso, l'intera serie viene trattata come un singolo gruppo, al quale viene applicata la clausola HAVING.

BNF (Backus-Naur Form) query ObjectGrid

Di seguito è riportato un riepilogo della notazione BNF (Backus-Naur Form) della query ObjectGrid.

Tabella 3. Riepilogo da chiave a BNF

Rappresentazione	Descrizione
{...}	Raggruppamento

Tabella 3. Riepilogo da chiave a BNF (Continua)

Rappresentazione	Descrizione
[...]	Costrutti facoltativi
grassetto	Parole chiave
*	Zero o più
|	Alternative

```
ObjectGrid QL ::=select_clause from_clause [where_clause] [group_by_clause]
  [having_clause] [order_by_clause]

from_clause ::=FROM identification_variable_declaration
  [,identification_variable_declaration]*

identification_variable_declaration ::=collection_member_declaration |
  range_variable_declaration

collection_member_declaration ::=IN ( collection_valued_path_expression |
  single_valued_navigation) [AS] identifier | [LEFT [OUTER]
  | INNER] JOIN collection_valued_path_expression |
  single_valued_navigation [AS] identifier

range_variable_declaration ::=abstract_schema_name [AS] identifier

single_valued_path_expression ::={single_valued_navigation | identification_variable}.
  { state_field | state_field.value_object_attribute } | single_valued_navigation

single_valued_navigation ::=identification_variable.[ single_valued_association_field. ]*
  single_valued_association_field

collection_valued_path_expression ::=identification_variable.[
  single_valued_association_field. ]* collection_valued_association_field

select_clause ::= SELECT [DISTINCT] [ selection , ]* selection

selection ::= {single_valued_path_expression |identification_variable | OBJECT
  ( identification_variable) |aggregate_functions } [[ AS ] id ]

order_by_clause ::= ORDER BY [ {identification_variable.[ single_valued_association_field.
  ]*state_field} [ASC|DESC],]* {identification_variable.[
  single_valued_association_field. ]*state_field}[ASC|DESC]

where_clause ::= WHERE conditional_expression

conditional_expression ::= conditional_term | conditional_expression OR conditional_term

conditional_term ::= conditional_factor | conditional_term AND conditional_factor

conditional_factor ::= [NOT] conditional_primary

conditional_primary ::= simple_cond_expression | (conditional_expression)

simple_cond_expression ::= comparison_expression | between_expression | like_expression |
  in_expression | null_comparison_expression | empty_collection_comparison_expression |
  exists_expression | collection_member_expression

between_expression ::= numeric_expression [NOT] BETWEEN numeric_expression
  AND numeric_expression | string_expression [NOT] BETWEEN
  string_expression AND string_expression | datetime_expression [NOT]
  BETWEEN datetime_expression AND datetime_expression

in_expression ::= identification_variable.[ single_valued_association_field. ]state_field
  [*NOT] IN { (subselect) | ( atom ,]* atom) }

atom ::= { string_literal | numeric_literal | input_parameter }

like_expression ::=string_expression [NOT] LIKE {string_literal | input_parameter}
  [ESCAPE {string_literal | input_parameter}]

null_comparison_expression ::= {single_valued_path_expression | input_parameter} IS
  [ NOT ] NULL

empty_collection_comparison_expression ::= collection_valued_path_expression IS
  [NOT] EMPTY

collection_member_expression ::={ ssingle_valued_path_expression | input_parameter }[
  NOT ] MEMBER [ OF ]collection_valued_path_expression

exists_expression ::= EXISTS {(subselect)}

subselect ::= SELECT [{ ALL | DISTINCT }] subselection from_clause
  [where_clause] [group_by_clause] [having_clause]

subselection ::= {single_valued_path_expression |identification_variable |
  aggregate_functions }

group_by_clause ::= GROUP BY[single_valued_path_expression,]*
  single_valued_path_expression
```

```
having_clause ::= HAVING conditional_expression

comparison_expression ::= numeric_ expression comparison_operator { numeric_expression
  | {SOME | ANY | ALL} (subselect) } | string_expression
  comparison_operator {

string_expression | {SOME | ANY | ALL}(subselect) } |

datetime_expression comparison_operator {

datetime_expression {SOME | ANY | ALL}(subselect) } |

boolean_expression {=|<>} {

boolean_expression {SOME | ANY | ALL}(subselect) } |

entity_expression {=|<>} {

entity_expression {SOME| ANY | ALL}(subselect) }

comparison_operator ::= = | > | >= | < | <= | <>

string_expression ::= string_primary | (subselect)

string_primary ::=state_field_path_expression |string_literal | input_parameter |
  functions_returning_strings

datetime_expression ::= datetime_primary |(subselect)

datetime_primary ::=state_field_path_expression | string_literal | long_literal
  | input_parameter | functions_returning_datetime

boolean_expression ::= boolean_primary |(subselect)

boolean_primary ::=state_field_path_expression | boolean_literal | input_parameter

entity_expression ::=single_valued_association_path_expression |
  identification_variable | input_parameter

numeric_expression ::= simple_numeric_expression |(subselect)

simple_numeric_expression ::= numeric_term | numeric_expression {+|-} numeric_term

numeric_term ::= numeric_factor | numeric_term {*|/} numeric_factor

numeric_factor ::= {+|-} numeric_primary

numeric_primary ::= single_valued_path_expression | numeric_literal |
  ( numeric_expression ) | input_parameter | functions

aggregate_functions :=

AVG([ALL|DISTINCT] identification_variable.
  [ single_valued_association_field. ]*state_field) |

COUNT([ALL|DISTINCT] {single_valued_path_expression |
  identification_variable}) |

MAX([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field) |

MIN([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field) |

SUM([ALL|DISTINCT] identification_variable.[
  single_valued_association_field. ]*state_field)

functions ::=

ABS (simple_numeric_expression) |

CONCAT (string_primary , string_primary) |

LOWER (string_primary) |

LENGTH(string_primary) |

LOCATE(string_primary, string_primary [, simple_numeric_expression]) |

MOD (simple_numeric_expression, simple_numeric_expression) |

SIZE (collection_valued_path_expression) |

SQRT (simple_numeric_expression) |

SUBSTRING (string_primary, simple_numeric_expression[, simple_numeric_expression]) |

UPPER (string_primary) |

TRIM ([[LEADING | TRAILING | BOTH] [trim_character]
  FROM] string_primary)
```

Ottimizzazione delle prestazioni della query

Per ottimizzare le prestazioni della propria query, utilizzare le seguenti tecniche e suggerimenti.

Utilizzo dei parametri

Quando si esegue una query, la stringa di query dove essere analizzata e si dove sviluppare un piano per eseguirla, entrambe le cose potrebbero essere dispendiose. WebSphere eXtreme Scale memorizza nella cache i piani di query vicino alla stringa di query. Poiché la cache non ha una dimensione infinita, è importante riutilizzare, ove possibile, le stringhe di query. L'utilizzo di parametri denominati e posizionali migliora le prestazioni adottando il riutilizzo dei piani di query.

```
Esempio di parametro posizionale  Query q = em.createQuery("select c from
Customer c where c.surname=?1");   q.setParameter(1, "Claus");
```

Utilizzo degli indici

Un'adeguata indicizzazione su una mappa può avere un impatto significativo sulle prestazioni della query, per quanto essa rappresenti un sovraccarico rispetto alle prestazioni generali di una mappa. Senza l'indicizzazione degli attributi degli oggetti coinvolti nelle query, il motore di query esegue la scansione della tabella per ciascun attributo. La scansione della tabella rappresenta l'operazione più dispendiosa durante l'esecuzione di una query. L'indicizzazione degli attributi degli oggetti coinvolti nelle query consente al motore di query di evitare una scansione non necessaria della tabella, migliorando così le prestazioni generali della query. Se l'applicazione è progettata per utilizzare in modo intensivo le query su una mappa prevalentemente in lettura, è bene configurare gli indici per gli attributi degli oggetti coinvolti nella query. Se la mappa viene prevalentemente aggiornata, si dovrà trovare un equilibrio tra il miglioramento delle prestazioni della query ed il sovraccarico di indicizzazione sulla mappa. Consultare Indicizzazione per ulteriori informazioni.

Quando nella mappa si memorizzano oggetti POJO (Java object), un'adeguata indicizzazione può evitare una reflection di Java. Nel seguente esempio, se sul campo budget è stato creato un indice, la query sostituisce la clausola WHERE con una ricerca di indice d'intervallo. Altrimenti la query effettua la scansione dell'intera mappa valutando le clausole WHERE, ottenendo prima il budget utilizzando la reflection di Java e quindi confrontandolo con il valore 50000:

```
SELECT d FROM DeptBean d WHERE d.budget=50000
```

Consultare “Piano di query” a pagina 121 per i dettagli su come sintonizzare al meglio una singola query e come una diversa sintassi, i modelli di oggetto e gli indici possano influenzare le prestazioni di una query.

Utilizzo della paginazione

In ambienti client-server, il motore di query trasporta l'intera mappa di risultato al client. I dati restituiti devono essere divisi in blocchi ragionevoli. Entrambe le interfacce EntityManager Query e ObjectMap ObjectQuery supportano i metodi setFirstResult e setMaxResults che consentono alla query di restituire un sottoinsieme dei risultati.

Restituzione di valori primitivi anziché delle entità

Con l'API EntityManager Query, le entità sono restituite come parametri della query. Il motore di query attualmente restituisce al client la chiave per queste entità. Quando il client effettua l'iterazione su queste entità utilizzando l'iteratore del metodo getResultIterator, ciascuna entità viene deserializzata automaticamente

e gestita come se fosse creata con il metodo find dell'interfaccia EntityManager. L'intero grafico dell'entità viene creato dall'entità ObjectMap sul client. Gli attributi del valore entità e qualsiasi entità relativa sono dettagliatamente risolti.

Per evitare di effettuare il dispendioso build del grafico, modificare la query in modo che restituisca gli attributi singoli con il percorso di navigazione.

Ad esempio:

```
// Returns an entity
SELECT p FROM Person p
// Returns attributes SELECT p.name, p.address.street, p.address.city, p.gender FROM Person p
```

Piano di query

Tutte le query di eXtreme Scale hanno un piano di query. Il piano descrive come il motore di query interagirà con le ObjectMap e gli indici. Visualizzare il piano di query per determinare se la stringa di query o gli indici vengono utilizzati correttamente. Il piano di query può essere utilizzato anche per provare le differenze che le minime modifiche in una stringa di query apportano nel modo in cui eXtreme Scale esegue una query.

Il piano di query può essere visualizzato in due modi:

- Metodi API getPlan Query o ObjectQuery di EntityManager
- Traccia di diagnosi ObjectGrid

Metodo getPlan

Il metodo getPlan sulle interfacce ObjectQuery e Query restituisce una stringa che descrive il piano di query. Questa stringa può essere visualizzata sull'output standard oppure in un log in modo da visualizzare il piano di query. Nota: in un ambiente distribuito, il metodo getPlan non viene eseguito sul server e non rifletterà alcun indice definito. Per visualizzare il piano su un server utilizzare un agent.

Traccia del piano di query

Il piano di query può essere visualizzato utilizzando la traccia ObjectGrid. Per abilitare il piano di query, utilizzare le seguenti specifiche di traccia:

QueryEnginePlan=debug=enabled

Consultare “Log e traccia” a pagina 383 per dettagli su come abilitare la traccia e ubicare i file di log di traccia.

Esempi di piano di query

Il piano di query utilizza la parola 'for' per indicare che la query è in iterazione su una raccolta ObjectMap o su una raccolta derivata quale: q2.getEmps(), q2.dept o una raccolta temporanea restituita da un loop interno. Se la raccolta proviene da un ObjectMap, il piano di query mostra se viene utilizzato un indice a scansione sequenziale (indicato da INDEX SCAN), univoco o non univoco. Il piano di query utilizza una stringa di filtro per elencare le espressioni di condizione applicate alla raccolta.

Solitamente, in una query di oggetto, non si utilizza un prodotto cartesiano. La seguente query esegue la scansione dell'intera mappa EmpBean nel loop esterno e quella della mappa DeptBean nel loop interno:

```
SELECT e, d FROM EmpBean e, DeptBean d

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in DeptBean ObjectMap using INDEX SCAN
   returning new Tuple( q2,  q3  )
```

La seguente query recupera tutti i nomi degli impiegati di un determinato reparto eseguendo una scansione sequenziale della mappa EmpBean in modo da ottenere un oggetto employee. Partendo dall'oggetto employee, la query naviga fino al suo relativo oggetto department ed applica il filtro d.no=1. In questo esempio, ciascun impiegato ha un unico riferimento ad un oggetto department, quindi il loop interno viene eseguito una sola volta:

```
SELECT e.name FROM EmpBean e JOIN e.dept d WHERE d.no=1

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in  q2.dept
      filter ( q3.getNo() = 1 )
   returning new Tuple( q2.name  )
```

La seguente query è equivalente alla precedente, ma ha migliori prestazioni poiché restringe i risultati ad un unico oggetto department utilizzando un indice univoco definito sul numero di campo della chiave primaria DeptBean. Dall'oggetto department la query su sposta sui relativi oggetti employee per ottenerne i nomi:

```
SELECT e.name FROM DeptBean d JOIN d.emps e WHERE d.no=1

Plan trace:

for q2 in DeptBean ObjectMap using UNIQUE INDEX key=(1)
     for q3 in  q2.getEmps()
   returning new Tuple( q3.name  )
```

La seguente query trova tutti gli impiegati che lavorano per i reparti sviluppo (Development) o vendite (Sales). La query esegue la scansione dell'intera mappa EmpBean ed esegue ulteriori filtri valutando le espressioni: d.name = 'Sales' o d.name='Dev'

```
SELECT e FROM EmpBean e, in (e.dept) d WHERE d.name = 'Sales'
 or d.name='Dev'

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
     for q3 in  q2.dept
      filter (( q3.getName() = Sales ) OR ( q3.getName() = Dev ) )
   returning new Tuple( q2  )
```

La seguente query è equivalente a quella precedente, ma esegue un piano di query differente ed utilizza l'indice d'intervallo creato sul nome campo. In generale questa query ha prestazioni migliori poiché l'indice sul campo nome viene utilizzato per restringere gli oggetti department, con un esecuzione veloce se solo pochi tra i reparti sono di sviluppo (Development) o vendite (Sales).

```
SELECT e FROM DeptBean d, in(d.emps) e WHERE d.name='Dev' or d.name='Sales'

Plan trace:

IteratorUnionIndex of

     for q2 in DeptBean ObjectMap using INDEX on name = (Dev)
```

```
       for q3 in  q2.getEmps()


     for q2 in DeptBean ObjectMap using INDEX on name = (Sales)
       for q3 in  q2.getEmps()
```

La seguente query trova i reparti che non hanno impiegati:

```
SELECT d FROM DeptBean d WHERE NOT EXISTS(select e from d.emps e)

Plan trace:

for q2 in DeptBean ObjectMap using INDEX SCAN
    filter ( NOT  EXISTS (    correlated collection defined as

       for q3 in  q2.getEmps()
       returning new Tuple( q3      )

   returning new Tuple( q2  )
```

La seguente query è equivalente alla precedente, ma utilizza la funzione scalare SIZE. Questa query ha prestazioni simili, ma è più semplice da scrivere.

```
SELECT d FROM DeptBean d WHERE SIZE(d.emps)=0
for q2 in DeptBean ObjectMap using INDEX SCAN
    filter (SIZE( q2.getEmps()) = 0 )
   returning new Tuple( q2  )
```

Il seguente esempio rappresenta un modo diverso di scrivere la stessa query con prestazioni simili, ma più semplice da scrivere:

```
SELECT d FROM DeptBean d WHERE d.emps is EMPTY

Plan trace:

for q2 in DeptBean ObjectMap using INDEX SCAN
    filter ( q2.getEmps() IS EMPTY  )
   returning new Tuple( q2  )
```

La seguente query trova qualsiasi impiegato con un domicilio che corrisponde ad almeno uno degli indirizzi dell'impiegato il cui nome equivale al valore del parametro. Il loop interno non ha alcuna dipendenza sul loop esterno. La query esegue il loop interno un'unica volta.

```
SELECT e FROM EmpBean e WHERE e.home =  any (SELECT e1.home FROM EmpBean e1
 WHERE e1.name=?1)
for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( q2.home =ANY     temp collection defined as

       for q3 in EmpBean ObjectMap using INDEX on name = ( ?1)
       returning new Tuple( q3.home      )
 )
   returning new Tuple( q2  )
```

La seguente query è equivalente alla precedente, ma ha una sottoquery correlata: inoltre il loop interno viene eseguito ripetutamente.

```
SELECT e FROM EmpBean e WHERE EXISTS(SELECT e1 FROM EmpBean e1 WHERE
 e.home=e1.home and e1.name=?1)

Plan trace:

for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( EXISTS (    correlated collection defined as

       for q3 in EmpBean ObjectMap using INDEX on name = (?1)
```

```
        filter ( q2.home =  q3.home )
       returning new Tuple( q3      )

  returning new Tuple( q2  )
```

Ottimizzazione della query mediante l'uso di indici.

Una definizione ed un utilizzo appropriato degli indici può incrementare significativamente le prestazioni di una query.

Le query di WebSphere eXtreme Scale possono utilizzare plug-in HashIndex incorporati per migliorare le prestazioni delle query. Gli indici possono essere definiti sulle entità o sugli attributi dell'oggetto. Il motore di query utilizzerà automaticamente gli indici definiti se la sua clausola WHERE utilizza una delle seguenti stringhe:

- Un'espressione di confronto con i seguenti operatori: =, <, >, <= o >= (qualsiasi espressione di confronto eccetto "non uguale" <>)
- Un espressione BETWEEN
- Operandi di espressioni che siano costanti o termini semplici

Requisiti

Gli indici hanno i seguenti requisiti quando sono utilizzati da Query:

- Tutti gli indici devono utilizzare il plug-in HashIndex incorporato.
- Tutti gli indici devono essere definiti staticamente. Gli indici dinamici non sono supportati.
- È possibile utilizzare l'annotazione @Index per creare automaticamente plug-in HashIndex statici.
- Tutti gli indici ad attributo singolo devono avere la proprietà RangeIndex impostata su true.
- Tutti gli indici composti devono avere la proprietà RangeIndex impostata su false.
- Tutti gli indici di associazione (relazione) devono avere la proprietà RangeIndex impostata su false.

Per informazioni sulla configurazione di HashIndex, fare riferimento a Configurazione di HashIndex.

Per informazioni relative all'indicizzazione, consultare Indicizzazione.

Per una ricerca più efficiente degli oggetti memorizzati nella cache, consultare “HashIndex composto” a pagina 246

Utilizzo degli hint nella scelta di un indice

Un indice può essere selezionato manualmente tramite l'utilizzo del metodo setHint sulle interfacce Query ed ObjectQuery con la costante HINT_USEINDEX. Ciò può essere utile per ottimizzare una query in modo che utilizzi l'indice con le prestazioni migliori.

Esempi di query che utilizzano gli indici di attributo

Gli esempi che seguono utilizzano termini semplici: e.empid, e.name, e.salary, d.name, d.budget ed e.isManager. Questi esempi presumono che siano stati definiti

degli indici sui campi name, salary e budget di un'entità o di un oggetto valore. Il campo empid rappresenta una chiave primaria e isManager non ha alcun indice definito.

La seguente query utilizza entrambi gli indici sui campi name e salary. Restituisce tutti gli impiegati con nomi uguali al valore del primo parametro o con salario uguale al valore del secondo parametro.

```
SELECT e FROM EmpBean e where e.name=?1 or e.salary=?2
```

La seguente query utilizza entrambi gli indici sui campi name e budget. La query restituisce tutti i reparti denominati 'DEV' con un budget superiore a 2000.

```
SELECT d FROM DeptBean dwhere d.name='DEV' and d.budget>2000
```

La seguente query restituisce tutti gli impiegati con un salario superiore a 3000 e con un valore indicatore isManager uguale al valore del parametro. La query utilizza l'indice definito sul campo salary ed esegue ulteriori operazioni di filtro valutando l'espressione di confronto: `e.isManager=?1.`

```
SELECT e FROM EmpBean e where e.salary>3000 and e.isManager=?1
```

La seguente query trova tutti gli impiegati che guadagnano più del primo parametro o quelli che sono manager. Per quanto il campo salary abbia un indice definito, la query effettua la scansione dell'indice incorporato sulle chiavi primarie del campo EmpBean e valuta l'espressione: `e.salary>?1` o `e.isManager=TRUE`.

```
SELECT e FROM EmpBean e WHERE e.salary>?1 or e.isManager=TRUE
```

La seguente query restituisce gli impiegati con un nome che contenga la lettera a. Per quanto il campo name abbia un indice definito, la query non lo utilizza perché il campo name viene utilizzato nell'espressione LIKE.

```
SELECT e FROM EmpBean e WHERE e.name LIKE '%a%'
```

La seguente query trova tutti gli impiegati con un nome che non sia "Smith". Per quanto il campo name abbia un indice definito, la query non lo utilizza perché essa utilizza l'operatore di confronto "non uguale" (<>).

```
SELECT e FROM EmpBean e where e.name<>'Smith'
```

La seguente query trova tutti i reparti con un budget minore del valore del parametro e con un impiegato che abbia un salario superiore a 3000. La query utilizza un indice per il salario, ma non per il budget poiché dept.budget non è un termine semplice. Gli oggetti dept derivano dalla raccolta e. Non è necessario utilizzare l'indice di budget per cercare oggetti dept.

```
SELECT dept from EmpBean e, in (e.dept) dept where e.salary>3000 and
dept.budget<?
```

La seguente query trova tutti gli impiegati con un salario superiore a quello degli impiegati che hanno un empid pari a 1, 2 e 3. L'indice di salary non viene utilizzato poiché il confronto coinvolge una sottoquery. La empid è una chiave primaria e viene utilizzata per un'unica ricerca d'indice poiché tutte le chiavi primarie hanno definito un indice incorporato.

```
SELECT e FROM EmpBean e WHERE e.salary > ALL (SELECT e1.salary FROM EmpBean
e1 WHERE e1.empid=1 or e1.empid=2 or e1.empid=99)
```

Per verificare se un indice è utilizzato da una query, è possibile visualizzare "Piano di query" a pagina 121. Di seguito viene riportato un piano di query di esempio per la query precedente:

```
for q2 in EmpBean ObjectMap using INDEX SCAN
    filter ( q2.salary >ALL     temp collection defined as
       IteratorUnionIndex of

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(1)
        )

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(2)
        )

         for q3 in EmpBean ObjectMap using UNIQUE INDEX key=(99)
        )
       returning new Tuple( q3.salary )
   returning new Tuple( q2  )

for q2 in EmpBean ObjectMap using RANGE INDEX on salary with range(3000,)
     for q3 in  q2.dept
      filter ( q3.budget <  ?1 )
   returning new Tuple( q3  )
```

Indicizzazione degli attributi

Gli indici possono essere definiti su qualsiasi tipo di attributo con i vincoli definiti in precedenza.

Definizione degli indici di entità con @Index

Per definire un indice su un'entità, bisogna semplicemente definire un'annotazione:

Entities using annotations

```
  @Entity
  public class Employee {
  @Id int empid;
  @Index String name
  @Index double salary
  @ManyToOne Department dept;
}
@Entity
  public class Department {
  @Id int deptid;
  @Index String name;
  @Index double budget;
  boolean isManager;
  @OneToMany Collection<Employee> employees;
  }
```

Con XML

Gli indici possono anche essere definiti utilizzando XML:

Entities without annotations

```
  public class Employee {
  int empid;
  String name
  double salary
  Department dept;
```

```
}

public class Department {
int deptid;
String name;
double budget;
boolean isManager;
Collection employees;
}
```

ObjectGrid XML with attribute indexes

```
<?xml version="1.0" encoding="UTF-8"?>
  <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
  <objectGrid name="DepartmentGrid" entityMetadataXMLFile="entity.xml">
  <backingMap name="Employee" pluginCollectionRef="Emp"/>
  <backingMap name="Department" pluginCollectionRef="Dept"/>
  </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
  <backingMapPluginCollection id="Emp">
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Employee.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Employee.salary"/>
  <property name="AttributeName" type="java.lang.String" value="salary"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  </backingMapPluginCollection>
  <backingMapPluginCollection id="Dept">
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Department.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Department.budget"/>
  <property name="AttributeName" type="java.lang.String" value="budget"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
  </objectGridConfig>
```

Entity XML

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

  <description>Department entities</description>
  <entity class-name="acme.Employee" name="Employee" access="FIELD">
  <attributes>
  <id name="empid" />
  <basic name="name" />
  <basic name="salary" />
  <many-to-one name="department"
  target-entity="acme.Department"
  fetch="EAGER">
  <cascade><cascade-persist/></cascade>
  </many-to-one>
  </attributes>
  </entity>
  <entity class-name="acme.Department" name="Department" access="FIELD">
  <attributes>
  <id name="deptid" />
  <basic name="name" />
  <basic name="budget" />
  <basic name="isManager" />
  <one-to-many name="employees"
  target-entity="acme.Employee"
  fetch="LAZY" mapped-by="parentNode">
  <cascade><cascade-persist/></cascade>
  </one-to-many>
  </attributes>
  </entity>
  </entity-mappings>
```

Definizione di indici per non-entità con XML

Gli indici per i tipi non-entità sono definiti in XML. Non vi sono differenze nella creazione di MapIndexPlugin per mappe di entità e per mappe di non-entità.

Java bean

```
public class Employee {
  int empid;
  String name
  double salary
  Department dept;

 public class Department {
  int deptid;
  String name;
  double budget;
  boolean isManager;
  Collection employees;
  }
```

ObjectGrid XML with attribute indexes

```
<?xml version="1.0" encoding="UTF-8"?>
  <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
  <objectGrid name="DepartmentGrid">
  <backingMap name="Employee" pluginCollectionRef="Emp"/>
  <backingMap name="Department" pluginCollectionRef="Dept"/>
  <querySchema>
  <mapSchemas>
  <mapSchema mapName="Employee" valueClass="acme.Employee"
  primaryKeyField="empid" />
  <mapSchema mapName="Department" valueClass="acme.Department"
  primaryKeyField="deptid" />
  </mapSchemas>
  <relationships>
  <relationship source="acme.Employee"
  target="acme.Department"
  relationField="dept" invRelationField="employees" />
  </relationships>
  </querySchema>
  </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
  <backingMapPluginCollection id="Emp">
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Employee.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Employee.salary"/>
  <property name="AttributeName" type="java.lang.String" value="salary"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  </backingMapPluginCollection>
  <backingMapPluginCollection id="Dept">
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Department.name"/>
  <property name="AttributeName" type="java.lang.String" value="name"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="Department.budget"/>
  <property name="AttributeName" type="java.lang.String" value="budget"/>
  <property name="RangeIndex" type="boolean" value="true"
  description="Ranges are must be set to true for attributes." />
  </bean>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
  </objectGridConfig>
```

Indicizzazione delle relazioni

WebSphere eXtreme Scale memorizza le chiavi esterne per le entità relative con l'oggetto parent. Per le entità, le chiavi sono memorizzate nella tupla sottostante. Per gli oggetti non-entità, le chiavi sono esplicitamente memorizzate nell'oggetto parent.

L'aggiunta di un indice su un attributo di relazione può velocizzare la query che utilizza riferimenti ciclici o che utilizza i filtri di query IS NULL, IS EMPTY, SIZE e MEMBER OF. Entrambe le associazioni a valore singolo e multiplo possono avere la configurazione dell'annotazione @Index o del plug-in HashIndex in un file XML descrittore di ObjectGrid.

Definizione di indici di relazione di entità con @Index

Il seguente esempio definisce entità con annotazioni @Index:

Entity with annotation

```
@Entity
public class Node {
    @ManyToOne @Index
    Node parentNode;

    @OneToMany @Index
    List<Node> childrenNodes = new ArrayList();

    @OneToMany @Index
    List<BusinessUnitType> businessUnitTypes = new ArrayList();
}
```

Definizione di indici di relazione di entità con XML

Il seguente esempio definisce le stesse entità ed indici utilizzando XML con i plug-in HashIndex:

Entity without annotations

```
public class Node {
int nodeId;
Node parentNode;
List<Node> childrenNodes = new ArrayList();
List<BusinessUnitType> businessUnitTypes = new ArrayList();
}
```

ObjectGrid XML

```
<?xml version="1.0" encoding="UTF-8"?>
  <objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
  <objectGrid name="ObjectGrid_Entity" entityMetadataXMLFile="entity.xml>
  <backingMap name="Node" pluginCollectionRef="Node"/>
  <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
  </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
  <backingMapPluginCollection id="Node">
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="parentNode"/>
  <property name="AttributeName" type="java.lang.String" value="parentNode"/>
<property name="RangeIndex" type="boolean" value="false"
  description="Ranges are not supported for association indexes." />  </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
  <property name="Name" type="java.lang.String" value="businessUnitType"/>
  <property name="AttributeName" type="java.lang.String" value="businessUnitTypes"/>

<property name="RangeIndex" type="boolean" value="false"
  description="Ranges are not supported for association indexes." />
 </bean>
  <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
```

```
  <property name="Name" type="java.lang.String" value="childrenNodes"/>
  <property name="AttributeName" type="java.lang.String" value="childrenNodes"/>
<property name="RangeIndex" type="boolean" value="false"
  description="Ranges are not supported for association indexes." />
  </bean>
  </backingMapPluginCollection>
  </backingMapPluginCollections>
  </objectGridConfig>
```

Entity XML

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

  <description>My entities</description>
  <entity class-name="acme.Node" name="Account" access="FIELD">
  <attributes>
  <id name="nodeId" />
  <one-to-many name="childrenNodes"
  target-entity="acme.Node"
  fetch="EAGER" mapped-by="parentNode">
  <cascade><cascade-all/></cascade>
  </one-to-many>
  <many-to-one name="parentNodes"
  target-entity="acme.Node"
  fetch="LAZY" mapped-by="childrenNodes">
  <cascade><cascade-none/></cascade>
  </one-to-many>
  <many-to-one name="businessUnitTypes"
  target-entity="acme.BusinessUnitType"
  fetch="EAGER">
  <cascade><cascade-persist/></cascade>
  </many-to-one>
</attributes>
  </entity>
  <entity class-name="acme.BusinessUnitType" name="BusinessUnitType" access="FIELD">
  <attributes>
  <id name="buId" />
  <basic name="TypeDescription" />
  </attributes>
  </entity>
  </entity-mappings>
```

Utilizzando gli indici precedentemente definiti si ottimizzano i seguenti esempi di query di entità:

```
SELECT n FROM Node n WHERE n.parentNode is null
SELECT n FROM Node n WHERE n.businessUnitTypes is EMPTY
  SELECT n FROM Node n WHERE size(n.businessUnitTypes)>=10
  SELECT n FROM BusinessUnitType b, Node n WHERE b member of n.businessUnitTypes and b.name='TELECOM'
```

Definizione di indici di relazione di non-entità

L'esempio seguente definisce un plug-in HashIndex per mappe non entità in un file XML descrittore ObjectGrid:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="ObjectGrid_POJO">
      <backingMap name="Node" pluginCollectionRef="Node"/>
      <backingMap name="BusinessUnitType" pluginCollectionRef="BusinessUnitType"/>
      <querySchema>
        <mapSchemas>
          <mapSchema mapName="Node"
     valueClass="com.ibm.websphere.objectgrid.samples.entity.Node"
            primaryKeyField="id" />
          <mapSchema mapName="BusinessUnitType"
            valueClass="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
            primaryKeyField="id" />
        </mapSchemas>
        <relationships>
          <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
            target="com.ibm.websphere.objectgrid.samples.entity.Node"
            relationField="parentNodeId" invRelationField="childrenNodeIds" />
          <relationship source="com.ibm.websphere.objectgrid.samples.entity.Node"
            target="com.ibm.websphere.objectgrid.samples.entity.BusinessUnitType"
            relationField="businessUnitTypeKeys" invRelationField="" />
        </relationships>
      </querySchema>
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
```

```
    <backingMapPluginCollection id="Node">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="parentNode"/>
<property name="Name" type="java.lang.String" value="parentNodeId"/>
<property name="AttributeName" type="java.lang.String" value="parentNodeId"/>
<property name="RangeIndex" type="boolean" value="false"
  description="Ranges are not supported for association indexes." />
      </bean>
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
        <property name="Name" type="java.lang.String" value="businessUnitType"/>
        <property name="AttributeName" type="java.lang.String" value="businessUnitTypeKeys"/>

<property name="RangeIndex" type="boolean" value="false"
  description="Ranges are not supported for association indexes." />
   </bean>
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="childrenNodeIds"/>
  <property name="AttributeName" type="java.lang.String" value="childrenNodeIds"/>
  <property name="RangeIndex" type="boolean" value="false"
      description="Ranges are not supported for association indexes." />
</bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

Date le precedenti configurazioni di indice, si ottimizzeranno i seguenti esempi di query di oggetti.

```
SELECT n FROM Node n WHERE n.parentNodeId is null
SELECT n FROM Node n WHERE n.businessUnitTypeKeys is EMPTY
SELECT n FROM Node n WHERE size(n.businessUnitTypeKeys)>=10
SELECT n FROM BusinessUnitType b, Node n WHERE
 b member of n.businessUnitTypeKeys and b.name='TELECOM'
```

Utilizzo degli oggetti diversi dalle chiavi per trovare le partizioni (interfaccia PartitionableKey)

Quando una configurazione eXtreme Scale utilizza la strategia di posizionamento a partizione fissa, esso dipende dalla funzione di hashing della chiave in una partizione per inserire, ottenere, aggiornare o rimuovere il valore. Il metodo hashCode viene richiamato nella chiave e deve essere ben definita se viene creata una chiave personalizzata. Tuttavia un'altra opzione è di utilizzare l'interfaccia PartitionableKey. Utilizzando l'interfaccia PartitionableKey, è possibile utilizzare un oggetto piuttosto che la chiave per eseguire l'hash su una partizione.

È possibile utilizzare l'interfaccia PartitionableKey in situazioni in cui esistono più mappe e i dati su cui si esegue il commit, sono relativi e perciò dovrebbero essere inseriti sulla stessa partizione. WebSphere eXtreme Scale non supporta i commit a due fasi perciò non dovrebbe essere eseguito il commit di più transazioni della mappa se espandono più partizioni. Se PartitionableKey esegue l'hash nella tessa partizione per le chiavi nelle differenti mappe nella stessa serie di mappe, può essere eseguito il commit insieme.

È possibile utilizzare l'interfaccia PartitionableKey quando i gruppi di chiavi dovrebbero essere inseriti sulla stessa partizione, ma non necessariamente durante una singola transazione. Se dovesse essere eseguito l'hash sulle chiavi in base alla location, al reparto, al tipo di dominio o a qualche altro tipo di identificativo, le chiavi child possono essere ottenute da un PartitionableKey parent.

Per esempio l'ash sugli impiegati dovrebbe essere eseguito nella stessa partizione del loro reparto. Ciascuna chiave employee dovrebbe avere un oggetto PartitionableKey che appartiene alla mappa department. Successivamente si dovrebbe eseguire l'ash su dipendenti e reparti nella stessa partizione.

L'interfaccia PartitionableKey fornisce un metodo denominato ibmGetPartition. L'oggetto restituito da questo metodo deve implementare il metodo hashCode. Il risultato restituito dall'utilizzo alternativo di hashCode verrà utilizzato per instradare la chiave alla partizione.

Programmazione per le transazioni

Le applicazioni che richiedono transazioni introducono considerazioni quali la gestione dei blocchi, la gestione delle collisioni e l'isolamento della transazione.

Panoramica sull'elaborazione della transazione

Sessioni ed elaborazione della transazione

WebSphere eXtreme Scale utilizza le transazioni come meccanismo di interazione con i dati.

Per interagire con i dati, il thread dell'applicazione necessita di una sessione propria. Quando l'applicazione desidera utilizzare l'ObjectGrid su un thread, chiama uno dei metodi ObjectGrid.getSession per ottenere un thread. Con la sessione, l'applicazione può utilizzare i dati memorizzati nelle mappe ObjectGrid.

Quando un'applicazione utilizza un oggetto Session, la sessione deve essere nel contesto di una transazione. Una transazione inizia ed esegue il commit o inizia ed esegue il rollback utilizzando i metodi begin, commit, e rollback sull'oggetto Session. Le applicazioni possono anche operare in modalità di commit automatico, in cui Session si avvia ed esegue il commit di una transazione automaticamente ogni volta che viene eseguita un'operazione sulla mappa. La modalità di commit automatico non può raggruppare più operazioni in una singola transazione, pertanto è l'opzione più lenta se si sta creando un batch di più operazioni in una singola transazione. Tuttavia, per le transazioni che contengono una sola operazione, il commit automatico è l'opzione più veloce.

Transazioni

Le transazioni hanno molti vantaggi per la memorizzazione e la manipolazione dei dati. È possibile utilizzare le transazioni per proteggere la griglia da modifiche simultanee, per applicare più modifiche come un'unità contemporanea, per replicare i dati e per implementare il ciclo di vita per i blocchi sulle modifiche.

Quando viene avviata una transazione, WebSphere eXtreme Scale assegna una speciale mappa delle differenze per contenere le modifiche o le copie correnti delle coppie chiave-valore utilizzate dalla transazione. In genere, quando si accede ad una coppia chiave-valore, il valore viene copiato prima che l'applicazione riceva il valore. La mappa delle differenze tiene traccia di tutte le modifiche per operazioni quali l'inserimento, l'aggiornamento, il richiamo, la rimozione e così via. Le chiavi non vengono copiate perché ritenute immutabili. Se viene specificato un oggetto ObjectTransformer, quell'oggetto viene utilizzato per copiare il valore. Se la transazione utilizza un blocco ottimistico, viene tenuta traccia anche delle immagini precedenti dei valori per il confronto quando viene eseguito il commit della transazione.

Se viene eseguito il rollback di una transazione, le informazioni della mappa delle differenze vengono eliminate, e i blocchi sulle voci vengono rilasciati. Quando viene eseguito il commit di una transazione, le modifiche vengono applicate alle mappe ed i blocchi vengono rilasciati. Se viene utilizzato il blocco ottimistico, eXtreme Scale confronta le versioni delle immagini precedenti dei valori con i valori contenuti nella mappa. Affinché possa essere seguito il commit della

transazione, questi valori devono corrispondere. Questo confronto consente uno schema di blocco di più versioni, ma questo comporta la creazione di due copie quando la transazione accede alla voce. Tutti i valori vengono copiati di nuovo e la nuova copia viene memorizzata nella mappa. WebSphere eXtreme Scale esegue questa copia per proteggersi dalla modifica del riferimento applicazione da parte dell'applicazione sul valore successivo ad un commit.

È possibile evitare di utilizzare diverse copie delle informazioni. L'applicazione può salvare una copia utilizzando un blocco pessimistico invece del blocco ottimistico come costo della limitazione della simultaneità. Anche la copia del valore durante il commit può essere evitata se l'applicazione decide di non modificare un valore dopo un commit.

Vantaggi delle transazioni

Utilizzare le transazioni per i seguenti motivi:

Utilizzando le transazioni, è possibile:

- Eseguire il rollback delle modifiche se si verifica un'eccezione o la logica di business deve annullare le modifiche di stato.
- Per applicare più modifiche come un'unità atomica al momento del commit.
- Mantenere e rilasciare blocchi sui dati per applicare più modifiche come un'unità atomica al momento del commit.
- Proteggere un thread da modifiche simultanee.
- Implementare un ciclo di vita per i blocchi sulle modifiche.
- Produrre un'unità atomica di replica.

Dimensione della transazione

Le transazioni di dimensioni maggiori sono più efficienti, specialmente per la replica. Tuttavia, le transazioni di dimensioni maggiori possono influire negativamente sulla simultaneità poiché i blocchi sulle voci vengono mantenuti per un periodo di tempo più lungo. Se si utilizzano transazioni di dimensioni maggiori, è possibile accrescere le prestazioni della replica. Questo accrescimento delle prestazioni è importante quando si precarica una mappa. Provare differenti dimensioni di batch per determinare cosa funziona meglio per lo scenario.

Transazioni più grandi sono utili anche con i programmi di caricamento. Se viene utilizzato un programma di caricamento che può eseguire il batch SQL, sono possibili significativi miglioramenti delle prestazioni a seconda della transazione e delle riduzioni significative del carico sul lato database. Questo miglioramento delle prestazioni dipende dall'implementazione di Loader.

Modalità di commit automatico

Se non è stata avviata attivamente alcuna transazione, quando un'applicazione interagisce con un oggetto ObjectMap viene eseguita un'operazione automatica di inizio e di commit per conto dell'applicazione. Questa operazione automatica di inizio e di commit funziona, ma impedisce il funzionamento corretto del rollback e del blocco. Questo ha un impatto sulla velocità di replica sincrona a causa della dimensione molto piccola della transazione. Se si sta usando un'applicazione gestore di entità, non utilizzare la modalità di commit automatico poiché gli oggetti che vengono cercati con il metodo EntityManager.find divengono immediatamente non gestiti alla restituzione del metodo e non utilizzabili.

Coordinatori di transazioni esterne

In genere le transazioni iniziano con il metodo session.begin e terminano con il metodo session.commit. Tuttavia, quando eXtreme Scale è incorporato, le transazioni potrebbero essere avviate e terminate da un coordinatore di transazioni esterne. Se si utilizza un coordinatore di transazioni esterne, non è necessario richiamare il metodo session.begin e terminare con il metodo session.commit. Per ulteriori informazioni su eXtreme Scale e l'interazione di transazione esterne, consultare *Guida alla programmazione*. Se si utilizza WebSphere Application Server, è possibile utilizzare il plug-in WebSphereTranscationCallback. Per ulteriori informazioni sui plug-in disponibili con WebSphere eXtreme Scale, consultare *Guida alla programmazione*.

Attributo CopyMode

È possibile ottimizzare il numero di copie definendo l'attributo CopyMode degli oggetti BackingMap o ObjectMap.

È possibile ottimizzare il numero di copie definendo l'attributo CopyMode degli oggetti BackingMap o ObjectMap. La modalità di copia ha i seguenti valori:

- COPY_ON_READ_AND_COMMIT
- COPY_ON_READ
- NO_COPY
- COPY_ON_WRITE
- COPY_TO_BYTES

Il valore COPY_ON_READ_AND_COMMIT è quello predefinito. Il valore COPY_ON_READ esegue la copia quando vengono inizialmente richiamati i dati, ma non durante il commit. Questa modalità è sicura se l'applicazione non modifica un valore dopo aver eseguito il commit di una transazione. Il valore NO_COPY non copia i dati, ed è una modalità sicura solo per i dati di sola lettura. Se i dati non cambiano mai, non è necessario copiarli per motivi di isolamento.

Prestare attenzione quando si utilizza il valore di attributo NO_COPY con mappe che possono essere aggiornate. WebSphere eXtreme Scale utilizza la copia al primo tocco per consentire il rollback della transazione. L'applicazione ha cambiato solo la copia, e come risultato, eXtreme Scale elimina la copia. Se viene utilizzato il valore di attributo NO_COPY, e l'applicazione modifica il valore sottoposto a commit, il completamento di un rollback non è possibile. La modifica del valore sottoposto a commit porta a problemi con gli indici, la replica e così via, poiché gli indici e le repliche si aggiornano quando viene eseguito il commit della transazione. Se si modificano i dati sottoposti a commit e poi si esegue il rollback della transazione, che di fatto non viene eseguito, gli indici non vengono aggiornati e la replica non viene eseguita. Altri thread possono immediatamente individuare le modifiche senza commit, anche se hanno i blocchi. Utilizzare il valore di attributo NO_COPY solo per le mappe di sola lettura o per le applicazioni che completano la copia appropriata prima di modificare il valore. Se si utilizza il valore attributo NO_COPY e si chiama il supporto IBM con un problema di integrità dei dati, viene richiesto di riprodurre il problema con la modalità di copia impostata su COPY_ON_READ_AND_COMMIT.

Il valore COPY_TO_BYTES memorizza i valori nella mappa in una formato serializzato. Alla lettura, eXtreme Scale deserializza il valore da un formato serializzato e al commit memorizza il valore in un formato serializzato. Con questo metodo viene eseguita una copia si alla lettura che al commit.

La modalità di copia predefinita per una mappa può essere configurata sull'oggetto BackingMap. È possibile anche cambiare la modalità di copia sulle mappe prima di avviare una transazione utilizzando il metodo ObjectMap.setCopyMode.

Segue un esempio di un frammento di mappa di backup da un file `objectgrid.xml` che mostra come impostare la modalità di copia per una determinata mappa di backup. Questo esempio presume che si stia utilizzando `cc` come spazio dei nomi `objectgrid/config`.

```
<cc:backingMap name="RuntimeLifespan" copyMode="NO_COPY"/>
```

Per ulteriori informazioni, consultare le informazioni relative alle migliori pratiche per copyMode in *Guida alla programmazione*.

Blocco della voce della mappa

Un ObjectGrid BackingMap supporta diverse strategie di blocco per le mappe al fine di mantenere la congruenza nella voce della cache.

Ciascuna BackingMap può essere configurata per utilizzare una delle seguenti strategie di blocco.

1. Modalità di blocco ottimistica
2. Modalità di blocco pessimistica
3. None

La strategia di blocco predefinita è OTTIMISTICA. Utilizzare il blocco ottimistico quando i dati vengono modificati non frequentemente. I blocchi sono conservati solo per un breve periodo mentre i dati sono letti dalla cache e copiati alla transazione. Quando la cache della transazione è sincronizzata con la cache principale, qualsiasi oggetto di cache che è stato aggiornato, viene controllato rispetto alla versione originale. Se la verifica non riesce, allora la transazione esegue il rollback e si genera un'eccezione OptimisticCollisionException.

La strategia di blocco PESSIMISTICA acquisisce i blocchi per le voci della cache e dovrebbe essere utilizzata quando i dati vengono modificati frequentemente. Qualche volta una voce dela cache è in lettura viene acquisito un blocco e conservato condizionatamente fino a quando non è completata la transazione. La durata di alcuni blocchi può essere regolata utilizzando i livelli di isolamento della transazione per la sessione.

Se non è richiesto il blocco perché i dati non vengono mai aggiornati o vengono aggiornati solo durante i periodi di attesa, è possibile disabilitare il blocco utilizzando la strategia di blocco NONE. Questa strategia è molto veloce perché non viene richiesto un gestore blocco. La strategia di blocco NONE è ideale per le tabelle di ricerca o per le mappe in sola lettura.

Per ulteriori informazioni relative alle strategie di blocco, consultare nella sezione *Panoramica sul prodotto*.

Specifica di una strategia di blocco

L'esempio di seguito riportato dimostra come la strategia di blocco possa essere impostata sulle BackingMaps map1, map2 e map3 , laddove ciascuna mappa utilizza una diversa strategia di blocco. Il primo frammento di codice mostra in che modo utilizzare XML per la configurazione della strategia di blocco e il secondo frammento di codice mostra un approccio programmatico.

Approccio XML

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

    <objectGrids>
        <objectGrid name="test">
            <backingMap name="map1"
                lockStrategy="PESSIMISTIC" numberOfLockBuckets="31"/>
            <backingMap name="map2"
                lockStrategy="OPTIMISTIC" numberOfLockBuckets="409"/>
            <backingMap name="map3"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Approccio programmatico

Configurazione BackingMap - esempio programmatico
```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.LockStrategy;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
...
ObjectGrid og =
 ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm.setNumberOfLockBuckets(31);
bm = og.defineMap("map2");
bm.setNumberOfLockBuckets(409);
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

Per evitare un'eccezione java.lang.IllegalStateException, il metodo setLockStrategy deve essere richiamato prima di utilizzare i metodi Initialize o getSession in un'istanza locale ObjectGrid instance.

Per ulteriori informazioni, consultare l'argomento relative alle strategie di blocco nella sezione *Panoramica sul prodotto*.

Configurazione di un gestore blocco

Quando vengono utilizzate sia una strategia di blocco PESSIMISTICA che una OTTIMISTICA, un gestore blocco viene creato per la BackingMap. Il gestore blocco utilizza una mappa di hash per tenere traccia delle voci che sono bloccate da una o più transazioni. Se esistono molte voci della mappa nell'associazione hash, più bucket di blocco possono condurre ad una migliore prestazione. better performance. Il rischio di conflitti di sincronizzazione Java è più basso quanto più cresce il numero di bucket. Più bucket di blocco conduce anche ad una maggiore concorrenza. Gli esempi precedenti mostrano in che modo un'applicazione può impostare il numero di bucket di blocco da utilizzare per una determinata istanza BackingMap.

Per evitare un'eccezione java.lang.IllegalStateException, il metodo setNumberOfLockBuckets deve essere richiamato prima di richiamare i metodi Initialize o getSession su un'istanza ObjectGrid. Il parametro del metodo setNumberOfLockBuckets è un primitivo Java intero che specifica il numero di bucket di blocco da utilizzare. L'utilizzo di un numero primo può consentire una

distribuzione uniforme delle voci della mappa sui bucket di bloccco. Un buon punto di partenza per migliorare le prestazioni è impostare il numero di bucket di blocco al 10 per cento circa del numero previsto di voci di BackingMap.

LockDeadlockException

Di seguito è riportato un esempio di codice che mostra il rilevamento delle eccezioni e il messaggio che ne risulta, viene successivamente visualizzato.

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

Il risultato è:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: _Message
```

Questo messaggio rappresenta la stringa che viene passata come parametro quando l'eccezione è creata e attivata.

Causa dell'eccezione

Il tipo più comune di eccezione di deadlock avviene quando si utilizza la strategia di blocco pessimisticoe due client separati ciascuno avente un blocco condiviso su un particolare oggetto. Successivamente, entrambi i client tentano di promuovere un blocco esclusivo su quell'oggetto. Il diagramma di seguito riportato illustra tale situazione, compreso i blocchi della transazione che causano 'eccezione.

Il frammento di codice Java di seguito riportato dimostra in che modo passare un file di configurazione XML per creare un ObjectGrid.

Questo è una visualizzazione dell'estratto di ciò che accade nel proprio programma quando si verifica l'eccezione. In un'appplicazione con molti thread che aggiornano lo stesso ObjecMap, è possibile incorrere in questa situazione. Di seguito è riportato un esempio di due client che esguono i blocchi del codice transazione come illustrato nella figura precedente.

Soluzioni possibili

È possibile ritrovare la situazione illustrata nella figura 1 quando numerosi thread avviano le transazioni in una particolare mappa. In questo caso, l'eccezione è attivata per evitare che il programma sia sospeso. È possibile avvisare se stessi e aggiungere codice al blocco rilevato per ulteriori dettagli sulla causa. Poiché questa eccezione si vede solo in una strategia di blocco pessimistico, una semplice soluzione è semplificare l'utilizzo della strategia di blocco. Se si acquisisce una strategia di blocco pessimistico, tuttavia è possibile utilizzare il metodo getForUpdate invece del metodo Get. Ciò elimina la ricezione delle eccezioni per la situazione descritta precedentemente.

Strategie di blocco

Le strategie di blocco includono le strategie pessimistica, ottimistica e none Per scegliere una strategia di blocco, è necessario considerare questioni come la percentuale di ciascun tipo di operazione che si ha se oppure se si utilizza o meno un programma di caricamento e così via.

I blocchi sono collegati dalle transazioni. È possibile specificare le seguenti impostazioni di blocco:

- **Nessun blocco**: l'esecuzione senza l'impostazione di blocco è l'opzione più veloce. Se si stanno utilizzando dati di sola lettura, il blocco potrebbe non essere necessario.
- **Blocco pessimistico**: acquisisce i blocchi sulle voci e mantiene i blocchi fino al commit. Questa strategia di blocco fornisce una buona congruenza a spese della velocità di trasmissione.
- **Blocco ottimistico**: prende un'immagine precedente di ogni record che la transazione tocca e la confronta con i valori correnti della voce quando viene seguito il commit della transazione. Se i valori della voce cambiano, la transazione esegue il rollback. Non vengono mantenuti blocchi fino al commit. Questa strategia di blocco fornisce una migliore simultaneità della strategia pessimistica, con il rischio del rollback della transazione e il dispendio di memoria per la creazione di un'ulteriore copia della voce.

Impostare la strategia di blocco sulla BackingMap. Non è possibile modificare la strategia di blocco di ogni transazione. Di seguito è riportato un esempio di frammento XML che mostra come impostare la modalità di blocco su una mappa utilizzando il file XML, presupponendo che `cc` sia lo spazio dei nomi per lo spazio dei nomi `objectgrid/config`:

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Blocco pessimistico

Utilizzare la strategia di blocco pessimistico per le mappe in lettura e scrittura quando altre strategie di blocco non sono possibili. Quando una mappa ObjectGrid è configurata per utilizzare una strategia di blocco pessimistico, si ottiene un blocco della transazione pessimistica per una voce della mappa quando una transazione prende prima la voce da BackingMap. Il blocco pessimistico è conservato fino a quando l'applicazione non completa la transazione. Generalmente la strategia di blocco pessimistico viene utilizzata nelle seguenti situazioni:

- Quando viene configurata la BackingMap con o senza un programma di caricamento e le informazioni sul controllo versioni non sono disponibili.
- Quando viene utilizzata la BackingMap direttamente da un'applicazione che ha necessità di una guida da eXtreme Scale per il controllo della concorrenza.
- Quando le informazioni sul controllo versioni sono disponibili, ma le transazioni di aggiornamento sono frequentemente in conflitto con le voci di backup risultando in un aggiornamento pessimistico non riuscito.

Poiché la strategia di blocco pessimistico ha il maggiore impatto sulle prestazioni e la scalabilità, questa strategia dovrebbe essere utilizzata per le mappe in lettura e scrittura solo quando altre strategie di blocco non sono praticabili. Ad esempio, queste situazioni potrebbero includere quando si verifica di frequente che un aggiornamento ottimistico non riesce o quando il recupero da un errore ottimistico è difficile da gestire per un'applicazione.

Blocco ottimistico

La strategia di blocco ottimistico presuppone che nessuna delle due transazioni possa tentare di aggiornare la stessa voce della mappa durante l'esecuzione contemporanea. A causa di questa convinzione, la modalità blocco non ha necessità di essere conservata per il ciclo di vita della transazione poiché è improbabile che più di una transazione possa aggiornare la voce della mappa contemporaneamente. La strategia di blocco ottimistico è generalmente utilizzata nelle seguenti situazioni:

- quando viene configurata la BackingMap con o senza un programma di caricamento e le informazioni sul controllo versioni sono disponibili.
- Quando una BackingMap ha per lo più transazioni che eseguono operazioni di lettura. Le operazioni di inserimento, aggiornamento e rimozione delle voci della mappa non si verificano spesso nella BackingMap.
- Quando una BackingMap viene inserita, aggiornata o rimossa più frequentemente di quanto non sia letta, ma le transazioni raramente sono in conflitto sulla stessa voce della mappa.

Come la strategia di blocco pessimistico, i metodi nell'interfaccia ObjectMap determinano in che modo eXtreme Scale tenta automaticamente di acquisire una modalità blocco per la voce della mappa che sta per essere acceduta. Tuttavia, esistono le seguenti differenze tra le strategie pessimistica e ottimistica:

- Come la strategia di blocco pessimistico, una modalità blocco S viene acquisita dai metodi Get e getAll quando viene richiamato il metodo. Tuttavia, con un blocco pessimistico, la modalità blocco S non è conservata fino a quando la transazione non è completa. Invece la modalità blocco S viene rilasciata prima che il metodo ritorni all'applicazione. Lo scopo per acquisire la modalità blocco è tale per cui eXtreme Scale può garantire che solo i dati su cui è stato eseguito il commit dalle altre transazioni è visibile alla transazione corrente. Dopo che eXtreme Scale ha verificato che è stato eseguito il commit dei dati, la modalità blocco S viene rilasciata. Al momento del commit viene eseguito un controllo sulla versione ottimistico per garantire che nessuna altra transazione abbia modificato la voce della mappa dopo che la transazione corrente ha rilasciato la propria modalità blocco S. Se non viene eseguito il fetch di una voce dalla mappa prima che essa sia aggiornata, invalidata o eliminata, eXtreme Scale il runtime implicitamente esegue il fetch della voce della mappa. Questa operazione implicita di caricamento viene eseguita per ottenere il valore corrente al momento in cui è stato richiesto di modificare la voce.
- Diversamente dalla strategia di blocco pessimistico, i metodi getForUpdate e getAllForUpdate vengono gestiti esattamente come i metodi Get e getAll quando viene utilizzata la strategia di blocco ottimistico. Cioè una modalità blocco S viene acquisita all'avvio del metodo e la modalità blocco S viene rilasciata prima di ritornare all'applicazione.

Tutti gli altri metodi ObjectMap vengono gestiti esattamente come viene gestita la strategia di blocco pessimistico. Cioè quando viene richiamato il metodo Commit, si ottiene una modalità blocco X per la voce della mappa che è stata inserita, aggiornata, rimossa,toccata o invalidata e la modalità blocco X è conservata fino a quando la transazione non completa l'elaborazione del commit.

La strategia di blocco ottimistico presuppone che nessuna transazione in esecuzione contemporaneamente tenti di aggiornare la stessa voce della mappa. A causa di questo assunto, la modalità blocco non necessita di essere conservata per il ciclo di vita della transazione poiché è improbabile che più di una transazione possa aggiornare contemporaneamente la voce della mappa. Tuttavia, poiché non è conservata una modalità di blocco, un'altra transazione concomitante potrebbe potenzialmente aggiornare la voce della mappa dopo che la transazione corrente ha rilasciato la modalità blocco S.

Per gestire questa possibilità, eXtreme Scale ottiene un blocco X al momento del commit ed esegue un controllo sulla versione ottimistico per verificare che nessun'altra transazione abbia modificato la voce della mappa dopo che la transazione corrente abbia letto la voce della mappa da BackingMap. Se un'altra transazione modifica la voce della mappa, il controllo sulla versione non riesce e si

verifica un'eccezione OptimisticCollisionException. Questa eccezione forza la transazione corrente ad eseguire il roll back e l'applicazione deve tentare l'intera transazione nuovamente. La strategia di blocco ottimistico è molto utile quando una mappa è per lo più letta ed è improbabile che si verifichino aggiornamenti alla stessa voce della mappa.

Nessun blocco

Quando una BackingMap viene configurata per non utilizzare alcuna strategia di blocco, non si ottiene alcun blocco della transazione per una voce della mappa.

L'utilizzo di nessuna strategia di blocco è utile quando un'applicazione utilizza Hibernate per ottenere i dati permanenti. In questo scenario, la BackingMap viene configurata senza un programma di caricamento e il gestore della persistenza utilizza la BackingMap come un cache di dati. In questo scenario, il gestore della persistenza fornisce un controllo sulla concorrenza tra le transazioni che accedono le stesse voci della mappa.

WebSphere eXtreme Scale non ha necessità di ottenere alcun blocco sulla transazione a scopo di controllo della concorrenza. Questa situazione presuppone che il gestore della persistenza non rilasci i propri blocchi sulla transazione prima di aggiornare la mappa ObjectGrid con le modifiche sulle quali è stato eseguito il commit. Se il gestore della persistenza rilascia i suoi blocchi, allora deve essere utilizzata una strategia di blocco ottimistico o pessimistico. Ad esempio, supponiamo che il gestore della persistenza di un contenitore EJB stia aggiornando una mappa ObjectGrid con i dati sui quali è stato eseguito il commit nella transazione gestita dal contenitore EJB. Se l'aggiornamento della mappa ObjectGrid si verifica prima che siano rilasciati i blocchi della transazione del gestore della persistenza, allora è possibile non utilizzare alcuna strategia di blocco. Se si verifica l'aggiornamento della mappa ObjectGrid dopo che sono stati rilasciati i blocchi della transazione del gestore della persistenza, allora devi utilizzare sia la strategia di blocco ottimistico che quella di blocco pessimistico.

Un altro scenario in cui non può essere utilizzata alcuna strategia di blocco, si ha quando l'applicazione utilizza direttamente una BackingMap e un programma di caricamento è configurato per la mappa. In questo scenario, il programma di caricamento utilizza il supporto del controllo della concorrenza che viene fornito da RDBMS (Sistema di gestione del database relazionale) utilizzando sia JDBC (Java database connectivity) che Hibernate per accedere i 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 con controllo sulla versione guida nel raggiungere i risultati migliori nella concorrenza e nelle prestazioni. Per ulteriori informazioni relative all'approccio di blocco ottimistico, consultare la sezione OptimisticCallback nelle per le informazioni relative alle considerazioni sul programma di caricamento in *Guida alla gestione*. Se si sta utilizzando un programma di caricamento che utilizza il supporto di blocco pessimistico di un backend sottostante si potrebbe desiderare di utilizzare il parametro forUpdate che viene passato nel metodo Get dell'interfaccia del programma di caricamento. Impostare questo parametro su true se è stato utilizzato il metodo getForUpdate dell'interfaccia ObjectMap da un'applicazione per ottenere i dati. Il programma di caricamento può utilizzare questo parametro per determinare se richiedere un blocco aggiornabile per la riga che si sta leggendo. Ad esempio, il DB2 ottiene un blocco aggiornabile quando l'istruzione select contiene una clausola `FOR UPDATE`. Questo approccio offre la stessa possibilità di prevenire un deadlock che è descritta in "Blocco pessimistico" a pagina 138.

JMS per la distribuzione delle modifiche della transazione

Utilizzare JMS (Java Message Service) per modifiche di transazioni distribuite tra livelli differenti o in ambienti su piattaforme miste.

JMS è un protocollo ideale per modifiche distribuite tra livelli differenti o in ambienti su piattaforme miste. Ad esempio, alcune applicazioni che utilizzano eXtreme Scale potrebbero essere distribuite su IBM WebSphere Application Server Community Edition, Apache Geronimo o Apache Tomcat, laddove altre applicazioni potrebbero essere eseguite su WebSphere Application Server Versione 6.x. JMS è ideale per le modifiche distribuite tra peer eXtreme Scale in questi diversi ambienti. Il sistema di trasporto messaggi del gestore HA (High Availability) è molto veloce ma può distribuire modifiche solo a Java virtual machine che sono in un gruppo principale singolo. JMS è più lento ma consente impostazioni di client dell'applicazione più ampie e varie per condividere un ObjectGrid. JMS è ideale quando si condividono dati in un ObjectGrid tra un client fat Swing ed un'applicazione distribuita su WebSphere Extended Deployment.

Il Client Invalidation Mechanism incorporato e il Peer-to-Peer Replication sono esempi di distribuzione delle modifiche transazionali basate su JMS. Consultare le informazioni sulla configurazione della replica peer-to-peer con JMS in*Guida alla gestione* per ulteriori informazioni.

Implementazione di JMS

JMS viene implementato per modifiche di transazioni distribuite mediante un oggetto Java che si comporta come un ObjectGridEventListener. Questo oggetto può propagare lo stato nei seguenti quattro modi:

1. Invalidazione: qualunque voce eliminata, aggiornata o cancellata viene rimossa su tutti i peer Java virtual machine quando viene ricevuto il messaggio.
2. Invalidazione condizionale: la voce viene eliminata solo se la versione locale è la stessa o è più vecchia rispetto alla versione sul publisher.
3. Invio: qualunque voce eliminata, aggiornata, cancellata o inserita viene aggiunta o sovrascritta su tutti i peer Java virtual machine quando viene ricevuto il messaggio JMS.
4. Invio condizionale: la voce viene solo aggiornata o aggiunta sul lato ricevente se la voce locale è meno recente della versione che è stata pubblicata.

Ascolto di modifiche per la pubblicazione

Il plug-in implementa l'interfaccia ObjectGridEventListener per intercettare l'evento transactionEnd. Quando eXtreme Scale richiama questo metodo, il plug-in prova a convertire l'elenco LogSequence per ogni mappa toccata dalla transazione in un messaggio JMS e poi lo pubblica. Il plug-in può essere configurato per pubblicare modifiche per tutte le mappe o per una serie secondaria di mappe. Gli oggetti LogSequence vengono elaborati per le mappe abilitate alla pubblicazione. La classe ObjectGrid di LogSequenceTransformer serializza un LogSequence filtrato per ogni mappa in un flusso. Dopo la serializzazione di tutti i LogSequences in un flusso, viene creato un ObjectMessage JMS e viene pubblicato in un argomento ben noto.

Ascolto di messaggi JMS e loro applicazione nell'ObjectGrid locale

Lo stesso plug-in avvia anche un thread che si avvita in un loop, ricevendo tutti i messaggi che vengono pubblicati in un argomento ben noto. Quando arriva un messaggio, passa il contenuto del messaggio alla classe LogSequenceTransformer dove viene convertito in una serie di oggetti LogSequence. Quindi, viene avviata

una transazione no-write-through. Ogni oggetto LogSequence viene fornito al metodo Session.processLogSequence, che aggiorna le mappe in locale con le modifiche. Il metodo processLogSequence capisce la modalità di distribuzione. Viene eseguito il commit della transazione e la cache locale ora riflette le modifiche. Per ulteriori informazioni sull'utilizzo di JMS per la distribuzione delle modifiche della transazione, consultare le informazioni sulla distribuzione delle modifiche tra Java Virtual Machines peer in *Guida alla gestione*.

Transazioni su partizione singola e sulle partizioni della griglia

La distinzione principale tra WebSphere eXtreme Scale e le soluzioni di memorizzazione dei dati tradizionali come i database relazionali o i database in memoria è rappresentata dall'utilizzo delle partizioni, che consentono alla cache di scalare in modo lineare. I tipi importanti di transazione da considerare sono le transazioni su partizione singola e su tutte le partizioni (sulla griglia).

In generale, le interazioni con la cache possono essere suddivise in transazioni su partizione singola o transazioni sulla griglia, come riportato di seguito.

Transazioni su partizione singola

Le transazioni su partizione singola rappresentano il metodo preferito per l'interazione con le cache ospitate da WebSphere eXtreme Scale. Quando una transazione è limitata ad una partizione singola, per impostazione predefinita è limitata ad una singola Java virtual machine e quindi ad un singolo computer server. Un server può completare un numero *M* di tali transazioni al secondo e, se si dispone di *N* computer, è possibile completare `M*N` transazioni al secondo. Se le proprie attività vengono incrementate ed è necessario raddoppiare il numero di transazioni al secondo, è possibile raddoppiare *N* acquistando altri computer. Quindi, è possibile soddisfare le richieste di capacità senza modificare l'applicazione, aggiornare l'hardware o portare l'applicazione fuori linea.

Oltre a consentire alla cache di scalare in modo così significativo, le transazioni su partizione singola incrementano al massimo la disponibilità della cache. Ciascuna transazione dipende solo da un computer. Anche in caso di malfunzionamento di uno degli altri `(N-1)` computer, il successo o il tempo di risposta della transazione non vengono modificati. Quindi, se vengono utilizzati 100 computer e si verifica un malfunzionamento di uno di essi, viene eseguito il rollback solo dell'1 percento delle transazioni in esecuzione al momento del malfunzionamento del server. In seguito al malfunzionamento del server, WebSphere eXtreme Scale assegna nuovamente le partizioni ospitate dal server malfunzionante agli altri 99 computer. Durante tale periodo, prima che l'operazione venga completata, gli altri 99 computer possono ancora completare le transazioni. Vengono bloccate solo le transazioni che coinvolgono le partizioni in fase di riassegnazione. Una volta completato il processo di failover, la cache può continuare a funzionare, completamente operativa, al 99 percento della propria capacità di trasmissione originale. Dopo che il server malfunzionante viene sostituito e restituito alla griglia, la cache ritorna al 100 percento della capacità di trasmissione.

Transazioni sulla griglia

In termini di prestazioni, disponibilità e scalabilità, le transazioni sulla griglia sono l'opposto delle transazioni su partizione singola. Le transazioni sulla griglia accedono a tutte le partizioni e quindi a tutti i computer nella configurazione. A ciascun computer nella griglia viene richiesto di ricercare alcuni dati e di restituire i risultati. La transazione non può essere completata fino a quando non hanno risposto tutti i computer e quindi il livello di prestazione dell'intera griglia è

limitato dal computer più lento. L'aggiunta di altri computer non rende più veloce il computer più lento e quindi non migliora il livello di prestazione della cache.

Le transazioni sulla griglia hanno un effetto simile sulla disponibilità. Facendo riferimento all'esempio precedente, se vengono utilizzati 100 server e si verifica il malfunzionamento di un server, viene eseguito il rollback del 100 percento delle transazioni in esecuzione al momento del malfunzionamento del server. Dopo il malfunzionamento del server, WebSphere eXtreme Scale inizia a riassegnare le partizioni ospitate da tale server agli altri 99 computer. Durante questo periodo di tempo, prima che venga completato il processo di failover, la griglia non è in grado di elaborare alcuna di tali transazioni. Una volta completato il processo di failover, la cache può continuare a funzionare, ma con una capacità ridotta. Se ciascun computer nella griglia gestiva 10 partizioni, 10 dei rimanenti 99 computer ricevono almeno una partizione supplementare in seguito al processo di failover. L'aggiunta di una partizione supplementare incrementa il carico di lavoro di tale computer di almeno il 10 percento. Poiché il livello di prestazione della griglia è limitato alla velocità di trasmissione del computer più lento in una transazione sulla griglia, il livello di prestazione viene ridotto in media del 10 percento.

È preferibile utilizzare le transazioni su partizione singola invece delle transazioni su griglia per eseguire lo scale out con una cache di oggetti distribuita, ad alta disponibilità come WebSphere eXtreme Scale. L'ottimizzazione delle prestazioni di questi tipi di sistemi richiede l'utilizzo di tecniche differenti dalle metodologie relazionali tradizionali, ma è possibile convertire le transazioni su griglia in transazioni su partizione singola scalabili.

Migliori pratiche per la creazione di modelli di dati scalabili

Le migliori pratiche per la creazione di applicazioni scalabili con prodotti come WebSphere eXtreme Scale includono due categorie: principi fondamentali e suggerimenti per l'implementazione. I principi fondamentali sono le idee principali da catturare nella progettazione dei dati. Un'applicazione che non osserva tali principi non sarà in grado di scalare correttamente, anche per le proprie transazioni principali. I suggerimenti per l'implementazione vengono applicati per transazioni problematiche in un'applicazione altrimenti ben progettata che osserva i principi generali per i modelli di dati scalabili.

Principi fondamentali

Alcuni degli importanti mezzi di ottimizzazione della scalabilità sono concetti di base o principi da tenere in considerazione.

Duplicazione invece della normalizzazione

Il concetto principale da ricordare quando si utilizzano prodotti come WebSphere eXtreme Scale è che tali prodotti sono progettati per distribuire i dati su un numero elevato di computer. Se l'obiettivo è il completamento della maggior parte o di tutte le transazioni su una singola partizione, il progetto del modello di dati deve garantire che tutti i dati che potrebbero essere richiesti dalla transazione si trovino nella partizione. Nella maggior parte dei casi, ciò è possibile solo duplicando i dati.

Ad esempio, considerare un'applicazione come una bacheca messaggi. Due transazioni molto importanti per una bacheca messaggi mostrano tutti i messaggi inviati da un determinato utente e tutti i messaggi relativi ad un determinato argomento. Innanzitutto, considerare il modo in cui tali transazioni funzionano con un modello di dati normalizzato che contiene

un record utente, un record dell'argomento ed un record del messaggio che contiene il testo reale. Se i messaggi sono partizionati con i record utente, la visualizzazione degli argomenti diventa una transazione su griglia e viceversa. Gli argomenti e gli utenti non possono essere partizionati insieme perché hanno una relazione molti-a-molti.

Il modo migliore per consentire a questa bacheca messaggi di scalare è quello di duplicare i messaggi, memorizzando una copia con il record dell'argomento ed una copia con il record utente. Quindi, la visualizzazione dei messaggi da un utente è una transazione su partizione singola, la visualizzazione dei messaggi su un argomento è una transazione su partizione singola e l'aggiornamento o eliminazione di un messaggio è una transazione su due partizioni. Tutte e tre tali transazioni scaleranno linearmente con l'incremento del numero di computer nella griglia.

Scalabilità piuttosto che risorse

L'ostacolo maggiore da superare quando si considerano modelli di dati denormalizzati è l'impatto di tali modelli sulle risorse. La conservazione di due, tre o più copie di alcuni dati può dare l'impressione di utilizzare un numero troppo elevato di risorse per essere pratica. In questo scenario, ricordare quanto riportato di seguito: ogni anno, le risorse hardware diventano meno costose. Secondo, e più importante, WebSphere eXtreme Scale elimina la maggior parte dei costi nascosti associati alla distribuzione di ulteriori risorse.

Misurare le risorse in termini di costi piuttosto che in termini di hardware, come megabyte e processori. Gli archivi di dati che utilizzano dati relazionali normalizzati generalmente devono essere ubicati sullo stesso computer. Tale collocazione obbligatoria indica che è necessario acquistare un singolo computer enterprise di capacità maggiori piuttosto che diversi computer di capacità minori. Con l'hardware enterprise, non è raro il caso in cui un computer in grado di completare un milione di transazioni al secondo costi molto più della soma dei costi di 10 computer in grado, ciascuno, di eseguire 100000 transazioni al secondo.

Inoltre, esiste un costo aziendale relativo all'aggiunta di risorse. Un business crescente alla fine esaurisce le proprie capacità. Quando si esaurisce la capacità, è necessario passare ad un computer più potente e veloce oppure creare un secondo ambiente di produzione. In entrambi i casi, derivano ulteriori costi a causa di affari non realizzati o della gestione di una capacità quasi doppia rispetto a quella necessaria durante il periodo di transizione.

Con WebSphere eXtreme Scale, non è necessario chiudere l'applicazione per aggiungere capacità. Se i propri progetti per l'anno successivo richiedono una capacità superiore del 10 percento, incrementare del 10 percento il numero di computer nella griglia. È possibile incrementare tale percentuale senza rendere indisponibile l'applicazione e senza acquistare capacità supplementare.

Evitare le trasformazioni dei dati

Quando si utilizza WebSphere eXtreme Scale, i dati devono essere memorizzati in un formato direttamente utilizzabile dalla logica di business. L'utilizzo di un formato più primitivo per i dati rappresenta un costo. È necessario eseguire la trasformazione quando vengono eseguite la scrittura e la lettura dei dati. Con i database relazionali tale trasformazione è necessaria, in quanto i dati vengono resi persistenti su disco abbastanza

frequentemente, ma con WebSphere eXtreme Scale, non è necessario eseguire tali trasformazioni. Per la maggior parte, i dati sono memorizzati nella memoria e possono essere memorizzati nel formato esatto richiesto dall'applicazione.

L'osservazione di questa semplice regola consente di denormalizzare i dati in base al primo principio. Il tipo più comune di trasformazione per i dati di business è rappresentato dalle operazioni JOIN, necessarie per convertire dati normalizzati in una serie di risultati che soddisfi le necessità dell'applicazione. La memorizzazione dei dati nel formato corretto evita, in modo implicito, l'esecuzione di tali operazioni JOIN e produce un modello di dati denormalizzato.

Eliminare le query illimitate

Anche se i dati sono strutturati correttamente, le query illimitate non vengono scalate correttamente. Ad esempio, non utilizzare una transazione che richiede un elenco di tutti gli elementi ordinati in base al valore. Questa transazione potrebbe funzionare inizialmente, quando il numero totale di elementi è uguale a 1000; quando, però, il numero totale di elementi raggiunge 10 milioni, la transazione restituisce tutti gli elementi. Se viene eseguita questa transazione, i due risultati più probabili sono la scadenza della transazione o la visualizzazione di un errore di memoria esaurita del client.

L'opzione migliore è quella di modificare la logica di business in modo che possano essere restituiti solo i primi 10 o 20 elementi. Tale modifica della logica rende gestibile la dimensione della transazione, indipendentemente dal numero di elementi presenti nella cache.

Definizione dello schema

Il vantaggio principale della normalizzazione dei dati consiste nel fatto che il sistema del database può occuparsi della congruenza dei dati in modo trasparente. Quando i dati vengono denormalizzati per la scalabilità, tale gestione della congruenza dei dati automatica non esiste più. Per garantire la congruenza dei dati, è necessario implementare un modello di dati che possa essere utilizzato nel livello dell'applicazione oppure come plug-in sulla griglia distribuita.

Considerare l'esempio relativo alla bacheca messaggi. Se una transazione rimuove un messaggio da un argomento, è necessario rimuovere il messaggio duplicato sul record utente. Senza un modello di dati, è possibile che uno sviluppatore scriva il codice dell'applicazione per rimuovere il messaggio dall'argomento, dimenticando di rimuovere il messaggio dal record utente. Tuttavia, se lo sviluppatore utilizzasse un modello di dati invece di interagire direttamente con la cache, il metodo removePost sul modello di dati potrebbe estrarre l'ID utente dal messaggio, ricercare il record utente e rimuovere il messaggio duplicato in modo trasparente.

In alternativa, è possibile implementare un listener che viene eseguito sulla partizione reale che rileva la modifica all'argomento e modifica automaticamente il record utente. Un listener può essere utile perché la modifica al record utente potrebbe essere eseguita in locale se la partizione dispone del record utente oppure, anche se il record utente si trova su una partizione differente, la transazione viene eseguita tra i server invece che tra client e server. È probabile che la connessione di rete tra i server sia più rapida rispetto alla connessione di rete tra il client ed il server.

Evitare i conflitti

Evitare scenari in cui è presente un contatore globale. La griglia non scala se un singolo record viene utilizzato un numero di volte sproporzionato in confronto agli altri record. Le prestazioni della griglia saranno limitate dalle prestazioni del computer che contiene il record indicato.

In queste situazioni, provare a suddividere il record, in modo che sia gestito per partizione. Ad esempio, considerare una transazione che restituisce il numero totale di voci nella cache distribuita. Invece di impostare il sistema in modo che tutte le operazioni di inserimento e rimozione accedono ad un singolo record, utilizzare, su ciascuna partizione, un listener che tiene traccia delle operazioni di inserimento e rimozione. Grazie alla traccia del listener, le operazioni di inserimento e rimozione possono diventare operazioni su partizione singola.

La lettura del contatore diventa un'operazione sulla griglia, ma per la maggior parte è già inefficiente come operazione sulla griglia perché le proprie prestazioni sono collegate a quelle del computer che contiene il record.

Suggerimenti per l'implementazione

Per ottenere la massima scalabilità, è possibile considerare i suggerimenti riportati di seguito.

Utilizzare gli indici di ricerca inversi

Considerare un modello di dati denormalizzato in modo appropriato in cui i record del cliente sono partizionati in base al numero ID del cliente. Tale metodo di partizionamento rappresenta la scelta logica in quanto quasi tutte le operazioni di business eseguite con il record del cliente utilizzano il numero ID del cliente. Tuttavia, un'importante transazione che non utilizza il numero ID del cliente è la transazione di login. Per il login, è più comune utilizzare nomi utente o indirizzi e-mail invece dei numeri ID del cliente.

L'approccio semplice allo scenario di login è quello di utilizzare una transazione sulla griglia per individuare il record del cliente. Come illustrato in precedenza, tale approccio non esegue la scalabilità.

L'opzione successiva potrebbe essere quella di eseguire il partizionamento sul nome utente o l'e-mail. Questa opzione non è pratica, in quanto tutte le operazioni basate sull'ID cliente diventano transazioni sulla griglia. Inoltre, i clienti del sito potrebbero desiderare di modificare il proprio nome utente o indirizzo e-mail. I prodotti come WebSphere eXtreme Scale richiedono che il valore utilizzato per partizionare i dati resti costante.

La soluzione corretta consiste nell'utilizzo di un indice di ricerca inverso. Con WebSphere eXtreme Scale, è possibile creare una cache nella stessa griglia distribuita della cache che contiene tutti i record utente. Questa cache è altamente disponibile, partizionata e scalabile. Tale cache può essere utilizzata per associare un nome utente o un indirizzo e-mail ad un ID cliente. Questa cache converte l'operazione di login in un'operazione su due partizioni invece di un'operazione sulla griglia. Tale scenario non è pratico come una transazione su partizione singola, ma il livello di prestazione viene scalato linearmente con l'aumento del numero di computer.

Calcolo al momento della scrittura

I valori calcolati comunemente, come le medie o i totali, possono essere costosi da creare perché tali operazioni generalmente richiedono la lettura di un numero di voci elevato. Poiché nella maggior parte delle applicazioni le letture sono più comuni delle scritture, è preferibile calcolare tali valori al momento della scrittura e memorizzare il risultato nella cache. Questa operazione rende le operazioni di scrittura più rapide e scalabili.

Campi facoltativi

Considerare un record utente che contenga un numero di telefono dell'ufficio, di casa e mobile. Per ciascun utente, potrebbero essere definiti tutti, nessuno o qualsiasi combinazione di tali numeri. Se i dati fossero normalizzati, esisterebbero una tabella utente ed una tabella dei numeri telefonici. I numeri telefonici di un determinato utente potrebbero essere individuati utilizzando un'operazione JOIN tra le due tabelle.

La denormalizzazione di tale record non richiede la duplicazione dei dati, perché la maggior parte degli utenti non condivide i numeri di telefono. Al contrario, deve essere possibile che alcuni alloggiamenti siano vuoti nel record utente. Invece di utilizzare una tabella dei numeri di telefono, aggiungere tre attributi a ciascun record utente, uno per ciascun tipo di numero di telefono. Tale aggiunta di attributi elimina l'operazione JOIN e converte l'operazione di ricerca di un numero di telefono per un utente in un'operazione su partizione singola.

Posizionamento di relazioni molti-a-molti

Considerare un'applicazione che tenga traccia dei prodotti e dei negozi in cui i prodotti vengono venduti. Un singolo prodotto viene venduto in molti negozi ed un singolo negozio vende molti prodotti. Si supponga che tale applicazione tracci 50 grandi venditori. Ciascun prodotto viene venduto in un massimo di 50 negozi, ciascuno dei quali vende migliaia di prodotti.

Conservare un elenco dei negozi all'interno dell'entità del prodotto (disposizione A) invece di conservare un elenco dei prodotti all'interno di ciascuna entità del negozio (disposizione B). Analizzando alcune delle transazioni che questa applicazione deve eseguire, è possibile comprendere per quale motivo la disposizione A è più scalabile.

Per prima cosa, si analizzino gli aggiornamenti. Con la disposizione A, la rimozione di un prodotto dall'inventario di un negozio blocca l'entità del prodotto. Se la griglia contiene 10000 prodotti, per eseguire l'aggiornamento è necessario bloccare solo 1/10000 della griglia. Con la disposizione B, la griglia contiene solo 50 negozi, per cui per completare l'aggiornamento è necessario bloccare solo 1/50 della griglia. Quindi, anche se entrambe queste operazioni possono essere considerate operazioni su partizione singola, la disposizione A esegue lo scale out in modo più efficiente.

Ora, considerando le letture con la disposizione A, la ricerca dei negozi in cui viene venduto un prodotto è una transazione su partizione singola che scala ed è rapida perché la transazione trasmette solo una piccola quantità di dati. Con la disposizione B, la transazione diventa una transazione sulla griglia, perché è necessario accedere a ciascuna entità del negozio per verificare se il prodotto viene venduto in tale negozio, rivelando un enorme vantaggio per le prestazioni per la disposizione A.

Scalabilità con dati normalizzati

Un utilizzo valido delle transazioni sulla griglia è quello di scalare l'elaborazione dei dati. Se una griglia dispone di 5 computer e viene distribuita una transazione sulla griglia che esamina circa 100000 record su ciascun computer, la transazione esamina 500000 record. Se il computer più lento nella griglia è in grado di eseguire 10 di tali transazioni al secondo, la griglia è in grado di esaminare 5000000 record al secondo. Se i dati nella griglia vengono raddoppiati, ciascun computer deve esaminare 200000 record e ciascuna transazione esamina 1000000 record. Tale incremento di dati riduce il livello di prestazione del computer più lento a 5 transazioni al secondo, riducendo di conseguenza livello di prestazione della griglia a 5 transazioni al secondo. La griglia esamina 5000000 di record al secondo.

In questo scenario, raddoppiando il numero di computer, ogni computer può tornare al proprio precedente carico di lavoro esaminando 100000 record, consentendo al computer più lento di elaborare 10 di tali transazioni al secondo. Il livello di prestazione della griglia resta invariato a 10 richieste al secondo, ma ora ciascuna transazione elabora 1000000 record, per cui la griglia ha raddoppiato la propria capacità in termini di elaborazione dei record, portandola a 10000000 al secondo.

Per le applicazioni come i motori di ricerca, che devono eseguire operazioni di scalabilità in termini di elaborazione dei dati per adattarsi all'incremento di Internet ed in termini di livello di prestazione per adattarsi al numero sempre crescente di utenti, è necessario creare più griglie, con una condivisione delle richieste sulle griglie. Se è necessario aumentare il livello di prestazione, aggiungere altri computer ed aggiungere un'altra griglia alle richieste di servizio. Se è necessario aumentare l'elaborazione dei dati, aggiungere altri computer e tenere costante il numero di griglie.

Gestione dei blocchi

I blocchi hanno dei cicli di vita e differenti tipi di blocchi sono compatibili con gli altri in vari modi. I blocchi devono essere gestiti nell'ordine corretto per evitare problemi di deadlock.

Blocco dei timeout

Ciascuna BackingMap ha un valore di timeout di attesa blocco predefinito. Il valore di timeout viene utilizzato per garantire che un'applicazione non attenda infinitamente in modalità blocco che è necessaria applicare a causa di una condizione di deadlock che si verifica per un errore dell'applicazione. L'applicazione può utilizzare l'interfaccia BackingMap per sovrascrivere il valore di timeout di attesa blocco predefinito L'esempio che segue illustra in che modo impostare il valore di timeout di attesa blocco per la mappa di backup map1 a 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 sia il metodo

initialize che il metodo getSession nell'istanza ObjectGrid. Il parametro del metodo setLockTimeout è un primitivo intero Java che specifica il numero di secondi che eXtreme Scale attende in modalità blocco da concedere. Se una transazione attende più tempo di quanto definito nel valore di timeout di attesa blocco configurato per BackingMap, ne deriverà un'eccezione com.ibm.websphere.objectgrid.LockTimeoutException

Quando si verifica un'eccezione LockTimeoutException, l'applicazione deve determinare se il timeout si è verificato perché l'esecuzione dell'applicazione è più lenta di quanto previsto o perché si è verificata una condizione di di deadlock. Se si è verificata un'effettiva condizione di deadlock, pur incrementando il valore di timeout di attesa blocco, non si eviterà l'eccezione. Incrementando il timeout, occorrerà più tempo prima che l'eccezione si verifichi. Tuttavia, se incrementando il valore di timeout di attesa blocco, non si evita l'eccezione, allora il problema si è verificato perché l'esecuzione dell'applicazione è avvenuta più lentamente di quanto previsto. L'applicazione in questo caso deve determinare quale prestazione è lenta.

Timeout di attesa blocco per ObjectMaps

Il timeout di attesa blocco può essere sovrascritto per una singola istanza ObjectMap utilizzando il metodo ObjectMap.setLockTimeout. Il valore di timeout di blocco influenza tutte le transazioni avviate dopo che sia stato impostato il nuovo valore di timeout. Questo metodo può essere utile quando sono possibili o previsti conflitti di blocco nelle transazioni di tipo select.

Blocchi condivisi, aggiornabili ed esclusivi.

Quando un'applicazione richiama qualsiasi metodo dell'interfaccia ObjectMap, utilizza i metodi find su un indice o esegue una query eXtreme Scale per tentare automaticamente di acquisire un blocco per una voce di mappa che sta per essere acceduta. WebSphere eXtreme Scale utilizza le seguenti modalità blocco basate sul metodo che l'applicazione richiama nell'interfaccia ObjectMap.

- I metodi Get e getAll nell'interfaccia ObjectMap, i metodi di indice e le query acquisiscono un blocco *S*, o una modalità blocco condivisa per la chiave della voce della mappa. La durata per cui è conservato il blocco S dipende dal livello di isolamento della transazione utilizzato. Una modalità blocco S consente la concorrenza tra le transazioni che tentano di acquisire una modalità blocco S o di blocco aggiornabile (blocco U) per la stessa chiave, ma blocca le altre transazioni che tentano di ottenere una modalità blocco esclusivo (blocco X) per la stessa chiave.
- I metodi getForUpdate e getAllForUpdate acquisiscono una modalità blocco *blocco U* , o una modalità blocco aggiornabile per la chiave di una voce della mappa. Il blocco U è conservato fino a quando la transazione non è completata. Una modalità blocco U consente la concorrenza tra le transazioni che acquisiscono una modalità blocco S per la stessa chiave, ma blocca le altre transazioni che tentano di acquisire una modalità blocco U o X per la stessa chiave.
- Le operazioni Put, putAll, Remove, removeAll, insert, update e touch acquisiscono una modalità blocco *blocco X* o una modalità blocco esclusivo per la chiave di una voce della mappa. Il blocco X è conservato fino a quando la transazione non è completa. Una modalità blocco X garantisce che solo una transazione inserisca, aggiorni o rimuova una voce della mappa di un dato valore della chiave. Un blocco X blocca tutte le altre transazioni che tentino di acquisire una modalità blocco S, U o X per la stessa chiave.

- I metodi invalidate global e invalidateAll global acquisiscono un blocco X per ciascuna voce della mappa che è invalidata. Il blocco X è conservato fino a quando la transazione non è compelta. Nessun blocco è richiesto per i metodi invalidate e invalidateAll in locale perché nessuna voce di BackingMap è invalidata dalle chiamate del metodo invalidate in locale.

Dalle definizioni precedenti, è ovvio che una modalità blocco S è più debole di una modalità blocco U perché consente a più transazioni di essere in esecuzione contemporaneamente quando accedono alla stessa voce della mappa. La modalità blocco U è leggermente più forte della modalità blocco S perché blocca le altre transazioni che richiedono una modalità blocco U o X. La modalità blocco S blocca solo le altre transazioni che richiedono una modalità blocco X. Questa piccola differenza è importante per evitare che si verifichino alcuni deadlock. La modalità blocco X è la modalità blocco più forte in assoluto perché essa blocca tutte le transazioni che tentano di ottenere una modalità blocco S, U o X per la stessa voce della mappa. L'effetto pratico di una modalità blocco X è garantire che solo una transazione possa inserire, aggiornare o rimuovere una voce della mappa ed evitare che gli aggiornamenti vadano persi quando più di una transazione sta tentando di aggiornare la stessa voce della mappa.

La tabella di seguito riportata rappresenta una matrice di compatibilità della modalità blocco che riassume le modalità blocco descritte e che è possibile utilizzare per determinare quali modalità blocco sono compatibili con le altre. Per leggere questa matrice, la riga nella matrice indica una modalità blocco che è già stata concessa. La colonna indica la modalità blocco che viene richiesta da un'altra transazione. Se nella colonna è visualizzato Sì, vuol dire che la modalità blocco richiesta dall'altra transazione viene concessa perché essa è compatibile con la modalità blocco che è già stata concessa. Nulla indica che la modalità blocco non è compatibile e l'altra transazione deve attendere che la prima transazione rilasci il blocco che le appartiene.

Tabella 4. Matrice della compatibilità della modalità di blocco

Blocco	Tipo blocco S (condiviso)	Tipo blocco U (aggiornabile)	Tipo blocco X (esclusivo)	Forza
S (condiviso)	Sì	Sì	No	più debole
U (aggiornabile)	Sì	No	Sì	normale
X (esclusivo)	No	No	No	il più forte

Deadlock di chiusura

Considerare la sequenza di modalità blocco richiesta:

1. blocco X è concesso alla transazione 1 per la chiave key1.
2. Blocco X è concesso alla transazione 2 per la chiave key2.
3. Blocco X richiesto dalla transazione 1 per la chiave key2. (La transazione 1 blocca attendendo il blocco posseduto dalla transazione 2.)
4. Blocco X richiesto dalla transazione 2 per la chiave key1. (La transazione 2 blocca in attesa del blocco posseduto dalla transazione 1.)

La precedente sequenza è il classico esempio di deadlock di due transazioni che tentano di acquisire più di un solo blocco e ciascuna transazione acquisisce i blocchi in un diverso ordine. Per evitare questo deadlock, ciascuna transazione deve ottenere più blocchi nello stesso ordine. Se viene utilizzata la strategia di blocco OTTIMISTICO e dall'applicazione non viene mai utilizzato il metod Flush nell'interfaccia ObjectMap allora le modalità blocco vengono richieste dalla transazione solo durante il ciclo di commit. Durante il ciclo di commit, eXtreme

Scale determina le chiavi per le voci della mappa che necessitano di essere bloccate e richiede le modalità blocco nella sequenza della chiave (comportamento deterministico). Con questo metodo, eXtreme Scale evita la grande maggioranza di deadlock classici. Tuttavia, eXtreme Scale non evita e non può farlo tutti i possibili scenari di deadlock. Esistono un po' di scenari che l'applicazione deve considerare. Di seguito sono riportati gli scenari di cui l'applicazione deve essere consapevole e per i quali assumere le relative azioni preventive.

Esiste uno scenario in cui eXtreme Scale è in grado di rilevare un deadlock senza dover attendere che si realizzi un timeout di attesa di blocco Se si verifica questo scenario, verrà generata un'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("Lynn");
// Lynn had a birthday, so we make her 1 year older.
p.setAge( p.getAge() + 1 );
person.put( "Lynn", p );
sess.commit();
```

In questa situazione l'amico di Lynn desidera renderla più vecchia di quanto non lo sia attualmente ed entrambi Lynn ed il suo amico eseguono contemporaneamente questa transazione. In questa situazione, entrambe le transazioni posseggono una modalità blocco S sulla voce Lynn della mappa PERSON in quanto è stato richiamato il metodo person.get("Lynn"). Per effetto della chiamata al metodo person.put ("Lynn", p,) entrambe le transazioni tentano di aggiornare la modalità blocco S in una modalità blocco X. Entrambi i blocchi delle transazioni attendono che le altre transazioni rilascino la modalità blocco S che esse posseggono. Come risultato, si verifica un deadlock perché esiste una condizione di attesa ciclica tra le due transazioni. Si realizza una condizione di attesa ciclica quando più di una transazione tenta di promuovere un blocco da una modalità più debole ad una più forte per la stessa voce della mappa. In questo scenario si genererà un'eccezione LockDeadlockException invece di un'eccezione LockTimeoutException.

L'applicazione può evitare l'eccezione LockDeadlockException nell'esempio precedente utilizzando una strategia di blocco ottimistico invece di una strategia di blocco pessimistico. L'utilizzo della strategia di blocco ottimistico è la soluzione preferibile quando la mappa è per lo più letta e gli aggiornamenti alla mappa non sono frequenti. Se deve essere utilizzata la strategia di blocco pessimistico, il metodo getForUpdate può essere utilizzato invece del metodo Get nell'esempio sopra riportato oppure può essere utilizzato un livello di isolamento della transazione di tipo TRANSACTION_READ_COMMITTED

Per Per ulteriori informazioni, consultare l'argomento sulle strategie di blocco nella *Panoramica sul prodotto.*

Utilizzando il livello di isolamento della transazione TRANSACTION_READ_COMMITTED si evita che il blocco S che è acquisito dal metodo Get sia conservato fino a quando la transazione non sia completata. Se la chiave non è mai invalidata nella cache transazionale, sono ancora garantite letture ripetibili.

consultare l'argomento relativo al blocco della voce della mappa in *Guida alla gestione* per ulteriori informazioni.

Un'alternativa per modificare il livello di isolamento è utilizzare il metodo getForUpdate. La prima transazione per chiamare il metodo getForUpdate acquisisce una modalità blocco U invece di un blocco S. Questa modalità blocco provoca il blocco della seconda transazione quando essa richiama il metodo getForUpdate poiché solo ad una transazione può essere concessa la modalità blocco U. Poiché la seconda transazione è bloccata essa non fa propria alcuna modalità blocco nella voce della mappa Lynn. La prima transazione non si blocca quando essa tenta di aggiornare la modalità blocco U in una modalità blocco X per effetto della chiamata al metodo Put dalla prima transazione. Questa funzione dimostra perché la modalità blocco U è denominata modalità blocco *aggiornabile*. Quando la prima transazione è completata, la seconda transazione si sblocca e viene concessa la modalità blocco U. Un'applicazione può evitare lo scenario di deadlock che promuove il blocco utilizzando il metodo getForUpdate invece del metodo Get quando viene utilizzata la strategia di blocco pessimistico.

Importante: Questa soluzione non impedisce alle transazioni in sola lettura di essere in grado di leggere una voce della mappa. Le transazioni in sola lettura richiamano il metodo Get, ma non richiamano mai i metodi Put, Insert, Update o Remove. La concorrenza è alta proprio come quando viene utilizzato il metodo Get. La riduzione nella concorrenza si verifica solo quando il metodo getForUpdate viene richiamato da più di una transazione per la stessa voce della mappa.

È necessario conoscere quando una transazione richiama il metodo getForUpdate per più di una voce della mappa per garantire che le modalità di blocco U siano acquisite nello stesso ordine da ciascuna transazione. Ad esempio, supponiamo che la prima transazione richiami il metodo getForUpdate per la chiave 1 e il metodo getForUpdate per la chiave 2. Un'altra transazione contemporaneamente richiama il metodo getForUpdate per le stesse chiavi, ma in ordine inverso. Questa sequenza provoca il classico deadlock perché più blocchi sono acquisiti in ordine diverso dalle differenti transazioni. L'applicazione ancora necessita di garantire che ciascuna transazione acceda più voci della mappa nella sequenza della chiave per garantire che non si verifichi il deadlock. Poiché il blocco U è conservato nel momento in cui viene richiamato il metodo getForUpdate piuttosto che al momento del commit, eXtreme Scale non può ordinare le richieste di blocco come fa durante il ciclo di commit. In questo caso, l'applicazione deve controllare l'ordinamento del blocco.

Utilizzando prima il metodo Flush nell'interfaccia ObjectMap, un commit può introdurre ulteriori considerazioni sull'ordinamento dei blocchi. Il metodo Flush viene generalmente utilizzato per forzare le modifiche apportate alla mappa al di fuori del backend attraverso il plug-in Loader. In questa situazione, il backend utilizza il proprio gestore di blocco per controllare la concorrenza in modo che la condizione di attesa del blocco e il deadlock possano verificarsi nel backend piuttosto che nel gestore del blocco. eXtreme Scale. Considerare la seguente transazione:

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

```
    p.setAge( p.getAge() + 1 );
    sess.commit();
    activeTran = false;
}
finally
{
    if ( activeTran ) sess.rollback();
}
```

Supponiamo che un'altra transazione anche abbia aggiornato la persona Tom, abbia richiamato il metodo Flush e successivamente aggiornato la persona Lynn. Se si è verificata questa situazione, l'alternanza delle due seguenti transazioni, provocherà una condizione di deadlock del database.

```
X lock is granted to transaction 1 for "Lynn" when flush is executed.
X lock is granted to transaction 2 for "Tom" when flush is executed..
X lock requested by transaction 1 for "Tom" during commit processing.
(Transaction 1 blocks waiting for lock owned by transaction 2.)
X lock requested by transaction 2 for "Lynn" during commit processing.
(Transaction 2 blocks waiting for lock owned by transaction 1.)
```

Questo esempio dimostra che l'utilizzo del metodo Flush può provocare che si verifichi un deadlock nel database invece che in eXtreme Scale. Questo esempio di deadlock può verificarsi indipendentemente dal tipo di strategia utilizzata. L'applicazione deve porre attenzione ad evitare che questo tipo di deadlock si verifichi quando si utilizza il metodo Flush e quando un programma di caricamento è collegato a BackingMap. L'esempio precedente illustra anche il motivo per cui eXtreme Scale dispone di un meccanismo di timeout di attesa del blocco Una transazione che è in attesa di un blocco del database potrebbe essere in attesa mentre possiede un blocco sulla voce della mappa eXtreme Scale. Conseguentemente, i problemi a livello di database possono causare tempi di attesa eccessivi per una modalità blocco eXtreme Scale e generare un'eccezione LockTimeoutException.

Scenari comuni di deadlock

Le sezioni di seguito riportate descrivono alcuni dei più comuni scenari di deadlock e suggeriscono il modo per evitarli.

Scenario: deadlock di chiave singola

Gli scenari di seguito riportati descrivono in che modo possono verificarsi i deadlock quando una singola chiave è acceduta utilizzando un blocco S e successivamente aggiornata. Quando ciò accade contemporaneamente in due transazioni, ne risulterà un deadlock.

Tabella 5. Scenario di deadlock di singola chiave

	Thread 1	**Thread 2**	
1	session.begin()	session.begin()	Ciascun thread stabilisce una transazione indipendente.
2	map.get(key1)	map.get(key1)	Blocco S concesso ad entrambe le transazioni per la chiave key1.
3	map.update(Key1,v)		Nessun blocco U. Aggiornamento eseguito nella cache transazionale.
4		map.update(key1,v)	Nessun blocco U. Aggiornamento eseguito nella cache transazionale.

Tabella 5. Scenario di deadlock di singola chiave (Continua)

	Thread 1	**Thread 2**	
5	session.commit()		Bloccato: il blocco S per la chiave key1 non può essere aggiornato in un blocco X perché il thread 2 ha un blocco S.
6		session.commit()	Deadlock: il blocco S per la chiave key1 non può essere aggiornato in un blocco X perché T1 ha un blocco S.

Tabella 6. deadlock di singola chiave, continuato

	Thread 1	**Thread 2**	
1	session.begin()	session.begin()	Ciascun thread stabilisce una transazione indipendente.
2	map.get(key1)		Blocco S concesso per la chiave key1
3	map.getForUpdate(key1,v)		Blocco S è aggiornato in un blocco U per la chiave key1.
4		map.get(key1)	Blocco S concesso per la chiave key1.
5		map.getForUpdate(key1,v)	Bloccato: T1 già disponne di un blocco U.
6	session.commit()		Deadlock: il blocco U per la chiave key1 non può essere aggiornato
7		session.commit()	Deadlock: il blocco S per la chiave key1 non può essere aggiornato.

Tabella 7. Deadlock di singola chiave, continuato,

	Thread 1	**Thread 2**	
1	session.begin()	session.begin()	Ciascun thread stabilisce una transazione indipendente
2	map.get(key1)		Blocco S concesso per la chiave key1.
3	map.getForUpdate(key1,v)		Blococ S è aggiornato in un blocco U per la chiave key1
4		map.get(key1)	Blocco S è concesso per la chiave key1.
5		map.getForUpdate(key1,v)	Bloccato: Thread 1 dispone già di un blocco U.
6	session.commit()		Deadlock: Il blocco U per una chiave key1 non può essere aggiornato in un blocco X perché il Thread 2 dispone di un blocco S.

Se viene utilizzato ObjectMap.getForUpdate per evitare il blocco S allora il deadlock è evitato:

Tabella 8. Deadlock di singola chiave, continuato

	Thread 1	Thread 2	
1	session.begin()	session.begin()	Ciascun thread stabilisce una transazione indipendente.
2	map.getForUpdate(key1)		Blocco U concesso al thread 1 per la chiave key1.
3		map.getForUpdate(key1)	La richiesta di blocco U è bloccata.
4	map.update(key1,v)	<blocked>	
5	session.commit()	<blocked>	Il blocco U per la chiave key1 può essere automaticamente aggiornato in un blocco X.
6		<released>	Il blocco U infine è concesso alla chiave key1 per il thread 2.
7		map.update(key2,v)	Blocco U concesso al thread 2 per la chiave key2.
8		session.commit()	Blocco U per la chiave key1 può essere aggiornato nel blocco X con esito positivo.

Soluzioni

1. Utilizzare il metodo getForUpdate invece del metodo Get per acquisire un blocco U invece di un blocco S.
2. Utilizzare un livello di isolamento della transazione read committed per evitare di conservare i blocchi S. Riducendo il livello di isolamento aumenta la possibilità di non-repeatable read. Tuttavia, le non-repeatable read sono possibili solo se la cache della transazione è esplicitamente invalidata.
3. Utilizzare la strategia di blocco ottimistico. L'utilizzo della strategia di blocco ottimistico richiede la gestione ottimistica di eccezioni di conflitto.

Scenario: deadlock di chiavi multiple ordinate

Questo scenario descrive cosa accade se due transazioni tentano di aggiornare la stessa voce direttamente e conservano i blocchi S sulle altre voci.

Tabella 9. Scenario di chiave multipla ordinata

	Thread 1	Thread 2	
1	session.begin()	session.begin()	Ciascun thread stabilisce una transazione indipendente.
2	map.get(key1)	map.get(key1)	Blocco S concesso ad entrambe le transazioni per la chiave key1.
3	map.get(key2)	map.get(key2)	Blococ S concesso ad entarmbe le transazioni per la chiave key2.
4	map.update(key1,v)		Nessun blocco U. Aggiornamento eseguito nella cache transazionale.
5		map.update(key2,v)	Nessun blocco U. Aggiornamento eseguito nella cache transazionale.

Tabella 9. Scenario di chiave multipla ordinata (Continua)

	Thread 1	Thread 2	
6.	session.commit()		Bloccato: Il blocco S per la chiave key 1 non può essere aggiornato in un blocco X perché il thread 2 dispone di un blocco S.
7		session.commit()	Deadlock: il blocco S per la chiave 2 non può essere aggiornato perché il thread 1 dispone di un blocco S.

È possibile utilizzare il metodo ObjectMap.getForUpdate per evitare il blocco S, allora è possibile evitare il deadlock:

Tabella 10. Scenario deadlock di chiave multipla ordinata, continuato

	Thread 1	Thread 2	
1	session.begin()	session.begin()	Ciascun thread stabilisce una transazione indipendente.
2	map.getForUpdate(key1)		Blocco U concesso alla transazione T1 per la chiave key1.
3		map.getForUpdate(key1)	La richiesta di blocco U è bloccata.
4	map.get(key2)	<blocked>	Blocco S concesso per T1 per la chiave key2.
5	map.update(key1,v)	<blocked>	
6	session.commit()	<blocked>	Il blocco U per la chiave key1 può essere automaticamente aggiornato in un blocco X.
7		<released>	Blocco U è infine concesso per la chiave key1 per T2
8		map.get(key2)	Blocco S concesso a T2 per la chiave key2.
9		map.update(key2,v)	Blocco U concesso a T2 per la chiave key2.
10		session.commit()	Il blocco U per la chiave key1 può essere automaticamente aggiornato in un blocco X.

Soluzioni

1. Utilizzare getForUpdate invece del metodo Get per acquisire un blocco U direttamente per la prima chiave. Questa strategia funziona solo se l'ordine del metodo è deterministico.
2. Utilizzare un livello di isolamento della transazione read committed per evitare di conservare i blocchi S. Questa soluzione è la più semplice da implementare se l'ordine del metodo non è deterministico. Riducendo il livello di isolamento aumenta la possibilità di non-repeatable read. Tuttavia, le non-repeatable read sono possibili solo se la cache della transazione è esplicitamente invalidata.
3. Utilizzare la strategia di blocco ottimistico. L'utilizzo della strategia di blocco ottimistico richiede la gestione di eccezioni di conflitto ottimistica.

Scenario: Al di fuori dell'ordine con il blocco U

Se non può essere garantito l'ordine con cui vengono richieste le chiavi, allora è possibile che si verifichi ancora un deadlock:

Tabella 11. Scenario al di fuori dell'ordine con un blocco U

	Thread 1	**Thread 2**	
1	session.begin()	session.begin()	Ciascun thread stabilisce una transazione indipendente.
2	map.getforUpdate(key1)	map.getForUpdate(key2)	Blocchi U concessi con esito positivo per le chiavi Key1 e key2.
3	map.get(key2)	map.get(key1)	Blocco S concesso per le chiavi key1 e key2.
4	map.update(key1,v)	map.update(key2,v)	
5	session.commit()		Blocco U non può essere aggiornato in un blocco X perché T2 dispone di un blocco S.
6		session.commit()	Blocco U non può essere aggiornato in un blocco X perché T1 dispone di un blocco S.

Soluzioni

1. Racchiudere tutte le attività con un blocco globale singolo (mutex). Questo metodo riduce la concorrenza, ma gestisce tutti gli scenari quando l'accesso e l'ordine non sono deterministici.
2. Utilizzare un livello di isolamento della transazione read committed per evitare di conservare i blocchi S. Questa soluzione è la più semplice per implementare se l'ordine del metodo non è deterministico e fornisce il maggior quantitativo di concorrenze. Riducendo il livello di isolamento aumenta la possibilità di non-repeatable read. Tuttavia, le non-repeatable read sono possibili solo se la cache della transazione è esplicitamente invalidata.
3. Utilizzare la strategia di blocco ottimistico. L'utilizzo della strategia di blocco ottimistico richiede la gestione ottimistica di eccezioni di conflitto.

Gestione dell'eccezione negli scenari di blocco

Gli esempi precedenti non prevedono alcuna gestione dell'eccezione. Per evitare che i blocchi siano conservati per un tempo eccessivo quando si verifica un'eccezione LockTimeoutException o un'eccezione LockDeadlockException, un'applicazione deve garantire che essa rilevi le eccezioni impreviste e richiami il metodo rollback quando si verifica qualcosa di imprevisto. Modificare il precedente frammento di codice come dimostrato nell'esempio di seguito riportato:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
    sess.begin();
    activeTran = true;
    Person p = (IPerson)person.get("Lynn");
    // Lynn had a birthday, so we make her 1 year older.
    p.setAge( p.getAge() + 1 );
    person.put( "Lynn", p );
    sess.commit();
    activeTran = false;
}
finally
{
    if ( activeTran ) sess.rollback();
}
```

Alla fine il blocco nel frammento di codice garantisce che una transazione esegua il rollback quando si verifica un'eccezione imprevista. Esso non solo gestisce l'eccezione LockDeadlockException, ma qualsiasi altra eccezione imprevista che potrebbe verificarsi. Infine il blocco gestisce il caso in cui si verifica un'eccezione durante il richiamo di un metodo commit. Questo esempio non è il solo modo di trattare le eccezioni impreviste e potrebbero esserci dei casi in cui un'applicazione desidera rilevare alcune delle eccezioni impreviste che possono verificarsi e visualizzare una delle sue eccezioni di applicazione. È possibile aggiungere i blocchi rilevati opportunamente, ma l'applicazione deve garantire che il frammento di codice non termini senza aver completato la transazione.

Strategie di blocco

Le strategie di blocco includono le strategie pessimistica, ottimistica e none Per scegliere una strategia di blocco, è necessario considerare questioni come la percentuale di ciascun tipo di operazione che si ha se oppure se si utilizza o meno un programma di caricamento e così via.

I blocchi sono collegati dalle transazioni. È possibile specificare le seguenti impostazioni di blocco:

- **Nessun blocco**: l'esecuzione senza l'impostazione di blocco è l'opzione più veloce. Se si stanno utilizzando dati di sola lettura, il blocco potrebbe non essere necessario.
- **Blocco pessimistico**: acquisisce i blocchi sulle voci e mantiene i blocchi fino al commit. Questa strategia di blocco fornisce una buona congruenza a spese della velocità di trasmissione.
- **Blocco ottimistico**: prende un'immagine precedente di ogni record che la transazione tocca e la confronta con i valori correnti della voce quando viene seguito il commit della transazione. Se i valori della voce cambiano, la transazione esegue il rollback. Non vengono mantenuti blocchi fino al commit. Questa strategia di blocco fornisce una migliore simultaneità della strategia pessimistica, con il rischio del rollback della transazione e il dispendio di memoria per la creazione di un'ulteriore copia della voce.

Impostare la strategia di blocco sulla BackingMap. Non è possibile modificare la strategia di blocco di ogni transazione. Di seguito è riportato un esempio di frammento XML che mostra come impostare la modalità di blocco su una mappa utilizzando il file XML, presupponendo che `cc` sia lo spazio dei nomi per lo spazio dei nomi `objectgrid/config`:

```
<cc:backingMap name="RuntimeLifespan" lockStrategy="PESSIMISTIC" />
```

Blocco pessimistico

Utilizzare la strategia di blocco pessimistico per le mappe in lettura e scrittura quando altre strategie di blocco non sono possibili. Quando una mappa ObjectGrid è configurata per utilizzare una strategia di blocco pessimistico, si ottiene un blocco della transazione pessimistica per una voce della mappa quando una transazione prende prima la voce da BackingMap. Il blocco pessimistico è conservato fino a quando l'applicazione non completa la transazione. Generalmente la strategia di blocco pessimistico viene utilizzata nelle seguenti situazioni:

- Quando viene configurata la BackingMap con o senza un programma di caricamento e le informazioni sul controllo versioni non sono disponibili.
- Quando viene utilizzata la BackingMap direttamente da un'applicazione che ha necessità di una guida da eXtreme Scale per il controllo della concorrenza.

- Quando le informazioni sul controllo versioni sono disponibili, ma le transazioni di aggiornamento sono frequentemente in conflitto con le voci di backup risultando in un aggiornamento pessimistico non riuscito.

Poiché la strategia di blocco pessimistico ha il maggiore impatto sulle prestazioni e la scalabilità, questa strategia dovrebbe essere utilizzata per le mappe in lettura e scrittura solo quando altre strategie di blocco non sono praticabili. Ad esempio, queste situazioni potrebbero includere quando si verifica di frequente che un aggiornamento ottimistico non riesce o quando il recupero da un errore ottimistico è difficile da gestire per un'applicazione.

Blocco ottimistico

La strategia di blocco ottimistico presuppone che nessuna delle due transazioni possa tentare di aggiornare la stessa voce della mappa durante l'esecuzione contemporanea. A causa di questa convinzione, la modalità blocco non ha necessità di essere conservata per il ciclo di vita della transazione poiché è improbabile che più di una transazione possa aggiornare la voce della mappa contemporaneamente. La strategia di blocco ottimistico è generalmente utilizzata nelle seguenti situazioni:

- quando viene configurata la BackingMap con o senza un programma di caricamento e le informazioni sul controllo versioni sono disponibili.
- Quando una BackingMap ha per lo più transazioni che eseguono operazioni di lettura. Le operazioni di inserimento, aggiornamento e rimozione delle voci della mappa non si verificano spesso nella BackingMap.
- Quando una BackingMap viene inserita, aggiornata o rimossa più frequentemente di quanto non sia letta, ma le transazioni raramente sono in conflitto sulla stessa voce della mappa.

Come la strategia di blocco pessimistico, i metodi nell'interfaccia ObjectMap determinano in che modo eXtreme Scale tenta automaticamente di acquisire una modalità blocco per la voce della mappa che sta per essere acceduta. Tuttavia, esistono le seguenti differenze tra le strategie pessimistica e ottimistica:

- Come la strategia di blocco pessimistico, una modalità blocco S viene acquisita dai metodi Get e getAll quando viene richiamato il metodo. Tuttavia, con un blocco pessimistico, la modalità blocco S non è conservata fino a quando la transazione non è completa. Invece la modalità blocco S viene rilasciata prima che il metodo ritorni all'applicazione. Lo scopo per acquisire la modalità blocco è tale per cui eXtreme Scale può garantire che solo i dati su cui è stato eseguito il commit dalle altre transazioni è visibile alla transazione corrente. Dopo che eXtreme Scale ha verificato che è stato eseguito il commit dei dati, la modalità blocco S viene rilasciata. Al momento del commit viene eseguito un controllo sulla versione ottimistico per garantire che nessuna altra transazione abbia modificato la voce della mappa dopo che la transazione corrente ha rilasciato la propria modalità blocco S. Se non viene eseguito il fetch di una voce dalla mappa prima che essa sia aggiornata, invalidata o eliminata, eXtreme Scale il runtime implicitamente esegue il fetch della voce della mappa. Questa operazione implicita di caricamento viene eseguita per ottenere il valore corrente al momento in cui è stato richiesto di modificare la voce.
- Diversamente dalla strategia di blocco pessimistico, i metodi getForUpdate e getAllForUpdate vengono gestiti esattamente come i metodi Get e getAll quando viene utilizzata la strategia di blocco ottimistico. Cioè una modalità blocco S viene acquisita all'avvio del metodo e la modalità blocco S viene rilasciata prima di ritornare all'applicazione.

Tutti gli altri metodi ObjectMap vengono gestiti esattamente come viene gestita la strategia di blocco pessimistico. Cioè quando viene richiamato il metodo Commit, si ottiene una modalità blocco X per la voce della mappa che è stata inserita, aggiornata, rimossa,toccata o invalidata e la modalità blocco X è conservata fino a quando la transazione non completa l'elaborazione del commit.

La strategia di blocco ottimistico presuppone che nessuna transazione in esecuzione contemporaneamente tenti di aggiornare la stessa voce della mappa. A causa di questo assunto, la modalità blocco non necessita di essere conservata per il ciclo di vita della transazione poiché è improbabile che più di una transazione possa aggiornare contemporaneamente la voce della mappa. Tuttavia, poiché non è conservata una modalità di blocco, un'altra transazione concomitante potrebbe potenzialmente aggiornare la voce della mappa dopo che la transazione corrente ha rilasciato la modalità blocco S.

Per gestire questa possibilità, eXtreme Scale ottiene un blocco X al momento del commit ed esegue un controllo sulla versione ottimistico per verificare che nessun'altra transazione abbia modificato la voce della mappa dopo che la transazione corrente abbia letto la voce della mappa da BackingMap. Se un'altra transazione modifica la voce della mappa, il controllo sulla versione non riesce e si verifica un'eccezione OptimisticCollisionException. Questa eccezione forza la transazione corrente ad eseguire il roll back e l'applicazione deve tentare l'intera transazione nuovamente. La strategia di blocco ottimistico è molto utile quando una mappa è per lo più letta ed è improbabile che si verifichino aggiornamenti alla stessa voce della mappa.

Nessun blocco

Quando una BackingMap viene configurata per non utilizzare alcuna strategia di blocco, non si ottiene alcun blocco della transazione per una voce della mappa.

L'utilizzo di nessuna strategia di blocco è utile quando un'applicazione utilizza Hibernate per ottenere i dati permanenti. In questo scenario, la BackingMap viene configurata senza un programma di caricamento e il gestore della persistenza utilizza la BackingMap come un cache di dati. In questo scenario, il gestore della persistenza fornisce un controllo sulla concorrenza tra le transazioni che accedono le stesse voci della mappa.

WebSphere eXtreme Scale non ha necessità di ottenere alcun blocco sulla transazione a scopo di controllo della concorrenza. Questa situazione presuppone che il gestore della persistenza non rilasci i propri blocchi sulla transazione prima di aggiornare la mappa ObjectGrid con le modifiche sulle quali è stato eseguito il commit. Se il gestore della persistenza rilascia i suoi blocchi, allora deve essere utilizzata una strategia di blocco ottimistico o pessimistico. Ad esempio, supponiamo che il gestore della persistenza di un contenitore EJB stia aggiornando una mappa ObjectGrid con i dati sui quali è stato eseguito il commit nella transazione gestita dal contenitore EJB. Se l'aggiornamento della mappa ObjectGrid si verifica prima che siano rilasciati i blocchi della transazione del gestore della persistenza, allora è possibile non utilizzare alcuna strategia di blocco. Se si verifica l'aggiornamento della mappa ObjectGrid dopo che sono stati rilasciati i blocchi della transazione del gestore della persistenza, allora devi utilizzare sia la strategia di blocco ottimistico che quella di blocco pessimistico.

Un altro scenario in cui non può essere utilizzata alcuna strategia di blocco, si ha quando l'applicazione utilizza direttamente una BackingMap e un programma di caricamento è configurato per la mappa. In questo scenario, il programma di

caricamento utilizza il supporto del controllo della concorrenza che viene fornito da RDBMS (Sistema di gestione del database relazionale) utilizzando sia JDBC (Java database connectivity) che Hibernate per accedere i 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 con controllo sulla versione guida nel raggiungere i risultati migliori nella concorrenza e nelle prestazioni. Per ulteriori informazioni relative all'approccio di blocco ottimistico, consultare la sezione OptimisticCallback nelle per le informazioni relative alle considerazioni sul programma di caricamento in *Guida alla gestione*. Se si sta utilizzando un programma di caricamento che utilizza il supporto di blocco pessimistico di un backend sottostante si potrebbe desiderare di utilizzare il parametro forUpdate che viene passato nel metodo Get dell'interfaccia del programma di caricamento. Impostare questo parametro su true se è stato utilizzato il metodo getForUpdate dell'interfaccia ObjectMap da un'applicazione per ottenere i dati. Il programma di caricamento può utilizzare questo parametro per determinare se richiedere un blocco aggiornabile per la riga che si sta leggendo. Ad esempio, il DB2 ottiene un blocco aggiornabile quando l'istruzione select contiene una clausola `FOR UPDATE`. Questo approccio offre la stessa possibilità di prevenire un deadlock che è descritta in "Blocco pessimistico" a pagina 138.

Migliori pratiche per la prestazione del blocco

Le strategie di blocco e le impostazioni di isolamento della transazione influenzano le prestazioni dell'applicazione.

Recupero di un'istanza della cache

per ulteriori informazioni, econsultare le informazioni relative al blocco della voce della mappa nella sezione *Guida alla gestione*.

Strategia di blocco pessimistico

Utilizzare la strategia di blocco pessimistico per operazioni di lettura e scrittura della mappa in cui le chiavi spesso sono in conflitto. La strategia di blocco pessimistico ha il maggiore impatto sulle prestazioni.

L'isolamento della transazione read committed e read uncommitted

Quando si utilizza una strategia di blocco pessimistico, impostare il livello di isolamento della transazione utilizzando il metodo Session.setTransactionIsolation. Per l'isolamento read committed o read uncommitted, utilizzare gli argomenti Session.TRANSACTION_READ_COMMITTED o Session.TRANSACTION_READ_UNCOMMITTED a seconda dell'isolamento. Per reimpostare il livello di isolamento della transazione per il comportamento di blocco pessimistico predefinito, utilizzare il metodo Session.setTransactionIsolation con l'argomento Session.REPEATABLE_READ.

L'isolamento read committed riduce la durata di blocchi condivisi il che può migliorare la concorrenza e ridurre la possibilità di deadlock. Questo livello di isolamento dovrebbe essere utilizzato quando una transazione non necessita di garantire che i valori in lettura restino non modificati per la durata della transazione.

Utilizzare un isolamento uncommitted read quando la transazione non necessita di vedere i dati sui quali è stato eseguito il commit.

Strategia di blocco ottimistico

La configurazione predefinita è il blocco ottimistico. Questa strategia migliora sia le prestazioni che la scalabilità rispetto alla strategia pessimistica. Utilizzare questa strategia quando le proprie applicazioni possono tollerare che alcuni aggiornamenti ottimistici non riescano pur essendo ancora le prestazioni migliori rispetto alla strategia pessimistica. Questa strategia è eccellente per le operazioni in lettura e per applicazioni di aggiornamento non frequenti.

Plug-in OptimisticCallback

La strategia di blocco ottimistico effettua una copia delle voci di cache e le confronta in base a quanto necessario. Questa operazione può essere onerosa perché copiare la voce potrebbe coinvolgere una clonazione o serializzazione. Per implementare le prestazioni in modo che siano quanto più veloci possibili, implementare il plug-in personalizzato per le mappe non entità.

Per ulteriori informazioni, vedere Per ulteriori informazioni, consultare le informazioni relative al plug-in OptimisticCallback in *Panoramica sul prodotto*.

Utilizzare i campi versione per le entità

Quando si utilizza un blocco ottimistico con le entità, utilizzare l'annotazione @Version o l'attributo equivalente nel file descrittore dei metadati dell'entità. L'annotazione della versione fornisce a ObjectGrid uno metodo molto efficace per tenere traccia della versione di un oggetto. Se l'entità non dispone di un campo versione ed è utilizzato il blocco ottimistico per l'entità, allora l'intera entità deve essere copiata e confrontata.

Strategia di blocco None

Utilizzare la strategia di blocco None per le applicazioni che sono solo in lettura. La strategia di blocco None non ottiene dei blocchi o utilizza il gestore blocco. Perciò, questa strategia offre più concorrenza, prestazioni e scalabilità.

Blocchi della voce della mappa con query ed indici

Questo argomento descrive come le API di Query eXtreme Scale e il plug-in di indicizzazione MapRangeIndex interagiscono con i blocchi e alcune migliori pratiche per incrementare la concorrenza e ridurre i deadlock quando si utilizza la strategia di blocco pessimistico per le mappe.

Panoramica

L'API di Query di ObjectGrid consente query SELECT su oggetti della cache ObjectMap ed entità. Quando è in esecuzione una query, il motore di query utilizza un MapRangeIndex quando possibile per trovare le chiavi corrispondenti che trovano corrispondenza nei valori nella clausola WHERE della query o per collegare le relazioni. Quando non è disponibile un indice, il motore di query effettua la scansione in una o più mappe per trovare le voci opportune. Sia il motore di query che i plug-in dell'indice acquisiranno blocchi per verificare i dati congrui a seconda della strategia di blocco, del livello di isolamento della transazione e dello stato della transazione.

Blocchi con plug-in HashIndex plug-in

Il plug-in HashIndex di eXtreme Scale consente di trovare le chiavi basate su un singolo attributo memorizzato nel valore della voce della cache. L'indice meomorizza il valore indicizzato in una struttura dati separata dalla mappa della cache. L'indice convalida le chiavi rispetto alle voci della mappa prima di ritornare all'utente per provare ad ottenere una serie precisa di risultati. Quando viene utilizzata la strategia di blocco pessimisticoe l'indice viene utilizzato rispetto ad un'istanza ObjectMap locale (verso un ObjectMap client/server), l'indice acquisirà i blocchi per ciascuna voce corrispondente. Quando si utilizza un blocco ottimistico o un ObjectMap remoto, i blocchi vengono sempre immediatamente rilasciati.

Il tipo di blocco che viene acquisito dipende dall'argomento forUpdate passato al metodo ObjectMap.getIndex. L'rgomento forUpdate specifica il tipo di blocco che l'indice dovrebbe acquisire. Se false, viene acquisito un blocco (S) condivisibile e se true viene acquisito un blocco aggiornabile (U).

Se il tipo di blocco è condivisibile, è valida l'impostazione dell'isolamento della transazione per la sessione e influenza la durata del blocco. Per i dettagli, consultare l'argomento relativo all'isolamento della transazione sul modo in cui viene utilizzato l'isolamento della transazione per aggiungere concorrenza alle applicazioni.

Blocchi condivisi con query

Il motore di query eXtreme Scale acquisisce i blocchi S quando è necessario per esaminare le voci della cache per rilevare se esse soddisfano il criterio del filtro della query. Quando si utilizza l'isolamento della transazione repeatable read con un blocco pessimistico, i blocchi S sono conservati solo per gli elementi che sono inclusi nel risultato della query e sono rilasciati per alcune voci che non sono incluse nel risultato. Se si utilizza un livello di isolamento della transazione più basso o un blocco ottimistico, i blocchi S non sono conservati.

Blocchi condivisi con client per query su server

Quando si utilizza la query di eXtreme Scale da un client, la query generalmente è in esecuzione sul server a meno che tutte le mappe o entità a cui si è fatto riferimento nella query non siano in locale per il client (ad esempio: una mappa replicata sul client o un'entità di risultato della query). Tutte le query che sono in esecuzione in una transazione in lettura/scrittura conserveranno i blocchi S come descritto nella sezione precedente. Se la transazione non è una transazione in lettura/scrittura, allora non viene conservata una sessione sul server e i blocchi S vengono rilasciati.

Una transazione in lettura/scrittura viene solo instradata alla partizione primaria ed una sessione viene gestita sul server per la sessione del client. Una transazione può essere promossa in lettura/scrittura nelle seguenti condizioni:

1. Alcune mappe configurate per utilizzare il blocco pessimistico vengono accedute utilizzando i metodi dell'API ObjectMap Get e getAll o i metodi EntityManager.find
2. La transazione viene ripulita facendo si che gli aggiornamenti siano inviati al server.
3. Alcune mappe configurate per utilizzare il blocco ottimistico vengono accedute utilizzando il metodo ObjectMap.getForUpdate o EntityManager.findForUpdate.

Blocchi aggiornabili con query

I blocchi condivisibili sono utili quando è importante la concorrenza e la congruenza. esso garantisce che un valore della voce non si modifica per la vita della transazione. Nessun'altra transazione può modificare il valore mentre è conservato qualche altro blocco S e solo un'altra transazione può determinare l'intento di aggiornare la voce. consultare l'argomento Modalità di blocco pessimistico per i dettagli sulle modalità blocco S, U e X.

I blocchi aggiornabili vengono utilizzati per identificare l'intento di aggiornare una voce della cache quando si utilizza la strategia di blocco pessimistico. Essa consente la sincronizzazione tra le transazioni che desiderano di modificare una voce della cache. Le transazioni possono ancor visualizzare la voce utilizzando un blocco S, ma alle altre transazioni è impedito acquisire un blocco U o un blocco X. In molti scenari, è necessaria l'acquisizione di un blocco U senza acquisire prima un blocco S per evitare i deadlock. consultare l'argomento Modalità di blocco pessimistica per gli esempi comuni di deadlock.

Le interfacce ObjectQuery e EntityManager forniscono il metodo setForUpdate per identificare l'utilizzo destinato al risultato della query. In particolare, il motore di query acquisisce i blocchi U invece dei blocchi S per ciascuna voce della mappa coinvolta nel risultato della query:

```
ObjectMap orderMap = session.getMap("Order");
ObjectQuery q = session.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
session.begin();
// Run the query.  Each order has  U lock
Iterator result = q.getResultIterator();
// For each order, update the status.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
    orderMap.update(o.getId(), o);
}
// When committed, the
session.commit();
```

```
Query q = em.createQuery("SELECT o FROM Order o WHERE o.orderDate=?1");
q.setParameter(1, "20080101");
q.setForUpdate(true);
emTran.begin();
// Run the query.  Each order has  U lock
Iterator result = q.getResultIterator();
// For each order, update the status.
while(result.hasNext()) {
    Order o = (Order) result.next();
    o.status = "shipped";
}
tmTran.commit();
```

Quando l'attributo **setForUpdate** viene abilitato, la transazione viene automaticamente convertita in una transazione in lettura/scrittura e i blocchi sono conservati sul server come previsto. Se la query non può utilizzare alcun indice, allora è necessario eseguire la scansione della mappa il che provocherà dei blocchi temporanei U per le voci della mappa che non soddisfano il risultato della query e l'attesa dei blocchi U per le voci che sono incluse nel risultato.

Isolamento della transazione

Per le transazioni, è possibile configurare ciascuna configurazione della mappa di backup con una delle tre strategie di blocco: pessimistico, ottimistico o none. Quando si utilizza un blocco pessimistico e ottimistico eXtreme Scale utilizza i blocchi condiviso (S), aggiornabile (U) ed esclusivo (X) per conservare la congruenza. Il comportamento di questo blocco è più importante quando si utilizza un blocco pessimistico perché i blocchi ottimistici non sono conservati. È possibile utilizzare uno dei tre livelli di isolamento per ottimizzare la semantica del blocco che eXtreme Scale utilizza per mantenere la congruenza in ciascuna mappa: repeatable read, read committed read committed e read uncommitted.

Panoramica sull'isolamento della transazione

L'isolamento della transazione definisce in che modo le modifiche che sono apportate per un'operazione diventano visibili ad altre operazioni concorrenti.

WebSphere eXtreme Scalesupporta tre livelli di isolamento della transazione con cui è possibile inoltre ottimizzare la semantica che utilizza eXtreme Scale per conservare la congruenza in ciascuna mappa della cache: repeatable read, read committed e read uncommitted. Il livello di isolamento viene impostato nell'interfaccia Session utilizzando il metodo setTransactionIsolation. L'isolamento della transazione può essere modificato in qualsiasi momento durante la vita della sessione se una transazione non è attualmente in corso.

Il prodotto impone le varie semantiche di isolamento della transazione adeguando il modo in cui i blocchi condivisi (S) sono richiesti e conservati. L'isolamento della transazione non influisce sulle mappe configurate per utilizzare le strategie di blocco ottimistico o pessimistico o none o quando sono richiesti dei blocchi aggiornabili (U).

Repeatable read con blocco pessimistico

Il livello di isolamento della transazione predefinito è repeatable read. Il livello di isolamento evita le dirty read e le non-repeatable read, ma non evita le phantom read. Una dirty read è un'operazione di lettura che avviene sui dati che sono stati modificati da una transazione, ma per i quali non è stato eseguito il commit. Una non repeatable read potrebbe verificarsi quando non sono stati acquisiti blocchi in lettura durante l'esecuzione di un'operazione di lettura. Una phantom read può verificarsi quando sono eseguite due operazioni di lettura identiche, ma vengono restituite due diverse serie di risultati perché si è verificato un aggiornamento dei dati tra le due operazioni di lettura. Il prodotto realizza una repeatable read rimanendo in attesa di un blocco S fino a quando la transazione che possiede i blocchi non è completata. Poiché il blocco X non è concesso fino a quando non sono rilasciati i blocchi S, tutte le transazioni in attesa del blocco S sono garantite per vedere lo stesso valore quando leggono nuovamente.

```
map = session.getMap("Order");
session.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session.begin();

// An S lock is requested and held and the value is copied into
// the transactional cache.
Order order = (Order) map.get("100");
// The entry is evicted from the transactional cache.
map.invalidate("100", false);

// The same value is requested again.  It already holds the
// lock, so the same value is retrieved and copied into the
```

```
// transactional cache.
Order order2 (Order) = map.get("100");

// All locks are released after the transaction is synchronized
// with cache map.
session.commit();
```

Le phantom read sono possibili quando si stanno utilizzando query o indici perché i blocchi non sono acquisiti per gli intervalli di dati, solo per le voci della cache che corrispondono al criterio della query o dell'indice Ad esempio:

```
session1.setTransactionIsolation(Session.TRANSACTION_REPEATABLE_READ);
session1.begin();

// A query is run which selects a range of values.
ObjectQuery query = session1.createObjectQuery
    ("SELECT o FROM Order o WHERE o.itemName='Widget'");

// In this case, only one order matches the query filter.
// The order has a key of "100".
// The query engine automatically acquires an S lock for Order "100".
Iterator result = query.getResultIterator();

// A second transaction inserts an order that also matches the query.
Map orderMap = session2.getMap("Order");
orderMap.insert("101", new Order("101", "Widget"));

// When the query runs again in the current transaction, the
// new order is visible and will return both Orders "100" and "101".
result = query.getResultIterator();

// All locks are released after the transaction is synchronized
// with cache map.
session.commit();
```

Read committed con blocco pessimistico

Il livello di isolamento della transazione read committed può essere utilizzato con eXtreme Scale, il che evita dirty read, ma non evita non-repeatable read o phantom read così eXtreme Scale continua ad utilizzare i blocchi S per leggere i dati dalla mappa della cache, ma rilascia immediatamente i blocchi.

```
map1 = session1.getMap("Order");
session1.setTransactionIsolation(Session.TRANSACTION_READ_COMMITTED);
session1.begin();

// An S lock is requested but immediately released and
//the value is copied into the transactional cache.

Order order = (Order) map1.get("100");

// The entry is evicted from the transactional cache.
map1.invalidate("100", false);

// A second transaction updates the same order.
// It acquires a U lock, updates the value, and commits.
// The ObjectGrid successfully acquires the X lock during
// commit since the first transaction is using read
// committed isolation.

Map orderMap2 = session2.getMap("Order");
session2.begin();
order2 = (Order) orderMap2.getForUpdate("100");
order2.quantity=2;
orderMap2.update("100", order2);
session2.commit();
```

```
// The same value is requested again.  This time, they
// want to update the value, but it now reflects
// the new value
Order order1Copy (Order) = map1.getForUpdate("100");
```

Read uncommitted con blocco pessimistico

Il livello di isolamento della transazione read uncommitted può essere utilizzato con eXtreme Scale, che è un livello che consente dirty read, non-repeatable read e phantom read.

Eccezione di conflitto ottimistica

È possibile ricevere un'eccezione OptimisticCollisionException direttamente o riceverla con, una ObjectGridException.

Il codice di seguito riportato è un esempio del modo in cui viene rilevata l'eccezione e successivamente visualizzato il messaggio:

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

Causa dell'eccezione

L'eccezione OptimisticCollisionException è generata in una situazione in cui due differenti client tentano di aggiornare la stessa voce della mappa relativamente allo stesso momento. Ad esempio, se un client tenta di eseguire il commit di una sessione e aggiorna la voce della mappa dopo che un altro client abbia letto i dati prima del commit, quei dati non saranno corretti. L'eccezione è generata quando l'altro client tenta di eseguire il commit dei dati non corretti.

Recupero della chiave che ha attivato l'eccezione

Potrebbe essere utile, quando la risoluzione del problema riguarda un'eccezione, recuperare la chiave corrispondente alla voce che ha attivato l'eccezione. Il vantaggio dell'eccezione OptimisticCollisionException è che essa contiene il metodo getKey il quale restituisce l'oggetto che rappresenta quella chiave. L'esempio di seguito riportato dimostra in che modo recuperare e stampare la chiave quando si rileva l'eccezione OptimisticCollisionException:

```
try {
...
} catch (OptimisticCollisionException oce) {
    System.out.println(oce.getKey());
}
```

L'eccezione ObjectGridException causa un'eccezione OptimisticCollisionException

L'eccezione OptimisticCollisionException potrebbe essere la causa della visualizzazione di ObjectGridException. Se si verifica questo caso, è possibile utilizzare il codice di seguito riportato per determinare l'eccezione e stampare la chiave. Il codice di seguito riportato utilizza il programma di utilità del metodo findRootCause come descritto nella sezione sotto riportata.

```
try {
...
}
catch (ObjectGridException oe) {
    Throwable Root = findRootCause( oe );
    if (Root instanceof OptimisticCollisionException) {
        OptimisticCollisionException oce = (OptimisticCollisionException)Root;
        System.out.println(oce.getKey());
    }
}
```

Tecnica di gestione dell'eccezione generale

Conoscendo la causa principale di un oggetto Throwable è utile per isolare l'origine dell'errore. L'esempio di seguito riportato dimostra in che modo un gestore dell'eccezione utilizza un metodo Utility per trovare la causa principale dell'oggetto Throwable.

Esempio:

```
static public Throwable findRootCause( Throwable t )
{
     // Start with Throwable that occurred as the root.
     Throwable root = t;

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

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

Configurazione dei client con WebSphere eXtreme Scale

È possibile configurare un client di eXtreme Scale in base ai propri requisiti, come, ad esempio, la necessità di sovrascrivere le impostazioni.

Configurazione del client mediante XML

È possibile utilizzare un file XML ObjectGrid per modificare le impostazioni sul lato client. Per modificare le impostazioni di un client eXtreme Scale, è necessario creare un file XML di ObjectGrid simile, nella struttura, al file utilizzato per il server eXtreme Scale.

Supporre che il seguente file XML sia associato ad un file XML della politica di distribuzione e che tali file siano utilizzati per avviare un server eXtreme Scale.

companyGridServerSide.xml

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

    <objectGrids>
        <objectGrid name="CompanyGrid">
            <bean id="TransactionCallback"
                className="com.company.MyTxCallback" />
            <bean id="ObjectGridEventListener"
                className="com.company.MyOgEventListener" />
            <backingMap name="Customer"
```

```
            pluginCollectionRef="customerPlugins" />
        <backingMap name="Item" />
        <backingMap name="OrderLine" numberOfBuckets="1049"
            timeToLive="1600" ttlEvictorType="LAST_ACCESS_TIME" />
        <backingMap name="Order" lockStrategy="PESSIMISTIC"
            pluginCollectionRef="orderPlugins" />
    </objectGrid>
</objectGrids>

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

Su un server eXtreme Scale, l'istanza ObjectGrid denominata CompanyGrid funziona nel modo definito dal file companyGridServerSide.xml. Per impostazione predefinita, il client CompanyGrid ha le stesse impostazioni dell'istanza CompanyGrid in esecuzione sul server. Tuttavia, è possibile sovrascrivere alcune impostazioni sul client, come riportato di seguito:

1. Creare un'istanza ObjectGrid specifica del client.
2. Copiare il file XML ObjectGrid utilizzato per aprire il server.
3. Modificare il nuovo file da personalizzare per il lato client.
   - Per impostare oppure aggiornare gli attributi sul client, specificare un nuovo valore oppure modificare il valore esistente.
   - Per rimuovere un plug-in dal client, utilizzare una stringa vuota come valore per l'attributo className.
   - Per modificare un plug-in esistente, specificare un nuovo valore per l'attributo className,
   - È anche possibile aggiungere qualsiasi plug-in supportato per una sostituzione client: TRANSACTION_CALLBACK, OBJECTGRID_EVENT_LISTENER, EVICTOR, MAP_EVENT_LISTENER.
4. Creare un client con il file XML di sostituzione client appena creato:

È possibile utilizzare il seguente file XML ObjectGrid per specificare alcuni degli attributi e plug-in sul client CompanyGrid.

companyGridClientSide.xml

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

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

    <backingMapPluginCollections>
```

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

- TransactionCallback sul client è com.company.MyClientTxCallback invece dell'impostazione sul lato server com.company.MyTxCallback.
- Il client non dispone di un plug-in ObjectGridEventListener perché il valore di className è una stringa vuota.
- Il client imposta numberOfBuckets sul valore 1429 per il backingMap Customer, conserva il relativo plug-in Evictor e rimuove il plug-in MapEventListener.
- Gli attributi numberOfBuckets e timeToLive del backingMap OrderLine sono stati modificati
- Sebbene sia specificato un attributo lockStrategy differente, non viene eseguita alcuna operazione perché l'attributo lockStrategy non è supportato per una sostituzione client.

Per creare il client CompanyGrid utilizzando il file companyGridClientSide.xml, passare il file XML ObjectGrid come URL ad uno dei metodi connect su ObjectGridManager.

Creating the client for XML

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

Configurazione del client in modo programmatico

È anche possibile sostituire le impostazioni di ObjectGrid sul lato client in modo programmatico. Creare un oggetto ObjectGridConfiguration simile nella struttura all'istanza ObjectGrid sul lato server. Il codice riportato di seguito crea un'istanza ObjectGrid sul lato client funzionalmente equivalente alla sostituzione client nella sezione precedente che utilizza un file XML.

client-side override programmatically

```
ObjectGridConfiguration companyGridConfig = ObjectGridConfigFactory
    .createObjectGridConfiguration("CompanyGrid");
Plugin txCallbackPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.TRANSACTION_CALLBACK, "com.company.MyClientTxCallback");
companyGridConfig.addPlugin(txCallbackPlugin);

Plugin ogEventListenerPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.OBJECTGRID_EVENT_LISTENER, "");
companyGridConfig.addPlugin(ogEventListenerPlugin);

BackingMapConfiguration customerMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("Customer");
customerMapConfig.setNumberOfBuckets(1429);
Plugin evictorPlugin = ObjectGridConfigFactory.createPlugin(PluginType.EVICTOR,
    "com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor");
customerMapConfig.addPlugin(evictorPlugin);

companyGridConfig.addBackingMapConfiguration(customerMapConfig);

BackingMapConfiguration orderLineMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("OrderLine");
orderLineMapConfig.setNumberOfBuckets(701);
orderLineMapConfig.setTimeToLive(800);
orderLineMapConfig.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);
```

```
companyGridConfig.addBackingMapConfiguration(orderLineMapConfig);

List ogConfigs = new ArrayList();
ogConfigs.add(companyGridConfig);

Map overrideMap = new HashMap();
overrideMap.put(CatalogServerProperties.DEFAULT_DOMAIN, ogConfigs);

ogManager.setOverrideObjectGridConfigurations(overrideMap);
ClientClusterContext client = ogManager.connect(catalogServerAddresses, null, null);
ObjectGrid companyGrid = ogManager.getObjectGrid(client, objectGridName);
```

L'istanza ogManager dell'interfaccia ObjectGridManager controlla le sostituzioni solo negli oggetti ObjectGridConfiguration e BackingMapConfiguration inclusi nella mappa overrideMap. Ad esempio, il codice precedente sostituisce il numero di bucket nella mappa OrderLine. Tuttavia, la mappa Order non viene modificata sul lato client perché non è inclusa alcune configurazione per tale mappa.

Configurazione del client in Spring Framework

È possibile sovrascrivere le impostazioni di ObjectGrid sul lato client anche utilizzando Spring Framework. Il file XML di esempio riportato di seguito illustra come creare un elemento ObjectGridConfiguration ed il relativo utilizzo per la sostituzione di alcune impostazioni sul lato client. In questo esempio vengono richiamate le stesse API indicate nella configurazione programmatica. Inoltre, l'esempio è funzionalmente equivalente all'esempio nella configurazione XML ObjectGrid.

client configuration with Spring

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
  "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="companyGrid" factory-bean="manager" factory-method="getObjectGrid"
    singleton="true">
    <constructor-arg type="com.ibm.websphere.objectgrid.ClientClusterContext">
      <ref bean="client" />
    </constructor-arg>
    <constructor-arg type="java.lang.String" value="CompanyGrid" />
</bean>

  <bean id="manager" class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
    factory-method="getObjectGridManager" singleton="true">
    <property name="overrideObjectGridConfigurations">
      <map>
        <entry key="DefaultDomain">
          <list>
            <ref bean="ogConfig" />
          </list>
        </entry>
      </map>
    </property>
  </bean>

  <bean id="ogConfig"
    class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
    factory-method="createObjectGridConfiguration">
    <constructor-arg type="java.lang.String">
      <value>CompanyGrid</value>
    </constructor-arg>
    <property name="plugins">
      <list>
   <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
          factory-method="createPlugin">
          <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
            value="TRANSACTION_CALLBACK" />
          <constructor-arg type="java.lang.String"
            value="com.company.MyClientTxCallback" />
        </bean>
        <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
          factory-method="createPlugin">
          <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
            value="OBJECTGRID_EVENT_LISTENER" />
          <constructor-arg type="java.lang.String" value="" />
```

```
        </bean>
      </list>
 </property>
      <property name="backingMapConfigurations">
        <list>
     <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
           factory-method="createBackingMapConfiguration">
           <constructor-arg type="java.lang.String" value="Customer" />
             <property name="plugins">
      <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
                factory-method="createPlugin">
                <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
     value="EVICTOR" />
   <constructor-arg type="java.lang.String"
                   value="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
               </bean>
             </property>
             <property name="numberOfBuckets" value="1429" />
           </bean>
           <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
             factory-method="createBackingMapConfiguration">
               <constructor-arg type="java.lang.String" value="OrderLine" />
                 <property name="numberOfBuckets" value="701" />
   <property name="timeToLive" value="800" />
   <property name="ttlEvictorType">
                    <value type="com.ibm.websphere.objectgrid.
          TTLType">LAST_ACCESS_TIME</value>
   </property>
           </bean>
         </list>
       </property>
     </bean>

     <bean id="client" factory-bean="manager" factory-method="connect"
       singleton="true">
       <constructor-arg type="java.lang.String">
    <value>localhost:2809</value>
       </constructor-arg>
   <constructor-arg
          type="com.ibm.websphere.objectgrid.security.
      config.ClientSecurityConfiguration">
          <null />
       </constructor-arg>
   <constructor-arg type="java.net.URL">
     <null />
       </constructor-arg>
     </bean>
</beans>
```

Una volta creato il file XML, caricarlo e creare ObjectGrid con il frammento di codice riportato di seguito.

```
BeanFactory beanFactory = new XmlBeanFactory(new

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

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

Per ulteriori informazioni sulla creazione di un file descrittore XML, consultare la sezione Panoramica sull'integrazione Spring Framework.

Disabilitazione della cache locale del client

La cache locale è abilitata per impostazione predefinita quando il blocco è configurato come optimistic oppure none. I client non gestiscono una cache locale quando l'impostazione relativa al blocco è configurata come pessimistic. Per disabilitare la cache locale, è necessario impostare l'attributo numberOfBuckets sul valore 0 nel file descrittore ObjectGrid di sostituzione client.

Tracciamento degli aggiornamenti della mappa mediante un'applicazione

Quando un'applicazione sta effettuando modifiche ad una mappa durante una transazione un oggetto LogSequence tiene traccia di queste modifiche Se l'applicazione modifica una voce nella mappa, un corrispondente oggetto LogElement fornisce i dettagli relativi alla modifica.

I programmi di caricamento forniscono un oggetto LogSequence per una mappa particolare ogni volta che un'applicazione richiama un'operazione di flush o di commit per la transazione. Il programma di caricamento itera gli oggetti LogElement all'interno dell'oggetto LogSequence e applica ciascun oggetto LogElement al backend.

I listener ObjectGridEventListener che sono registrati con ObjectGrid utilizzano anche oggetti LogSequence. Questi listener vengono forniti di un oggetto LogSequence per ciascuna mappa nella transazione sulla quale è stato eseguito il commit. Le applicazioni possono utilizzare questi listner per attendere che determinate voci siano modificate, simile ad un trigger in un database convenzionale.

Le seguenti interfacce relative al log o alle classi vengono fornite dal framework di eXtreme Scale:

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

Interfaccia LogElement

Un LogElement rappresenta un'operazione su una voce durante una transazione. Un oggetto LogElement ha diversi metodi per ottenere i suoi vari attributi. Gli attributi più comunemente utilizzati sono gli attributi tipo e valore corrente sui quali è stato eseguito il fetch da getType() e getCurrentValue().

Il tipo è rappresentato da una delle costanti definite nell'interfaccia LogElement: INSERT, UPDATE, DELETE, EVICT, FETCH, o TOUCH.

Il valore corrente rappresenta il nuovo valore per l'operazione se essa è di tipo INSERT, UPDATE o FETCH. Se l'operazione è di tipo TOUCH, DELETE, o EVICT, allora il valore corrente è null. Può essere eseguito il cast del valore per ValueProxyInfo quando è in uso una ValueInterface

consultare la documentazione relativa all'API per ulteriori dettagli sull'interfaccia LogElement.

Interfaccia LogSequence

Nella maggior parte delle transazioni, si realizzano operazioni su più di una voce nella mappa, così vengono creati più oggetti LogElement. Si dovrebbe creare un oggetto che si comporta come un composito di più oggetti LogElement. L'interfaccia LogSequence serve allo scopo di contenere un elenco di oggetti LogElement.

consultare la documentazione relativa all'API per ulteriori dettagli sull'interfaccia LogSequence.

Utilizzo di LogElement e LogSequence

LogElement e LogSequence sono ampiamente utilizzati in eXtreme Scale e dai plug-in ObjectGrid che sono scritte dagli utenti quando le operazioni vengono propagate da un componente o server ad un altro componente o server. Ad esempio, un oggetto LogSequence può essere utilizzato dalla funzione di propagazione di una transazione ObjectGrid distribuita per propagare le modifiche sugli altri server oppure può essere applicato all'archivio permanente dal programma di caricamento. LogSequence è principalmente utilizzato 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

Esempio di programma di caricamento

Questa sezione dimostra in che modo gli oggetti LogSequence e LogElement vengono utilizzati in un programma di caricamento. Un programma di caricamento viene utilizzato per caricare i dati in un archivio persistente. Il metodo batchUpdate dell'interfaccia di un programma di caricamento utilizza l'oggetto LogSequence:

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

Il metodo batchUpdate viene richiamato quando un ObjectGrid ha necessità di applicare tutte le modifiche correnti al programma di caricamento. Il programma di caricamento è fornito di un elenco di oggetti LogElement per la mappa, incapsulati in un oggetto LogSequence. L'implementazione del metodo batchUpdate deve iterare le modifiche e applicarle al backend. Il frammento di codice riportato di seguito dimostra il modo in cui il programma di caricamento utilizza un oggetto LogSequence. Il frammento itera la serie di modifiche ed esegue il build in batch di tre istruzioni di JDBC (Java database connectivity): istruzione insert , update e delete:

```
public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException
{
    // Get a SQL connection to use.
    Connection conn = getConnection(tx);
    try
    {
    // Process the list of changes and build a set of prepared
    // statements for executing a batch update, insert, or delete
    // SQL operations. The statements are cached in stmtCache.
    Iterator iter = sequence.getPendingChanges();
    while ( iter.hasNext() )
    {
        LogElement logElement = (LogElement)iter.next();
        Object key = logElement.getCacheEntry().getKey();
        Object value = logElement.getCurrentValue();
        switch ( logElement.getType().getCode() )
        {
            case LogElement.CODE_INSERT:
                buildBatchSQLInsert( key, value, conn );
                break;
            case LogElement.CODE_UPDATE:
```

```
                buildBatchSQLUpdate( key, value, conn );
                break;
            case LogElement.CODE_DELETE:
                buildBatchSQLDelete( key, conn );
                break;
        }
    }
    // Run the batch statements that were built by above loop.
    Collection statements = getPreparedStatementCollection( tx, conn );
    iter = statements.iterator();
    while ( iter.hasNext() )
    {
        PreparedStatement pstmt = (PreparedStatement) iter.next();
        pstmt.executeBatch();
    }
} catch (SQLException e)
{
    LoaderException ex = new LoaderException(e);
    throw ex;
}
}
```

L'esempio precedente illustra la logica ad alto livello dell'elaborazione dell'argomento LogSequence. Tuttavia, l'esempio non illustra i dettagli del modo in cui viene costruita un'istruzione SQL insert, update, o delete. Il metodo getPendingChanges è richiamato sull'argomento LogSequence per ottenere un iteratore degli oggetti LogElement che un programma di caricamento ha necessità di elaborare e il metodo LogElement.getType().getCode() viene utilizzato per determinare se un LogElement è per un'operazione SQL insert, update, o delete.

Esempio di programma di eliminazione

È possibile anche utilizzare gli oggetti LogSequence e LogElement con un programma di eliminazione. Un programma di eliminazione viene utilizzato per eliminare le voci della mappa dalla mappa di backup basati su determinati criteri Il metodo Apply di un'interfaccia del programma di eliminazione utilizza LogSequence.

```
/**
* This is called during cache commit to allow the evictor to track object usage
* in a backing map. This will also report any entries that have been successfully
* evicted.
*
* @param sequence LogSequence of changes to the map
*/
void apply(LogSequence sequence);
```

Interfacce LogSequenceFilter e LogSequenceTransformer

Qualche volta, è necessario filtrare gli oggetti LogElement in modo che solo gli oggetti LogElement con determinati criteri siano accettati e rigettati gli altri oggetti. Ad esempio, si potrebbe desiderare serializzare un determinato LogElement basato su alcuni criteri.

LogSequenceFilter risolve questo problema con il seguente metodo.

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

Questo metodo restituisce true se il LogElement fornito dovesse essere utilizzato nell'operazione e restituisce false se il LogElement fornito non dovesse essere utilizzato.

LogSequenceTransformer è una classe che utilizza la funzione LogSequenceFilter. Essa utilizza LogSequenceFilter per filtrare alcuni oggetti LogElement e successivamente serializzare gli oggetti LogElement accettati. Questa classe dispone di due metodi: Il primo metodo è riporttao di seguito.

```
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 quali LogElements 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 invalidazione, allora non esiste la necessità di serializzare il valore. Il secondo metodo di questa classe è il metodo Inflate riportato di seguito.

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

Il metodo Inflate legge il modulo serializzato della sequenza di log, che è stato creato dal metodo Serialize, dal flusso di input dell'oggetto fornito.

Abilitazione della replica di mappa lato client

È inoltre possibile abilitare la replica di mappe sul lato client per rendere i dati disponibili in modo più rapido.

Con eXtreme Scale, è possibile replicare una mappa del server su uno o più client utilizzando la replica asincrona. Un client può richiedere una copia locale in sola lettura della mappa lato server utilizzando il metodo ClientReplicableMap.enableClientReplication.

```
void enableClientReplication(Mode mode, int[] partitions,
 ReplicationMapListener listener) throws ObjectGridException;
```

Il primo parametro rappresenta la modalità di replica. Questa modalità può essere una replica continua o istantanea. Il secondo parametro è un array di ID di partizione che rappresentano le partizioni da cui replicare i dati. Se il valore è null o un array vuoto, i dati vengono replicati da tutte le partizioni. L'ultimo parametro è un listener per ricevere gli eventi di replica del client. Per i dettagli, consultare le informazioni relative a ClientReplicableMap e ReplicationMapListener nella documentazione API.

Una volta che la replica è abilitata, il server avvia la replica della mappa sul client. In questo modo, il client resta indietro solo di poche transazioni rispetto al server in ogni memento.

Esempio di API DataGrid

Le API DataGrid supportano due pattern di programmazione della griglia comuni: mappa parallela e riduzione parallela.

Mappa parallela

La mappa parallela consente l'elaborazione delle voci per una serie di chiavi e restituisce un risultato per ciascuna voce elaborata. L'applicazione crea un elenco di chiavi e riceve una mappa di coppie chiave/risultato dopo il richiamo di un'operazione Map. Il risultato è il risultato dell'applicazione di una funzione alla voce di ciascuna chiave. La funzione è fornita dall'applicazione.

Flusso della chiamata MapGridAgent

Quando viene richiamato il metodo AgentManager.callMapAgent con una raccolta di chiavi, l'istanza MapGridAgent viene serializzata ed inviata a ciascuna partizione primaria in cui viene risolta la chiave. Ciò significa che tutti i dati dell'istanza memorizzati nell'agent possono essere inviati al server. Ciascuna partizione primaria, quindi, dispone di un'istanza dell'agent. Il metodo process viene richiamato per ogni istanza una volta per ciascuna chiave che risolve la partizione. Il risultato di ciascun metodo process viene quindi serializzato al client e restituito al chiamante in un'istanza Map, in cui il risultato è rappresentato come il valore nella mappa.

Quando il metodo AgentManager.callMapAgent viene richiamato senza una raccolta di chiavi, l'istanza MapGridAgent viene serializzata ed inviata a ciascuna partizione primaria. Ciò significa che tutti i dati dell'istanza memorizzati nell'agent possono essere inviati al server. Ciascuna partizione primaria, quindi, dispone di un'istanza (partizione) dell'agent. Il metodo processAllEntries viene richiamato per ciascuna partizione. Il risultato di ciascun metodo processAllEntries viene quindi serializzato al client e restituito al chiamante in un'istanza Map. Nell'esempio riportato di seguito, si suppone che esista un'entità Person con il seguente formato:

```
import com.ibm.websphere.projector.annotations.Entity;
import com.ibm.websphere.projector.annotations.Id;
@Entity
public class Person
{
  @Id String ssn;
  String firstName;
  String surname;
  int age;
}
```

La funzione fornita dall'applicazione è scritta come una classe che implementa l'interfaccia MapAgentGrid. Di seguito è riportato un agent di esempio che mostra una funzione che restituisce l'età di una persona (Person) moltiplicata per due.

```
public class DoublePersonAgeAgent implements MapGridAgent, EntityAgentMixin
{
  private static final long serialVersionUID = -2006093916067992974L;

  int lowAge;
  int highAge;

  public Object process(Session s, ObjectMap map, Object key)
  {
    Person p = (Person)key;
    return new Integer(p.age * 2);
  }

  public Map processAllEntries(Session s, ObjectMap map)
  {
    EntityManager em = s.getEntityManager();
    Query q = em.createQuery("select p from Person p where p.age > ?1 and p.age < ?2");
    q.setParameter(1, lowAge);
    q.setParameter(2, highAge);
    Iterator iter = q.getResultIterator();
    Map<Person, Interger> rc = new HashMap<Person, Integer>();
   while(iter.hasNext())
    {
   Person p = (Person)iter.next();
   rc.put(p, (Integer)process(s, map, p));
  }
  return rc;
 }
 public Class getClassForEntity()
 {
  return Person.class;
 }
}
```

È visualizzato l'agent Map per il raddoppio dell'età di una persona. Si considerino i metodi process. Il primo metodo process viene fornito con l'oggetto Person con

cui lavorare. Tale metodo semplicemente restituisce l'età raddoppiata di tale voce. Il secondo metodo process viene richiamato per ciascuna partizione e rileva tutti gli oggetti Person con età compresa tra lowAge ed highAge e restituisce le relative età raddoppiate.

```
Session s = grid.getSession();
ObjectMap map = s.getMap("Person");
AgentManager amgr = map.getAgentManager();

DoublePersonAgeAgent agent = new DoublePersonAgeAgent();

// make a list of keys
ArrayList<Person> keyList = new ArrayList<Person>();
Person p = new Person();
p.ssn = "1";
keyList.add(p);
p = new Person ();
p.ssn = "2";
keyList.add(p);

// get the results for those entries
Map<Tuple, Object> = amgr.callMapAgent(agent, keyList);
```

Questo esempio illustra un client che ottiene una Session ed un riferimento alla mappa Person. L'operazione dell'agent viene eseguita su una mappa specifica. L'interfaccia AgentManager viene richiamata da tale mappa. Viene creata un'istanza dell'agent da richiamare e tutti gli stati necessari vengono aggiunti all'oggetto impostando degli attributi, nessuno in questo caso. Viene quindi creato un elenco di chiavi. Viene restituita una mappa con i valori per la persona 1 raddoppiati e gli stessi valori per la persona 2.

Quindi, viene richiamato l'agent per tale serie di dati. Il metodo process dell'agent viene richiamato su ciascuna partizione con alcune delle chiavi specificate nella griglia in parallelo. Viene restituita una mappa che fornisce i risultati uniti per la chiave specificata. In questo caso, viene restituita una mappa con i valori relativi all'età della persona 1 raddoppiata e gli stessi valori per la persona 2.

Se la chiave non esiste, l'agent viene comunque richiamato. Ciò fornisce all'agent l'opportunità di creare la voce della mappa. Se viene utilizzato EntityAgentMixin, la chiave da elaborare non sarà l'entità, ma il valore della chiave Tuple reale per l'entità. Se la chiavi sono sconosciute, è possibile richiedere a tutte le partizioni di individuare oggetti Person di un certo tipo e di restituire le relative età raddoppiate. Di seguito è riportato un esempio:

```
Session s = grid.getSession();
  ObjectMap map = s.getMap("Person");
  AgentManager amgr = map.getAgentManager();

  DoublePersonAgeAgent agent = new DoublePersonAgeAgent();
  agent.lowAge = 20;
  agent.highAge = 9999;

  Map m = amgr.callMapAgent(agent);
```

L'esempio precedente mostra l'AgentManager ottenuto per la mappa Person e l'agent creato ed inizializzato con le età minima e massima per gli oggetti Person di interesse. L'agent viene quindi richiamato utilizzando il metodo callMapAgent. Notare che non vengono fornite chiavi. In questo modo, ObjectGrid richiama l'agent su ciascuna partizione nella griglia in parallelo e restituisce al client i risultati uniti. Verranno rilevati tutti gli oggetti Person nella griglia con un'età compresa tra il valore minimo ed il valore massimo e verrà calcolata l'età raddoppiata di tali oggetti Person. Questo esempio mostra il modo in cui è

possibile utilizzare le API della griglia per eseguire una query che consente di individuare entità che corrispondono ad una determinata query. L'agent viene semplicemente serializzato e trasportato da ObjectGrid alle partizioni con le voci richieste. I risultati sono allo stesso modo serializzati per essere inviati nuovamente al client. È necessario prestare attenzione quando si utilizzano le API Map. Se ObjectGrid ospita terabyte di oggetti ed è in esecuzione su un elevato numero di server, potrebbero verificarsi dei problemi sulle macchine più grandi su cui è in esecuzione il client. Questo metodo deve essere utilizzato per elaborare un sottoinsieme di piccole dimensioni. Se è necessario elaborare un sottoinsieme di grandi dimensioni, è preferibile utilizzare un agent di riduzione per eseguire l'elaborazione in una griglia piuttosto che su un client.

Riduzione parallela o agent di aggregazione

Questo stile di programmazione elabora un sottoinsieme delle voci e calcola un singolo risultato per il gruppo di voci. Alcuni esempi di questo risultato sono:

- valore minimo
- valore massimo
- altre funzioni specifiche aziendali

L'agent di riduzione viene codificato e richiamato in modo molto simile agli agent Map.

Flusso di chiamata ReduceGridAgent

Quando il metodo AgentManager.callReduceAgent viene richiamato con una raccolta di chiavi, l'istanza ReduceGridAgent viene serializzata ed inviata a ciascuna partizione primaria in cui vengono risolte le chiavi. Ciò significa che tutti i dati dell'istanza memorizzati nell'agent possono essere inviati al server. Ciascuna partizione primaria, quindi, dispone di un'istanza dell'agent. Il metodo reduce(Session s, ObjectMap map, Collection keys) viene richiamato una volta per istanza (partizione) con il sottoinsieme di chiavi che risolvono la partizione. Il risultato di ciascun metodo reduce viene quindi serializzato al client. Il metodo reduceResults viene richiamato sull'istanza ReduceGridAgent del client con la raccolta di ciascun risultato da ciascun richiamo del metodo reduce remoto. Il risultato del metodo reduceResults viene restituito al chiamante del metodo callReduceAgent.

Quando il metodo AgentManager.callReduceAgent viene richiamato senza una raccolta di chiavi, l'istanza ReduceGridAgent viene serializzata ed inviata a ciascuna partizione primaria. Ciò significa che tutti i dati dell'istanza memorizzati nell'agent possono essere inviati al server. Ciascuna partizione primaria, quindi, dispone di un'istanza dell'agent. Il metodo reduce(Session s, ObjectMap map) viene richiamato una volta per istanza (partizione). Il risultato di ciascun metodo reduce viene quindi serializzato al client. Il metodo reduceResults viene richiamato sull'istanza ReduceGridAgent del client con la raccolta di ciascun risultato da ciascun richiamo del metodo reduce remoto. Il risultato del metodo reduceResults viene restituito al chiamante del metodo callReduceAgent. Di seguito è riportato un esempio di agent di riduzione che semplicemente aggiunge le età delle voci corrispondenti.

```
public class SumAgeReduceAgent implements ReduceGridAgent, EntityAgentMixin
{
  private static final long serialVersionUID = 2521080771723284899L;

  int lowAge;
  int highAge;

  public Object reduce(Session s, ObjectMap map, Collection keyList)
  {
```

```
    Iterator<Person> iter = keyList.iterator();
  int sum = 0;
  while (iter.hasNext())
  {
   Person p = iter.next();
   sume += p.age;
  }
  return new Integer(sum);
 }

 public Object reduce(Session s, ObjectMap map)
 {
  EntityManager em = s.getEntityManager ();
  Query q = em.createQuery("select p from Person p where p.age > ?1 and p.age < ?2");
  q.setParameter(1, lowAge);
  q.setParameter(2, highAge);
  Iterator<Person> iter = q.getResultIterator();
  int sum = 0;
  while(iter.hasNext())
  {
   sum += iter.next().age;
  }
  return new Integer(sum);
 }

 public Class getClassForEntity()
 {
  return Person.class;
 }
}
```

L'esempio precedente mostra l'agent. L'agent è composto da tre parti importanti. La prima consente l'elaborazione di una serie specifica di voci senza una query. Tale serie interagisce con la serie di voci aggiungendo le età. La somma viene restituita dal metodo. La seconda parte utilizza una query per selezionare le voci da aggregare. Quindi, somma tutte le età degli oggetti Person corrispondenti. Il terzo metodo viene utilizzato per unire in un singolo risultato i risultati provenienti da ciascuna partizione. ObjectGrid esegue l'aggregazione delle voci in parallelo sulla griglia. Ciascuna partizione produce un risultato intermedio che deve essere unito agli altri risultati intermedi delle partizione. Tale attività viene eseguita da questo terzo metodo. Nell'esempio riportato di seguito, l'agent viene richiamato e vengono aggregate esclusivamente le età di tutte le persone con età comprese tra 10 e 20:

```
Session s = grid.getSession();
  ObjectMap map = s.getMap("Person");
  AgentManager amgr = map.getAgentManager();

  SumAgeReduceAgent agent = new SumAgeReduceAgent();

  Person p = new Person();
  p.ssn = "1";
  ArrayList<Person> list = new ArrayList<Person>();
 list.add(p);
 p = new Person ();
 p.ssn = "2";
 list.add(p);
 Integer v = (Integer)amgr.callReduceAgent(agent, list);
```

Funzioni dell'agent

L'agent è libero di eseguire operazioni ObjectMap oppure EntityManager all'interno del frammento locale in cui è in esecuzione. L'agent riceve un oggetto Sessione e può aggiungere, aggiornare, eseguire query, leggere o rimuovere dati dalla partizione rappresentata da Session. Alcune applicazioni eseguono solo query di dati dalla griglia, ma è possibile anche scrivere un agent per incrementare di 1 tutte le età di Person che corrispondono ad una determinata query. Quando l'agent viene richiamato, è presente una transazione su Session, di cui viene eseguito il commit quando l'agent restituisce i valori, a meno che non venga generata un'eccezione

Gestione degli errori

Se viene richiamato un agent map con una chiave sconosciuta, il valore restituito è un oggetto di errore che implementa l'interfaccia EntryErrorValue.

Transazioni

L'agent map viene eseguito in una transazione separata rispetto al client. È possibile raggruppare i diversi richiami dell'agent in una singola transazione. In caso di errore di un agent (viene generata un'eccezione), viene eseguito il rollback della transazione. Per tutti gli agent eseguiti correttamente in una transazione, viene eseguito il rollback con l'agent per cui si è verificato l'errore. AgentManager eseguirà nuovamente gli agent eseguiti correttamente e di cui è stato eseguito il rollback in una nuova transazione.

Per ulteriori informazioni, consultare la documentazione relativa all'API DataGrid.

Capitolo 4. Accesso ai dati con il servizio dati REST

Sviluppo di applicazioni che eseguono operazioni utilizzando i protocolli del servizio dati REST.

Operazioni con il servizio dati REST

Dopo aver avviato il servizio dati REST di eXtreme Scale, è possibile utilizzare qualsiasi client HTTP per interagire con esso. È possibile utilizzare un browser Web, un client PHP, un client Java o un client WCF Data Services per inviare qualsiasi operazione di richiesta supportata.

Il servizio REST implementa un sottoinsieme della specifica Microsoft Atom Publishing Protocol: Data Services URI and Payload Extensions, Versione 1.0 che fa parte del protocollo OData. Questo capitolo descrive quali delle funzioni della specifica sono supportate e come vengono associate a eXtreme Scale.

URI della root del servizio

Microsoft WCF Data Services in genere definisce un servizio per origine dati o modello di entità. Il servizio dati REST di eXtreme Scale definisce un servizio per ciascun ObjectGrid definito. Ciascun ObjectGrid definito nel file XML di sostituzione del client ObjectGrid di eXtreme Scale viene automaticamente esposto come una root del servizio REST separata.

L'URI per la root del servizio è:

`http://host:port/contextroot/restservice/gridname`

Dove:

- *contextroot* viene definita al momento della distribuzione dell'applicazione di servizio dati e dipende dal server delle applicazioni
- *gridname* è il nome della ObjectGrid

Tipi di richiesta

L'elenco riportato di seguito descrive i tipi di richiesta di Microsoft WCF Data Services supportati dal servizio dati REST di eXtreme Scale. Per i dettagli su ciascun tipo di richiesta supportato da WCF Data Services, consultare: MSDN: Request Types.

Tipi di richiesta di inserimento

I client possono inserire le risorse utilizzando il verbo HTTP POST con le seguenti limitazioni:

- Richiesta InsertEntity: supportata.
- Richiesta InsertLink: supportata.
- Richiesta InsertMediaResource: non supportata a causa di una limitazione di supporto risorse media.

Per ulteriori informazioni, consultare: MSDN: Insert Request Types.

Tipi di richiesta di aggiornamento

I client possono aggiornare le risorse utilizzando i verbi HTTP PUT e MERGE con le seguenti limitazioni:

- Richiesta UpdateEntity: supportata.
- Richiesta UpdateComplexType: non supportata a causa di una limitazione del tipo complesso.
- Richiesta UpdatePrimitiveProperty: supportata.
- Richiesta UpdateValue: supportata.
- Richiesta UpdateLink: supportata.
- Richiesta UpdateMediaResource: non supportata a causa di una limitazione di supporto risorse media.

Per ulteriori informazioni, consultare: MSDN: Insert Request types.

Tipi di richiesta di eliminazione

I client possono eliminare le risorse utilizzando il verbo HTTP DELETE con le seguenti limitazioni:

- Richiesta DeleteEntity: supportata.
- Richiesta DeleteLink: supportata.
- Richiesta DeleteValue: supportata.

Per ulteriori informazioni, consultare: MSDN: Delete Request Types.

Tipi di richiesta di richiamo

I client possono richiamare le risorse utilizzando il verbo HTTP GET con le seguenti limitazioni:

- Richiesta RetrieveEntitySet: supportata.
- Richiesta RetrieveEntity: supportata.
- Richiesta RetrieveComplexType: non supportata a causa di una limitazione del tipo complesso.
- Richiesta RetrievePrimitiveProperty: supportata.
- Richiesta RetrieveValue: supportata.
- Richiesta RetrieveServiceMetadata: supportata.
- Richiesta RetrieveServiceDocument: supportata.
- Richiesta RetrieveLink: supportata.
- Richiesta Retrieve che contiene un'associazione di feed personalizzabile: non supportata
- Richiesta RetrieveMediaResource: non supportata a causa di una limitazione di risorse media.

Per ulteriori informazioni, consultare: MSDN: Retrieve Request Types.

Opzioni delle query di sistema

Le query sono supportate e consentono ai client di identificare una raccolta di entità o una singola entità. Le opzioni della query di sistema sono specificate in un URI del servizio dati e sono supportate con le seguenti limitazioni:

- $expand: supportata
- $filter: supportata.
- $orderby: supportata.
- $format: non supportata. Il formato accettabile viene identificato nell'intestazione della richiesta HTTP Accept.

- $skip: supportata
- $top: supportata

Per ulteriori informazioni, consultare: MSDN: System Query Options.

Instradamento della partizione

L'instradamento della partizione si basa sull'entità root. Un URI di richiesta deduce un'entità root se il suo percorso risorsa inizia con un'entità root o con un'entità che ha un'associazione diretta o indiretta con l'entità. In un ambiente con partizioni, qualsiasi richiesta che non può dedurre un'entità root verrà respinta. Tutte le richieste che deducono un'entità root verranno instradate alla partizione corretta.

Per ulteriori informazioni sulla definizione di uno schema con le associazioni e le entità root, consultare Modello di dati scalabile in eXtreme Scale e Partizionamento.

Richieste Invoke

Le richieste Invoke non sono supportate. Per ulteriori informazioni, consultare: MSDN: Invoke Request.

Richiesta batch

I client possono eseguire il batch di più serie di modifiche o operazioni di query in un'unica richiesta. Questo riduce il numero di viaggi verso il server e consente a più richieste di partecipare in un'unica transazione. Per ulteriori informazioni, consultare MSDN: Batch Request.

Richieste con tunnel

Le richieste con tunnel non sono supportate. Per ulteriori informazioni, consultare MSDN: Tunneled Requests.

Protocolli di richiesta per il servizio dati REST

In generale, i protocolli per l'interazione con i servizi dati REST sono uguali a quelli descritti nel protocollo WCF Data Services AtomPub. Tuttavia, eXtreme Scale fornisce dettagli aggiuntivi dalla prospettiva di Entity Model di eXtreme Scale. Prima di leggere questa sezione, è necessario avere dimestichezza con i protocolli WCF Data Services. In alternativa, è possibile leggere questa sezione insieme alla sezione relativa al protocollo WCF Data Services.

Sono forniti degli esempi che illustrano la richiesta e la risposta. Tali esempi si applicano sia ai servizi dati REST di eXtreme Scale sia a WCF Data Services. Poiché i browser Web possono solo recuperare dati, le operazioni CUD (create, update and delete) devono essere eseguite da un altro client, come ad esempio Java, JavaScript™, RUBY o PHP.

Richieste di richiamo con il servizio dati REST

La richiesta RetrieveEntity viene utilizzata da un client per richiamare un'entità eXtreme Scale. Il payload della risposta contiene i dati dell'entità in formato AtomPub o JSON. Inoltre, è possibile utilizzare l'operatore di sistema $expand per espandere le relazioni. Le relazioni sono rappresentate in linea nella risposta del servizio dati come documento feed Atom (relazione a molti) o documento voce Atom (relazione ad uno).

Suggerimento: Per ulteriori dettagli relativi al protocollo RetrieveEntity definito in WCF Data Services, fare riferimento a: MSDN: RetrieveEntity Request.

Richiamo di un'entità

L'esempio di RetrieveEntity riportato di seguito richiama un'entità Customer con chiave.

AtomPub

- Metodo

  GET
- URI richiesta:

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')
- Intestazione richiesta:

  Accept: application/atom+xml
- Payload richiesta:

  None
- Intestazione risposta:

  Content-Type: application/atom+xml
- Payload risposta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/
restservice" xmlns:d= "http://schemas.microsoft.com/ado/2007/
08/dataservices" xmlns:m = "http://schemas.microsoft.com/ado/2007/
08/dataservices/metadata" xmlns = "http://www.w3.org/2005/Atom">

<category term = "NorthwindGridModel.Customer" scheme = "http://
schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
<id>http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('ACME')</id>
    <title type = "text"/>
    <updated>2009-12-16T19:52:10.593Z</updated>
    <author>
        <name/>
    </author>
    <link rel = "edit" title = "Customer" href = "Customer('ACME')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/
     dataservices/related/
orders" type = "application/atom+xml;type=feed" title =
  "orders" href ="Customer('ACME')/orders"/>
    <content type = "application/xml">
        <m:properties>
            <d:customerId>ACME</d:customerId>
            <d:city m:null = "true"/>
            <d:companyName>RoaderRunner</d:companyName>
            <d:contactName>ACME</d:contactName>
            <d:country m:null = "true"/>
            <d:version m:type = "Edm.Int32">3</d:version>
        </m:properties>
    </content>
</entry>
```

- Codice risposta:

  200 OK

JSON

- Metodo

GET

- URI richiesta:

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('ACME')
- Intestazione richiesta:

  Accept: application/json
- Payload richiesta:

  None
- Intestazione risposta:

  Content-Type: application/json
- Payload risposta:

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Customer('ACME')",
   "type":"NorthwindGridModel.Customer"},
   "customerId":"ACME",
   "city":null,
   "companyName":"RoaderRunner",
   "contactName":"ACME",
   "country":null,
   "version":3,
   "orders":{"__deferred":{"uri":"http://localhost:8080/
      wxsrestservice/restservice/
   NorthwindGrid/Customer('ACME')/orders"}}}}
```

- Codice risposta:

  200 OK

Query

È possibile utilizzare una query anche con una richiesta RetrieveEntitySet o RetrieveEntity. La query è specificata dall'operatore $filter del sistema.

Per i dettagli relativi all'operatore $filter, fare riferimento a: MSDN: Filter System Query Option ($filter)

Il protocollo OData supporta diverse espressioni comuni. Il servizio dati REST eXtreme Scale supporta un sottoinsieme delle espressioni definite nella specifica:

- Espressioni booleane:
  - eq, ne, lt, le, gt, ge
  - negate
  - not
  - parentesi
  - and, or
- Espressioni aritmetiche:
  - add
  - sub
  - mul
  - div
- Letterali primitivi
  - String
  - date-time
  - decimal

  - single
  - double
  - int16
  - int32
  - int64
  - binary
  - null
  - byte

Le seguenti espressioni *non* sono disponibili:

- Espressioni booleane:
  - isof
  - cast
- Espressioni di richiamo dei metodi
- Espressioni aritmetiche:
  - mod
- Letterali primitivi:
  - Guid
- Espressioni del membro

Per un elenco completo ed una descrizione delle espressioni disponibili in Microsoft WCF Data Services, consultare la sezione 2.2.3.6.1.1: Common Expression Syntax

L'esempio riportato di seguito illustra una richiesta RetrieveEntity con una query. In questo esempio, vengono richiamati tutti i clienti il cui nome di contatto è “RoadRunner”. L'unico cliente che corrisponde a tale filtro è Customer('ACME'), come illustrato nel payload di risposta.

Limitazione: questa query funzionerà solo per le entità non partizionate. Se Customer è partizionato, è necessaria la chiave che appartiene al cliente.

AtomPub

- Metodo: GET
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?$filter=contactName eq 'RoadRunner'
- Intestazione richiesta: Accept: application/atom+xml
- Payload di input: None
- Intestazione risposta: Content-Type: application/atom+xml
- Payload risposta:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<feed
 xml:base="http://localhost:8080/wxsrestservice/restservice"
 xmlns:d="http://schemas.microsoft.com/ado/2007/08/
   dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/
   dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <title type="text">Customer</title>
 <id>http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer </id>
 <updated>2009-09-16T04:59:28.656Z</updated>
```

```
 <link rel="self" title="Customer" href="Customer" />
 <entry>
  <category term="NorthwindGridModel.Customer"
   scheme="http://schemas.microsoft.com/ado/2007/08/
   dataservices/scheme" />
  <id>
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
  Customer('ACME')</id>
  <title type="text" />
  <updated>2009-09-16T04:59:28.656Z</updated>
  <author>
   <name />
  </author>
  <link rel="edit" title="Customer" href="Customer('ACME')" />
  <link
   rel="http://schemas.microsoft.com/ado/2007/08/dataservices/
      related/orders"
   type="application/atom+xml;type=feed" title="orders"
   href="Customer('ACME')/orders" />
  <content type="application/xml">
     <m:properties>
            <d:customerId>ACME</d:customerId>
            <d:city m:null = "true"/>
            <d:companyName>RoaderRunner</d:companyName>
            <d:contactName>ACME</d:contactName>
            <d:country m:null = "true"/>
            <d:version m:type = "Edm.Int32">3</d:version>
       </m:properties>
  </content>
 </entry>
</feed>
```

- Codice risposta: 200 OK

JSON

- Metodo: GET
- URI richiesta:

  http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer?$filter=contactName eq 'RoadRunner'
- Intestazione richiesta: Accept: application/json
- Payload richiesta: None
- Intestazione risposta: Content-Type: application/json
- Payload risposta:

```
{"d":[{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Customer('ACME')",
   "type":"NorthwindGridModel.Customer"},
   "customerId":"ACME",
   "city":null,
   "companyName":"RoaderRunner",
   "contactName":"ACME",
   "country":null,
   "version":3,
   "orders":{"__deferred":{"uri":"http://localhost:8080/
     wxsrestservice/restservice/NorthwindGrid/
     Customer('ACME')/orders"}}}]}
```

- Codice risposta: 200 OK

Operatore di sistema $expand

L'operatore di sistema $expand può essere utilizzato per espandere le associazioni. Le associazioni sono rappresentate in linea nella risposta del servizio dati. Le

associazioni a più valori (a molti) sono rappresentate come documento feed Atom o array JSON. Le associazioni a valore singolo (a uno) sono rappresentate come documento voce Atom o oggetto JSON.

Per ulteriori dettagli relativi all'operatore di sistema $expand, fare riferimento a: Expand System Query Option ($expand).

Di seguito è riportato un esempio di utilizzo dell'operatore di sistema $expand. In questo esempio, viene richiamata l'entità Customer('IBM') a cui sono associati gli ordini 5000, 5001 ed altri. La clausola $expand è impostata su “orders”, per cui la raccolta degli ordini viene espansa come serializzata nel payload della risposta. Vengono visualizzati solo gli ordini 5000 e 5001.

AtomPub

- Metodo: GET
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders
- Intestazione richiesta: Accept: application/atom+xml
- Payload richiesta: None
- Intestazione risposta: Content-Type: application/atom+xml
- Payload risposta:

```
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/restservice"
   xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
   xmlns:m = "http://schemas.microsoft.com/ado/2007/08/dataservices/
   metadata" xmlns = "http://www.w3.org/2005/Atom">
<category term = "NorthwindGridModel.Customer" scheme = "http://schemas.

microsoft.com/ado/2007/08/dataservices/scheme"/>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
    Customer('IBM')</id>
    <title type = "text"/>
    <updated>2009-12-16T22:50:18.156Z</updated>
    <author>
        <name/>
    </author><link rel = "edit" title = "Customer" href = "Customer('IBM')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/dataservices/
    related/orders" type = "application/atom+xml;type=feed" title =
    "orders" href = "Customer('IBM')/orders">
        <m:inline>
            <feed>
                <title type = "text">orders</title>
                <id>http://localhost:8080/wxsrestservice/restservice/
             NorthwindGrid/Customer('IBM')/orders</id>
                <updated>2009-12-16T22:50:18.156Z</updated>
                <link rel = "self" title = "orders" href = "Customer('IBM')/orders"/>
                <entry>
                    <category term = "NorthwindGridModel.Order" scheme =
              "http://schemas.microsoft.com/ado/2007/08/
              dataservices/scheme"/>
                <id>http://localhost:8080/wxsrestservice/restservice/
            NorthwindGrid/Order(orderId=5000,customer_customerId=
              'IBM')</id>
                <title type = "text"/>
                <updated>2009-12-16T22:50:18.156Z</updated>
                <author>
                    <name/>
                </author>
                <link rel = "edit" title = "Order" href =
            "Order(orderId=5000,customer_customerId='IBM')"/>
                <link rel = "http://schemas.microsoft.com/ado/2007/08/
```

```
            dataservices/related/customer" type = "application/
            atom+xml;type=entry" title ="customer" href =
            "Order(orderId=5000,customer_customerId='IBM')/customer"/>
                     <link rel = "http://schemas.microsoft.com/ado/2007/08/
            dataservices/related/orderDetails" type = "application/
             atom+xml;type=feed" title ="orderDetails" href =
             "Order(orderId=5000,customer_customerId='IBM')/orderDetails"/>
                     <content type = "application/xml">
                         <m:properties>
                             <d:orderId m:type = "Edm.Int32">5000</d:orderId>
                             <d:customer_customerId>IBM</d:customer_customerId>
                             <d:orderDate m:type = "Edm.DateTime">
                  2009-12-16T19:46:29.562</d:orderDate>
                             <d:shipCity>Rochester</d:shipCity>
                             <d:shipCountry m:null = "true"/>
                             <d:version m:type = "Edm.Int32">0</d:version>
                         </m:properties>
                     </content>
                  </entry>
                  <entry>
                     <category term = "NorthwindGridModel.Order" scheme =
             "http://schemas.microsoft.com/ado/2007/08/
             dataservices/scheme"/>
                     <id>http://localhost:8080/wxsrestservice/restservice/
            NorthwindGrid/Order(orderId=5001,customer_customerId=
            'IBM')</id>
                     <title type = "text"/>
                     <updated>2009-12-16T22:50:18.156Z</updated>
                     <author>
                         <name/></author>
                     <link rel = "edit" title = "Order" href = "Order(
orderId=5001,customer_customerId='IBM')"/>
                     <link rel = "http://schemas.microsoft.com/ado/2007/
              08/dataservices/related/customer" type =
              "application/atom+xml;type=entry" title =
              "customer" href = "Order(orderId=5001,customer_customerId=
              'IBM')/customer"/>
                     <link rel = "http://schemas.microsoft.com/ado/2007/08/
              dataservices/related/orderDetails" type = "application/atom+xml;type=feed" title =
              "orderDetails" href = "Order(orderId=5001,customer_customerId='IBM')/orderDetails"/
                     <content type = "application/xml">
                         <m:properties>
                             <d:orderId m:type = "Edm.Int32">5001</d:orderId>
                             <d:customer_customerId>IBM</d:customer_customerId>
                             <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:
                50:11.125</d:orderDate>
                             <d:shipCity>Rochester</d:shipCity>
                             <d:shipCountry m:null = "true"/>
                             <d:version m:type = "Edm.Int32">0</d:version>
                         </m:properties>
                     </content>
                  </entry>
               </feed>
            </m:inline>
         </link>
         <content type = "application/xml">
            <m:properties>
               <d:customerId>IBM</d:customerId>
               <d:city m:null = "true"/>
               <d:companyName>IBM Corporation</d:companyName>
               <d:contactName>John Doe</d:contactName>
               <d:country m:null = "true"/>
               <d:version m:type = "Edm.Int32">4</d:version>
            </m:properties>
         </content>
      </entry>
```

- Codice risposta: 200 OK

JSON

- Metodo: GET
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')?$expand=orders
- Intestazione richiesta: Accept: application/json
- Payload richiesta: None
- Intestazione risposta: Content-Type: application/json
- Payload risposta:

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Customer('IBM')",
 "type":"NorthwindGridModel.Customer"},
"customerId":"IBM",
"city":null,
"companyName":"IBM Corporation",
"contactName":"John Doe",
"country":null,
"version":4,
"orders":[{"__metadata":{"uri":"http://localhost:8080/
    wxsrestservice/restservice/NorthwindGrid/Order(
    orderId=5000,customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
"orderId":5000,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260992789562)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(
   orderId=5000,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:
     8080/wxsrestservice/restservice/NorthwindGrid/
     Order(orderId=5000,customer_customerId='IBM')/
     orderDetails"}}},
{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
     restservice/NorthwindGrid/Order(orderId=5001,
     customer_customerId='IBM')","type":
     "NorthwindGridModel.Order"},
"orderId":5001,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260993011125)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/
NorthwindGrid/Order(orderId=5001,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(
   orderId=5001, customer_customerId='IBM')/
   orderDetails"}}}]}}
```

- Codice risposta: 200 OK

Richiamo di non-entità con i servizi dati REST

Il servizio dati REST consente di richiamare, oltre alle entità, anche le raccolte di entità e le proprietà.

Richiamo di una raccolta di entità

Un client può utilizzare una richiesta RetrieveEntitySet per richiamare una serie di entità eXtreme Scale. Le entità sono rappresentate come documento feed Atom o

come array JSON nel payload della risposta. Per ulteriori dettagli relativi al protocollo RetrieveEntitySet definito in WCF Data Services, fare riferimento a: MSDN: RetrieveEntitySet Request.

L'esempio di richiesta RetrieveEntitySet riportato di seguito richiama tutte le entità Order associate all'entità Customer('IBM'). Vengono visualizzati solo gli ordini 5000 e 5001.

AtomPub

- Metodo: GET
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders
- Intestazione richiesta: Accept: application/atom+xml
- Payload richiesta: None
- Intestazione risposta: Content-Type: application/atom+xml
- Payload risposta:

```
<?xml version="1.0" encoding="utf-8"?>
<feed xml:base = "http://localhost:8080/wxsrestservice/restservice"
   xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
   xmlns:m = "http://schemas.microsoft.com/ado/2007/08/dataservices/
   metadata" xmlns = "http://www.w3.org/2005/Atom">
    <title type = "text">Order</title>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
     Order</id>
    <updated>2009-12-16T22:53:09.062Z</updated>
    <link rel = "self" title = "Order" href = "Order"/>
    <entry>
        <category term = "NorthwindGridModel.Order" scheme = "http://
      schemas.microsoft.com/
      ado/2007/08/dataservices/scheme"/>
        <id>http://localhost:8080/wxsrestservice/restservice/
      NorthwindGrid/Order(orderId=5000,customer_customerId=
      'IBM')</id>
        <title type = "text"/>
        <updated>2009-12-16T22:53:09.062Z</updated>
        <author>
            <name/>
        </author>
        <link rel = "edit" title = "Order" href = "Order(orderId=5000,
      customer_customerId='IBM')"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
      dataservices/related/customer"
      type = "application/atom+xml;type=entry"
      title = "customer" href = "Order(orderId=5000,
      customer_customerId='IBM')/customer"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
      dataservices/related/orderDetails"
      type = "application/atom+xml;type=feed"
      title = "orderDetails" href = "Order(orderId=5000,customer_customerId='IBM')/
        orderDetails"/>
        <content type = "application/xml">
            <m:properties>
                <d:orderId m:type = "Edm.Int32">5000</d:orderId>
                <d:customer_customerId>IBM</d:customer_customerId>
                <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:
          46:29.562</d:orderDate>
                <d:shipCity>Rochester</d:shipCity>
                <d:shipCountry m:null = "true"/>
                <d:version m:type = "Edm.Int32">0</d:version>
            </m:properties>
        </content>
    </entry>
```

```
    <entry>
        <category term = "NorthwindGridModel.Order" scheme = "http://
      schemas.microsoft.com/ado/2007/08/dataservices/scheme"/>
        <id>http://localhost:8080/wxsrestservice/restservice/
      NorthwindGrid/Order(orderId=5001, customer_customerId='IBM')
    </id>
        <title type = "text"/>
        <updated>2009-12-16T22:53:09.062Z</updated>
        <author>
            <name/>
        </author>
        <link rel = "edit" title = "Order" href = "Order(orderId=5001,
       customer_customerId='IBM')"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
       dataservices/related/customer"
       type = "application/atom+xml;type=entry"
       title = "customer" href = "Order(orderId=5001,
         customer_customerId='IBM')/customer"/>
        <link rel = "http://schemas.microsoft.com/ado/2007/08/
         dataservices/related/orderDetails"
       type = "application/atom+xml;type=feed"
       title = "orderDetails" href = "Order(orderId=5001,
         customer_customerId='IBM')/orderDetails"/>
        <content type = "application/xml">
            <m:properties>
                <d:orderId m:type = "Edm.Int32">5001</d:orderId>
                <d:customer_customerId>IBM</d:customer_customerId>
                <d:orderDate m:type = "Edm.DateTime">2009-12-16T19:50:
          11.125</d:orderDate>
                <d:shipCity>Rochester</d:shipCity>
                <d:shipCountry m:null = "true"/>
                <d:version m:type = "Edm.Int32">0</d:version>
            </m:properties>
        </content>
    </entry>
</feed>
```

- Codice risposta: 200 OK

JSON

- Metodo: GET
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/orders
- Intestazione richiesta: Accept: application/json
- Payload richiesta: None
- Intestazione risposta: Content-Type: application/json
- Payload risposta:

```
{"d":[{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
   restservice/NorthwindGrid/Order(orderId=5000,
   customer_customerId='IBM')",
"type":"NorthwindGridModel.Order"},
"orderId":5000,
"customer_customerId":"IBM",
"orderDate":"\/Date(1260992789562)\/",
"shipCity":"Rochester",
"shipCountry":null,
"version":0,
"customer":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5000,customer_customerId='IBM')/customer"}},
"orderDetails":{"__deferred":{"uri":"http://localhost:8080/
   wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5000,customer_customerId='IBM')/orderDetails"}}},
{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/
```

```
      restservice/NorthwindGrid/
     Order(orderId=5001,
     customer_customerId='IBM')",
  "type":"NorthwindGridModel.Order"},
  "orderId":5001,
  "customer_customerId":"IBM",
  "orderDate":"\/Date(1260993011125)\/",
  "shipCity":"Rochester",
  "shipCountry":null,
  "version":0,
  "customer":{"__deferred":{"uri":"http://localhost:8080/
     wxsrestservice/restservice/NorthwindGrid/Order(orderId=
     5001,customer_customerId='IBM')/customer"}},
  "orderDetails":{"__deferred":{"uri":"http://localhost:8080/
     wxsrestservice/restservice/NorthwindGrid/Order(orderId=
     5001,customer_customerId='IBM')/orderDetails"}}}]}
```

- Codice risposta: 200 OK

Richiamo di una proprietà

È possibile utilizzare una richiesta RetrievePrimitiveProperty per ottenere il valore di una proprietà di un'istanza dell'entità eXtreme Scale. Il valore della proprietà è rappresentato come formato XML per le richieste AtomPub e come oggetto JSON per le richieste JSON nel payload della risposta. Per ulteriori dettagli relativi alla richiesta RetrievePrimitiveProperty, fare riferimento a MSDN: RetrievePrimitiveProperty Request.

L'esempio di richiesta RetrievePrimitiveProperty riportato di seguito richiama la proprietà contactName dell'entità Customer('IBM').

AtomPub

- Metodo: GET
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Intestazione richiesta: Accept: application/xml
- Payload richiesta: None
- Intestazione risposta: Content-Type: application/atom+xml
- Payload risposta:

```
<contactName xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
    John Doe
</contactName>
```

- Codice risposta: 200 OK

JSON

- Metodo: GET
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Intestazione richiesta: Accept: application/json
- Payload richiesta: None
- Intestazione risposta: Content-Type: application/json
- Payload risposta: `{"d":{"contactName":"John Doe"}}`
- Codice risposta: 200 OK

Richiamo del valore di una proprietà

È possibile utilizzare una richiesta RetrieveValue per ottenere il valore non elaborato di una proprietà su un'istanza dell'entità eXtreme Scale. Il valore della proprietà è rappresentato come valore non elaborato nel payload della risposta. Se il tipo di entità è uno dei tipi riportati di seguito, il tipo di supporto della risposta è "text/plain". In caso contrario, il tipo di supporto della risposta è "application/octet-stream". Tali tipi sono:

- Tipi Java primitivi e rispettivi wrapper
- java.lang.String
- byte[]
- Byte[]
- char[]
- Character[]
- enums
- java.math.BigInteger
- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp

Per ulteriori dettagli relativi alla richiesta RetrieveValue, fare riferimento a MSDN: RetrieveValue Request.

L'esempio di richiesta RetrieveValue riportato di seguito richiama il valore non elaborato della proprietà contactName dell'entità Customer('IBM').

- Metodo richiesta: GET
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName/$value
- Intestazione richiesta: Accept: text/plain
- Payload richiesta: None
- Intestazione risposta: Content-Type: text/plain
- Payload risposta: John Doe
- Codice risposta: 200 OK

Richiamo di un link

È possibile utilizzare una richiesta RetrieveLink per ottenere i link che rappresentano un'associazione ad uno oppure un'associazione a molti. Per l'associazione a uno, il link è da un'istanza dell'entità eXtreme Scale ad un'altra ed è rappresentato nel payload della risposta. Per l'associazione a molti, i link sono da un'istanza dell'entità eXtreme Scale a tutte le altre in una raccolta di entità eXtreme Scale specificata e la risposta è rappresentata come una serie di link nel payload della risposta. Per ulteriori dettagli relativi alla richiesta RetrieveLink, fare riferimento a MSDN: RetrieveLink Request.

Di seguito è riportato un esempio di richiesta RetrieveLink. In questo esempio, viene richiamata l'associazione tra l'entità Order(orderId=5000,customer_customerId='IBM') ed il relativo cliente. La risposta visualizza l'URI dell'entità Customer.

AtomPub

- Metodo: GET
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- Intestazione richiesta: Accept: application/xml
- Payload richiesta: None
- Intestazione risposta: Content-Type: application/xml
- Payload risposta:

```
<?xml version="1.0" encoding="utf-8"?>
<uri>http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('IBM')</uri>
```

- Codice risposta: 200 OK

JSON

- Metodo: GET
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- Intestazione richiesta: Accept: application/json
- Payload richiesta: None
- Intestazione risposta: Content-Type: application/json
- Payload risposta: {"d":{"uri":"http:\/\/localhost:8080\/wxsrestservice\/restservice\/NorthwindGrid\/Customer('IBM')"}}

Richiamo dei metadati del servizio

È possibile utilizzare una richiesta RetrieveServiceMetadata per ottenere il documento CSDL (conceptual schema definition language), che descriva il modello di dati associato al servizio dati REST di eXtreme Scale. Per ulteriori dettagli relativi alla richiesta RetrieveServiceMetadata, fare riferimento a MSDN: RetrieveServiceMetadata Request.

Richiamo del documento del servizio

È possibile utilizzare una richiesta RetrieveServiceDocument per richiamare il documento del servizio che descrive la raccolta di risorse esposte dal servizio dati REST di eXtreme Scale. Per ulteriori dettagli relativi alla richiesta RetrieveServiceDocument, fare riferimento a MSDN: RetrieveServiceDocument Request.

Richieste di inserimento con i servizi dati REST

È possibile utilizzare una richiesta InsertEntity per inserire una nuova istanza dell'entità eXtreme Scale, potenzialmente con nuove entità correlate, nel servizio dati REST di eXtreme Scale.

Richiesta di inserimento di entità

È possibile utilizzare una richiesta InsertEntity per inserire una nuova istanza dell'entità eXtreme Scale, potenzialmente con nuove entità correlate, nel servizio dati REST di eXtreme Scale. Durante l'inserimento di un'entità, il client può specificare se la risorsa o entità deve essere automaticamente collegata ad altre entità esistenti nel servizio dati.

Il client deve includere le informazioni di bind richieste nella rappresentazione della relazione associata nel payload della richiesta.

Oltre a supportare l'inserimento di una nuova istanza EntityType (E1), la richiesta InsertEntity consente l'inserimento di nuove entità relative a E1 (descritte da una relazione di entità) in una singola richiesta. Ad esempio, quando viene inserito un Customer('IBM'), è possibile inserire tutti gli ordini con Customer('IBM'). Questa forma di richiesta InsertEntity è conosciuta anche come *inserimento completo*. Con un "inserimento completo", le entità correlate devono essere rappresentate utilizzando la rappresentazione serializzata della relazione associata a E1 che identifica il link alle entità correlate (da inserire).

Le proprietà dell'entità da inserire sono specificate nel payload della richiesta. Le proprietà sono analizzate dal servizio dati REST e quindi impostate sulla proprietà corrispondente sull'istanza dell'entità. Per il formato AtomPub, la proprietà è specificata come un elemento XML <d:PROPERTY_NAME>. Per JSON, la proprietà è specificata come una proprietà di un oggetto JSON.

Se una proprietà non è presente nel payload della richiesta, il servizio dati REST imposta il valore della proprietà dell'entità sul valore predefinito Java. Tuttavia, il backend del database potrebbe non accettare tale valore predefinito, ad esempio, se la colonna non è nullable nel database. In questo caso, viene restituito il codice risposta 500 per indicare un errore del server interno.

Se nel payload sono specificate proprietà duplicate, viene utilizzata l'ultima proprietà. Tutti i valori precedenti per la stessa proprietà sono ignorati dal servizio dati REST.

Se il payload contiene una proprietà non esistente, il servizio dati REST restituisce il codice risposta 400 (richiesta errata) per indicare che la sintassi della richiesta inviata dal client non era corretta.

Se mancano proprietà chiave, il servizio dati REST restituisce il codice risposta 400 (richiesta errata) per indicare una proprietà chiave mancante.

Se il payload contiene un link ad un'entità correlata con una chiave non esistente, il servizio dati REST restituisce il codice risposta 404 (non trovato) per indicare che non è possibile trovare l'entità collegata.

Se il payload contiene un link ad un'entità correlata con un nome associazione non corretto, il servizio dati REST restituisce il codice risposta 400 (richiesta errata) per indicare che non è possibile trovare il link.

Se il payload contiene più di un link ad una relazione ad uno, verrà utilizzato l'ultimo link. Tutti i link precedenti per la stessa associazione vengono ignorati.

Per ulteriori dettagli sulla richiesta InsertEntity, consultare MSDN Library: InsertEntity Request.

Una richiesta InsertEntity inserisce un'entità Customer con chiave 'IBM'.

AtomPub

- Metodo: POST
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Intestazione richiesta: Accept: application/atom+xml Content-Type: application/atom+xml
- Payload richiesta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<content type="application/xml">
  <m:properties>
   <d:customerId>Rational</d:customerId>
   <d:city>Rochester</d:city>
   <d:companyName>Rational</d:companyName>
   <d:contactName>John Doe</d:contactName>
   <d:country>USA</d:country>
  </m:properties>
 </content>
</entry>
```

- Intestazione risposta: Content-Type: application/atom+xml
- Payload risposta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
<content type="application/xml">
  <m:properties>
   <d:customerId>Rational</d:customerId>
   <d:city>Rochester</d:city>
   <d:companyName>Rational</d:companyName>
   <d:contactName>John Doe</d:contactName>
   <d:country>USA</d:country>
  </m:properties>
 </content>
</entry>
Response Header:
Content-Type: application/atom+xml
Response Payload:
<?xml version="1.0" encoding="utf-8"?>
<entry xml:base = "http://localhost:8080/wxsrestservice/restservice" xmlns:d =
   "http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m =
     "http://schemas.microsoft.com/
   ado/2007/08/dataservices/metadata" xmlns = "http://www.w3.org/2005/Atom">
    <category term = "NorthwindGridModel.Customer" scheme = "http://schemas.
     microsoft.com/ado/2007/08/dataservices/scheme"/>
    <id>http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/
     Customer('Rational')</id>
    <title type = "text"/>
    <updated>2009-12-16T23:25:50.875Z</updated>
    <author>
        <name/>
    </author>
    <link rel = "edit" title = "Customer" href = "Customer('Rational')"/>
    <link rel = "http://schemas.microsoft.com/ado/2007/08/dataservices/related/
```

```
    orders" type = "application/atom+xml;type=feed"
    title = "orders" href = "Customer('Rational')/orders"/>
    <content type = "application/xml">
        <m:properties>
            <d:customerId>Rational</d:customerId>
            <d:city>Rochester</d:city>
            <d:companyName>Rational</d:companyName>
            <d:contactName>John Doe</d:contactName>
            <d:country>USA</d:country>
            <d:version m:type = "Edm.Int32">0</d:version>
        </m:properties>
    </content>
</entry>
```

- Codice risposta: 201 Created

JSON

- Metodo: POST
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer
- Intestazione richiesta: Accept: application/json Content-Type: application/json
- Payload richiesta:

```
{"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country": "USA",}
```

- Intestazione risposta: Content-Type: application/json
- Payload risposta:

```
{"d":{"__metadata":{"uri":"http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('Rational')",
"type":"NorthwindGridModel.Customer"},
"customerId":"Rational",
"city":null,
"companyName":"Rational",
"contactName":"John Doe",
"country":"USA",
"version":0,
"orders":{"__deferred":{"uri":"http://localhost:8080/wxsrestservice/restservice/
   NorthwindGrid/Customer('Rational')/orders"}}}}
```

- Codice risposta: 201 Created

Richiesta di inserimento link

È possibile utilizzare una richiesta InsertLink per creare un nuovo Link tra due istanze dell'entità eXtreme Scale . L'URI della richiesta deve essere risolto in un'associazione eXtreme Scale a molti. Il payload della richiesta contiene un singolo link che punta all'entità di destinazione dell'associazione a molti.

Se l'URI della richiesta InsertLink rappresenta un'associazione a uno, il servizio dati REST restituisce una risposta 400 (richiesta errata).

Se l'URI della richiesta InsertLink punta ad un'associazione che non esiste, il servizio dati REST restituisce una risposta 404 (non trovato) che indica che non è possibile trovare il link.

Se il payload contiene un link con una chiave che non esiste, il servizio dati REST restituisce una risposta 404 (non trovato) che indica che non è possibile trovare l'entità collegata.

Se il payload contiene più di un link, il servizio dati REST di eXtreme Scale analizzerà il primo link. I link rimanenti vengono ignorati.

Per ulteriori dettagli sulla richiesta InsertLink, consultare: MSDN Library: InsertLink Request.

L'esempio di richiesta InsertLink riportato di seguito crea un link da Customer('IBM') a Order(orderId=5000,customer_customerId='IBM').

AtomPub

- Metodo: POST
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$link/orders
- Intestazione richiesta: Content-Type: application/xml
- Payload richiesta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId=
   5000,customer_customerId='IBM')</uri>
```

- Payload risposta: nessuno
- Codice risposta: 204 No Content

JSON

- Metodo: POST
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/$links/orders
- Intestazione richiesta: Content-Type: application/json
- Payload richiesta:

```
{"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Order(orderId
    =5000,customer_customerId='IBM')"}
```

- Payload risposta: nessuno
- Codice risposta: 204 No Content

Richieste di aggiornamento con i servizi dati REST

Il servizio dati REST di WebSphere eXtreme Scale supporta le richieste di aggiornamento per entità, proprietà primitive dell'entità e così via.

Aggiornamento di un'entità

È possibile utilizzare una richiesta UpdateEntity per aggiornare un'entità eXtreme Scale esistente. Il client può utilizzare un metodo HTTP PUT per sostituire un'entità eXtreme Scale esistente oppure un metodo HTTP MERGE per unire le modifiche in un'entità eXtreme Scale esistente.

Durante l'aggiornamento dell'entità, il client può specificare se l'entità (oltre ad essere aggiornata) deve essere automaticamente collegata ad altre entità esistenti nel servizio dati e correlate mediante associazioni con valore singolo (ad uno).

La proprietà dell'entità da aggiornare è nel payload della richiesta. La proprietà viene analizzata dal servizio dati REST e quindi impostata sulla proprietà corrispondente sull'entità. Per il formato AtomPub, la proprietà è specificata come un elemento XML <d:PROPERTY_NAME>. Per JSON, la proprietà è specificata come una proprietà di un oggetto JSON.

Se una proprietà non è presente nel payload della richiesta, il servizio dati REST imposta il valore della proprietà dell'entità sul valore predefinito java per il metodo HTTP PUT. Tuttavia, il backend del database potrebbe non accettare tale valore predefinito se, ad esempio, la colonna non è nullable nel database. Viene restituito il codice di risposta 500 (errore del server interno) per indicare un errore del server interno. Se una proprietà non è presente nel payload della richiesta HTTP MERGE, il servizio dati REST non modificherà il valore della proprietà esistente.

Se nel payload sono specificate proprietà duplicate, viene utilizzata l'ultima proprietà. Tutti i valori precedenti con lo stesso nome proprietà sono ignorati dal servizio dati REST.

Se il payload contiene una proprietà non esistente, il servizio dati REST restituisce il codice di risposta 400 (richiesta errata) per indicare che la sintassi della richiesta inviata dal client non era corretta.

Come parte della serializzazione di una risorsa, se il payload di una richiesta Update contiene una delle proprietà chiave per l'entità, il servizio dati REST ignora tali valori chiave perché le chiavi dell'entità non sono modificabili.

Per dettagli sulla richiesta UpdateEntity, consultare: MSDN Library: UpdateEntity Request.

Una richiesta UpdateEntity aggiorna il nome della città di Customer('IBM') in 'Raleigh'.

AtomPub

- Metodo: PUT
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Intestazione richiesta: Content-Type: application/atom+xml
- Payload richiesta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<entry xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
 xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
 xmlns="http://www.w3.org/2005/Atom">
 <category term="NorthwindGridModel.Customer"
  scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
 <title />
 <updated>2009-07-28T21:17:50.609Z</updated>
 <author>
  <name />
 </author>
 <id />
 <content type="application/xml">
  <m:properties>
   <d:customerId>IBM</d:customerId>
   <d:city>Raleigh</d:city>
   <d:companyName>IBM Corporation</d:companyName>
   <d:contactName>Big Blue</d:contactName>
   <d:country>USA</d:country>
  </m:properties>
 </content>
</entry>
```

- Payload risposta: nessuno
- Codice risposta: 204 No Content

JSON

- Metodo: PUT
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
- Intestazione richiesta: Content-Type: application/json
- Payload richiesta:

```
{"customerId":"IBM",
"city":"Raleigh",
"companyName":"IBM Corporation",
"contactName":"Big Blue",
"country":"USA",}
```

- Payload risposta: nessuno
- Codice risposta: 204 No Content

Aggiornamento di una proprietà primitiva dell'entità

La richiesta UpdatePrimitiveProperty può aggiornare un valore della proprietà di un'entità eXtreme Scale. La proprietà ed il valore da aggiornare sono nel payload della richiesta. La proprietà non può essere una proprietà chiave perché eXtreme Scale non consente ai client di modificare le chiavi dell'entità.

Per ulteriori dettagli sulla richiesta UpdatePrimitiveProperty, consultare: MSDN Library: UpdatePrimitiveProperty Request.

Di seguito è riportato un esempio di richiesta UpdatePrimitiveProperty. In questo esempio, il nome della città di Customer('IBM') viene aggiornato in 'Raleigh'.

AtomPub

- Metodo: PUT
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city
- Intestazione richiesta: Content-Type: application/xml
- Payload richiesta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<city xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices">
 Raleigh
</city>
```

- Payload risposta: nessuno
- Codice risposta: 204 No Content

JSON

- Metodo: PUT
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city
- Intestazione richiesta: Content-Type: application/json
- Payload richiesta: {"city":"Raleigh"}
- Payload risposta: nessuno
- Codice risposta: 204 No Content

Aggiornamento del valore di una proprietà primitiva dell'entità

La richiesta UpdateValue può aggiornare un valore della proprietà non elaborato di un'entità eXtreme Scale. Il valore da aggiornare è rappresentato come valore non elaborato nel payload della richiesta. La proprietà non può essere una proprietà chiave perché eXtreme Scale non consente ai client di modificare le chiavi dell'entità.

Il tipo di contenuto della richiesta può essere “text/plain” o “application/octet-stream”, in base al tipo di proprietà. Per ulteriori dettagli, fare riferimento alla sezione 6.3.1.4.

Per ulteriori dettagli sulla richiesta UpdateValue, consultare: MSDN Library: UpdateValue Request

Di seguito è riportato un esempio di richiesta UpdateValue. In questo esempio, il nome della città di Customer('IBM') viene aggiornato in 'Raleigh'.

- Metodo: PUT
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/city/$value
- Intestazione richiesta: Content-Type: text/plain
- Payload richiesta: Raleigh
- Payload risposta: nessuno
- Codice risposta: 204 No Content

Aggiornamento di un link

È possibile utilizzare la richiesta UpdateLink per stabilire una associazione tra due istanze dell'entità eXtreme Scale. L'associazione può essere una relazione con valore singolo (ad uno) o con più valori (a molti).

L'aggiornamento di un link tra due istanze dell'entità eXtreme Scale può non solo stabilire ma anche rimuovere le associazioni. Ad esempio, se il client stabilisce un'associazione ad uno tra un'entità Order(orderId=5000,customer_customerId='IBM') e l'entità e l'istanza Customer('ALFKI'), deve dividere l'entità Order(orderId=5000,customer_customerId='IBM') e l'entità dalla relativa istanza Customer attualmente associata.

Se non è possibile trovare alcuna delle istanze dell'entità specificate nella richiesta UpdateLink, il servizio dati REST restituisce una risposta 404 (non trovato).

Se l'URI della richiesta UpdateLink specifica un'associazione non esistente, il servizio dati REST restituisce una risposta 404 (non trovato) per indicare che non è possibile trovare il link.

Se l'URI specificato nel payload della richiesta UpdateLink non viene risolto nella stessa entità o chiave specificate nell'URI, se esistenti, il servizio dati REST di eXtreme Scale restituisce la risposta 400 (richiesta errata - Bad Request).

Se il payload della richiesta UpdateLink contiene più link, il servizio dati REST analizza solo il primo link. Gli altri link vengono ignorati.

Per ulteriori dettagli sulla richiesta UpdateLink, consultare: MSDN Library: UpdateLink Request.

Di seguito è riportato un esempio di richiesta UpdateLink. In questo esempio, viene aggiornata la relazione cliente dell'entità Order(orderId=5000,customer_customerId='IBM') e da Customer('IBM') a Customer('IBM').

Attenzione: L'esempio precedente è solo a scopo illustrativo. Poiché tutte le associazioni sono generalmente associazioni chiave per una griglia partizionata, non è possibile modificare il link.

AtomPub

- Metodo: PUT
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer
- Intestazione richiesta: Content-Type: application/xml
- Payload richiesta:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<uri>
 http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')
</uri>
```

- Payload risposta: nessuno
- Codice risposta: 204 No Content

JSON

- Metodo: PUT
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(orderId=5000,customer_customerId='IBM')/$links/customer
- Intestazione richiesta: Content-Type: application/xml
- Payload richiesta: {"uri": "http://host:1000/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')"}
- Payload risposta: nessuno
- Codice risposta: 204 No Content

Richieste di eliminazione con i servizi dati REST

Il servizio dati REST di WebSphere eXtreme Scale può eliminare entità, valori di proprietà e collegamenti.

Eliminazione di un'entità

La richiesta DeleteEntity può eliminare un'entità eXtreme Scale dal servizio dati REST.

Se una relazione all'entità da eliminare ha l'impostazione cascade-delete, il servizio dati Rest di eXtreme Scale eliminerà la/le entità correlata/e. Per ulteriori dettagli sulla richiesta DeleteEntity, fare riferimento a MSDN Library: DeleteEntity Request.

Le seguente richiesta DeleteEntity elimina il cliente con chiave 'IBM'.

- Metodo: DELETE
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')

- Payload richiesta: None
- Payload risposta: None
- Codice risposta: 204 No Content

Eliminazione di un valore di proprietà

La richiesta DeleteValue imposta su null una proprietà di un'entità eXtreme Scale.

Qualunque proprietà di un'entità eXtreme Scale può essere impostata su null con una richiesta DeleteValue. Per impostare una proprietà su null, accertarsi che:

- Per ciascun tipo di numero primitivo ed il relativo wrapper, BigInteger o BigDecimal, il valore della proprietà sia impostato su 0.
- Per il tipo Boolean o boolean, il valore della proprietà sia impostato su false.
- Per il tipo char o Character, il valore della proprietà sia impostato sul carattere #X1 (NIL).
- Per il tipo enum, il valore della proprietà sia impostato sul valore enum con ordinale 0.
- Per tutti gli altri tipi, il valore della proprietà sia impostato su null.

Tuttavia, una richiesta di eliminazione di questo genere può essere respinta dal backend del database se, ad esempio, non è possibile impostare la proprietà su null nel database. In tal caso, il servizio dati REST restituisce un codice risposta 500 (errore server interno). Per ulteriori dettagli sulla richiesta DeleteValue, fare riferimento a: MSDN Library: DeleteValue Request.

Di seguito è riportato un esempio di richiesta DeleteValue. In questo esempio, si imposta su null il nome del contatto (contactName) di Customer('IBM').

- Metodo: DELETE
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Customer('IBM')/contactName
- Payload richiesta: None
- Payload risposta: None
- Codice risposta: 204 No Content

Eliminazione di un link

La richiesta DeleteLink può rimuovere un'associazione tra due istanze di entità eXtreme Scale. L'associazione può essere una relazione a-uno o una relazione a-molti. Tuttavia, una richiesta di eliminazione di questo genere può essere respinta dal backend del database se, ad esempio, è impostato il vincolo di chiave esterna. In tal caso, il servizio dati REST restituisce un codice risposta 500 (errore server interno). Per ulteriori dettagli sulla richiesta DeleteLink, fare riferimento a: MSDN Library: DeleteLink Request.

La seguente richiesta DeleteLink rimuove l'associazione tra Order(101) ed il relativo Customer associato.

- Metodo: DELETE
- URI richiesta: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/Order(101)/$links/customer
- Payload richiesta: None
- Payload risposta: None
- Codice risposta: 204 No Content

Contemporaneità ottimistica

Il servizio dati REST di eXtreme Scale utilizza un modello di blocco ottimistico utilizzando le intestazioni HTTP native: If-Match, If-None-Match e ETag. Queste intestazioni vengono inviate nei messaggi di richiesta e di risposta per trasmettere le informazioni sulla versione di un'entità dal server al client e dal client al server.

Per maggiori dettagli sulla contemporaneità ottimistica, consultare MSDN Library: Optimistic Concurrency (ADO.NET).

Il servizio dati REST di eXtreme Scale abilita la contemporaneità ottimistica per un'entità se un attributo di versione viene definito nello schema per quella entità. È possibile definire una proprietà della versione nello schema entità mediante un'annotazione @Version per classi Java o un attributo <version/> per entità definite utilizzando un file XML descrittore di entità. Il servizio dati REST di eXtreme Scale propaga automaticamente il valore della proprietà della versione al client nell'intestazione ETag per singole risposte di entità, utilizzando un attributo m:etag nel payload per più risposte XML di entità e un attributo etag nel payload per più risposte JSON di entità.

Per maggiori dettagli sulla definizione di uno schema di entità eXtreme Scale, consultare “Definizione di uno schema di entità” a pagina 62.

Capitolo 5. Programmazione con API di sistema e plug-in

Un plug-in è un componente che fornisce una funzione ai componenti collegabili, che includono ObjectGrid e BackingMap. Per utilizzare in modo più efficace eXtreme Scale come griglia di dati in-memory o come spazio di elaborazione del database, determinare attentamente come ottimizzare al massimo le prestazioni con i plug-in disponibili.

Introduzione ai plug-in

Un plug-in WebSphere eXtreme Scale è un componente che fornisce un certo tipo di funzioni ai componenti collegabili che includono ObjectGrid e BackingMap. WebSphere eXtreme Scale fornisce svariati punti di collegamento per consentire ai provider di applicazioni e di cache di integrare con vari archivi dati, API di client alternativi e di migliorare le prestazioni complessive della cache. Il prodotto viene commercializzato con svariati plug-in preinstallati e predefiniti ma è possibile anche costruire plug-in personalizzati con le applicazioni.

Tutti i plug-in sono classi concrete che implementano uno o più interfacce di plug-in eXtreme Scale. Di queste classi vengono poi create istanze e vengono inizializzate e richiamate al momento opportuno dall'ObjectGrid. ObjectGrid e BackingMaps consentono che i plug-in personalizzati vengano registrati.

Plug-in di ObjectGrid

Sono disponibili i seguenti plug-in per un'istanza ObjectGrid:

- **TransactionCallback**: un plug-in TransactionCallback fornisce eventi del ciclo di vita della transazione.
- **ObjectGridEventListener**: un plug-in ObjectGridEventListener fornisce eventi del ciclo di vita di ObjectGrid per l'ObjectGrid, i frammenti e le transazioni.
- **SubjectSource, ObjectGridAuthorization, SubjectValidation**: eXtreme Scale fornisce svariati endpoint di sicurezza per consentire meccanismi di autenticazione da integrare con eXtreme Scale. (Solo lato server)
- **MapAuthorization** (solo lato server)
- **JPATxCallback** (solo lato server)
- **Sottoclassi di JPATxCallback**

Requisiti comuni del plug-in ObjectGrid

ObjectGrid crea istanze ed inizializza istanze di plug-in utilizzando le convenzioni JavaBeans. Tutte le precedenti implementazioni di plug-in hanno i seguenti requisiti:

- La classe plug-in deve essere una classe pubblica di livello superiore.
- La classe plug-in deve fornire un costruttore pubblico, senza argomento.
- La classe plug-in deve essere disponibile nel percorso di classe sia per i server che per i client (secondo quanto appropriato).
- Gli attributi devono essere impostati utilizzando i metodi delle proprietà di stile JavaBeans.

- I plug-in, se non espressamente annotato, vengono registrati prima che ObjectGrid venga inizializzato e non possono essere modificati dopo l'inizializzazione di ObjectGrid.

Plug-in BackingMap

I plug-in di seguito riportati sono disponibili per una BackingMap:

- **Programma di eliminazione**: un plug-in Evictor è un meccanismo predefinito che viene fornito per eliminare le voci della cache e un plug-in per creare programmi di eliminazione personalizzati.
- **ObjectTransformer**: un plug-in ObjectTransformer consente di serializzare, deserializzare e copiare oggetti nella cache.
- **OptimisticCallback**: un plug-in OptimisticCallback consente di personalizzare le operazioni di controllo versioni e confronto degli oggetti della cache quando si utilizza la strategia di blocco ottimistico.
- **MapEventListener**: un plug-in MapEventListener fornisce notifiche di callback e modifiche significative allo stato della cache che si verificano in una BackingMap.
- **Indicizzazione**: utilizzare la funzione di indicizzazione, rappresentata dal plug-in MapIndexplug-in, per costruire un indice o svariati indici su una mappa BackingMap per supportare l'accesso ai dati non chiave.
- **Programma di caricamento**: un plug-in Loader in una mappa ObjectGrid agisce come una cache della memoria per i dati che tipicamente vengono conservati in un archivio persistente o sullo stesso sistema o su qualche altro sistema. (Solo lato server)

Cicli di vita del plug-in

La maggior parte dei plug-in dispone sia di metodi di inizializzazione che di distruzione o metodi equivalenti, in aggiunta ai metodi per i quali sono stati progettati ad operare. Questi metodi specializzati per ogni plug-in sono disponibili per essere richiamati nei punti funzionali designati. Entrambi i metodi di inizializzazione e di distruzione definiscono il ciclo di vita dei plug-in, che sono controllati dai loro oggetti "owner". Un oggetto owner è l'oggetto che effettivamente utilizza il dato plug-in. Un owner può essere un client di griglia, un server o una mappa di backup.

Quando vengono inizializzati gli oggetti owner, questi richiamano il metodo initialize dei loro plug-in posseduti. Durante il ciclo di distruzione degli oggetti owner, verrà di conseguenza richiamato anche il metodo destroy di plug-in. Per i dettagli sulle specifiche dei metodi initialize e destroy, insieme agli altri metodi adatti a ciascun plug-in, fare riferimento agli argomenti pertinenti a ciascun plug-in.

Come esempio, considerare un ambiente distribuito. Sia l'ObjectGrids lato client che l'ObjectGrids lato server possono avere i loro propri plug-in. Il ciclo di vita di un ObjectGrid lato client e, pertanto, le sue istanze di plug-in sono indipendenti da tutte le istanze di plug-in e dall'ObjectGrid lato server.

In una tale topologia distribuita, supponiamo di avere un ObjectGrid che si chiama "myGrid" definito nel file objectGrid.xml e configurato con un ObjectGridEventListener personalizzato che si chiama myObjectGridEventListener. Il file objectGridDeployment.xml definisce la politica di distribuzione per l'ObjectGrid myGrid. Sia objectGrid.xml che objectGridDeployment.xml vengono

utilizzati per avviare i server contenitore. Durante l'avvio del server contenitore, l'istanza ObjectGrid myGrid lato server verrà inizializzata e verrà richiamato il metodo initialize dell'istanza myObjectGridEventListener posseduta dall'istanza myObjectGrid. Dopo l'avvio del server contenitore, la propria applicazione può collegarsi all'istanza ObjectGrid myGrid lato server e ottenere un'istanza lato client.

Quando si ottiene l'istanza ObjectGrid myGrid lato client, l'istanza myGrid lato client avanzerà attraverso il suo ciclo di inizializzazione e richiamerà il metodo initialize della sua istanza myObjectGridEventListener lato client. Questa istanza myObjectGridEventListener lato client è indipendente dall'istanza myObjectGridEventListener lato server. Il suo ciclo di vita è controllato dal suo proprietario che è l'istanza ObjectGrid myGrid lato client.

Se la propria applicazione si scollega o distrugge l'istanza ObjectGrid myGrid lato client, verrà richiamato automaticamente il metodo destroy dell'istanza posseduta myObjectGridEventListener lato client. Tuttavia, ciò non impatta l'istanza myObjectGridEventListener lato server. Il metodo destroy dell'istanza myObjectGridEventListener lato server verrà richiamato solo durante il ciclo di distruzione dell'istanza ObjectGrid myGrid lato server durante l'interruzione di un server contenitore. Cioè, quando si interrompe un server contenitore, le istanze ObjectGrid contenute verranno distrutte e verrà richiamato il metodo destroy di tutti i plug-in posseduti.

Sebbene l'esempio precedente si applichi specificamente al caso di un'istanza del client e di un server di un ObjectGrid, il possessore di un plug-in può anche essere una BackingMap e bisogna prestare attenzione alla determinazione delle proprie configurazioni per i plug-in che si potrebbero scrivere sulla base di queste considerazioni sul ciclo di vita.

Plug-in per l'eliminazione degli oggetti di cache

WebSphere eXtreme Scale fornisce un meccanismo predefinito per l'eliminazione delle voci della cache ed un plug-in per la creazione di programmi di eliminazione personalizzati. Un programma di eliminazione controlla l'appartenenza delle voci in ogni BackingMap. Il programma di eliminazione predefinito utilizza una politica di eliminazione TTL (time-to-live) per ogni BackingMap. Se si fornisce un meccanismo di programma di eliminazione collegabile, in genere utilizza una politica di eliminazione che si basa sul numero di voci, invece che sul tempo.

Proprietà TimeToLive

Un programma di eliminazione TTL predefinito viene creato con ogni mappa di backup. Il programma di eliminazione predefinito rimuove le voci sulla base del concetto time-to-live. Questo comportamento è definito dall'attributo ttlType, che presenta i seguenti tipi.

Nessuna

Specifica che le voci non scadono mai e quindi non vengono mai rimosse dalla mappa.

Creazione

Specifica che le voci vengono eliminate in base a quando sono state create.

Se si utilizza l'attributo ttlType CREATION_TIME, il programma di eliminazione elimina una voce quando il tempo dalla creazione è uguale al valore dell'attributo TimeToLive, che è impostato in millisecondi nella

configurazione dell'applicazione. Se si imposta il valore dell'attributo TimeToLive su 10 secondi, la voce viene eliminata automaticamente dieci secondi dopo il suo inserimento.

È importante prestare attenzione quando si imposta questo valore per il ttlType CREATION_TIME. Questo programma di eliminazione viene utilizzato al meglio quando esistono quantità ragionevolmente elevate di aggiunte alla cache che vengono utilizzate per una determinata quantità di tempo. Con questa strategia, tutto ciò che viene creato viene rimosso dopo una determinata quantità di tempo.

Il ttlType CREATION_TIME è utile in scenari quali quello in cui le quotazioni delle azioni vengono aggiornate ogni 20 minuti o meno. Si supponga che un'applicazione Web ottenga le quotazioni delle azioni e che non sia importante che essa riceva quelle più aggiornate. In questo caso, le quotazioni delle azioni vengono inserite nella cache in un ObjectGrid per 20 minuti. Dopo 20 minuti, le voci della mappa ObjectGrid scadono e vengono eliminate. Circa ogni venti minuti la mappa ObjectGrid utilizza il plug-in Loader per aggiornare i dati della mappa con i dati recenti del database. Il database viene aggiornato ogni 20 minuti con le quotazioni più recenti delle azioni.

Ultimo accesso

Specifica che le voci vengono eliminate in base a quando sono state accedute l'ultima volta, siano esse state lette o aggiornate.

Ultimo aggiornamento

Specifica che le voci vengono eliminate in base a quando sono state aggiornate l'ultima volta.

Se si sta utilizzando l'attributo ttlType LAST_ACCESS_TIME o LAST_UPDATE_TIME, impostare il TimeToLive su un numero inferiore rispetto a quello che si utilizzerebbe per l'attributo ttlType CREATION_TIME, poiché l'attributo TimeToLive delle voci viene reimpostato ogni volta che queste vengono accedute. In altre parole, se l'attributo TimeToLive è uguale a 15 e una voce esiste da 14 secondi ma viene acceduta, non scade di nuovo per altri 15 secondi. Se si imposta TimeToLive su un numero relativamente elevato, molte voci potrebbero non essere mai eliminate. Tuttavia, se si imposta il valore su qualcosa come 15 secondi, le voci potrebbero essere rimosse quando non vengono accedute spesso.

Il ttlType LAST_ACCESS_TIME o LAST_UPDATE_TIME è utile in scenari quali quello in cui si gestiscono i dati della sessione di un client, utilizzando la mappa ObjectGrid. I dati della sessione devono essere distrutti se il client non li utilizza per un determinato periodo di tempo. Ad esempio, i dati della sessione scadono dopo 30 minuti se non vi è attività da parte del client. In questo caso, per questa applicazione risulta appropriato l'utilizzo di un TTL di tipo LAST_ACCESS_TIME o LAST_UPDATE_TIME con l'attributo TimeToLive impostato su 30 minuti.

Il seguente esempio crea una mappa di backup, imposta l'attributo ttlType del programma di eliminazione predefinito ed imposta la proprietà TimeToLive.

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

La maggior parte delle impostazioni del programma di eliminazione devono essere definite prima di inizializzare l'ObjectGrid.

È possibile anche scrivere dei propri programmi di eliminazione: per ulteriori informazioni, consultare le informazioni relative alla scrittura di un programma di eliminazione personalizzato in *Guida alla programmazione*.

Programmi di eliminazione facoltativi

Il programma di eliminazione TTL predefinito utilizza una politica di eliminazione che si basa sul tempo, e il numero di voci nella BackingMap non influisce sulla scadenza di una voce. È possibile utilizzare un programma di eliminazione collegabile facoltativo per eliminare le voci in base al numero di voci esistenti, invece che in base al tempo.

I seguenti programmi di eliminazione collegabili facoltativi forniscono alcuni algoritmi comunemente utilizzati per decidere quali voci eliminare quando una BackingMap cresce superando i limiti di dimensione definiti:

- Il programma di eliminazione LRUEvictor utilizza l'algoritmo LRU (Least Recently Used) per decidere quali voci eliminare quando una BackingMap supera il numero massimo di voci.
- Il programma di eliminazione LFUEvictor utilizza l'algoritmo LFU (Least Frequently Used) per decidere quali voci eliminare quando la BackingMap supera il numero massimo di voci.

La BackingMap informa un programma di eliminazione ogni volta che vengono create, modificate o rimosse voci in una transazione. La BackingMap tiene traccia di queste voci e sceglie quando eliminare una o più voci dall'istanza BackingMap.

Una BackingMap non ha informazioni di configurazione relative ad una dimensione massima. Le proprietà del programma di eliminazione, invece, vengono impostate per controllare il comportamento del programma di eliminazione. Sia LRUEvictor che LFUEvictor hanno una proprietà relativa alla dimensione massima utilizzata per far sì che il programma di eliminazione inizi ad eliminare le voci dopo che la dimensione massima è stata superata. Così come per il programma di eliminazione TTL, per ridurre al minimo l'impatto sulle prestazioni, i programmi di eliminazione LRU e LFU potrebbero non eliminare immediatamente una voce quando viene raggiunto il numero massimo di voci.

Se gli algoritmi di eliminazione LRU o LFU non risultano adatti ad una determinata applicazione, è possibile scrivere programmi di eliminazione propri per creare una propria strategia di eliminazione.

Eliminazione basata sulla memoria

Importante: L'eliminazione basata sulla memoria è supportata solo su Java Platform, Enterprise Edition Versione 5 o successiva.

Tutti i programmi di eliminazione incorporati supportano l'eliminazione basata sulla memoria, che può essere abilitata sull'interfaccia BackingMap impostando l'attributo evictionTriggers di BackingMap su `MEMORY_USAGE_THRESHOLD`. Per ulteriori informazioni su come impostare l'attributo evictionTriggers su BackingMap, consultare le informazioni relative all'interfaccia BackingMap e al file XML descrittore ObjectGrid in *Guida alla programmazione*.

L'eliminazione basata sulla memoria si fonda sulla soglia di utilizzo dell'heap. Quando su BackingMap si abilita l'eliminazione basata sulla memoria e la BackingMap ha un programma di eliminazione incorporato, se non precedentemente impostata, la soglia di utilizzo viene impostata su una percentuale predefinita della memoria totale.

Quando si utilizza l'eliminazione basata sulla memoria, è necessario configurare la soglia di raccolta dati obsoleti sullo stesso valore dell'utilizzo dell'heap di destinazione. Ad esempio, se la soglia dell'eliminazione basata sulla memoria è impostata sul 50 percento e la soglia della raccolta dati obsoleti è al livello predefinito del 70 percento, l'utilizzo dell'heap può arrivare fino al 70 percento. Questo aumento dell'utilizzo dell'heap si verifica perché l'eliminazione basata sulla memoria viene attivata solo dopo un ciclo di raccolta dati obsoleti.

L'algoritmo dell'eliminazione basata sulla memoria utilizzato da WebSphere eXtreme Scale è sensibile al comportamento dell'algoritmo della raccolta dati obsoleti in uso. Il miglior algoritmo per l'eliminazione basata sulla memoria è il programma di raccolta del livello di prestazione predefinito IBM. Gli algoritmi di raccolta dati obsoleti generazionali possono causare un comportamento indesiderato, pertanto non utilizzare tali algoritmi con l'eliminazione basata sulla memoria.

Per modificare la percentuale della soglia di utilizzo, impostare la proprietà memoryThresholdPercentage nei file delle proprietà del contenitore e del server per i processi server eXtreme Scale.

Durante il runtime, se l'utilizzo della memoria supera la soglia di utilizzo di destinazione, i programma di eliminazione basati sulla memoria avviano l'eliminazione delle voci e cercano di mantenere l'utilizzo della memoria al di sotto della soglia di utilizzo di destinazione. Tuttavia, non vi è alcuna garanzia che la velocità di eliminazione sia tale da evitare un potenziale errore di memoria esaurita se il runtime del sistema continua a consumare memoria velocemente.

Programma di eliminazione TTL (TimeToLive)

WebSphere eXtreme Scale fornisce un meccanismo predefinito per l'eliminazione delle voci della cache ed un plug-in per la creazione di programmi di eliminazione personalizzati. Un programma di eliminazione controlla l'appartenenza delle voci in ogni istanza BackingMap.

Abilitazione del programma di eliminazione TTL in modo programmatico

I programmi di eliminazione TTL sono associati ad istanze BackingMap. Il programma di eliminazione predefinito utilizza una politica di eliminazione TTL (time-to-live) per ogni istanza BackingMap. Se si fornisce un meccanismo di programma di eliminazione collegabile, in genere utilizza una politica di eliminazione che si basa sul numero di voci, invece che sul tempo.

Il seguente frammento di codice utilizza l'interfaccia BackingMap per impostare l'orario di scadenza di ogni voce su 10 minuti dopo la creazione della voce.

```
programmatic time-to-live evictorimport 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 perché indica il valore time-to-live è in secondi. Il codice precedente deve essere eseguito prima che venga richiamato il metodo di inizializzazione sull'istanza ObjectGrid. Gli attributi BackingMap non possono essere modificati dopo l'inizializzazione dell'istanza ObjectGrid. Dopo l'esecuzione del codice, qualsiasi voce inserita nella BackingMap myMap ha un orario di scadenza. Una volta raggiunto l'orario di scadenza, il programma di eliminazione TTL rimuove la voce.

Per impostare la scadenza sull'orario dell'ultimo accesso più 10 minuti, modificare l'argomento trasmesso al metodo setTtlEvictorType da TTLType.CREATION_TIME a TTLType.LAST_ACCESS_TIME. Con questo valore, l'orario di scadenza viene calcolato come orario dell'ultimo accesso più 10 minuti. Quando una voce viene inizialmente creata, l'orario dell'ultimo accesso è l'orario di creazione. Per stabilire l'orario di scadenza sull'ultimo *aggiornamento* anziché semplicemente sull'ultimo *accesso* (se un aggiornamento è interessato o meno), sostituire l'impostazione TTLType.LAST_UPDATE_TIME per l'impostazione TTLType.LAST_ACCESS_TIME.

Quando si utilizza l'impostazione TTLType.LAST_ACCESS_TIME o TTLType.LAST_UPDATE_TIME, è possibile utilizzare le interfacce ObjectMap e JavaMap per sostituire il valore TTL (Time To Live) di BackingMap. Questo meccanismo consente ad un'applicazione di utilizzare un valore time-to-live diverso per ogni voce creata. Si presupponga che il precedente frammento di codice abbia impostato l'attributo ttlType su LAST_ACCESS_TIME e il valore time-to-live su 10 minuti. Un'applicazione può sovrascrivere il valore time-to-live di ogni voce eseguendo il seguente codice 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" );
```

Nel precedente frammento di codice la voce con chiave key1 ha un orario di scadenza pari all'orario di inserimento più 30 minuti come risultato del richiamo del metodo setTimeToLive(1800) sull'istanza ObjectMap. La variabile oldTimeToLive1 è impostata su 600 perché il valore time-to-live della BackingMap viene utilizzato come valore predefinito, se il metodo setTimeToLive non è stato precedentemente richiamato sull'istanza ObjectMap.

La voce con chiave key2 ha un orario di scadenza pari all'orario di inserimento più 20 minuti come risultato del richiamo del metodo setTimeToLive(1200) sull'istanza ObjectMap. La variabile oldTimeToLive2 è impostata su 1800 perché il valore time-to-live del precedente richiamo del metodo ObjectMap.setTimeToLive ha impostato il valore time-to-live su 1800.

L'esempio precedente mostra due voci di mappa inserite nella mappa myMap per le chiavi key1 e key2. In un momento successivo, l'applicazione può ancora aggiornare le voci della mappa pur mantenendo i valori time-to-live utilizzati al momento dell'inserimento per ogni voce di mappa. Il seguente esempio illustra come mantenere i valori time-to-live utilizzando una costante definita nell'interfaccia ObjectMap:

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

Poiché il valore speciale ObjectMap.USE_DEFAULT viene utilizzato nella chiamata al metodo setTimeToLive, la chiave key1 mantiene il proprio valore time-to-live di 1800 secondi e la chiave key2 mantiene il proprio valore time-to-live di 1200 perché quei valori sono stati utilizzati quando queste voci di mappa sono state inserite dalla transazione precedente.

L'esempio precedente mostra anche una nuova voce di mappa per l'inserimento della chiave key3. In questo caso, il valore speciale USE_DEFAULT indica di utilizzare l'impostazione predefinita del valore time-to-live per questa mappa. Il valore predefinito è definito dall'attributo time-to-live di BackingMap. Vedere gli attributi dell'interfaccia BackingMap per informazioni su come viene definito l'attributo time-to-live nell'istanza BackingMap.

Consultare la documentazione dell'API per il metodo setTimeToLive sulle interfacce ObjectMap e JavaMap. La documentazione descrive che viene generata un'eccezione IllegalStateException se il metodo BackingMap.getTtlEvictorType restituisce qualcosa di diverso dal valore TTLType.LAST_ACCESS_TIME o TTLType.LAST_UPDATE_TIME. Le interfacce ObjectMap e JavaMap possono sovrascrivere il valore TTL (Time To Live) solo quando si sta utilizzando l'impostazione LAST_ACCESS_TIME o TTLType.LAST_UPDATE_TIME per il tipo di programma di eliminazione TTL. Il metodo setTimeToLive non può essere utilizzato per sovrascrivere il valore time-to-live quando si utilizza l'impostazione del tipo di programma di eliminazione CREATION_TIME o NONE.

Abilitazione del programma di eliminazione TTL tramite la configurazione XML

Invece di utilizzare l'interfaccia BackingMap per impostare in modo programmatico gli attributi di BackingMap che verranno utilizzati dal programma di eliminazione TTL, utilizzare un file XML per configurare ogni istanza BackingMap. Il seguente codice mostra come impostare questi attributi per tre diverse mappe BackingMap:

enabling time-to-live evictor using XML

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid1">
        <backingMap name="map1" ttlEvictorType="NONE" />
        <backingMap name="map2" ttlEvictorType="LAST_ACCESS_TIME|LAST_UPDATE_TIME"
     timeToLive="1800" />
        <backingMap name="map3" ttlEvictorType="CREATION_TIME"
     timeToLive="1200" />
    </objectgrid>
</objectGrids>
```

L'esempio precedente mostra che l'istanza BackingMap map1 utilizza un tipo di programma di eliminazione TTL NONE. L'istanza BackingMap map2 utilizza un

tipo di programma di eliminazione TTL LAST_ACCESS_TIME o LAST_UPDATE_TIME - specificare solo una o l'altra di queste impostazioni - e impostare un valore TTL di 1800 secondi o 30 minuti. L'istanza BackingMap map3 è definita per l'utilizzo di un tipo di programma di eliminazione TTL CREATION_TIME ed ha come valore time-to-live 1200 secondi, o 20 minuti.

Collegamento di un programma di eliminazione collegabile

Poiché i programmi di eliminazione sono associati con le BackingMap, è necessario utilizzare l'interfaccia BackingMap per specificare il programma di eliminazione collegabile.

Programmi di eliminazione collegabili facoltativi

Il programma di eliminazione TTL predefinito utilizza una politica di eliminazione basata sul tempo, il numero di voci nella BackingMap non influenza l'orario di scadenza di una voce. È possibile utilizzare un programma di eliminazione collegabile facoltativo per eliminare le voci in base al numero di voci esistenti, invece che in base al tempo.

I seguenti programmi di eliminazione collegabili facoltativi forniscono alcuni algoritmi comunemente utilizzati per decidere quali voci eliminare quando una BackingMap cresce superando i limiti di dimensione definiti.

- Il programma di eliminazione LRUEvictor utilizza l'algoritmo LRU (least recently used) per decidere quali voci eliminare quando una BackingMap supera il numero massimo di voci.
- Il programma di eliminazione LFUEvictor utilizza l'algoritmo LFU (least frequently used) per decidere quali voci eliminare quando una BackingMap supera il numero massimo di voci.

La BackingMap informa un programma di eliminazione delle voci create, modificate o rimosse in una transazione. La BackingMap tiene traccia di queste voci e decide quando eliminarne una o più dall'istanza BackingMap.

Un'istanza BackingMap non ha alcuna informazione di configurazione relativa alla dimensione massima. Le proprietà del programma di eliminazione sono impostate, invece, per controllare il comportamento del programma di eliminazione. Sia LRUEvictor che LFUEvictor hanno una proprietà relativa alla massima dimensione utilizzata per far sì che il programma di eliminazione elimini delle voci dopo che la dimensione massima è stata superata. Così come il programma di eliminazione TTL, per ridurre al minimo l'impatto sulle prestazioni, anche i programmi di eliminazione LRU e LFU potrebbero non eliminare subito una voce quando il numero massimo di voci è stato raggiunto.

Se gli algoritmi di eliminazione LRU o LFU non risultano adatti ad una determinata applicazione, è possibile scrivere dei propri programmi di eliminazione per creare la propria strategia di eliminazione.

Utilizzo di programmi di eliminazione collegabili facoltativi

Per aggiungere programmi di eliminazione collegabili facoltativi nella configurazione della BackingMap, è possibile utilizzare la configurazione programmatica o XML.

Collegamento programmatico di un programma di eliminazione collegabile

Poiché i programmi di eliminazione sono associati alle BackingMap, utilizzare l'interfaccia BackingMap per specificare il programma di eliminazione collegabile. Il seguente frammento di codice rappresenta un esempio che riporta come specificare un programma di eliminazione LRUEvictor per la BackingMap map1 ed un programma di eliminazione LFUEvictor per l'istanza della BackingMap map2:

```
plugging in an evictor programmatically
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 precedente frammento mostra un programma di eliminazione LRUEvictor utilizzato per la BackingMap map1 con un numero massimo approssimativo di voci pari a 53.000 (53 * 1000). Per la BackingMap map2 viene utilizzato il programma di eliminazione LFUEvictor con un numero massimo approssimativo di voci pari a 422.000 (211 * 2000). Sia il programma di eliminazione LRU che quello LFU hanno un periodo di attesa che indica quanto tempo il programma di eliminazione attende prima di verificare se vi siano voci da eliminare. Il tempo di attesa viene specificato in secondi. Un valore di 15 secondi rappresenta un buon compromesso per minimizzare l'impatto sulle prestazioni ed evitare che la BackingMap cresca troppo. Lo obiettivo è quello di utilizzare il tempo di attesa maggiore evitando di far crescere troppo la BackingMap.

Il metodo setNumberOfLRUQueues imposta la proprietà LRUEvictor che indica quante code LRU sono utilizzate dal programma di eliminazione per gestire le informazioni LRU. Viene utilizzata una raccolta di code per evitare che ciascuna voce mantenga le informazioni LRU nella stessa coda. Questo approccio migliora le prestazioni minimizzando il numero di voci della mappa da sincronizzare nello stesso oggetto coda. L'aumento del numero di code rappresenta una buon metodo per ridurre l'impatto del programma di eliminazione LRU sulle prestazioni. Un buon punto di partenza è quello di utilizzare come numero di code il 10% del numero di voci. È solitamente meglio utilizzare un numero primo. Il metodo setMaxSize indica quante voci possono essere contenute in un coda. Quando un coda raggiunge il limite massimo di voci, quelle meno utilizzate di recente vengono eliminate appena il programma di eliminazione verifica nuovamente la presenza di voci da eliminare.

Il metodo setNumberOfHeaps imposta la proprietà LFUEvictor che definisce il numero massimo di oggetti heap binari utilizzati da LFUEvictor per gestire le informazioni LFU. Verrà nuovamente utilizzata una raccolta per migliorare le

prestazioni. L'utilizzo del 10% del numero massimo di voci è un buon punto di partenza ed è solitamente meglio utilizzare un numero primo. Il metodo setMaxSize indica il numero di voci massimo per ciascun heap. Quando un'heap raggiunge il limite massimo di voci, quelle meno frequentemente utilizzate vengono eliminate appena il programma di eliminazione verifica nuovamente la presenza di voci da eliminare.

Approccio con configurazione XML al collegamento di un programma di eliminazione collegabile

Invece di utilizzare diverse API per collegare programmaticamente un programma di eliminazione ed impostarne le proprietà, è possibile utilizzare un file XML per configurare ciascuna BackingMap come riportato di seguito:

```
plugging in an evictor using XML
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid">
        <backingMap name="map1" ttlEvictorType="NONE" pluginCollectionRef="LRU" />
        <backingMap name="map2" ttlEvictorType="NONE" pluginCollectionRef="LFU" />
    </objectGrid>
</objectGrids>
<backingMapPluginCollections>
    <backingMapPlugincollection id="LRU">
        <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
            <property name="maxSize" type="int" value="1000" description="set max size
       for each LRU queue" />
            <property name="sleepTime" type="int" value="15" description="evictor
       thread sleep time" />
            <property name="numberOfLRUQueues" type="int" value="53" description="set number
       of LRU queues" />
        </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="LFU">
        <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
            <property name="maxSize" type="int" value="2000" description="set max size for each LFU heap" />
            <property name="sleepTime" type="int" value="15" description="evictor thread sleep time" />
            <property name="numberOfHeaps" type="int" value="211" description="set number of LFU heaps" />
        </bean>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Eliminazione basata sulla memoria

Tutti i programmi di eliminazione incorporati supportano l'eliminazione basata sulla memoria che può essere abilitata in un'interfaccia BackingMap impostandone l'attributo evictionTriggers su "MEMORY_USAGE_THRESHOLD". Per ulteriori informazioni sull'impostazione dell'attributo evictionTriggers sulla BackingMap, consultare i riferimenti all'interfaccia BackingMap ed alla configurazione di eXtreme Scale.

L'eliminazione basata sulla memoria si fonda sulla soglia di utilizzo dell'heap. Quando su BackingMap si abilita l'eliminazione basata sulla memoria e la BackingMap ha un programma di eliminazione incorporato, se non precedentemente impostata, la soglia di utilizzo viene impostata su una percentuale predefinita della memoria totale.

Per modificare la percentuale della soglia di utilizzo impostare la proprietà memoryThresholdPercentage nel file delle proprietà del contenitore e del server del processo server di eXtreme Scale. Per impostare la soglia di utilizzo di destinazione in un processo client di eXtreme Scale, è possibile utilizzare MemoryPoolMXBean. Consultare anche: File containerServer.props ed Avvio del processo server eXtreme Scale.

Durante il runtime, se l'utilizzo della memoria supera la soglia di utilizzo di destinazione, i programma di eliminazione basati sulla memoria avviano l'eliminazione delle voci e tentano di mantenere l'utilizzo della memoria al di sotto della soglia di utilizzo di destinazione. Tuttavia, non vi è alcuna garanzia che la velocità di eliminazione sia tale da evitare un errore di memoria esaurita se il runtime del sistema continua a consumare velocemente memoria.

Scrittura di un programma di eliminazione personalizzato

WebSphere eXtreme Scale consente di scrivere un'implementazione di un programma di eliminazione personalizzato.

È necessario creare un programma di eliminazione personalizzato che implementa l'interfaccia evictor e segue le convenzioni comuni del plug-in eXtreme Scale. Di seguito è riportata l'interfaccia:

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

- Il metodo initialize viene richiamato durante l'inizializzazione dell'oggetto BackingMap. Tale metodo inizializza un plug-in Evictor con un riferimento a BackingMap ed un riferimento ad un oggetto che implementa l'interfaccia com.ibm.websphere.objectgrid.plugins.EvictionEventCallback.
- Il metodo activate viene richiamato per attivare Evictor. Una volta richiamato tale metodo, Evictor può utilizzare l'interfaccia EvictionEventCallback per eliminare le voci della mappa. Se Evictor prova ad utilizzare EvictionEventCallbackinterface per eliminare le voci della mappa prima che venga richiamato il metodo activate, viene generata un'eccezione IllegalStateException.
- Il metodo apply viene richiamato quando viene eseguito il commit delle transazioni che accedono ad una o più entità di BackingMap. Il metodo apply viene passato come riferimento ad un oggetto che implementa l'interfaccia com.ibm.websphere.objectgrid.plugins.LogSequence. L'interfaccia LogSequence consente al plug-in Evictor di determinare le voci BackingMap create, modificate o rimosse dalla transazione. Evictor utilizza tali informazioni per decidere quali voci eliminare ed il momento in cui eseguire tale operazione.
- Il metodo deactivate viene richiamato per disattivare Evictor. Una volta richiamato tale metodo, è necessario arrestare Evictor utilizzando l'interfaccia EvictionEventCallback per eliminare le voci della mappa. Se Evictor utilizza l'interfaccia EvictionEventcallback dopo il richiamo di tale metodo, viene generata un'eccezione IllegalStateException.
- Il metodo destroy viene richiamato durante l'eliminazione di BackingMap. Tale metodo consente ad Evictor di arrestare tutti i thread creati.

L'interfaccia EvictionEventCallback dispone dei 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 vengono utilizzati dal plug-in Evictor per eseguire chiamate al framework eXtreme Scale come riportato di seguito:

- Il metodo setEvictorData viene utilizzato da un programma di eliminazione per richiedere il framework utilizzato per memorizzare ed associare alcuni oggetti evictor creati alla voce indicata dall'argomento della chiave. I dati sono specifici del programma di eliminazione e sono determinati dalle informazioni richieste dal programma di eliminazione per implementare l'algoritmo utilizzato. Ad esempio, in un algoritmo LFU (least frequently used), il programma di eliminazione esegue un conteggio nel proprio oggetto di dati per tenere traccia del numero di volte in cui viene richiamato il metodo apply con un LogElement che fa riferimento ad una voce per una chiave determinata.
- Il metodo getEvictorData viene utilizzato da un programma di eliminazione per richiamare i dati passati al metodo setEvictorData durante un precedente richiamo del metodo apply. Se i dati del programma di eliminazione per l'argomento key specificato non vengono trovati, viene restituito uno speciale oggetto KEY_NOT_FOUND definito sull'interfaccia EvictorCallback.
- Il metodo evictMapEntries viene utilizzato da un programma di eliminazione per richiedere l'eliminazione di una o più voci della mappa. Ciascun oggetto nel parametro evictorDataList deve implementare l'interfaccia com.ibm.websphere.objectgrid.plugins.EvictorData. Inoltre, la stessa istanza EvictorData passata al metodo setEvictorData deve essere presente nel parametro dell'elenco di dati del programma di eliminazione di questo metodo. Il metodo getKey dell'interfaccia EvictorData viene utilizzato per determinare la voce della mappa da eliminare. La voce della mappa viene eliminata se la voce della cache contiene la stessa istanza EvictorData presente nell'elenco dei dati del programma di eliminazione per questa voce della cache.
- Il metodo evictEntries viene utilizzato dal programma di eliminazione per richiedere l'eliminazione di una o più voci della mappa. Questo metodo viene utilizzato solo se l'oggetto passato al metodo setEvictorData non implementa l'interfaccia com.ibm.websphere.objectgrid.plugins.EvictorData.

Una volta completata una transazione, eXtreme Scale richiama il metodo apply dell'interfaccia Evictor. Tutti i blocchi della transazione acquisiti dalla transazione completata non vengono conservati. Potenzialmente, più thread possono richiamare contemporaneamente il metodo apply e ciascun thread può completare la propria transazione. Poiché i blocchi della transazione sono già rilasciati dalla transazione completata, il metodo apply deve fornire la propria sincronizzazione per garantire di essere sicuro per il thread.

Il motivo dell'implementazione dell'interfaccia EvictorData e dell'utilizzo del metodo evictMapEntries invece del metodo evictEntries è la chiusura di una potenziale finestra temporale. 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 la chiave 1.
2. La transazione 2 viene completata e richiama il metodo apply con un LogSequence che inserisce una nuova voce della mappa per la chiave 1. In altre parole, la transazione 2 crea nuovamente la voce della mappa eliminata dalla transazione 1.

Poiché il programma di eliminazione viene eseguito in modo asincrono rispetto ai thread che eseguono le transazioni, è possibile che quando il programma di eliminazione decide di eliminare la chiave 1, potrebbe eliminare la voce della mappa esistente prima del completamento della transazione 1 oppure la voce della mappa creata dalla transazione 2. Per eliminare finestre temporali e per eliminare

l'incertezza riguardo la versione della voce della mappa della chiave 1 che deve essere eliminata dal programma di eliminazione, implementare l'interfaccia EvictorData mediante l'oggetto passato al metodo setEvictorData. Utilizzare la stessa istanza EvictorData per la durata di una voce della mappa. Quando tale voce della mappa viene eliminata e creata nuovamente da un'altra transazione, il programma di eliminazione deve utilizzare una nuova istanza dell'implementazione EvictorData. Utilizzando l'implementazione EvictorData ed il metodo evictMapEntries, il programma di eliminazione può garantire che la voce della mappa venga eliminata se e solo se la voce della cache associata alla voce della mappa contiene l'istanza EvictorData corretta.

Le interfacce Evictor ed EvictonEventCallback consentono ad un'applicazione di utilizzare un programma di eliminazione che implementa un algoritmo definito dall'utente per l'eliminazione. Il frammento di codice riportato di seguito 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;
// Instance variables
private BackingMap bm;
private EvictionEventCallback evictorCallback;
private LinkedList queue;
private Thread evictorThread;
public void initialize(BackingMap map, EvictionEventCallback callback)
{
    bm = map;
    evictorCallback = callback;
    queue = new LinkedList();
    // spawn evictor thread
    evictorThread = new Thread( this );
    String threadName = "MyEvictorForMap-" + bm.getName();
    evictorThread.setName( threadName );
    evictorThread.start();
}
```

Il codice sopra riportato salva i riferimento agli oggetti map e callback in variabili dell'istanza, in modo che siano disponibili per i metodi apply e destroy. In questo esempio, viene creato un elenco collegato utilizzato come coda FIFO (first in, first out) per l'implementazione di un algoritmo LRU (least recently used). Viene creato un thread ed un riferimento al thread viene conservato come variabile dell'istanza. Utilizzando tale riferimento, il metodo destroy può interrompere e terminare il thread creato.

Ignorando i requisiti di sincronizzazione per rendere il codice sicuro per il thread, il frammento di codice riportato di seguito illustra il modo in cui è possibile 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.getKey();
        LogElement.Type type = elem.getType();
        if ( type == LogElement.INSERT )
        {
            // do insert processing here by adding to front of LRU queue.
            EvictorData data = new EvictorData(key);
            evictorCallback.setEvictorData(key, data);
            queue.addFirst( data );
        }
        else if ( type == LogElement.UPDATE || type == LogElement.FETCH || type == LogElement.TOUCH )
        {
```

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

L'elaborazione di inserimento nel metodo apply generalmente gestisce la creazione di un oggetto di dati del programma di eliminazione che viene passato al metodo setEvictorData dell'interfaccia EvictionEventCallback. Poiché questo programma di eliminazione illustra un'implementazione LRU, EvictorData viene aggiunto alla parte anteriore della coda creata dal metodo initialize. L'elaborazione di aggiornamento nel metodo apply generalmente aggiorna l'oggetto di dati del programma di eliminazione creato da un precedente richiamo del metodo apply (ad esempio, dall'elaborazione insert del metodo apply). Poiché questo programma di eliminazione è un'implementazione LRU, deve spostare l'oggetto EvictorData dalla posizione della coda corrente alla parte anteriore della coda. Il thread del programma di eliminazione creato rimuove l'ultimo oggetto EvictorData nella coda perché l'ultimo elemento della coda rappresenta la voce LRU (least recently used). Il presupposto è che l'oggetto disponga di un metodo getKey in modo che il thread del programma di eliminazione riconosca le chiavi delle entità che devono essere rimosse. Ricordare che in questo esempio i requisiti di sincronizzazione vengono ignorati per rendere sicuro il codice del thread. Un programma di eliminazione personalizzato reale è più complicato in quanto deve prendere in considerazione i colli di bottiglia delle prestazioni e la sincronizzazione che si verificano come risultato dei punti di sincronizzazione.

Il frammento di codice riportato di seguito illustra i metodi destroy e run del thread eseguibile creato dal metodo initialize:

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

// Here is the run method of the thread that was spawned by the initialize method.
public void run()
{
    // Loop until destroy method interrupts this thread.
    boolean continueToRun = true;
    while ( continueToRun )
    {
        try
```

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

Interfaccia RollBackEvictor facoltativa

È possibile implementare l'interfaccia com.ibm.websphere.objectgrid.plugins.RollbackEvictor mediante un plug-in Evictor. Implementando tale interfaccia, il programma di eliminazione 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 è implementata dal programma di eliminazione, viene richiamato il metodo rollingBack. Se l'interfaccia RollbackEvictor non è implementata e viene eseguito il rollback della transazione, i metodi apply e rollingBack non vengono richiamati.

Migliori pratiche per le prestazioni del plug-in Evictor

Se si utilizzano programmi di eliminazione di plug-in, questi non saranno attivi finché non vengono creati e associati ad una mappa di backup. Le migliori pratiche di seguito riportate aumenteranno le prestazioni dei programmi di eliminazione LFU (least frequently used) e LRU (least recently used).

Programma di eliminazione LFU (Least frequently used)

Il concetto di programma di eliminazione LFU è quello di rimuovere dalla mappa le voci che vengono utilizzate con poca frequenza. Le voci della mappa vengono diffuse su un quantitativo impostato di heap binari. Via via che cresce l'utilizzo di una voce particolare della cache, diventa ordinata ad un livello più alto nell'heap. Quando il programma di eliminazione tenta una serie di eliminazioni, rimuove solo le voci della cache posizionate più in basso rispetto ad un punto specifico sull'heap binario. Come risultato, vengono eliminate le voci utilizzate meno frequentemente.

Programma di eliminazione LRU (Least recently used)

Il programma di eliminazione LRU segue lo stesso concetto del programma di eliminazione LFU con alcune differenze. La differenza principale consiste nel fatto che LRU utilizza una coda FIFO (first in, first out) invece di una serie di heap binari. Ogni volta che si accede ad una voce della cache, la voce si sposta in testa alla coda. Di conseguenza, l'inizio della coda contiene le voci della mappa utilizzate più di recente e la fine della coda diventa la parte con le voci della mappa utilizzate meno di recente. Ad esempio, la voce A della cache viene utilizzata 50 volte e la voce B della cache viene utilizzata solo una volta proprio dopo la voce A della cache. In questa situazione, la voce B della cache si trova all'inizio della coda in quanto è stata usata più di recente e la voce A della cache si trova alla fine della coda. Il programma di eliminazione LRU espelle le voci della cache che si trovano alla fine della coda e che sono le voci della mappa utilizzate meno di recente.

Proprietà LFU e LRU e le migliori pratiche per migliorare le prestazioni

Numero di heap

Quando si utilizza il programma di eliminazione LFU, tutte le voci della cache di una mappa particolare vengono ordinate sul numero di heap che è stato specificato, migliorando drasticamente le prestazioni e prevenendo che tutte le eliminazioni vengano sincronizzate su un heap binario che contiene tutto l'ordinamento della mappa. Un quantitativo maggiore di heap velocizza inoltre il tempo richiesto per il riordinamento dell'heap in quanto ogni heap ha un numero minore di voci. Impostare il numero di heap sul 10% del numero di voci presenti nella propria BaseMap.

Numero di code

Quando si utilizza il programma di eliminazione LRU, tutte le voci della cache di una mappa particolare vengono ordinate sul numero di code LRU che viene specificato, migliorando drasticamente le prestazioni e prevenendo che tutte le eliminazioni vengano sincronizzate su una coda che contiene tutto l'ordinamento della mappa. Impostare il numero delle code sul 10% del numero di voci presenti nella propria BaseMap.

Proprietà MaxSize

Quando un programma di eliminazione LFU o LRU inizia ad eliminare le voci, utilizza le proprietà del programma di eliminazione MaxSize per determinare quanti heap binari o elementi di coda LRU eliminare. Ad esempio, si supponga di impostare il numero di heap o di code in modo che abbia circa dieci voci di mappa

in ciascuna coda della mappa. Se la propria proprietà MaxSize è impostata su 7, il programma di eliminazione elimina 3 voci da ogni oggetto coda o heap in modo da portare la dimensione di ogni heap o coda indietro su 7. Il programma di eliminazione elimina voci di mappa da un'heap o coda solo quando quell'heap o coda ha un valore più alto della proprietà MaxSize di elementi in essa contenuti. Impostare MaxSize al 70% della dimensione della propria coda o heap. Per questo esempio, il valore è impostato su 7. È possibile ottenere una dimensione approssimativa di ogni heap o coda dividendo il numero di voci di BaseMap per il numero di heap o di code che sono utilizzate.

Proprietà SleepTime

Un programma di eliminazione non rimuove costantemente voci dalla propria mappa. Infatti è inattivo per un certo quantitativo di tempo, controllando solo la mappa ogni n numero di secondi, dove n si riferisce alla proprietà SleepTime. Questa proprietà influisce positivamente anche sulle prestazioni: l'esecuzione troppo frequente di una curva di eliminazione riduce le prestazioni a causa delle risorse che sono necessarie per l'elaborazione. Tuttavia, il non utilizzo del programma di eliminazione può generare una mappa con voci che non sono necessarie. Una mappa piena di voci non necessarie può influire in modo negativo sia sui requisiti della memoria che sulle risorse di elaborazione richieste per la propria mappa. Un intervallo della curva di eliminazione su quindici secondi è una buona impostazione per la maggior parte delle mappe. Se la mappa è scritta frequentemente e viene utilizzata ad un'elevata velocità di transazione, impostare il valore su un tempo più basso. Se alla mappa si accede poco di frequente, è possibile impostare il tempo su un valore più alto.

Esempio

L'esempio di seguito riportato definisce una mappa, crea un nuovo programma di eliminazione LFU, imposta le proprietà del programma di eliminazione ed imposta la mappa in modo che utilizzi il programma:

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

//Set properties assuming 50,000 map entries
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

L'utilizzo del programma di eliminazione LRU è molto simile all'uso del programma di eliminazione LFU. Segue un esempio:

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");

//Set properties assuming 50,000 map entries
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Notare che solo due righe sono differenti dall'esempio di LFUEvictor.

Plug-in per convertire gli oggetti memorizzati nella cache

Prendere in considerazione la conversione degli oggetti memorizzati nella cache per aumentare le prestazioni della propria cache. È possibile utilizzare il plug-in ObjectTransformer quando l'utilizzo del processore è elevato. Più del 60-70 per cento del tempo totale del processore è utilizzato per serializzare e copiare le voci. Implementando il plug-in ObjectTransformer, è possibile serializzare e deserializzare gli oggetti utilizzando la propria implementazione. È possibile utilizzare il plug-in CollisionArbiter per definire in che modo vengono gestiti i conflitti di modifica nei propri domini.

Sviluppo di arbitri personalizzati per repliche multi-master

Si possono verificare conflitti di modifica se gli stessi record possono essere contemporaneamente modificati in due posti. In una topologia di replica multi-master, i domini rilevano automaticamente i conflitti. Quando un dominio rileva un conflitto, esso richiama un arbitro. Generalmente, i conflitti vengono risolti utilizzando l'arbitro dei conflitti predefinito. Tuttavia, un'applicazione può fornire un arbitro di conflitti personalizzato.

Informazioni su questa attività

Se un dominio riceve una voce replicata che è in conflitto con un record locale, l'arbitro predefinito utilizza le modifiche provenienti dal dominio con il nome lessicale più basso. Ad esempio, se un dominio A e un dominio B generano un conflitto per un record, la modifica del dominio B viene ignorata. Il dominio A conserva la propria versione e il record nel dominio B viene modificato per corrispondere al record del dominio A. I nomi del dominio vengono trasformati in maiuscolo per il confronto.

Un'opzione alternativa per la topologia di replica multi-master è richiamare un plug-in di conflitto personalizzato per decidere il risultato. Queste istruzioni illustrano in che modo sviluppare un arbitro di conflitti personalizzato e configurare una topologia di replica multi-master per utilizzarla.

Procedura

1. Sviluppare un arbitro di conflitti personalizzato e integrarlo nella propria applicazione.

   La classe dovrebbe implementare l'interfaccia:

   ```
   com.ibm.websphere.objectgrid.revision.CollisionArbiter
   ```

   Un plug-in di conflitto dispone di tre opzioni per decidere il risultato di un conflitto. Può scegliere la copia in locale o la copia in remoto o può fornire una versione corretta della voce. Un dominio fornisce le seguenti informazioni per un arbitro di conflitti personalizzato:

   - la versione esterna del record
   - La versione locale del record
   - Un oggetto Session che è necessario utilizzare per creare la versione corretta della voce che è andata in conflitto

   Il metodo del plug-in restituisce un oggetto che indica la sua decisione. Il metodo richiamato dal dominio per richiamare i plug-in deve restituire true o false, dove false indica di ignorare il conflitto – la versione locale resta immutata e eXtreme Scale dimenticherà che essa mai ha ricevuto la versione

esterna. Il metodo restituisce un valore true se il metodo ha utilizzato la sessione fornita per creare una nuova versione unita del record riconciliando la modifica.

2. Nel file `objectgrid.xml` specificare di utilizzare il plug-in dell'arbitro personalizzato.

   L'ID deve essere “CollisionArbiter."

```
<dgc:objectGrid name="revisionGrid" txTimeout="10">
     <dgc:bean className="com.you.your_application.
      CustomArbiter" id="CollisionArbiter">
      <dgc:property name="property" type="java.lang.String"
      value="propertyValue"/>
     </dgc:bean>
</dgc:objectGrid>
```

Plug-in ObjectTransformer

Con il plug-in ObjectTransformer, è possibile serializzare, deserializzare e copiare nella cache per ottenere prestazioni migliori.

Se si verificano problemi nelle prestazioni utilizzando il processore, aggiungere un plug-in ObjectTransformer a ciascuna mappa. Se non si fornisce un plug-in ObjectTransformer, più del 60 70 per cento del tempo totale del processore è utilizzato per serializzare e copiare le voci.

Scopo

Utilizzando il plug-in ObjectTransformer, le applicazioni possono fornire metodi personalizzati per le seguenti operazioni:

- serializzare o deserializzare la chiave per una voce
- serializzare o deserializzare il valore per una voce
- copiare una chiave o un valore per una voce

Se non è fornito alcun plug-in ObjectTransformer, è necessario poter serializzare le chiavi e i valori poiché ObjectGrid utilizza una sequenza serializzata e deserializzata per copiare gli oggetti. Questo metodo è oneroso, perciò utilizzare un plug-in ObjectTransformer quando le prestazioni sono critiche. La copia avviene quando un'applicazione ricerca un oggetto in una transazione per la prima volta. È possibile evitare questa copia impostando la modalità copia della mappa a NO_COPY o ridurre la copia impostando la modalità copia a COPY_ON_READ. Ottimizzare l'operazione di copia quando necessario mediante l'applicazione fornendo un metodo di copia personalizzato in questo plug-in. In tal modo un plug-in può ridurre il sovraccarico dovuto alla copia dal 65-70 per cento al 2/3 per cento del tempo totale del processore.

Le implementazioni predefinite copyKey ecopyValue del metodo prima tentano di utilizzare il metodo clone se viene fornito il metodo. Se nessuna implementazione del metodo clone viene fornita l'implementazione è impostata per impostazione predefinita alla serializzazione.

La serializzazione dell'oggetto viene utilizzata anche quando eXtreme Scale è in esecuzione in modalità distribuita. Il plug-in LogSequence utilizza ObjectTransformer per guidare nella serializzazione delle chiavi e dei valori prima di trasmettere le modifiche ai peer in ObjectGrid. È necessario porre attenzione quando si fornisce un metodo di serializzazione personalizzato invece di utilizzare la serializzazione del kit dello sviluppatore Java incorporato. L'oggetto controllo

versione è una questione complessa e si potrebbero avere problemi con la compatibilità della versione se non si garantisce che i metodi personalizzati sono progettati per il controllo versione.

L'elenco di seguito riportato descrive in che modo eXtreme Scale tenta di serializzare sia le chiavi che i valori:

- se un plug-in ObjectTransformer personalizzato è scritto ed è stato eseguito il plug-in eXtreme Scale richiama i metodi nell'interfaccia ObjectTransformer per serializzare le chiavi e i valori e ottenere le copie delle chiavi oggetto e dei valori.
- Se un plug-in ObjectTransformer personalizzato, eXtreme Scale serializza e deserializza i valori in base all'impostazione predefinita. Se il plug-in predefinito viene utilizzato, ciascun oggetto viene implementato come esternalizzabile o viene implementato come serializzabile.
  - Se l'oggetto supporta l'interfaccia Externalizable, viene richiamato il metodo writeExternal. Gli oggetti che vengono implementati come esternalizzabili conducono a migliori prestazioni.
  - Se l'oggetto non supporta l'interfaccia Externalizable ed implementa l'interfaccia Serializable, l'oggetto viene salvato utilizzando il metodo ObjectOutputStream.

Utilizzo dell'interfaccia ObjectTransformer

Un oggetto ObjectTransformer deve implementare l'interfaccia ObjectTransformer e seguire le convenzioni comuni del plug-in ObjectGrid.

Vengono utilizzati due approcci la configurazione XML e la configurazione programmatica per aggiungere un oggetto ObjectTransformer alla configurazione BackingMap come di seguito riportato.

L'approccio configurazione XML per eseguire il plug-in di un ObkjecyTransformer

Presuppone che il nome della classe dell'implementazione ObjectTransformer sia la casse com.company.org.MyObjectTransformer. Questa classe implementa l'interfaccia ObjectTransformer. Un'implementazione ObjectTransformer può essere configurata utilizzando il seguente XML:

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

    <backingMapPluginCollections>
 <backingMapPluginCollection id="myMap">
      <bean id="ObjectTransformer" className="com.company.org.MyObjectTransformer" />
 </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Plug-in di un oggetto ObjectTransformer in modo programmatico

Il frammento di codice di seguito riportato crea l'oggetto ObjectTransformer personalizzato e lo aggiunge a BackingMap:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap backingMap = myGrid.getMap("myMap");
MyObjectTransformer myObjectTransformer = new MyObjectTransformer();
backingMap.setObjectTransformer(myObjectTransformer);
```

Scenari per l'utilizzo di ObjectTransformer

È possibile utilizzare il plug-in ObjectTransformer nelle seguenti situazioni:

- l'oggetto è Non-serializzabile
- l'oggetto è serializzabile ma si desidera migliorare le prestazioni di serializzazione
- per una copia della chiave o del valore

Nell'esempio di seguito riportato, ObjectGrid viene utilizzato per memorizzare la classe Stock:

```
/**
* Stock object for ObjectGrid demo
*
*
*/
public class Stock implements Cloneable {
    String ticket;
    double price;
    String company;
    String description;
    int serialNumber;
    long lastTransactionTime;
    /**
    * @return Returns the description.
    */
    public String getDescription() {
        return description;
    }
    /**
    * @param description The description to set.
    */
    public void setDescription(String description) {
        this.description = description;
    }
    /**
    * @return Returns the lastTransactionTime.
    */
    public long getLastTransactionTime() {
        return lastTransactionTime;
    }
    /**
    * @param lastTransactionTime The lastTransactionTime to set.
    */
    public void setLastTransactionTime(long lastTransactionTime) {
        this.lastTransactionTime = lastTransactionTime;
    }
    /**
    * @return Returns the price.
    */
    public double getPrice() {
        return price;
    }
    /**
    * @param price The price to set.
    */
    public void setPrice(double price) {
        this.price = price;
    }
    /**
    * @return Returns the serialNumber.
    */
    public int getSerialNumber() {
        return serialNumber;
    }
    /**
    * @param serialNumber The serialNumber to set.
    */
    public void setSerialNumber(int serialNumber) {
```

```
        this.serialNumber = serialNumber;
    }
    /**
    * @return Returns the ticket.
    */
    public String getTicket() {
        return ticket;
    }
    /**
    * @param ticket The ticket to set.
    */
    public void setTicket(String ticket) {
        this.ticket = ticket;
    }
    /**
    * @return Returns the company.
    */
    public String getCompany() {
        return company;
    }
    /**
    * @param company The company to set.
    */
    public void setCompany(String company) {
        this.company = company;
    }
    //clone
    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}
```

È possibile scrivere una classe del trasformatore dell'oggetto personalizzato per la classe Stock:

```
/**
* Custom implementation of ObjectGrid ObjectTransformer for stock object
*
*/
public class MyStockObjectTransformer implements ObjectTransformer {
/* (non-Javadoc)
* @see
* com.ibm.websphere.objectgrid.plugins.ObjectTransformer#serializeKey
* (java.lang.Object,
* java.io.ObjectOutputStream)
*/
public void serializeKey(Object key, ObjectOutputStream stream) throws IOException {
    String ticket= (String) key;
    stream.writeUTF(ticket);
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#serializeValue(java.lang.Object,
java.io.ObjectOutputStream)
*/
public void serializeValue(Object value, ObjectOutputStream stream) throws IOException {
    Stock stock= (Stock) value;
    stream.writeUTF(stock.getTicket());
    stream.writeUTF(stock.getCompany());
    stream.writeUTF(stock.getDescription());
    stream.writeDouble(stock.getPrice());
    stream.writeLong(stock.getLastTransactionTime());
    stream.writeInt(stock.getSerialNumber());
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateKey(java.io.ObjectInputStream)
*/
public Object inflateKey(ObjectInputStream stream) throws IOException, ClassNotFoundException {
    String ticket=stream.readUTF();
    return ticket;
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateValue(java.io.ObjectInputStream)
*/

public Object inflateValue(ObjectInputStream stream) throws IOException, ClassNotFoundException {
    Stock stock=new Stock();
    stock.setTicket(stream.readUTF());
    stock.setCompany(stream.readUTF());
```

```
    stock.setDescription(stream.readUTF());
    stock.setPrice(stream.readDouble());
    stock.setLastTransactionTime(stream.readLong());
    stock.setSerialNumber(stream.readInt());
    return stock;
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyValue(java.lang.Object)
*/
public Object copyValue(Object value) {
    Stock stock = (Stock) value;
    try {
        return stock.clone();
    }
    catch (CloneNotSupportedException e)
    {
        // display exception message    }
}

/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyKey(java.lang.Object)
*/
public Object copyKey(Object key) {
    String ticket=(String) key;
    String ticketCopy= new String (ticket);
    return ticketCopy;
}
}
```

Successivamente, eseguire il plug-in in questa classe personalizzata MyStockObjectTransformer nella BackingMap:

```
ObjectGridManager ogf=ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogf.getObjectGrid("NYSE");
BackingMap bm = og.defineMap("NYSEStocks");
MyStockObjectTransformer ot = new MyStockObjectTransformer();
bm.setObjectTransformer(ot);
```

Prestazione della serializzazione

WebSphere eXtreme Scale utilizza più processi Java per contenere i dati. Questi processi serializzano i dati: convertono, cioè, i dati (che si trovano nel modulo delle istanze dell'oggetto Java) in byte e poi di nuovo in oggetti secondo quanto necessario per spostare i dati tra i processi del client e del server. L'esecuzione del marshalling dei dati è l'operazione più dispendiosa e deve essere indirizzata dallo sviluppatore dell'applicazione durante la progettazione dello schema, configurando la griglia e interagendo con le API di accesso ai dati.

Le routine predefinite di copia e serializzazione Java sono relativamente lente e possono consumare il 60 - 70 per cento del processore in una configurazione tipica. Le seguenti sezioni rappresentano delle scelte per il miglioramento della prestazione della serializzazione.

Scrittura di un ObjectTransformer per ogni BackingMap

Un ObjectTransformer può essere associato ad una BackingMap. La propria applicazione può avere una classe che implementa l'interfaccia di ObjectTransformer e che fornisce implementazioni per le operazioni di seguito riportate:

- Copia di valori
- Chiavi di serializzazione e deserializzazione verso e dai flussi
- Valori di serializzazione e deserializzazione verso e dai flussi.

L'applicazione non ha bisogno di copiare le chiavi poiché le chiavi sono considerate non mutabili.

Per ulteriori informazioni, consultare Plug-in per la serializzazione e la copia di oggetti memorizzati nella cache e Migliori prassi per l'interfaccia ObjectTransformer.

Nota: ObjectTransformer è l'unico richiamato quando ObjectGrid è informato sui dati in fase di trasformazione. Ad esempio, quando vengono utilizzati gli agent dell'API DataGrid, gli stessi agent così come i dati dell'istanza dell'agent o i dati restituiti dall'agent, devono essere ottimizzati utilizzando le tecniche personalizzate di serializzazione. ObjectTransformer non viene richiamato per gli agent delle API DataGrid.

Utilizzo delle entità

Quando si utilizzano le API EntityManager con entità, l'ObjectGrid non memorizza gli oggetti dell'entità direttamente nelle BackingMaps. L'API EntityManager converte l'oggetto dell'entità in oggetti Tupla. Per ulteriori informazioni, consultare Per ulteriori informazioni, consultare l'argomento sull'utilizzo di un programma di caricamento con mappe di entità e tuple in *Guida alla programmazione*. Le mappe dell'entità sono associate automaticamente ad un ObjectTransformer altamente ottimizzato. Ogni volta che viene utilizzata l'API ObjectMap o l'API EntityManager per interagire con le mappe dell'entità, viene richiamata l'entità ObjectTransformer.

Personalizzazione della serializzazione

Esistono alcuni casi in cui gli oggetti devono essere modificati per utilizzare la serializzazione personalizzata, come l'implementazione dell'interfaccia java.io.Externalizable o l'implementazione dei metodi writeObject e readObject per classi che implementano l'interfaccia java.io.Serializable. Le tecniche di serializzazione personalizzata devono essere impiegate quando vengono serializzati gli oggetti utilizzando meccanismi diversi dai metodi delle API ObjectGrid o EntityManager.

Ad esempio, quando oggetti o entità vengono memorizzati come dati dell'istanza in un agent dell'API DataGrid oppure quando l'agent restituisce oggetti o entità, quegli oggetti non vengono trasformati utilizzando un ObjectTransformer. L'agent, tuttavia, utilizzerà automaticamente l'ObjectTransformer quando utilizza l'interfaccia `EntityMixin`. Per ulteriori dettagli, consultare Agent DataGrid e Mappe basate su entità.

Array di byte

Quando si utilizzano le API ObjectMap o DataGrid, gli oggetti chiave e valore vengono serializzati ogni volta che il client interagisce con la griglia e quando viene replicato l'oggetto. Per evitare il sovraccarico della serializzazione, utilizzare gli array di byte invece degli oggetti Java. Gli array di byte sono molto più semplici da inserire in memoria poiché JDK ha un numero inferiore di oggetti da cercare per la raccolta dati obsoleti e possono essere deserializzati solo all'occorrenza. Gli array di byte devono essere utilizzati solo se non è necessario accedere agli oggetti usando query o indici. Poiché i dati vengono memorizzati come byte, è possibile accedere ai dati solo attraverso chiavi.

WebSphere eXtreme Scale può memorizzare dati automaticamente come array di byte utilizzando l'opzione di configurazione della mappa CopyMode.COPY_TO_BYTES oppure può essre gestito manualmente dal client.

Questa opzione memorizzerà in modo efficiente i dati in memoria e può anche deserializzare gli oggetti all'interno dell'array di byte per l'uso mediante query e indici on demand.

Migliori pratiche per l'interfaccia ObjectTransformer

L'interfaccia ObjectTransformer utilizza i callback all'applicazione per fornire implementazioni personalizzate di operazioni comuni e dispendiose quali la serializzazione degli oggetti e le copie complete sugli oggetti.

Panoramica

Per i dettagli relativi all'interfaccia ObjectTransformer, consultare "Plug-in ObjectTransformer" a pagina 228. Da un punto di vista delle prestazioni, e dalle informazioni relative al metodo CopyMode contenute nell'argomento Migliori pratiche del metodo CopyMode, eXtreme Scale chiaramente copia i valori per tutti i casi tranne quando viene utilizzata la modalità NO_COPY. Il meccanismo di copia predefinito impiegato in eXtreme Scale è la serializzazione, nota come un'operazione dispendiosa. In questa situazione viene utilizzata l'interfaccia ObjectTransformer. L'interfaccia ObjectTransformer utilizza i callback all'applicazione per fornire un'implementazione personalizzata di operazioni comuni e dispendiose, come la serializzazione degli oggetti e le copie complete sugli oggetti.

Un'applicazione può fornire un'implementazione dell'interfaccia ObjectTransformer ad una mappa, e eXtreme Scale quindi delega ai metodi su questo oggetto e si affida all'applicazione per fornire una versione ottimizzata di ciascun 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 possibile associare un'interfaccia ObjectTransformer ad una BackingMap utilizzando il seguente codice di esempio:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ottimizzazione della serializzazione e deserializzazione degli oggetti

La serializzazione degli oggetti è in genere la considerazione più importante sulle prestazioni con eXtreme Scale, che utilizza il meccanismo serializzabile predefinito se l'applicazione non fornisce un plug-in ObjectTransformer. Un'applicazione può fornire le implementazioni readObject e writeObject serializzabili o può lasciare che siano gli oggetti ad implementare l'interfaccia esternalizzabile, che è circa dieci volte più rapido. Se gli oggetti nella mappa non possono essere modificati, un'applicazione può associare un'interfaccia ObjectTransformer all'ObjectMap. I metodi di serializzazione e deserializzazione vengono forniti per consentire all'applicazione di fornire codice personalizzato per ottimizzare queste operazioni, dato il loro vasto impatto sulle prestazioni nel sistema. Il metodo di serializzazione serializza l'oggetto nel flusso fornito. Il metodo di deserializzazione fornisce il flusso di input e si aspetta che l'applicazione crei l'oggetto, lo deserializzi utilizzando i dati contenuti nel flusso e restituisce l'oggetto. Le implementazioni dei metodi di serializzazione e deserializzazione devono riflettersi reciprocamente.

Ottimizzazione delle operazioni di copia completa

Dopo che l'applicazione ha ricevuto un oggetto da una ObjectMap, eXtreme Scale esegue una copia completa sul valore dell'oggetto per assicurarsi che la copia nella mappa BaseMap mantenga l'integrità dei dati. L'applicazione può quindi modificare tranquillamente il valore dell'oggetto. Quando viene eseguito il commit della transazione, la copia del valore dell'oggetto nella mappa BaseMap viene aggiornata al nuovo valore modificato e l'applicazione si arresta utilizzando il valore da quel punto in poi. Nella fase del commit si sarebbe potuto copiare l'oggetto di nuovo per effettuare una copia privata. Tuttavia, in questo caso il costo delle prestazioni di questa azione è stato contraccambiato richiedendo al programmatore dell'applicazione di non utilizzare il valore dopo il commit della transazione. L'ObjectTransformer predefinito cerca di utilizzare un clone o una coppia serializzazione-deserializzazione per generare una copia. La coppia serializzazione-deserializzazione è il peggior caso di scenario delle prestazioni. Se l'analisi del profilo rivela che la serializzazione e la deserializzazione è un problema per l'applicazione, scrivere un metodo clone appropriato per creare una copia completa. Se non è possibile alterare la classe, creare un plug-in ObjectTransformer personalizzato ed implementare metodi copyValue e copyKey più efficienti.

Plug-in per il controllo versioni e il confronto degli oggetti della cache.

Utilizzare il plug-in OptimisticCallback per personalizzare le operazioni di controllo versioni e di confronto di oggetti della cache quando si utilizza la strategia di blocco ottimistico.

È possibile fornire un oggetto callback ottimistico puggable che implementi l'interfaccia com.ibm.websphere.objectgrid.plugins.OptimisticCallback. Per le mappe di entità, un plug-in OptimisticCallback ad elevate prestazioni viene automaticamente configurato.

Scopo

Utilizzare l'interfaccia OptimisticCallback per fornire operazioni di confronto ottimistico per i valori di una mappa. Un plug-in OptimisticCallback viene richiesto quando si utilizza una strategia di blocco ottimistico. Il prodotto fornisce un'implementazione OptimisticCallback predefinita. Tuttavia, generalmente la propria applicazione deve collegare la propria implementazione dell'interfaccia OptimisticCallback.

Implementazione predefinita

Il framework eXtreme Scale fornisce un'implementazione predefinita dell'interfaccia OptimisticCallback che viene utilizzata se l'applicazione non è collegata ad un oggetto OptimisticCallback fornito dall'applicazione. L'implementazione predefinita restituisce sempre il valore speciale NULL_OPTIMISTIC_VERSION come oggetto versione per il valore e non aggiorna mai l'oggetto versione. Questa azione esegue il confronto ottimistico per la funzione "no operation". In molti casi, si può non desiderare che si verifichi la funzione "no operation" quando si utilizza una strategia di blocco ottimistico. Le applicazioni devono implementare l'interfaccia OptimisticCallback e collegare le proprie implementazioni OptimisticCallback in modo che non venga utilizzata l'implementazione predefinita. Tuttavia, esiste almeno uno scenario in cui l'implementazione OptimisticCallback predefinita è utile. Considerare la situazione di seguito riportata.

- Un programma di caricamento è collegato alla mappa di backup.

- Il programma di caricamento individua il confronto ottimistico senza necessità dell'assistenza del plug-in OptimisticCallback.

In che modo il programma di caricamento esegue il controllo versione ottimistico senza l'assistenza dell'oggetto OptimisticCallback? Il programma di caricamento ha conoscenza dell'oggetto classe valore e conosce quale campo dell'oggetto valore viene utilizzato come valore del controllo versione ottimistico. Ad esempio, supponiamo che l'interfaccia di seguito illustrata ha utilizzato l'oggetto valore per la mappa Impiegati:

```
public interface Employee
{
    // Sequential sequence number used for optimistic versioning.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Other get/set methods for other fields of Employee object.
}
```

In questo esempio, il programma di caricamento sa che può utilizzare il metodo getSequenceNumber per ottenere le informazioni sulla versione corrente per l'oggetto valore Employee. Il programma di caricamento incrementa il valore restituito per generare un nuovo numero di versione prima di aggiornare la memoria persistente con un nuovo valore Employee. Per un programma di caricamento JDBC (Java database connectivity), viene utilizzato il numero di sequenza corrente nella clausola WHERE di un'istruzione SQL UPDATE sovraqualificata e utilizza il nuovo numero di sequenza generato per impostare la colonna numero di sequenza sul nuovo valore del numero di sequenza. Un'altra possibilità è che il programma di caricamento utilizzi alcune funzioni fornite dal backend che automaticamente aggiornano una colonna nascosta che può essere utilizzata per il controllo versione ottimistico.

In alcune situazioni, una procedura memorizzata o un trigger possono probabilmente essere utilizzati per guidare nella gestione di una colonna che attende le informazioni sul controllo versione. Se il programma di caricamento utilizza una di queste tecniche per la gestione delle informazioni sul controllo versione ottimistico, allora l'applicazione non ha la necessità di fornire l'implementazione OptimisticCallback. L'implementazione predefinita OptimisticCallback è utilizzabile in questo scenario perchè ilprogramma di caricamento può gestire il controllo versione ottimistico senza l'assistenza dell'oggetto OptimisticCallback.

Implementazione predefinita per le entità

Le entità sono memorizzate in ObjectGrid utilizzando gli oggetti tuple. Il comportamento dell'implementazione predefinita OptimisticCallback è simile al comportamento delle mappe non entità. Tuttavia, il campo versione nell'entità viene identificato utilizzando l'annotazione @Version o l'attributo versione nel file XML descrittore dell'entità.

L'attributo versione può essere di uno dei seguenti tipi: int, Integer, short, Short, long, Long o java.sql.Timestamp. Un'entità deve avere solo un attributo versione definito. Impostare solo l'attributo versione durante la costruzione. Dopo che l'entità è divenuta persistente, il valore dell'attributo versione non dovrebbe essere modificato.

Se un attributo versione non è configurato e viene utilizzata la strategia di blocco ottimistico, allora l'intera tupla è implicitamente fornita di versione utilizzando l'intero stato della tupla il che è molto più oneroso.

Nell'esempio di seguito riportato, l'entità Employee ha un attributo versione lungo denominato SequenceNumber:

```
@Entity
public class Employee
{
private long sequence;
    // Sequential sequence number used for optimistic versioning.
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Other get/set methods for other fields of Employee object.
}
```

Scrittura di un plug-in OptimisticCallback plug-in

Un plug-in OptimisticCallback deve implementare l'interfaccia OptimisticCallback e seguire le comuni convenzioni del plug-in ObjectGrid. Per ulteriori informazioni, consultare l'interfaccia OptimisticCallback nella documentazione dell'API.

L'elenco di seguito riportato fornisce una descrizione o considerazione per ciascuno dei metodi nell'interfaccia OptimisticCallback:

NULL_OPTIMISTIC_VERSION

Questo valore speciale viene restituito dal metodo getVersionedObjectForValue se l'implementazione OptimisticCallback non richiede il controllo versione. L'implementazione del plug-in incorporato della casse com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback utilizza questo valore perché il controllo versione è disabilitato quando si specifica questa implementazione del plug-in.

metodo getVersionedObjectForValue

Il metodo getVersionedObjectForValue potrebbe restituire una copia del valore o un attributo del valore che può essere utilizzato per scopi di controllo versione. Questo metodo è richiamato se un oggetto è associato ad una transazione. Quando nessun programma di caricamento è collegato nella mappa di backup, la mappa di backup utilizza questo valore al momento del commit per eseguire un confronto di versione ottimistico. Il confronto di versione ottimistico viene utilizzato dalla mappa di backup per garantire che la versione non sia stata modificata dopo che questa transazione abbia prima eseguito l'accesso alla voce della mappa che è stata modificata da questa transazione. Se un'altra transazione è già stata modificata per questa voce della mappa, il confronto di versione non riesce e la mappa di backup visualizza un'eccezione OptimisticCollisionException per forzare la transazione ad eseguire il roll back. Se un programma di caricamento è collegato alla mappa di backup non utilizza le informazioni sul controllo versione ottimistico. Invece, il programma di caricamento è responsabile di eseguire il confronto di controllo versione ottimistico ed aggiornare le informazioni sul controllo versione quando necessario. Il programma di caricamento generalmente ottiene l'oggetto di controllo versione iniziale dal LogElement passato al metodo batchUpdate nel programma di caricamento, che viene richiamato quando avviene un'operazione di flush o viene eseguito il commit di una transazione.

Il codice riportato di seguito 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 si è dimostrato nell'esempio precedente, l'attributo sequenceNumber viene restituito in un oggetto java.lang.Long object come previsto dal programma di caricamento il che implica che la stessa persona che ha scritto il programma di caricamento ha scritto anche l'implementazione EmployeeOptimisticCallbackImpl o ha lavorato a contatto con la persona che ha implementato EmployeeOptimisticCallbackImpl - ad esempio, ha concordato il valore restituito dal metodo getVersionedObjectForValue. Il plug-in OptimisticCallback predefinito restituisce il valore speciale NULL_OPTIMISTIC_VERSION come oggetto versione.

Metodo updateVersionedObjectForValue

Questo metodo viene richiamato se una transazione è stata aggiornata con un valore ed è necessario un nuovo oggetto fornito di versione. Se il metodo getVersionedObjectForValue restituisce un attributo del valore, questo metodo generalmente aggiorna il valore dell'attributo con un nuovo oggetto versione. Se il metodo getVersionedObjectForValue restituisce una copia del valore, questo metodo generalmente non completa alcuna azione. Il plug-in predefinito OptimisticCallback non completa alcuna azione con questo metodo perché l'implementazione predefinita di getVersionedObjectForValue restituisce sempre il valore speciale NULL_OPTIMISTIC_VERSION come oggetto versione. L'esempio di seguito riportato mostra l'implementazione utilizzata dall'oggetto EmployeeOptimisticCallbackImpl che viene utilizzato nella sezione OptimisticCallback:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Come si è dimostrato nell'esempio precedente, l'attributo sequenceNumber incrementa di uno in modo che la volta successiva quando viene richiamato il metodo getVersionedObjectForValue, il valore java.lang.Long che viene restituito ha un valore a lunga scadenza che è il il valore del numero di sequenza originale più uno, ad esempio, è il valore della versione successiva per questa istanza Employee. Questo esempio implica che la stessa persona che ha scritto il programma di caricamento abbia scritto anche EmployeeOptimisticCallbackImpl o ha lavorato a contatto con la persona che ha implementato EmployeeOptimisticCallbackImpl.

Metodo serializeVersionedValue

Questo metodo scrive il valore fornito di versione per il flusso specificato. A seconda dell'implementazione il valore fornito di versione può essere utilizzato per identificare i conflitti di aggiornamento ottimistici. In alcune implementazioni, il valore fonito di versione, è una copia del valore originale. Altre implementazioni potrebbero avere un numero di sequenza o qualche altro oggetto per indicare la versione del valore. Poiché non si conosce l'implementazione effettiva, questo metodo è fornito per eseguire l'opportuna serializzazione. L'implementazione predefinita richiama il metodo writeObject.

Metodo inflateVersionedValue

Questo metodo prende la versione serializzata e restituisce l'effettivo oggetto valore fornito di versione. A seconda dell'implementazione il valore fornito di versione può essere utilizzato per identificare i conflitti di aggiornamento ottimistici. In alcune implementazioni, il valore fonito di versione, è una copia del valore originale. Altre implementazioni potrebbero avere un numero di sequenza o qualche altro oggetto per indicare la versione del valore. Poiché non si conosce l'implementazione effettiva, questo metodo è fornito per eseguire l'opportuna deserializzazione. L'implementazione predefinita richiama il metodo readObject

Utilizzo dell'oggetto OptimisticCallback fornito dall'applicazione

Si hanno due approcci per aggiungere un oggetto OptimisticCallback fornito dall'applicazione nella configurazione di BackingMap: la configurazione XML e la configurazione programmatica.

L'approccio della configurazione XML per eseguire il plug-in di un oggetto OptimisticCallback

L'applicazione può utilizzare un file XML per eseguire il plug-in del proprio oggetto OptimisticCallback come è mostrato nell'esempio di seguito riportato:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
    <objectGrid name="grid1">
        <backingMap name="employees" pluginCollectionRef="employees" lockStrategy="OPTIMISTIC" />
    </objectGrid>
</objectGrids>

<backingMapPluginCollections>
    <backingMapPluginCollection id="employees">
        <bean id="OptimisticCallback" className="com.xyz.EmployeeOptimisticCallbackImpl" />
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

Plug-in di un oggetto OptimisticCallback in modo programmatico

L'esempio di seguito riportato dimostra come un'applicazione può eseguire il plug-in in un oggetto OptimisticCallback in modo programmatico per la mappa di backup Employee nell'istanza locale ObjectGrid grid1:

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
```

```
ObjectGrid og = ogManager.createObjectGrid( "grid1" );
BackingMap bm = dg.defineMap("employees");
EmployeeOptimisticCallbackImpl cb = new EmployeeOptimisticCallbackImpl();
bm.setOptimisticCallback( cb );
```

Plug-in per la fornitura di listener di eventi

È possibile utilizzare i plug-in ObjectGridEventListener e MapEventListener per configurare le notifiche di diversi eventi nella cache eXtreme Scale. I plug-in del listener sono registrati con un'istanza ObjectGrid o BackingMap come altri plug-in eXtreme Scale e aggiungono punti di personalizzazione e integrazione per applicazioni e fornitori della cache.

Plug-in ObjectGridEventListener

Un plug-in ObjectGridEventListener fornisce eventi del ciclo di vita eXtreme Scale per l'istanza ObjectGrid, frammenti e transazioni. Utilizzare il plug-in ObjectGridEventListener per ricevere notifiche quando si verificano eventi significativi su un ObjectGrid. Questi eventi comprendono l'inizializzazione di ObjectGrid, l'inizio e la fine di una transazione e l'eliminazione di un ObjectGrid. Per eseguire l'ascolto di questi eventi, creare una classe che implementi l'interfaccia ObjectGridEventListener e aggiungerla a eXtreme Scale.

Per ulteriori informazioni sulla scrittura di un plug-in ObjectGridEventListener, consultare “Plugin ObjectGridEventListener” a pagina 242. È possibile inoltre fare riferimento alla documentazione API per ulteriori informazioni.

Aggiunta e rimozione di istanze ObjectGridEventListener

Un ObjectGrid può avere più listener ObjectGridEventListener. Aggiungere e rimuovere i listener utilizzando i metodi addEventListener, setEventListeners e removeEventListener sull'interfaccia ObjectGrid. È possibile inoltre registrare in modo dichiarativo i plug-in ObjectGridEventListener con il file descrittore ObjectGrid. Per gli esempi, consultare “Plugin ObjectGridEventListener” a pagina 242.

Plug-in MapEventListener

Un plug-in MapEventListener fornisce notifiche di callback e modifiche significative allo stato della cache che si verificano per un'istanza BackingMap. Per dettagli sulla scrittura di un plug-in MapEventListener, consultare “Plug-in MapEventListener”. È possibile inoltre fare riferimento alla documentazione API per ulteriori informazioni.

Aggiunte e rimozioni di istanze MapEventListener

Un eXtreme Scale può avere più listener MapEventListener. Aggiungere e rimuovere i listener con i metodi addMapEventListener, setMapEventListeners e removeMapEventListener sull'interfaccia BackingMap. È possibile inoltre registrare in modo dichiarativo i listener MapEventListener con il file descrittore ObjectGrid. Per gli esempi, consultare “Plug-in MapEventListener”.

Plug-in MapEventListener

Un plug-in MapEventListener fornisce notifiche di callback e modifiche significative allo stato della cache che si verificano per un oggetto BackingMap: quando una mappa ha terminato il precaricamento o quando una voce viene

eliminata dalla mappa. Un particolare plug-in MapEventListener è una classe personalizzata che viene scritta implementando l'interfaccia MapEventListener.

Convenzioni del plug-in MapEventListener

Quando si sviluppa un plug-in MapEventListener, è necessario attenersi alle convenzioni di plug-in comuni. Per ulteriori informazioni sulle convenzioni di plug-in, consultare “Introduzione ai plug-in” a pagina 209. Per altri tipi di plug-in listener, consultare “Plug-in per la fornitura di listener di eventi” a pagina 240.

Dopo la scrittura di un'implementazione MapEventListener, è possibile collegarla alla configurazione BackingMap in modo programmatico o con una configurazione XML.

Scrittura di un'implementazione MapEventListener

La propria applicazione può includere un'implementazione del plug-in MapEventListener. Il plug-in deve implementare l'interfaccia MapEventListener per ricevere eventi significativi relativi a una mappa. Gli eventi vengono inviati al plug-in MapEventListener quando una voce viene eliminata dalla mappa e quando termina il precaricamento di una mappa.

Esecuzione del plug-in di un'implementazione MapEventListener utilizzando XML

È possibile configurare un'implementazione MapEventListner utilizzando XML. Il seguente codice XML 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>
    </backingMapPluginCollections>
</objectGridConfig>
```

Fornire questo file all'istanza ObjectGridManager agevola la creazione di questa configurazione. Il seguente frammento di codice mostra il modo in cui creare un'istanza ObjectGrid utilizzando questo file XML. L'istanza ObjectGrid appena creata ha un MapEventListener impostato sulla BackingMap myMap.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
 objectGridManager.createObjectGrid("myGrid", new URL("file:etc/test/myGrid.xml"),
  true, false);
```

Collegamento in modo programmatico di un'implementazione MapEventListener

Il nome classe per l'oggetto MapEventListener personalizzato è la classe com.company.org.MyMapEventListener. Questa classe implementa l'interfaccia

MapEventListener. Il seguente frammento di codice crea l'oggetto MapEventListener personalizzato e lo aggiunge a un oggetto BackingMap:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
BackingMap myMap = myGrid.defineMap("myMap");
MyMapEventListener myListener = new MyMapEventListener();
myMap.addMapEventListener(myListener);
```

Plugin ObjectGridEventListener

Un plug-in ObjectGridEventListener fornisce eventi con ciclo di vita WebSphere eXtreme Scale per ObjectGrid, frammenti e transazioni. Un plugin ObjectGridEventListener fornisce notifiche quando si inizializza o si elimina un ObjectGrid e quando si avvia o si termina una transazione. I plugin ObjectGridEventListener sono classi personalizzate scritte implementando l'interfaccia ObjectGridEventListener. Facoltativamente, l'implementazione include interfacce secondarie ObjectGridEventGroup e segue le convenzioni di plugin eXtreme Scale comuni.

Panoramica

Un plugin ObjectGridEventListener è utile quando si dispone di un plugin Loader ed è necessario inizializzare le connessioni JDBC (Java Database Connectivity) o le connessioni a un backend quando si avviano e si terminano le transazioni. Normalmente, un plugin ObjectGridEventListener e un plugin Loader vengono scritti contemporaneamente.

Scrittura di un plugin ObjectGridEventListener

È necessario che un plugin ObjectGridEventListener implementi l'interfaccia ObjectGridEventListener per ricevere notifiche di eventi eXtreme Scale significativi. Per ricevere ulteriori notifiche di eventi, è possibile implementare le seguenti interfacce. Le seguenti interfacce secondarie sono contenute nell'interfaccia ObjectGridEventGroup:

- Interfaccia ShardEvents
- Interfaccia ShardLifecycle
- Interfaccia TransactionEvents

Per ulteriori informazioni su queste interfacce, consultare la documentazione API.

Eventi frammento (shard)

Quando il servizio catalogo colloca l'elemento primario di partizione o i frammenti di replica in una JVM (Java Virtual Machine), viene creata una nuova istanza ObjectGrid in quella JVM per ospitare quel frammento. Alcune applicazioni che necessitano di avviare i thread sulla JVM ospitano la notifica di richiesta primaria di questi eventi. L'interfaccia ObjectGridEventGroup.ShardEvents dichiara i metodi shardActivate e shardDeactivate. Tali metodi vengono richiamati solo quando un frammento viene attivato come elemento primario e disattivato da un elemento primario. Questi due eventi consentono all'applicazione di avviare thread aggiuntivi quando il frammento è un elemento primario, e di arrestare i thread quando il frammento ritorna alla funzione di replica o diventa inattivo.

Un'applicazione può determinare quale partizione è stata attivata ricercando una BackingMap specifica nel riferimento ObjectGrid fornito al metodo shardActivate

utilizzando il metodo ObjectGrid#getMap. L'applicazione può quindi verificare il numero di partizione utilizzando il metodo BackingMap#getPartitionId(). Le partizioni vengono numerate da 0 fino al numero di partizioni presenti nel descrittore di distribuzione meno uno.

Eventi con ciclo di vita di frammenti

Gli eventi metodo ObjectGridEventListener.initialize e ObjectGridEventListener.destroy vengono consegnati utilizzando l'interfaccia ObjectGridEventGroup.ShardLifecycle.

Eventi di transazione

I metodi ObjectGridEventListener.transactionBegin e ObjectGridEventListener.transactionEnd vengono consegnati attraverso l'interfaccia ObjectGridEventGroup.TransactionEvents.

Vantaggi di questo approccio

Se un plug-in ObjectGridEventListener implementa le interfacce ObjectGridEventListener e ShardLifecycle, gli eventi con ciclo di vita di frammenti sono gli unici eventi consegnati al listener. Dopo aver implementato una qualsiasi delle nuove interfacce interne ObjectGridEventGroup, eXtreme Scale consegna solo gli eventi specifici con le nuove interfacce. Con questa implementazione, il codice può essere compatibile con le versioni precedenti. Se si utilizzano le nuove interfacce interne, il listener può ora ricevere solo gli eventi specifici richiesti.

Utilizzo del plug-in ObjectGridEventListener

Per utilizzare un plug-in ObjectGridEventListener personalizzato, creare prima una classe che implementi l'interfaccia ObjectGridEventListener e tutte le eventuali interfacce secondarie ObjectGridEventGroup facoltative. Aggiungere il listener personalizzato a un ObjectGrid per ricevere la notifica di eventi significativi. Sono possibili due approcci per aggiungere un plug-in ObjectGridEventListener alla configurazione di eXtreme Scale: configurazione programmatica e configurazione XML.

Configurazione di un plug-in ObjectGridEventListener in modo programmatico

Si presupponga che il nome classe del listener eventi eXtreme Scale sia la classe com.company.org.MyObjectGridEventListener. Questa classe implementa l'interfaccia ObjectGridEventListener. Il seguente frammento di codice crea l'interfaccia ObjectGridEventListener personalizzata e la aggiunge a un ObjectGrid.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
MyObjectGridEventListener myListener = new MyObjectGridEventListener();
myGrid.addEventListener(myListener);
```

Configurazione di un plug-in ObjectGridEventListener con XML

È possibile inoltre configurare un plug-in ObjectGridEventListener utilizzando XML. Il seguente codice XML crea una configurazione che è uguale al listener di eventi ObjectGrid creato in modo programmatico descritto. 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" />
            <backingMap name="Book"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Si noti che le dichiarazioni bean vengono prima delle dichiarazioni backingMap. Fornire questo file al plug-in ObjectGridManager per agevolare la creazione di questa configurazione. Il seguente frammento di codice descrive in che modo creare un'istanza ObjectGrid utilizzando questo file XML. L'istanza ObjectGrid creata ha un listener ObjectGridEventListener impostato sull'ObjectGrid myGrid.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid",
 new URL("file:etc/test/myGrid.xml"), true, false);
```

Plug-in per l'indicizzazione personalizzata di oggetti cache

Con un plug-in MapIndexPlugin, o indice, è possibile scrivere strategie indicizzate personalizzate che vanno oltre gli indici integrati forniti da eXtreme Scale.

Per informazioni generali sull'indicizzazione, consultare Indicizzazione.

Per informazioni sull'utilizzo dell'indicizzazione, vedere “Utilizzo dell'indicizzazione per l'accesso ai dati non chiave” a pagina 249.

Le implementazioni MapIndexPlugin devono utilizzare l'interfaccia MapIndexPlugin e devono seguire le comuni convenzioni dei plug-in eXtreme Scale.

Le sezioni riportate di seguito includono alcuni degli metodi importanti dell'interfaccia di indice.

Metodo setProperties

Utilizzare il metodo setProperties per inizializzare il plug-in indice tramite un programma. Il parametro dell'oggetto Properties passato al metodo deve contenere le informazioni di configurazione richieste per inizializzare correttamente il plug-in indice. L'implementazione del metodo setProperties, insieme all'implementazione del metodo getProperties è richiesta in un ambiente distribuito poiché la configurazione del plug-in indice si sposta tra i processi client e server. Di seguito è riportato un esempio di implementazione di questo metodo.

```
setProperties(Properties properties)

// setProperties method sample code
    public void setProperties(Properties properties) {
        ivIndexProperties = properties;

        String ivRangeIndexString = properties.getProperty("rangeIndex");
        if (ivRangeIndexString != null && ivRangeIndexString.equals("true")) {
            setRangeIndex(true);
        }
        setName(properties.getProperty("indexName"));
        setAttributeName(properties.getProperty("attributeName"));

        String ivFieldAccessAttributeString = properties.getProperty("fieldAccessAttribute");
        if (ivFieldAccessAttributeString != null && ivFieldAccessAttributeString.equals("true")) {
            setFieldAccessAttribute(true);
        }

        String ivPOJOKeyIndexString = properties.getProperty("POJOKeyIndex");
```

```
        if (ivPOJOKeyIndexString != null && ivPOJOKeyIndexString.equals("true")) {
            setPOJOKeyIndex(true);
        }
    }
```

Metodo getProperties

Il metodo getProperties estrae la configurazione del plug-in indice da un'istanza MapIndexPlugin. È possibile utilizzare le proprietà estratte per inizializzare un'altra istanza MapIndexPlugin per avere gli stessi stati interni. L'implementazione dei metodi getProperties e setProperties è richiesta in un ambiente distribuito. Di seguito è riportato un esempio di implementazione del metodo getProperties.

```
getProperties()

// getProperties method sample code
    public Properties getProperties() {
        Properties p = new Properties();
        p.put("indexName", indexName);
        p.put("attributeName", attributeName);
        p.put("rangeIndex", ivRangeIndex ? "true" : "false");
        p.put("fieldAccessAttribute", ivFieldAccessAttribute ? "true" : "false");
        p.put("POJOKeyIndex", ivPOJOKeyIndex ? "true" : "false");
        return p;
    }
```

Metodo setEntityMetadata

Il metodo setEntityMetadata viene richiamato dal runtime di WebSphere eXtreme Scale durante l'inizializzazione per impostare gli EntityMetadata della BackingMap associata sull'istanza MapIndexPlugin. Gli EntityMetadata sono richiesti per supportare l'indicizzazione degli oggetti tupla. Una tupla è un dataset che rappresenta un oggetto entità o la relativa chiave. Se la BackingMap è per un'entità, è necessario implementare questo metodo.

Il seguente codice campione implementa il metodo setEntityMetadata.

```
setEntityMetadata(EntityMetadata entityMetadata)

// setEntityMetadata method sample code
    public void setEntityMetadata(EntityMetadata entityMetadata) {
        ivEntityMetadata = entityMetadata;
        if (ivEntityMetadata != null) {
            // this is a tuple map
            TupleMetadata valueMetadata = ivEntityMetadata.getValueMetadata();
            int numAttributes = valueMetadata.getNumAttributes();
            for (int i = 0; i < numAttributes; i++) {
                String tupleAttributeName = valueMetadata.getAttribute(i).getName();
                if (attributeName.equals(tupleAttributeName)) {
                    ivTupleValueIndex = i;
                    break;
                }
            }

            if (ivTupleValueIndex == -1) {
                // did not find the attribute in value tuple, try to find it on key tuple.
                // if found on key tuple, implies key indexing on one of tuple key attributes.
                TupleMetadata keyMetadata = ivEntityMetadata.getKeyMetadata();
                numAttributes = keyMetadata.getNumAttributes();
                for (int i = 0; i < numAttributes; i++) {
                    String tupleAttributeName = keyMetadata.getAttribute(i).getName();
                    if (attributeName.equals(tupleAttributeName)) {
                        ivTupleValueIndex = i;
                        ivKeyTupleAttributeIndex = true;
                        break;
                    }
                }
            }

            if (ivTupleValueIndex == -1) {
                // if entityMetadata is not null and we could not find the
      // attributeName in entityMetadata, this is an
                // error
                throw new ObjectGridRuntimeException("Invalid attributeName.  Entity: " +
```

```
            ivEntityMetadata.getName());
                }
            }
        }
```

Metodi nome attributo

Il metodo setAttributeName imposta il nome dell'attributo da indicizzare. La classe oggetto in cache deve fornire il metodo get per l'attributo indicizzato. Ad esempio, se l'oggetto ha un attributo employeeName o EmployeeName, l'indice richiama il metodo getEmployeeName sull'oggetto per estrarre il valore dell'attributo. Il nome attributo deve essere uguale al nome contenuto nel metodo get e l'attributo deve implementare l'interfaccia Comparable. Se l'attributo è di tipo booleano, è possibile anche utilizzare il pattern del metodo isAttributeName.

Il metodo getAttributeName restituisce il nome dell'attributo indicizzato.

Metodo getAttribute

Il metodo getAttribute restituisce il valore dell'attributo indicizzato dell'oggetto specificato. Ad esempio, se un oggetto Employee ha un attributo denominato employeeName che è indicizzato, è possibile utilizzare il metodo getAttribute per estrarre il valore dell'attributo employeeName da un oggetto Employee specificato. Questo metodo è richiesto in un ambiente WebSphere eXtreme Scale distribuito.

```
getAttribute(Object value)

// getAttribute method sample code
    public Object getAttribute(Object value) throws ObjectGridRuntimeException {
        if (ivPOJOKeyIndex) {
            // In the POJO key indexing case, no need to get attribute from value object.
            // The key itself is the attribute value used to build the index.
            return null;
        }

        try {
            Object attribute = null;
            if (value != null) {
                // handle Tuple value if ivTupleValueIndex != -1
                if (ivTupleValueIndex == -1) {
                    // regular value
                    if (ivFieldAccessAttribute) {
                        attribute = this.getAttributeField(value).get(value);
                    } else {
                        attribute = getAttributeMethod(value).invoke(value, emptyArray);
                    }
                } else {
                    // Tuple value
                    attribute = extractValueFromTuple(value);
                }
            }
            return attribute;
        } catch (InvocationTargetException e) {
             throw new ObjectGridRuntimeException(
                    "Caught unexpected Throwable during index update processing,
               index name = " + indexName + ": " + t,
                    t);
        } catch (Throwable t) {
             throw new ObjectGridRuntimeException(
                    "Caught unexpected Throwable during index update processing,
              index name = " + indexName + ": " + t,
                    t);
        }
    }
```

HashIndex composto

L'HashIndex composto migliora le prestazioni della query ed evita dispendiose scansioni della mappa. La funzione fornisce inoltre un modo conveniente per l'API HashIndex per trovare oggetti memorizzati nella cache quando il criterio di ricerca coinvolge molti attributi.

Prestazioni migliorate

Un HashIndex composto fornisce un modo rapido e conveniente per cercare oggetti memorizzati nella cache con più attributi nel criterio di ricerca del corrispondente. L'indice composto supporta le ricerche di corrispondenza completa dell'attributo ma non supporta le ricerche di intervallo.

Nota: Gli indici composti non supportano l'operatore BETWEEN nel linguaggio di query ObjectGrid poiché BETWEEN richiede il supporto dell'intervallo. Non funzionano neanche i condizionali maggiore di (>) e minore di (<) in quanto richiedono gli indici di intervallo.

Un indice composto può migliorare le prestazioni delle query se è disponibile l'indice composto appropriato per la condizione WHERE. Ciò vuol dire che l'indice composto ha esattamente gli stessi attributi di quelli nella condizione WHERE con corrispondenza completa degli attributi.

Una query può avere molti attributi coinvolti in una condizione come nell'esempio seguente.

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND
a.zipcode='55901'
```

L'indice composto può migliorare la prestazione della query evitando di passare a scansione la mappa o di unire i risultati multipli dell'indice ad attributo singolo. Nell'esempio, se un indice composto viene definito con attributi (città, stato, CAP), il motore di query può utilizzare l'indice composto per cercare la voce con `city='Rochester'`, `state='MN'` e `zipcode='55901'`. Senza indice composto e indice dell'attributo sugli attributi relativi a città, stato e CAP (city, state, zipcode), il motore di query dovrà sottoporre a scansione la mappa oppure unire le ricerche di più attributi singoli, il che di solito comporta un carico di lavoro impegnativo. Inoltre, l'operazione di query per indice composto supporta solo un pattern a corrispondenza totale.

Configurazione di un indice composto

L'indicizzazione composta può essere configurata in tre modi: utilizzando XML, in modo programmatico e (solo per le mappe di entità) con annotazioni di entità.

Uso di XML

Per poter configurare un indice composto con XML, includere codice come di seguito indicato nell'elemento backingMapPluginCollections del file di configurazione.

```
Indice composto - approccio configurazione XML
<bean id="MapIndexPlugin"  className="com.ibm.websphere.objectgrid.plugins.index.HashIndex">
<property name="Name" type="java.lang.String" value="Address.CityStateZip"/>
<property name="AttributeName" type="java.lang.String" value="city,state,zipcode"/>
</bean>
```

Configurazione programmatica

Il codice dell'esempio programmatico di seguito indicato creerà l'indice composto come l'XML precedente.

```
HashIndex mapIndex = new HashIndex();
 mapIndex.setName("Address.CityStateZip");
 mapIndex.setAttributeName(("city,state,zipcode"));
```

```
    mapIndex.setRangeIndex(true);

    BackingMap bm = objectGrid.defineMap("mymap");
    bm.addMapIndexPlugin(mapIndex);
```

Notare che configurare un indice composto equivale a configurare un indice regolare con XML tranne che per il valore della proprietà attributeName. Nel caso di un indice composto, il valore di attributeName è un elenco di attributi delimitato da virgola. Ad esempio, la classe del valore Address (indirizzo) ha 3 attributi: city, state e zipcode (città, stato e CAP). Un indice composto può essere definito con il valore della proprietà attributeName come `"city,state,zipcode"` indicando che city, state e zipcode sono inclusi nell'indice composto.

Inoltre, notare che gli HashIndexe composti non supportano le ricerche di intervallo e pertanto non possono avere la proprietà RangeIndex impostata su true.

Con annotazioni di entità

Nel caso della mappa di entità, l'approccio dell'annotazione può essere utilizzato per definire un indice composto. È possibile definire un elenco di CompositeIndex all'interno dell'annotazione CompositeIndexes sul livello della classe dell'entità. CompositeIndex ha un nome ed una proprietà attributeNames. Ogni CompositeIndex è associato ad un'istanza HashIndex applicata alla BackingMap associata dell'entità. L'HashIndex è configurato come un indice di non-intervallo.

```
@Entity
@CompositeIndexes({
    @CompositeIndex(name="CityStateZip", attributeNames="city,state,zipcode"),
    @CompositeIndex(name="lastnameBirthday", attributeNames="lastname,birthday")
 })
public class Address {
    @Id int id;
    String street;
    String city;
    String state;
    String zipcode;
    String lastname;
    Date birthday;
}
```

Il nome della proprietà di ciascun indice composto deve essere univoco all'interno dell'entità e di BackingMap. Se il nome non è specificato, verrà utilizzato un nome generato. La proprietà attributeNames viene utilizzata per riempire l'attributeName di HashIndex con l'elenco degli attributi delimitato da virgola. I nomi degli attributi coincidono con i nomi dei campi persistenti quando le entità sono configurate per utilizzare l'accesso di tipo campo oppure con il nome della proprietà come definito per le convenzioni di assegnazione dei nomi di JavaBeans per le entità di accesso di tipo proprietà. Ad esempio: se il nome dell'attributo è "street" ("strada") il metodo per richiamare la proprietà si chiama getStreet.

Esecuzione dei lookup dell'indice composto

Dopo la configurazione di un indice composto, un'applicazione può utilizzare il metodo findAll(Object) dell'interfaccia MapIndex per eseguire ricerche nel seguente modo.

```
Session sess = objectgrid.getSession();
ObjectMap map = sess.getMap("MAP_NAME");
MapIndex codeIndex = (MapIndex) map.getIndex("INDEX_NAME");
Object[] compositeValue = new Object[]{ MapIndex.EMPTY_VALUE,
   "MN", "55901"};
Iterator iter = mapIndex.findAll(compositeValue);
```

MapIndex.EMPTY_VALUE viene assegnato a compositeValue[0] che indica l'attributo di city è escluso dalla valutazione. Nel risultato verranno inclusi solo oggetti con attributo di stato uguale a "MN" con attributo zipcode uguale a "55901".

Le query seguenti traggono benefici dalla precedente configurazione dell'indice composto:

```
SELECT a FROM Address a WHERE a.city='Rochester' AND a.state='MN' AND
a.zipcode='55901'

SELECT a FROM Address a WHERE a.state='MN' AND a.zipcode='55901'
```

Il motore di query troverà l'indice composto appropriato e lo utilizzerà per migliorare le prestazioni di query nei casi di corrispondenza totale dell'attributo.

In alcuni scenari, l'applicazione potrebbe aver bisogno di definire più indici composti con attributi sovrapposti per poter soddisfare tutte le query con corrispondenza totale degli attributi. Uno svantaggio derivante dall'incremento del numero di indici consiste nella possibilità di un rallentamento delle prestazioni nelle operazioni di associazione.

Migrazione e interoperabilità

L'unico vincolo all'utilizzo dell'indice composto è che un'applicazione non può configurarlo in un ambiente distribuito con contenitori eterogenei. I vecchi e i nuovi contenitori non possono essere mischiati poiché i vecchi contenitori non riconoscerebbero una configurazione di indice composto. L'indice composto è proprio come l'indice esistente dell'attributo regolare, tranne per il fatto che il primo consente l'indicizzazione di più attributi. Quando si utilizza solo l'indice dell'attributo regolare, è sempre praticabile l'ambiente a contenitore-misto.

Utilizzo dell'indicizzazione per l'accesso ai dati non chiave

L'utilizzo dell'indicizzazione come alternativa all'accesso chiave per i dati può risultare più efficiente.

Passi necessari

1. Aggiungere plug-in dell'indice statici o dinamici a BackingMap.
2. Acquisire l'oggetto proxy dell'indice, mediante il metodo getIndex di ObjectMap.
3. Eseguire il cast dell'oggetto proxy dell'indice su un'interfaccia dell'indice dell'applicazione appropriata, come MapIndex, MapRangeIndex o un'interfaccia dell'indice personalizzata.
4. Utilizzare i metodi definiti nell'interfaccia dell'indice dell'applicazione per individuare gli oggetti memorizzati nella cache.

La classe HashIndex è l'implementazione del plug-in dell'indice incorporato in grado di supportare entrambe le interfacce dell'indice dell'applicazione incorporate: MapIndex e MapRangeIndex. È anche possibile creare i propri indici. È possibile aggiungere HashIndex sia come indice statico che come indice dinamico nella BackingMap, ottenere l'oggetto proxy dell'indice MapIndex o MapRangeIndex e utilizzare l'oggetto proxy dell'indice per trovare gli oggetti nella cache.

Nota: in un ambiente distribuito, se l'oggetto indice viene ottenuto da un ObjectGrid del client, avrà tipo oggetto oggetto indice del client e tutte le

operazioni di indice verranno eseguite in un ObjectGrid del server remoto. Se la mappa è suddivisa in partizioni, l'operazione di indice verrà eseguita in remoto su ciascuna partizione ed i risultati provenienti da ciascuna partizione verranno uniti prima di essere restituiti all'applicazione. Le prestazioni saranno determinate dal numero di partizioni e dalle dimensioni del risultato restituito da ciascuna partizione. Le prestazioni potrebbero peggiorare se entrambi i fattori sono elevati.

Per informazioni sulla configurazione di HashIndex, consultare Configurazione di HashIndex.

Se si desidera scrivere il proprio plug-in dell'indice, consultare "Plug-in per l'indicizzazione personalizzata di oggetti cache" a pagina 244.

Per informazioni relative all'indicizzazione, consultare Indicizzazione e "HashIndex composto" a pagina 246.

Aggiunta di plug-in dell'indice statici

È possibile utilizzare due approcci per aggiungere i plug-in dell'indice statici alla configurazione BackingMap: configurazione XML e configurazione programmatica. L'esempio riportato di seguito illustra l'approccio mediante la configurazione XML.

Aggiunta di plug-in dell'indice statici: approccio mediante 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>
```

In questo esempio di configurazione XML, la classe HashIndex incorporata viene utilizzata come plug-in dell'indice. La classe HashIndex supporta proprietà che possono essere configurate dagli utenti, come Name, RangeIndex ed AttributeName nell'esempio precedente.

- La proprietà Name è configurata come "CODE", una stringa che identifica questo plug-in dell'indice. Il valore della proprietà Name deve essere univoco nell'ambito di BackingMap e può essere utilizzato per richiamare l'oggetto dell'indice mediante il nome dall'istanza ObjectMap per BackingMap.
- La proprietà RangeIndex è configurata come "true", che indica che l'applicazione può eseguire il cast dell'oggetto dell'indice richiamato sull'interfaccia MapRangeIndex. Se la proprietà RangeIndex è configurata come "false", l'applicazione può eseguire il cast dell'oggetto dell'indice richiamato solo sull'interfaccia MapIndex. MapRangeIndex supporta funzioni per individuare i dati utilizzando funzioni di confronto, come maggiore di, minore di o entrambi, mentre MapIndex supporta solo funzioni di uguaglianza. Se l'indice verrà utilizzato da una query, la proprietà RangeIndex deve essere configurata su "true" su indici ad attributo singolo. Per un indice di relazione ed un indice composto, la proprietà RangeIndex deve essere configurata su "false".
- La proprietà AttributeName è configurata come "employeeCode", che indica che l'attributo employeeCode dell'oggetto memorizzato nella cache viene utilizzato per creare un indice ad attributo singolo. Se l'applicazione deve ricercare oggetti

memorizzati nella cache con più attributi, è possibile impostare la proprietà AttributeName su un elenco di attributi separati da virgole, che costituiscono un indice composito.

Consultare le informazioni relative alla configurazione di HashIndex in *Guida alla gestione* per ulteriori informazioni.

L'interfaccia BackingMap dispone di due metodi che è possibile utilizzare per aggiungere plug-in dell'indice statici: addMapIndexplugin e setMapIndexplugins. Per ulteriori informazioni, consultare la documentazione API.

L'esempio di codice riportato di seguito illustra l'approccio mediante configurazione programmatica:

Aggiunta di plug-in dell'indice statici: approccio mediante 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");

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

Utilizzo degli indici statici

Una volta aggiunto un plug-in dell'indice statico ad una configurazione BackingMap ed inizializzata l'istanza ObjectGrid che lo contiene, le applicazioni possono richiamare l'oggetto dell'indice mediante il nome dall'istanza ObjectMap per BackingMap. Eseguire il cast dell'oggetto dell'indice sull'interfaccia dell'indice dell'applicazione. Ora le operazioni supportate dall'interfaccia dell'indice dell'applicazione possono essere eseguite.

L'esempio di codice riportato di seguito illustra come richiamare ed utilizzare gli indici statici.

Esempio di 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, rimozione ed utilizzo degli indici dinamici

È possibile creare e richiamare gli indici dinamici da un'istanza BackingMap in modo programmatico in qualsiasi momento. Un indice dinamico è diverso da un indice statico in quanto l'indice dinamico 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 e deve essere nello stato di pronto prima di poter essere utilizzata. Questo metodo utilizza

lo stesso approccio per il richiamo e l'utilizzo degli indici dinamici utilizzato per gli indici statici. Se non più necessario, è possibile rimuovere un indice dinamico. L'interfaccia BackingMap dispone di metodi per la creazione e la rimozione degli indici dinamici.

Consultare l'API BackingMap per informazioni relative ai metodi createDynamicIndex e removeDynamicIndex.

Esempio di utilizzo degli indici dinamici

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

        ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
        ObjectGrid og = ogManager.createObjectGrid("grid");
        BackingMap bm = og.getMap("person");
        og.initialize();

        // create index after ObjectGrid initialization without DynamicIndexCallback.
        bm.createDynamicIndex("CODE", true, "employeeCode", null);

        try {
            // If not using DynamicIndexCallback, need to wait for the Index to be ready.
            // The waiting time depends on the current size of the map
            Thread.sleep(3000);
        } catch (Throwable t) {
            // ...
        }

        // When the index is ready, applications can try to get application index
        // interface instance.
        // Applications have to find a way to ensure that the index is ready to use,
        // if not using DynamicIndexCallback interface.
        // The following example demonstrates the way to wait for the index to be ready
        // Consider the size of the map in the total waiting time.

        Session session = og.getSession();
        ObjectMap m = session.getMap("person");
        MapRangeIndex codeIndex = null;

        int counter = 0;
        int maxCounter = 10;
        boolean ready = false;
        while (!ready && counter < maxCounter) {
            try {
                counter++;
                codeIndex = (MapRangeIndex) m.getIndex("CODE");
                ready = true;
            } catch (IndexNotReadyException e) {
                // implies index is not ready, ...
                System.out.println("Index is not ready. continue to wait.");
                try {
                    Thread.sleep(3000);
                } catch (Throwable tt) {
                    // ...
                }
            } catch (Throwable t) {
                // unexpected exception
                t.printStackTrace();
            }
        }

        if (!ready) {
            System.out.println("Index is not ready.  Need to handle this situation.");
        }

        // Use the index to peform queries
        // Refer to the MapIndex or MapRangeIndex interface for supported operations.
        // The object attribute on which the index is created is the EmployeeCode.
        // Assume that the EmployeeCode attribute is Integer type: the
        // parameter that is passed into index operations has this data type.

        Iterator iter = codeIndex.findLessEqual(new Integer(15));

        // remove the dynamic index when no longer needed

        bm.removeDynamicIndex("CODE");
```

Interfaccia DynamicIndexCallback

L'interfaccia DynamicIndexCallback è progettata per le applicazioni che devono ricevere notifiche in caso degli eventi di indicizzazione di ready, error o destroy. DynamicIndexCallback è un parametro facoltativo per il metodo createDynamicIndex di BackingMap. Con un'istanza DynamicIndexCallback registrata, le applicazioni possono eseguire la logica di business quando viene ricevuta la notifica di un evento di indicizzazione. Ad esempio, l'evento ready indica che l'indice è pronto per l'utilizzo. Quando viene ricevuta una notifica per

questo evento, un'applicazione può richiamare ed utilizzare l'istanza dell'interfaccia dell'indice dell'applicazione. Consultare API DynamicIndexCallback API nella documentazione API per ulteriori informazioni.

L'esempio di codice riportato di seguito illustra l'utilizzo dell'interfaccia DynamicIndexCallback:

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);

            // Simulate what an application would do when notified that the index is ready.
            // Normally, the application would wait until the ready state is reached and then proceed
            // with any index usage logic.
            if("CODE".equals(indexName)) {
                ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
                ObjectGrid og = ogManager.createObjectGrid( "grid" );
                Session session = og.getSession();
                ObjectMap map = session.getMap("person");
                MapIndex codeIndex = (MapIndex) map.getIndex("CODE");
                Iterator iter = codeIndex.findAll(codeValue);
            }
        }

        public void error(String indexName, Throwable t) {
            System.out.println("DynamicIndexCallbackImpl.error() -> indexName = " + indexName);
            t.printStackTrace();
        }

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

Plug-in per la comunicazione con archivi persistenti

Con un plug-in programma di caricamento eXtreme Scale, una mappa ObjectGrid può agire da cache di memoria per i dati che in genere vengono conservati in un archivio persistente sullo stesso sistema o su un altro sistema. In genere viene utilizzato un database o un file system come archivio persistente. È possibile utilizzare anche una JVM (Java Virtual Machine) remota come origine dei dati, consentendo la creazione di cache basate su hub utilizzando ObjectGrid. Un programma di caricamento dispone della logica per la lettura e la scrittura di dati da e verso un archivio persistente.

I programmi di caricamento sono plug-in mappa di backup che vengono richiamati quando vengono apportate modifiche alla mappa di backup o quando la mappa di backup non è in grado di soddisfare una richiesta di dati (una mancata corrispondenza nella cache).

Per ulteriori informazioni, consultare le informazioni relative ai concetti sulla memorizzazione nella cache in *Panoramica sul prodotto*.

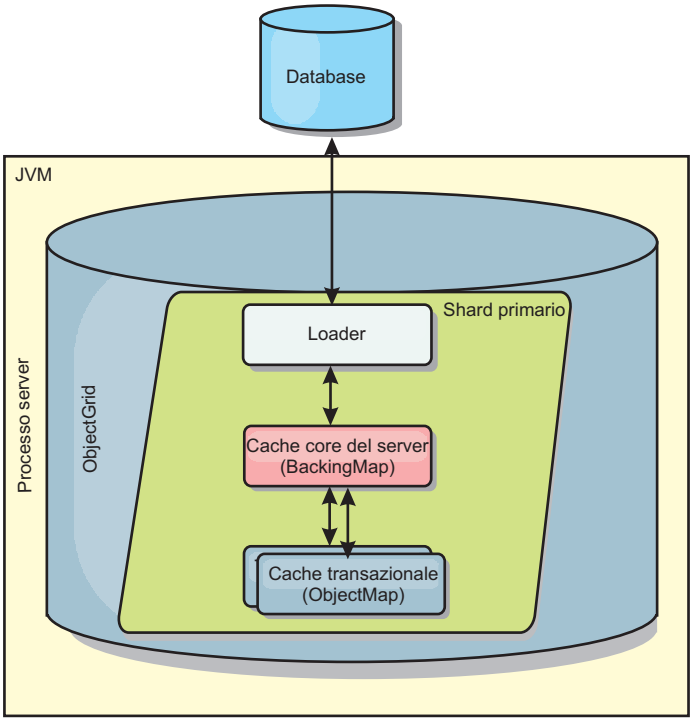


Figura 3. Programma di caricamento

WebSphere eXtreme Scale include due programmi di caricamento integrati per l'integrazione con i back-end dei database relazionali. I programmi di caricamento di JPA (Java Persistence API) utilizzano le capability ORM (Object-Relational Mapping) di entrambe le implementazioni OpenJPA e Hibernate della specifica JPA.

Utilizzo di un programma di caricamento

Per aggiungere un programma di caricamento alla configurazione di BackingMap, è possibile utilizzare la configurazione tramite un programma o la configurazione XML. Un programma di caricamento ha la seguente relazione con una mappa di backup:

- Una mappa di backup può avere un unico programma di caricamento.
- Una mappa di backup client (cache locale) non può avere un programma di caricamento.
- Una definizione di programma di caricamento può essere applicata a più mappe di backup, ma ognuna ha una propria istanza di programma di caricamento.

Collegamento di un programma di caricamento tramite un programma

Il seguente frammento di codice mostra come collegare un programma di caricamento fornito da un'applicazione alla mappa di backup per 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 );
```

Questo frammento presuppone che la classe 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 con una mappa di backup non può essere modificata dopo l'inizializzazione di ObjectGrid, il codice deve essere eseguito prima di richiamare il metodo initialize dell'interfaccia ObjectGrid richiamata. Si verifica un'eccezione IllegalStateException sul metodo setLoader se questo viene richiamato dopo l'inizializzazione.

Il programma di caricamento fornito dall'applicazione può avere proprietà impostate. Nell'esempio viene utilizzato il programma di caricamento MyLoader per leggere e scrivere i dati da una tabella in un database relazionale. Il programma di caricamento deve specificare il nome del database e il livello di isolamento SQL. Il programma di caricamento MyLoader dispone dei metodi setDataBaseName e setIsolationLevel che consentono all'applicazione di impostare queste due proprietà del programma di caricamento.

Approccio configurazione XML per il collegamento di un programma di caricamento

Un programma di caricamento fornito da un'applicazione può essere collegato anche tramite un file XML. L'esempio riportato di seguito mostra come collegare il programma di caricamento MyLoader alla mappa di backup map1 con le stesse proprietà nome database e livello di isolamento del programma di caricamento:

```
<?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>
```

Scrittura di un programma di caricamento

È possibile scrivere una propria implementazione del plug-in Loader nella propria applicazione il che deve avvenire seguendo le comuni convenzioni del plug-in WebSphere eXtreme Scale.

Inclusione di plug-in Loader

L'interfaccia del programma di caricamento 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;
}
```

Per ulteriori informazioni, consultare Per ulteriori informazioni relative ai programmi di caricamento nella sezione *Panoramica sul prodotto*

Metodo Get

La mappa di backup richiama il metodo Get del programma di caricamento per ottenere i valori che sono associati ad un elenco di chiavi che viene passato come argomento keyList. Il metodo Get è necessario per restituire un elenco java.lang.util.List di valori, un valore per ciascuna chiave che si trova nell'elenco delle chiavi. Il primo valore che viene restituito nell'elenco dei valori corrisponde alla prima chiave nell'elenco delle chiavi, il secondo valore restituito nell'elenco dei valori corrisponde alla seconda chiave nell'elenco delle chiavi e così via. Se il programma di caricamento non trova il valore per una chiave nell'elenco delle chiavi, al programma di caricamento viene richiesto di restituire il valore oggetto speciale KEY_NOT_FOUND che viene definito nell'interfaccia del programma di caricamento. Poichè una mappa di backup può essere configurata per consentire un valore `null` come valore valido, è molto importante che il programma di caricamento restituisca l'oggetto speciale KEY_NOT_FOUND quando non è in grado di trovare la chiave. Questo valore speciale consente alla mappa di backup di distinguere tra un valore null e un valore che non esiste perché non è stata trovata la chiave. Se la mappa di backup non supporta valori null, si verificherà un'eccezione nel programma di caricamento che restituisce un valore null invece dell'oggetto KEY_NOT_FOUND per una chiave che non esiste.

L'argomento forUpdate informa il programma di caricamento se l'applicazione ha richiamato nella mappa un metodo Get o un metodo getForUpdate. Per ulteriori informazioni, consultare Interfaccia ObjectMap nella documentazione relativa alle API for Il programma di caricamento è responsabile di implementare una politica di controllo della concorrenza che controlli l'accesso concorrente all'archivio persistente. Ad esempio, molti sistemi di gestione di database relazionali supportano la sintassi for update nell'istruzione SQL 'select' che viene utilizzata per leggere i dati da una tabella relazionale. Il programma di caricamento può scegliere di utilizzare la sintassi for update nell'istruzione SQL 'select' in base al fatto che `boolean true` venga passato come il valore dell'argomento per il parametro forUpdate di questo metodo. Generalmente, il programma di caricamento utilizza la sintassi for update solo quando viene utilizzata la politica di controllo della concorrenza pessimistica. Per un controllo della concorrenza ottimistico, il programma di caricamento non utilizzerà mai la sintassi `for update` nell'istruzione SQL 'select'. Il programma di caricamento è responsabile di decidere di utilizzare l'argomento forUpdate in base alla politica di controllo della concorrenza che sta utilizzando il programma di caricamento.

Per una spiegazione del parametro txid, consultare “Plug-in per la gestione degli eventi del ciclo di vita della transazione” a pagina 274.

metodo batchUpdate

Il metodo batchUpdate è importante nell'interfaccia del programma di caricamento. Questo metodo viene richiamato se eXtreme Scale ha necessità di applicare tutte le modifiche correnti al programma di caricamento. Il programma di caricamento ha

fornito un elenco delle modifiche per la mappa selezionata. Le modifiche sono reiterate e applicate al backend. Il metodo riceve il valore TxID corrente e le modifiche da applicare. L'esempio di seguito riportato itera la serie di modifiche ed esegue in batch tre istruzioni JDBC (Java database connectivity) una con 'insert, un'altra con 'update' e una con 'delete'.

```
import java.util.Collection;
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;

    public void batchUpdate(TxID tx, LogSequence sequence) throws LoaderException {
        // Get a SQL connection to use.
        Connection conn = getConnection(tx);
        try {
            // Process the list of changes and build a set of prepared
            // statements for executing a batch update, insert, or delete
            // SQL operation.
            Iterator iter = sequence.getPendingChanges();
            while (iter.hasNext()) {
                LogElement logElement = (LogElement) iter.next();
                Object key = logElement.getKey();
                Object value = logElement.getCurrentValue();
                switch (logElement.getType().getCode()) {
                case LogElement.CODE_INSERT:
                    buildBatchSQLInsert(tx, key, value, conn);
                    break;
                case LogElement.CODE_UPDATE:
                    buildBatchSQLUpdate(tx, key, value, conn);
                    break;
                case LogElement.CODE_DELETE:
                    buildBatchSQLDelete(tx, key, conn);
                    break;
                }
            }
            // Execute the batch statements that were built by above loop.
            Collection statements = getPreparedStatementCollection(tx, conn);
            iter = statements.iterator();
            while (iter.hasNext()) {
                PreparedStatement pstmt = (PreparedStatement) iter.next();
                pstmt.executeBatch();
            }
        } catch (SQLException e) {
            LoaderException ex = new LoaderException(e);
            throw ex;
        }
    }
```

L'esempio precedente illustra la logica di alto livello dell'elaborazione dell'argomento LogSequence, ma i dettagli di come viene costruita un'istruzione SQL insert, update o delete non vengono illustrati. Alcuni punti chiave illustrati comprendono:

- il metodo getPendingChanges è richiamato nell'argomento LogSequence per ottenere un iteratore sull'elenco di LogElements che il programma di caricamento ha necessità di elaborare.
- Il metodo LogElement.getType().getCode() viene utilizzato per determinare se LogElement è per un'operazione SQL insert, update, o delete.
- Un'eccezione SQLException viene rilevata ed è concatenata ad un'eccezione LoaderException che stampa nel report che si è verificata un'eccezione durante l'aggiornamento in batch.
- Il supporto di aggiornamento batch JDBC viene utilizzato per ridurre al minimo il numero di query che devono essere eseguite nel backend.

metodo preloadMap

Durante l'inizializzazione eXtreme Scale, ciascuna mappa di backup che è definita, viene inizializzata. Se il programma di caricamento è collegato ad una mappa di backup, la mappa di backup richiama il metodo preloadMap nell'interfaccia Loader

per consentire al programma di caricamento di eseguire il fetch preventivo dei dati dal suo backend e di caricare i dati nella mappa. L'esempio di seguito riportato assume che le prime 100 righe di una tabella Employee sono lette dal database e caricate nella mappa. La classe EmployeeRecord è una classe fornita dall'applicazione che conserva i dati relativi agli impiegati che sono letti dalla tabella Employee.

Nota: Questo esempio esegue il fetch di tutti i dati dal database e successivamente li inserisce nella mappa di base di una partizione. In uno scenario di distribuzione eXtreme Scale realmente distribuito, i dati dovrebbero essere distribuiti in tutte le partizioni. Fare riferimento a "Programmazione utilità di precaricamento JPA basata su client" a pagina 309 per ulteriori informazioni.

```
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException

    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException {
        boolean tranActive = false;
        ResultSet results = null;
        Statement stmt = null;
        Connection conn = null;
        try {
            session.beginNoWriteThrough();
            tranActive = true;
            ObjectMap map = session.getMap(backingMap.getName());
            TxID tx = session.getTxID();
            // Get a auto-commit connection to use that is set to
            // a read committed isolation level.
            conn = getAutoCommitConnection(tx);
            // Preload the Employee Map with EmployeeRecord
            // objects. Read all Employees from table, but
            // limit preload to first 100 rows.
            stmt = conn.createStatement();
            results = stmt.executeQuery(SELECT_ALL);
            int rows = 0;
            while (results.next() && rows < 100) {
                int key = results.getInt(EMPNO_INDEX);
                EmployeeRecord emp = new EmployeeRecord(key);
                emp.setLastName(results.getString(LASTNAME_INDEX));
                emp.setFirstName(results.getString(FIRSTNAME_INDEX));
                emp.setDepartmentName(results.getString(DEPTNAME_INDEX));
                emp.updateSequenceNumber(results.getLong(SEQNO_INDEX));
                emp.setManagerNumber(results.getInt(MGRNO_INDEX));
                map.put(new Integer(key), emp);
                ++rows;
            }
            // Commit the transaction.
            session.commit();
            tranActive = false;
        } catch (Throwable t) {
            throw new LoaderException("preload failure: " + t, t);
        } finally {
            if (tranActive) {
                try {
                    session.rollback();
                } catch (Throwable t2) {
                    // Tolerate any rollback failures and
                    // allow original Throwable to be thrown.
                }
            }
            // Be sure to clean up other databases resources here
            // as well such a closing statements, result sets, etc.
        }
    }
```

Questo esempio illustra i seguenti punti chiave:

- La mappa di backup utilizza un oggetto Session che le viene passato come un argomento session.
- Il metodo Session.beginNoWriteThrough viene utilizzato per iniziare la transazione invece del metodo begin.
- Il programma di caricamento non può essere richiamato per ciascuna operazione di tipo put che avviene con questo metodo per caricare la mappa.

- Il programma di caricamento può associare colonne della tabella Employee ad un campo nell'oggetto Java EmployeeRecord. Il programma di caricamento rileva tutte le eccezioni Throwable che si verificano e genera un'eccezione LoaderException con l'eccezione Throwable rilevata e ad essa collegata. to it.
- Infine il blocco garantisce che qualsiasi eccezione Throwable che si verifichi tra il momento in cui viene richiamato il metodo beginNoWriteThrough e il momento in cui viene richiamato il metodo Commit causi infine il blocco per eseguire il roll back delle transazioni attive. Questa operazione è delicata per garantire che qualsiasi transazione che è stata avviata dal metodo preloadMap sia completata prima di essere restituita al chiamante. Infine il blocco è idoneo ad eseguire altre azioni di ripulitura che potrebbero essere necessarie come la chiusura della connessione JDBC (Java Database Connectivity) e degli altri oggetti JDBC.

L'esempio preloadMap utilizza un'istruzione SQL select che seleziona tutte le righe della tabella. Nel proprio programma di caricamento fornito dall'applicazione potrebbe essere necessario impostare una o più proprietà del programma di caricamento per controllare quanto della tabella necessita di essere precaricato nella mappa.

Poiché il metodo preloadMap viene richiamato solo una volta durante l'inizializzazione di BackingMap, esso è idoneo anche ad eseguire un'unica volta il codice di inizializzazione del programma di caricamento. Anche se il programma di caricamento sceglie di non eseguire preventivamente il fetch dei dati dal backend e di caricare i dati nella mappa, esso probabilmente necessita di eseguire qualche altra volta l'inizializzazione per rendere gli altri metodi del programma di caricamento più efficienti. L'esempio che segue, illustra la memorizzazione nella cache dell'oggetto TransactionCallback e dell'oggetto OptimisticCallback come variabili di istanza del programma di caricamento in modo che gli altri metodi del programma di caricamento non hanno necessità di richiamare altri metodi per accedere a questi oggetti. Questa memorizzazione nella cache dei valori di plug-in ObjectGrid può essere eseguita perché dopo che BackingMap è stata inizializzata, gli oggetti TransactionCallback e OptimisticCallback non possono essere modificati o sostituiti. È accettabile memorizzare in cache questi oggetti di riferimento come variabili di istanza 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;

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

    // extends TransactionCallback
    MyOptimisticCallback ivOcb; // MyOptimisticCallback

    // implements OptimisticCallback
    // ...
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
   [Replication programming]
        // Cache TransactionCallback and OptimisticCallback objects
        // in instance variables of this Loader.
        ivTcb = (MyTransactionCallback) session.getObjectGrid().getTransactionCallback();
        ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
        // The remainder of preloadMap code (such as shown in prior example).
    }
```

Per ulteriori informazioni relative al precaricamento e al recupero del precaricamento quando esso riguarda un errore di replica, consultare le informazioni relative alla nella sezione *Panoramica sul prodotto*.

Programmi di caricamento con le mappe entità

Se il programma di caricamento è collegato ad una mappa entità, il programma di caricamento deve gestire oggetti tupla. Gli oggetti tupla sono un formato dati speciale dell'entità. Il programma di caricamento deve convertire i dati in tupla e in altri formati. Ad esempio, il metodo Get restituisce un elenco di valori che corrispondono alla serie di chiavi che vengono passate al metodo. Le chiavi passate sono nel tipo di tupla dette tuple chiavi. Presupponendo che il programma di caricamento persiste la mappa con un database utilizzando JDBC, il metodo Get deve convertire ciascuna tupla chiave in un elenco di valori di attributi che corrispondano alle colonne della chiave dell'elemento primario della tabella che è associata alla mappa entità, eseguire l'istruzione SELECT con la clausola WHERE che utilizza i valori attributi convertiti come criterio per eseguire il fetch dei dati dal database e successivamente convertire i dati restituiti nelle tuple del valore. Il metodo Get prende i dati dal database e li converte in tuple di valore per le tuple di chiavi passate e successivamente restituisce un elenco di tuple di valore corrispondenti alla serie di chiavi della tupla che sono state passate al chiamante. Il metodo Get può eseguire un'istruzione SELECT per eseguire il fetch di tutti i dati ad un determinato momento o eseguire l'istruzione SELECT per ciascuna tupla di chiave. Per i dettagli sulla programmazione che mostra in che modo utilizzare il programma di caricamento quando i dati sono memorizzati utilizzando il gestore di entità, vedere "Utilizzo del programma di caricamento con mappe di entità e tuple" a pagina 264.

Considerazioni sulla programmazione del programma di caricamento JPA

Un Loader JPA (Java Persistence API) è un'implementazione plug-in del programma di caricamento che utilizza JPA per interagire con il database. Utilizzare le seguenti considerazioni quando si sviluppa un'applicazione che utilizza un programma di caricamento JPA.

Entità eXtreme Scale e entità JPA

È possibile progettare una qualsiasi classe POJO come entitàeXtreme Scale utilizzando le annotazioni entità eXtreme Scale, le configurazioni XML o entrambe. È possibile inoltre progettare la stessa classe POJO come entità JPA utilizzando le annotazioni entità JPA, la configurazione XML o entrambe.

Entità eXtreme Scale: un'entità eXtreme Scale rappresenta dati persistenti memorizzati nelle mappe ObjectGrid. Un oggetto entità viene trasformato in una tupla di chiavi e una tupla di valori, che vengono poi memorizzate come coppie chiave-valore nelle mappe. Una tupla è un array di attributi primitivi.

Entità JPA: un'entità JPA rappresenta dati persistenti memorizzati automaticamente in un database relazionale utilizzando la persistenza gestita da contenitore. I dati persistono in un tipo di sistema di archiviazione dati nel formato appropriato, come ad esempio, tuple di database in un database.

Quando persiste un'entità eXtreme Scale, le relative relazioni vengono memorizzate in altre mappe di entità. Ad esempio, quando si persiste un'entità Consumer con una relazione una-a-molti in un'entità ShippingAddress, se è abilitata la funzione cascade-persist, l'entità ShippingAddress viene memorizzata nella mappa shippingAddress in formato tupla. Se si persiste un'entità JPA, anche le entità JPA associate persistono nelle tabelle di database se la funzione cascade-persist è abilitata. Quando una classe POJO è progettata sia come entità eXtreme Scale che

come entità JPA, i dati possono persistere sia nei database che nelle mappe entità ObjectGrid. Di seguito vengono riportati gli utilizzi comuni:

- **Scenario di precaricamento**: un'entità viene caricata da un database utilizzando un provider JPA e persiste nelle mappe entità ObjectGrid.
- **Scenario di caricamento**: viene collegata un'implementazione Loader per mappe di entità ObjectGrid in modo che un'entità memorizzata nelle mappe di entità ObjectGrid possa persistere in un database o essere caricata da un database utilizzando i provider JPA.

È inoltre frequente che una classe POJO sia progettata solo come entità JPA. In quel caso, nelle mappe ObjectGrid vengono memorizzate le istanze POJO rispetto alle tuple di entità nel caso dell'entità ObjectGrid.

Considerazioni di progettazione dell'applicazione per mappe di entità

Quando si collega un'interfaccia JPALoader, le istanze dell'oggetto vengono memorizzate direttamente nelle mappe ObjectGrid.

Tuttavia, quando si collega un JPAEntityLoader, la classe entità viene progettata sia come entità eXtreme Scale che come entità JPA. In quel caso, gestire questa entità come se avesse due archivi persistenti: le mappe di entità ObjectGrid e l'archivio di persistenza JPA. L'architettura è più complessa rispetto al caso JPALoader.

Per ulteriori informazioni sul plug-in JPAEntityLoader e considerazioni sulla progettazione dell'applicazione, consultare le informazioni sul plug-in JPAEntityLoader in *Guida alla gestione*. Queste informazioni consentono inoltre di pianificare l'implementazione del proprio programma di caricamento per le mappe di entità.

Considerazioni sulle prestazioni

Assicurarsi che per le relazioni venga impostato il tipo fetch EAGER o LAZY appropriato. Ad esempio, una relazione uno-a-molti bidirezionale Consumer con ShippingAddress, con OpenJPA per descrivere le differenze di prestazioni. In questo esempio, una query JPA esegue `select o from Consumer o where . . .` per effettuare un caricamento di massa e per caricare anche tutti gli oggetti ShippingAddress associati. Di seguito è riportata una relazione uno-a-molti definita nella classe Consumer:

```
@Entity
public class Consumer implements Serializable {

    @OneToMany(mappedBy="consumer",cascade=CascadeType.ALL, fetch =FetchType.EAGER)
    ArrayList <ShippingAddress> addresses;
```

Di seguito è riportata la relazione molti-a-uno consumer definita nella classe ShippingAddress:

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.EAGER)
    Consumer consumer;
}
```

Se i tipi fetch di entrambe le relazioni vengono configurati come `eager`, OpenJPA utilizza le query N+1+1 per ottenere tutti gli oggetti Consumer e ShippingAddress, dove N è il numero di oggetti ShippingAddress. Tuttavia, se l'oggetto

ShippingAddress viene modificato per utilizzare il tipo fetch LAZY come indicato di seguito, per ottenere tutti i dati sono richieste solo due query.

```
@Entity
public class ShippingAddress implements Serializable{

    @ManyToOne(fetch=FetchType.LAZY)
    Consumer consumer;
}
```

Sebbene la query restituisca gli stessi risultati, l'utilizzo di un minor numero di query riduce significativamente l'interazione con il database, aumentando le prestazioni dell'applicazione.

Plug-in JPAEntityLoader

Il plug-in JPAEntityLoader è una implementazione Loader incorporata che utilizza JPA (Java Persistence API) per comunicare con il database quando si sta utilizzando l'API EntityManager. Quando si sta utilizzando l'API ObjectMap utilizzare il programma di caricamento JPALoader.

Dettagli del programma di caricamento

Utilizzare il plug-in JPALoader quando si stanno memorizzando i dati utilizzando l'API ObjectMap. Utilizzare il plug-in JPAEntityLoader quando si stanno memorizzando i dati mediante l'API EntityManager.

I programmi di caricamento forniscono due funzioni principali:

1. **get**: nel metodo get il plug-in JPAEntityLoader prima richiama il metodo javax.persistence.EntityManager.find(Class entityClass, Object key) per trovare l'entità JPA. Quindi il plug-in proietta questa entità JPA in tuple di entità. Durante la proiezione, gli attributi delle tuple e le chiavi di associazione vengono memorizzati nella tupla del valore. Dopo l'elaborazione di ogni chiave, il metodo get restituisce un elenco di tuple di valori entità.
2. **batchUpdate**: il metodo batchUpdate prende un oggetto LogSequence che contiene un elenco di oggetti LogElement. Ogni oggetto LogElement contiene una tupla di chiavi e una tupla di valori. Per interagire con il provider JPA, si deve prima trovare l'entità eXtreme Scale basata sulla tupla delle chiavi. In base al tipo LogElement eseguire le seguenti chiamate JPA:
   - **insert**: javax.persistence.EntityManager.persist(Object o)
   - **update**: javax.persistence.EntityManager.merge(Object o)
   - **remove**: javax.persistence.EntityManager.remove(Object o)

Un LogElement con il tipo **update** fa in modo che JPAEntityLoader richiami il metodo javax.persistence.EntityManager.merge(Object o) per unificare l'entità. Tuttavia, un LogElement di tipo **update** potrebbe essere il risultato di una chiamata com.ibm.websphere.objectgrid.em.EntityManager.merge(object o) o di una modifica attributo dell'istanza gestita dall'EntityManager eXtreme Scale. Consultare il seguente esempio:

```
com.ibm.websphere.objectgrid.em.EntityManager em = og.getSession().getEntityManager();
em.getTransaction().begin();
Consumer c1 = (Consumer) em.find(Consumer.class, c.getConsumerId());
c1.setName("New Name");
em.getTransaction().commit();
```

In questo esempio, un LogElement di tipo update viene inviato al JPAEntityLoader del consumer della mappa. Il metodo javax.persistence.EntityManager.merge(Object o) viene richiamato nel gestore entità JPA invece di un aggiornamento degli

attributi nell'entità gestita da JPA. A causa di questo comportamento modificato, vi sono alcune limitazioni riguardo all'utilizzo di questo modello di programmazione.

Regole di progettazione delle applicazioni

Le entità hanno relazioni con altre entità. Riguardo alla progettazione di un'applicazione dove sono coinvolte relazioni e con il plug-in JPAEntityLoader sono necessarie ulteriori considerazioni. L'applicazione deve attenersi alle seguenti quattro regole, descritte nelle seguenti sezioni.

Supporto in profondità di relazioni limitate

JPAEntityLoader viene supportato solo quando si stanno utilizzando entità senza alcuna relazione o entità con relazioni con un singolo livello. Le relazioni con più livelli, come `Company > Department > Employee` non sono supportate.

Un programma di caricamento per mappa

Utilizzando le relazioni di entità Consumer-ShippingAddress come un esempio, quando si carica un Consumer con un fetch di tipo eager abilitato, è possibile caricare tutti gli oggetti ShippingAddress correlati. Quando si esegue la persistenza di un oggetto Consumer o lo si unifica, è possibile eseguire la persistenza degli oggetti ShippingAddress o unificare quelli correlati se la funzione cascade-persist o cascade-merge è abilitata.

Non è possibile eseguire il plug-in in un programma di caricamento per la mappa di entità root che memorizza le tuple dell'entità Consumer. È necessario configurare un programma di caricamento per ciascuna mappa di entità.

Tipo di cascata uguale per JPA e eXtreme Scale

Ripensare allo scenario in cui l'entità Consumer ha una relazione uno-a-molti con ShippingAddress. È possibile esaminare lo scenario in cui la funzione cascade-persist è abilitata per la relazione. Quando si esegue la persistenza di un oggetto Consumer in eXtreme Scale, viene anche eseguita la persistenza del numero N associato di oggetti in eXtreme Scale.

Una chiamata di persistenza dell'oggetto Consumer con una relazione cascade-persist a ShippingAddress si traduce in una chiamata di metodo javax.persistence.EntityManager.persist(consumer) e *N* chiamate di metodo javax.persistence.EntityManager.persist(shippingAddress) da parte del livello JPAEntityLoader. Tuttavia, queste N extra chiamate di persistenza agli oggetti ShippingAddress non sono necessarie a seguito dell'impostazione di cascade-persist dal punto di vista del provider JPA. Per risolvere questo problema, eXtreme Scale fornisce un nuovo metodo isCascaded nell'interfaccia LogElement. Il metodo isCascaded indica se LogElement è il risultato di un'operazione in cascata di EntityManager eXtreme Scale. In questo esempio, JPAEntityLoader della mappa ShippingAddress riceve *N* oggetti LogElement a seguito delle chiamate di persistenza a cascata. JPAEntityLoader rileva che il metodo isCascaded restituisce true e quindi li ignora senza effettuare alcuna chiamata JPA. Pertanto, da un punto di vista di JPA, viene ricevuta solo una chiamata di metodo javax.persistence.EntityManager.persist(consumer).

Si manifesta lo stesso comportamento se si unifica un'entità o la si rimuove con la cascata abilitata. Le operazioni in cascata vengono ignorate dal plug-in JPAEntityLoader.

La progettazione del supporto in cascata è di ripetere le operazioni EntityManager eXtreme Scale nei provider JPA. Queste operazioni includono operazioni di persistenza, unione e rimozione. Per abilitare il supporto in cascata, verificare che l'impostazione della cascata per JPA e quella EntityManager eXtreme Scale sono uguali.

Utilizzo accorto dell'aggiornamento di entità

Come descritto in precedenza, la progettazione del supporto in cascata è ripetere le operazioni EntityManager eXtreme Scale nei provider JPA. Se l'applicazione richiama il metodo ogEM.persist(consumer) sull'EntityManager eXtreme Scale, viene eseguita la persistenza anche degli oggetti ShippingAddress associati a causa dell'impostazione cascade-persist e JPAEntityLoader richiama solo il metodo jpAEM.persist(consumer) sui provider JPA.

Tuttavia, se l'applicazione aggiorna un'entità gestita, questo aggiornamento si traduce in una chiamata di unione JPA da parte del plug-in JPAEntityLoader. In questo scenario, non è garantito il supporto per più livelli di relazioni e di associazioni chiave. In questo caso, è consigliabile utilizzare il metodo javax.persistence.EntityManager.merge(o) invece di aggiornare un'entità gestita.

Utilizzo del programma di caricamento con mappe di entità e tuple

Il gestore entità converte tutti gli oggetti entità in oggetti tuple prima che essi siano memorizzati nella mappa WebSphere eXtreme Scale. Ogni entità ha una tupla chiave e una tupla valore. Questa coppia chiave-valore viene memorizzata nella mappa associata eXtreme Scale all'entità. Quando si sta utilizzando una mappa eXtreme Scale con un programma di caricamento il programma di caricamento deve interagire con gli oggetti tupla.

Panoramica

eXtreme Scale comprende i plug-in Loader che semplificano l'integrazione con i database relazionali. I programmi di caricamento JPA (Java Persistence API) utilizzano una Java Persistence API per interagire con il database e creare gli oggetti entità. I programmi di caricamento JPA sono compatibili con le entità eXtreme Scale.

Tuple

Una tupla contiene le informazioni relative agli attributi e alle associazioni di un'entità. I valori primitivi sono memorizzati utilizzando i loro wrapper primitivi. Gli altri tipi di oggetti supportati sono memorizzati nel loro formato nativo. Le associazioni alle altre entità sono memorizzate come una raccolta di oggetti di tupla chiave che rappresentano le chiavi delle entità di destinazione.

Ciascun attributo o associazione è memorizzato utilizzando un indice su base zero. È possibile recuperare l'indice di ciascun attributo utilizzando i metodi getAttributePosition o getAssociationPosition. Dopo che è stata recuperata la posizione, questa resta immutata per la durata del ciclo di vita eXtreme Scale. La posizione può essere modificata quando viene riavviato eXtreme Scale. I metodi setAttribute, setAssociation e setAssociations sono utilizzati per aggiornare gli elementi nella tupla.

Attenzione: Quando si sta creando o aggiornando oggetti tupla, aggiornare ciascun campo primitivo con un valore non null. I valori primitivi come `int` non dovrebbero essere null. Se non si modifica il valore con un valore predefinito, possono verificarsi problemi di basse prestazioni che influenzano anche i campi indicati con l'annotazione @Version o l'attributo di versione nel file XML del descrittore dell'entità.

Il seguente esempio illustra ulteriormente come elaborare le tuple Per ulteriori informazioni relative alla definizione di entità per questo esempio, consultare le informazioni relative allo schema di entità Order, che si trova nel supporto didattico del gestore entità in *Panoramica sul prodotto*.WebSphere eXtreme Scale è configurato per l'utilizzo dei programmi di caricamento con ciascuna delle entità. Inoltre, solo l'entità Order è considerata e questa specifica entità è in relazione di tipo molti-a-uno con l'entità Customer. Il nome dell'attributo è `customer`, ed è in relazione di tipo uno-a-molti con l'entità OrderLine.

Utilizzare il Projector per costruire oggetti tupla automaticamente dalle entità. L'utilizzo di Projector può semplificare i programmi di caricamento quando si sta utilizzando un programma di utilità ORM (object-relational mapping) come Hibernate o JPA.

order.java

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

customer.java

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

orderLine.java

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

Una classe OrderLoader che implementa l'interfaccia del programma di caricamento è mostrata nel seguente codice. L'esempio di seguito riportato assume che sia definito il plug-in associato a TransactionCallback.

orderLoader.java

```
public class OrderLoader implements com.ibm.websphere.objectgrid.plugins.Loader {

 private EntityMetadata entityMetaData;
 public void batchUpdate(TxID txid, LogSequence sequence)
           throws LoaderException, OptimisticCollisionException {
       ...
```

```
        }
        public List get(TxID txid, List keyList, boolean forUpdate)
            throws LoaderException {
        ...
        }
        public void preloadMap(Session session, BackingMap backingMap)
            throws LoaderException {
  this.entityMetaData=backingMap.getEntityMetadata();
 }

}
```

La variabile di istanza entityMetaData viene inizializzata durante la chiamata al metodo preLoadMap da eXtreme Scale. La variabile *entityMetaData* non ha valore null se la mappa è configurata per utilizzare le entità. Altrimenti, il valore è null.

Il metodo batchUpdate

Il metodo batchUpdate fornisce la possibilità di conoscere quale azione l'applicazione intendeva eseguire. Una connessione, basata su un'operazione di inserimento, aggiornamento o cancellazione, può essere stabilita con il database ed eseguire il lavoro. Poiché la chiave e i valori sono di tipo tupla, essi devono essere trasformati in modo che i valori siano compatibili con l'istruzione SQL.

La tabella ORDER è stata creata con la seguente definizione DDL (Data Definition Language) come è mostrato nel codice di seguito riportato:

```
CREATE TABLE ORDER (ORDERNUMBER VARCHAR(250) NOT NULL, DATE TIMESTAMP, CUSTOMER_ID VARCHAR(250))
ALTER TABLE ORDER ADD CONSTRAINT PK_ORDER PRIMARY KEY (ORDERNUMBER)
```

Il codice di seguito riportato descrive come convertire una tupla in un oggetto:

```
public void batchUpdate(TxID txid, LogSequence sequence)
            throws LoaderException, OptimisticCollisionException {
        Iterator iter = sequence.getPendingChanges();
        while (iter.hasNext()) {
            LogElement logElement = (LogElement) iter.next();
            Object key = logElement.getKey();
            Object value = logElement.getCurrentValue();

            switch (logElement.getType().getCode()) {
            case LogElement.CODE_INSERT:

1)              if (entityMetaData!=null) {

// The order has just one key orderNumber
2)                  String ORDERNUMBER=(String) getKeyAttribute("orderNumber", (Tuple) key);
// Get the value of date
3)                  java.util.Date unFormattedDate = (java.util.Date) getValueAttribute("date",(Tuple)value);
// The values are 2 associations. Lets process customer because
// the our table contains customer.id as primary key
4)                  Object[] keys= getForeignKeyForValueAssociation("customer","id",(Tuple) value);
                    //Order to Customer is M to 1. There can only be 1 key
5)                  String CUSTOMER_ID=(String)keys[0];
// parse variable unFormattedDate and format it for the database as formattedDate
6)                   String formattedDate = "2007-05-08-14.01.59.780272"; // formatted for DB2
// Finally, the following SQL statement to insert the record
7) //INSERT INTO ORDER (ORDERNUMBER, DATE, CUSTOMER_ID) VALUES(ORDERNUMBER,formattedDate, CUSTOMER_ID)
                }
                break;
            case LogElement.CODE_UPDATE:
                break;
            case LogElement.CODE_DELETE:
                break;
            }
        }

    }
// returns the value to attribute as stored in the key Tuple
 private Object getKeyAttribute(String attr, Tuple key) {
        //get key metadata
        TupleMetadata keyMD = entityMetaData.getKeyMetadata();
        //get position of the attribute
        int keyAt = keyMD.getAttributePosition(attr);
        if (keyAt > -1) {
            return key.getAttribute(keyAt);
        } else { // attribute undefined
            throw new IllegalArgumentException("Invalid position index for  "+attr);
        }

    }
```

```
// returns the value to attribute as stored in the value Tuple
    private Object getValueAttribute(String attr, Tuple value) {
        //similar to above, except we work with value metadata instead
        TupleMetadata valueMD = entityMetaData.getValueMetadata();

        int keyAt = valueMD.getAttributePosition(attr);
        if (keyAt > -1) {
            return value.getAttribute(keyAt);
        } else {
            throw new IllegalArgumentException("Invalid position index for  "+attr);
        }
    }
// returns an array of keys that refer to association.
    private Object[] getForeignKeyForValueAssociation(String attr, String fk_attr, Tuple value) {
        TupleMetadata valueMD = entityMetaData.getValueMetadata();
        Object[] ro;

        int customerAssociation = valueMD.getAssociationPosition(attr);
        TupleAssociation tupleAssociation = valueMD.getAssociation(customerAssociation);

        EntityMetadata targetEntityMetaData = tupleAssociation.getTargetEntityMetadata();

        Tuple[] customerKeyTuple = ((Tuple) value).getAssociations(customerAssociation);

        int numberOfKeys = customerKeyTuple.length;
        ro = new Object[numberOfKeys];

        TupleMetadata keyMD = targetEntityMetaData.getKeyMetadata();
        int keyAt = keyMD.getAttributePosition(fk_attr);
        if (keyAt < 0) {
            throw new IllegalArgumentException("Invalid position index for  " + attr);
        }
        for (int i = 0; i < numberOfKeys; ++i) {
            ro[i] = customerKeyTuple[i].getAttribute(keyAt);
        }

        return ro;

    }
```

1. Accertarsi che entityMetaData non abbia un valore null il che implica che le voci della cache chiave e valore siano di tipo tupla. Da entityMetaData, viene recuperata la chiave TupleMetaData che in realtà rappresenta solo la parte di chiave dei metadati Order.
2. Elaborare KeyTuple e prendere il valore dell'attributo Key orderNumber
3. Elaborare ValueTuple e prendere il valore dell'attributo Date
4. Elaborare ValueTuple e prendere il valore delle chiavi dall'associazione customer.
5. Estrarre CUSTOMER_ID. In base alla relazione un Order può solo avere un customer, avremo solo una chiave. Di conseguenza, la dimensione delle chiavi è 1. Per semplicità, abbiamo saltato l'analisi dei dati per correggere il formato.
6. Poiché si tratta di un'operazione di inserimento, l'istruzione SQL viene passata durante la connessione dell'origine dati per completare l'operazione di inserimento.

La demarcazione della transazione e l'accesso al database sono descritti in “Scrittura di un programma di caricamento” a pagina 255.

Il metodo Get

Se la chiave non è stata trovata nella cache, richiamare il metodo Get nel plug-in Loader per trovare la chiave.

La chiave è una tupla. La prima operazione è convertire la tupla in valori primitivi che possono essere passati nell'istruzione SQL SELECT. Dopo che tutti gli attributi sono stati recuperati dal database, è necessario convertirli in tuple. Il codice riportato di seguito descrive la classe Order.

```
public List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException {
  System.out.println("OrderLoader: Get called");
  ArrayList returnList = new ArrayList();

1)  if (entityMetaData != null) {
```

```
   int index=0;
   for (Iterator iter = keyList.iterator(); iter.hasNext();) {
2)    Tuple orderKeyTuple=(Tuple) iter.next();

    // The order has just one key orderNumber
3)    String ORDERNUMBERKEY = (String) getKeyAttribute("orderNumber",orderKeyTuple);
    //We need to run a query to get values of
4)    // SELECT CUSTOMER_ID, date FROM ORDER WHERE ORDERNUMBER='ORDERNUMBERKEY'

5)    //1) Foreign key: CUSTOMER_ID
6)    //2) date
    // Assuming those two are returned as
7)                              String  CUSTOMER_ID = "C001"; // Assuming Retrieved and initialized
8)    java.util.Date retrievedDate = new java.util.Date();
                                // Assuming this date reflects the one in database

    // We now need to convert this data into a tuple before returning

    //create a value tuple
9)    TupleMetadata valueMD = entityMetaData.getValueMetadata();
    Tuple valueTuple=valueMD.createTuple();


    //add retrievedDate object to Tuple
    int datePosition = valueMD.getAttributePosition("date");
10)    valueTuple.setAttribute(datePosition, retrievedDate);

    //Next need to add the Association
11)    int customerPosition=valueMD.getAssociationPosition("customer");
    TupleAssociation customerTupleAssociation =
                                  valueMD.getAssociation(customerPosition);
    EntityMetadata customerEMD = customerTupleAssociation.getTargetEntityMetadata();
    TupleMetadata customerTupleMDForKEY=customerEMD.getKeyMetadata();
12)    int customerKeyAt=customerTupleMDForKEY.getAttributePosition("id");


    Tuple customerKeyTuple=customerTupleMDForKEY.createTuple();
    customerKeyTuple.setAttribute(customerKeyAt, CUSTOMER_ID);
13)    valueTuple.addAssociationKeys(customerPosition, new Tuple[] {customerKeyTuple});


14)    int linesPosition = valueMD.getAssociationPosition("lines");
    TupleAssociation linesTupleAssociation = valueMD.getAssociation(linesPosition);
    EntityMetadata orderLineEMD = linesTupleAssociation.getTargetEntityMetadata();
    TupleMetadata orderLineTupleMDForKEY = orderLineEMD.getKeyMetadata();
    int lineNumberAt = orderLineTupleMDForKEY.getAttributePosition("lineNumber");
    int orderAt = orderLineTupleMDForKEY.getAssociationPosition("order");

    if (lineNumberAt < 0 || orderAt < 0) {
     throw new IllegalArgumentException(
                                      "Invalid position index for lineNumber or order "+
                                      lineNumberAt + " " + orderAt);
    }
15) // SELECT LINENUMBER FROM ORDERLINE WHERE ORDERNUMBER='ORDERNUMBERKEY'
    // Assuming two rows of line number are returned with values 1 and 2

    Tuple orderLineKeyTuple1 = orderLineTupleMDForKEY.createTuple();
    orderLineKeyTuple1.setAttribute(lineNumberAt, new Integer(1));// set Key
    orderLineKeyTuple1.addAssociationKey(orderAt, orderKeyTuple);

    Tuple orderLineKeyTuple2 = orderLineTupleMDForKEY.createTuple();
    orderLineKeyTuple2.setAttribute(lineNumberAt, new Integer(2));// Init Key
    orderLineKeyTuple2.addAssociationKey(orderAt, orderKeyTuple);

16)    valueTuple.addAssociationKeys(linesPosition, new Tuple[]
                                  {orderLineKeyTuple1, orderLineKeyTuple2 });

    returnList.add(index, valueTuple);

    index++;

  }
  }else {
   // does not support tuples
  }
  return returnList;
 }
```

1. Il metodo Get viene richiamato quando la cache ObjectGrid potrebbe non trovare la chiave e richiede al programma di caricamento di eseguire il fetch. Verificare il valore per entityMetaData e procedere se non è null.
2. Il keyList contiene tuple.
3. Recuperare il valore dell'attributo orderNumber.
4. Eseguire la query per recuperare la data (valore) e l'ID customer (chiave esterna).

5. CUSTOMER_ID è una chiave esterna che deve essere impostata nella tupla di associazione.
6. La data è un valore e dovrebbe essere già impostato.
7. Poiché questo esempio non esegue chiamate JDBC, il CUSTOMER_ID viene assunto.
8. Poiché questo esempio non esegue chiamate JDBC, la data viene assunta.
9. Creare una tupla di valore.
10. Impostare il valore della data nella tupla sulla base della sua posizione.
11. Order dispone di due associazioni. Iniziare con l'attributo customer che fa riferimento all'entità customer. È necessario avere il valore dell'ID per impostarlo nella tupla.
12. Trovare la posizione dell'ID nell'entità customer.
13. Impostare solo i valori delle chiavi di associazione.
14. Anche lines è un'associazione che deve essere impostata come un gruppo di chiavi di associazione allo stesso modo di come è stato fatto per l'associazione customer.
15. Poiché è necessario impostare le chiavi per il lineNumber associato a questo Order, eseguire SQL per recuperare i valori di lineNumber.
16. Impostare le chiavi di associazione in valueTuple. Ciò completa la creazione della tupla che è stata restituita alla BackingMap.

Questa sezione descrive le operazioni per creare le tuple e una descrizione solo dell'entità Order. Completare operazioni simili per le altre entità e l'intero processo legato al plug-in TransactionCallback. Consultare “Plug-in per la gestione degli eventi del ciclo di vita della transazione” a pagina 274 per i dettagli.

Scrittura di un programma di caricamento utilizzando un controller di precaricamento della replica

Un programma di caricamento con un controller di precaricamento della replica è un programma di caricamento che implementa l'interfaccia ReplicaPreloadController in aggiunta all'interfaccia del programma di caricamento.

Panoramica

L'interfaccia ReplicaPreloadController è progettata per fornire un metodo ad una replica che diviene il frammento dell'elemento primario di conoscere se il precedente frammento dell'elemento primario ha completato il processo di precaricamento. Se il precaricamento è parzialmente completato, viene fornita l'informazione per riprendere da dove si è fermato il precedente elemento primario. Con l'implementazione dell'interfaccia ReplicaPreloadController, una replica che diviene elemento primario continua il processo di precaricamento dove si è fermato il precedente elemento primario e continua per terminare il precaricamento completamente.

In un ambiente distribuito WebSphere eXtreme Scale, una mappa può avere repliche e potrebbe precaricare volumi considerevoli di dati durante l'inizializzazione. Il precaricamento è un'attività del programma di caricamento e si realizza solo nella mappa dell'elemento primario durante l'inizializzazione. Il precaricamento potrebbe richiedere molto tempo per essere completato se viene precaricato un volume considerevole di dati. Se la mappa dell'elemento primario ha completato il precaricamento di una parte considerevole di dati, ma viene arrestata per motivi imprevisti durante l'inizializzazione, una replica diviene

elemento primario . In questa situazione i dati precaricati che furono completati da un precedente elemento primario, andranno persi poiché il nuovo elemento primario normalmente esegue un precaricamento non condizionale. Con un precaricamento non condizionale, il nuovo elemento primario avvia il processo di precaricamento e i dati precedentemente precaricati, verranno ignorati. Se si desidera che il nuovo elemento primario riprenda da dove si è fermato il precedente durante il processo di precaricamento, fornire un programma di caricamento che implementi l'interfaccia ReplicaPreloadController. Per ulteriori informazioni, consultare la documentazione relativa alle API.

Per informazioni relative ai programmi di caricamento, consultare le informazioni sui in *Panoramica sul prodotto*. Se si è interessati alla scrittura di un normale plug-in Loader, consultare "Scrittura di un programma di caricamento" a pagina 255.

L'interfaccia ReplicaPreloadController ha la seguente definizione:

```
public interface ReplicaPreloadController
{
    public static final class Status
    {
        static public final Status PRELOADED_ALREADY = new Status(K_PRELOADED_ALREADY);
        static public final Status FULL_PRELOAD_NEEDED = new Status(K_FULL_PRELOAD_NEEDED);
        static public final Status PARTIAL_PRELOAD_NEEDED = new Status(K_PARTIAL_PRELOAD_NEEDED);
    }

    Status checkPreloadStatus(Session session, BackingMap bmap);
}
```

La seguente sezione descrive alcuni metodi dell'interfaccia Loader e ReplicaPreloadController.

metodocheckPreloadStatus

Quando un programma di caricamento implementa l'interfaccia ReplicaPreloadController, il metodocheckPreloadStatus viene richiamato prima del metodopreloadMapdurante l'inizializzazione della mappa. Lo stato di ritorno di questo metodo determina se il metodo preloadMap è stato richiamato. Se questo metodo restituisce `Status#PRELOADED_ALREADY`, il metodo di precaricamento non è stato richiamato. Altrimenti, il metodo preload viene eseguito. A causa di questo comportamento, questo metodo dovrebbe essere utilizzato come metodo di inizializzazione del programma di caricamento. È necessario inizializzare le proprietà del programma di caricamento in questo metodo. Questo metodo deve restituire lo stato corretto o il precaricamento potrebbe non funzionare come previsto.

```
public Status checkPreloadStatus(Session session, BackingMap backingMap) {
        // When a loader implements ReplicaPreloadController interface,
    // this method will be called before preloadMap method during
    // map initialization. Whether the preloadMap method will be
    // called depends on teh returned status of this method. So, this
    // method also serve as Loader's initialization method. This method
    // has to return the right staus, otherwise the preload may not
    // work as expected.

        // Note: must initialize this loader instance here.
        ivOptimisticCallback = backingMap.getOptimisticCallback();
        ivBackingMapName = backingMap.getName();
        ivPartitionId = backingMap.getPartitionId();
        ivPartitionManager = backingMap.getPartitionManager();
        ivTransformer = backingMap.getObjectTransformer();
        preloadStatusKey = ivBackingMapName + "_" + ivPartitionId;

        try {
            // get the preloadStatusMap to retrieve preload status that
      // could be set by other JVMs.
            ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);
```

```
            // retrieve last recorded preload data chunk index.
            Integer lastPreloadedDataChunk = (Integer) preloadStatusMap.get(preloadStatusKey);

            if (lastPreloadedDataChunk == null) {
                preloadStatus = Status.FULL_PRELOAD_NEEDED;
            } else {
                preloadedLastDataChunkIndex = lastPreloadedDataChunk.intValue();
                if (preloadedLastDataChunkIndex == preloadCompleteMark) {
                    preloadStatus = Status.PRELOADED_ALREADY;
                } else {
                    preloadStatus = Status.PARTIAL_PRELOAD_NEEDED;
                }
            }

            System.out.println("TupleHeapCacheWithReplicaPreloadControllerLoader.checkPreloadStatus()
     -> map = " + ivBackingMapName + ", preloadStatusKey = " + preloadStatusKey
                    + ", retrieved lastPreloadedDataChunk =" + lastPreloadedDataChunk + ",
         determined preloadStatus = "
                    + getStatusString(preloadStatus));

        } catch (Throwable t) {
            t.printStackTrace();
        }

        return preloadStatus;
    }
```

metodo preloadMap

L'esecuzione del metodo preloadMap dipende dal risultato restituito dal metodo checkPreloadStatus. Se il metodo preloadMap è richiamato, esso generalmente deve recuperare le informazioni sullo stato del precaricamento dalla mappa dello stato di precaricamento progettata e determinare come procedere. L'ideale è che il metodo preloadMap conosca se il precaricamento è stato completato parzialmente ed esattamente da dove ricominciare. Durante il precaricamento dei dati, il metodo preloadMap dovrebbe aggiornare lo stato di precaricamento nella mappa dello stato di precaricamento progettata. Lo stato di precaricamento che è memorizzato nella mappa dello stato di precaricamento viene recuperato dal metodo checkPreloadStatus quando ha necessità di verificare lo stato di precaricamento.

```
public void preloadMap(Session session, BackingMap backingMap)
     throws LoaderException {
        EntityMetadata emd = backingMap.getEntityMetadata();
        if (emd != null && tupleHeapPreloadData != null) {
            // The getPreLoadData method is similar to fetching data
      // from database. These data will be push into cache as
      // preload process.
            ivPreloadData = tupleHeapPreloadData.getPreLoadData(emd);

            ivOptimisticCallback = backingMap.getOptimisticCallback();
            ivBackingMapName = backingMap.getName();
            ivPartitionId = backingMap.getPartitionId();
            ivPartitionManager = backingMap.getPartitionManager();
            ivTransformer = backingMap.getObjectTransformer();
            Map preloadMap;

            if (ivPreloadData != null) {
                try {
                    ObjectMap map = session.getMap(ivBackingMapName);

                    // get the preloadStatusMap to record preload status.
                    ObjectMap preloadStatusMap = session.getMap(ivPreloadStatusMapName);

                    // Note: when this preloadMap method is invoked, the
        // checkPreloadStatus has been called, Both preloadStatus

        // and preloadedLastDataChunkIndex have been set. And the
        // preloadStatus must be either PARTIAL_PRELOAD_NEEDED

        // or FULL_PRELOAD_NEEDED that will require a preload again.

                    // If large amount of data will be preloaded, the data usually
        // is divided into few chunks and the preload process will
        // process each chunk within its own tran. This sample only
        // preload few entries and assuming each entry represent a chunk.
```

```
            // so that the preload process an entry in a tran to simulate
// chunk preloading.

            Set entrySet = ivPreloadData.entrySet();
            preloadMap = new HashMap();
            ivMap = preloadMap;

            // The dataChunkIndex represent the data chunk that is in
// processing
            int dataChunkIndex = -1;
            boolean shouldRecordPreloadStatus = false;
            int numberOfDataChunk = entrySet.size();
            System.out.println("    numberOfDataChunk to be preloaded = "
  + numberOfDataChunk);

            Iterator it = entrySet.iterator();
            int whileCounter = 0;
            while (it.hasNext()) {
                whileCounter++;
                System.out.println("preloadStatusKey = " + preloadStatusKey + " ,
   whileCounter = " + whileCounter);

                dataChunkIndex++;

                // if the current dataChunkIndex <= preloadedLastDataChunkIndex
                // no need to process, becasue it has been preloaded by
  // other JVM before. only need to process dataChunkIndex
  // > preloadedLastDataChunkIndex
                if (dataChunkIndex <= preloadedLastDataChunkIndex) {
                    System.out.println("ignore current dataChunkIndex =
     " + dataChunkIndex + " that has been previously
     preloaded.");
                    continue;
                }

                // Note: This sample simulate data chunk as an entry.
                // each key represent a data chunk for simplicity.
                // If the primary server or shard stopped for unknown
   // reason, the preload status that indicates the progress
   // of preload should be available in preloadStatusMap. A
  // replica that become a primary can get the preload status
  // and determine how to preload again.
                // Note: recording preload status should be in the same
   // tran as putting data into cache; so that if tran
  // rollback or error, the recorded preload status is the
  // actual status.

                Map.Entry entry = (Entry) it.next();
                Object key = entry.getKey();
                Object value = entry.getValue();
                boolean tranActive = false;

                System.out.println("processing data chunk. map = " +
    this.ivBackingMapName + ", current dataChunkIndex = " +
    dataChunkIndex + ", key = " + key);

                try {
                    shouldRecordPreloadStatus = false; // re-set to false
                    session.beginNoWriteThrough();
                    tranActive = true;

                    if (ivPartitionManager.getNumOfPartitions() == 1) {
                        // if just only 1 partition, no need to deal with
      // partition.
                        // just push data into cache
                        map.put(key, value);
                        preloadMap.put(key, value);
                        shouldRecordPreloadStatus = true;
                    } else if (ivPartitionManager.getPartition(key) == ivPartitionId) {
                        // if map is partitioned, need to consider the
      // partition key only preload data that belongs
      // to this partition.
                        map.put(key, value);
                        preloadMap.put(key, value);
                        shouldRecordPreloadStatus = true;
                    } else {
                        // ignore this entry, because it does not belong to
      // this partition.
                    }
```

```
                    if (shouldRecordPreloadStatus) {
                        System.out.println("record preload status. map = " +
                  this.ivBackingMapName + ", preloadStatusKey = " +
                  preloadStatusKey + ", current dataChunkIndex ="
                  + dataChunkIndex);
                        if (dataChunkIndex == numberOfDataChunk) {
                            System.out.println("record preload status. map = " +
                     this.ivBackingMapName + ", preloadStatusKey = " +
                    preloadStatusKey + ", mark complete =" +
                    preloadCompleteMark);
                            // means we are at the lastest data chunk, if commit
                // successfully, record preload complete.
                // at this point, the preload is considered to be done
                            // use -99 as special mark for preload complete status.

                            preloadStatusMap.get(preloadStatusKey);

                            // a put follow a get will become update if the get
                 // return an object, otherwise, it will be insert.
                            preloadStatusMap.put(preloadStatusKey, new Integer(preloadCompleteMark));

                        } else {
                            // record preloaded current dataChunkIndex into
                 // preloadStatusMap a put follow a get will become
                 // update if teh get return an object, otherwise, it
                 // will be insert.
                            preloadStatusMap.get(preloadStatusKey);
                            preloadStatusMap.put(preloadStatusKey, new Integer(dataChunkIndex));
                        }
                    }

                    session.commit();
                    tranActive = false;

                    // to simulate preloading large amount of data
                    // put this thread into sleep for 30 secs.
                    // The real app should NOT put this thread to sleep
                    Thread.sleep(10000);

                } catch (Throwable e) {
                    e.printStackTrace();
                    throw new LoaderException("preload failed with exception: " + e, e);
                } finally {
                    if (tranActive && session != null) {
                        try {
                            session.rollback();
                        } catch (Throwable e1) {
                            // preload ignoring exception from rollback
                        }
                    }
                }
            }

            // at this point, the preload is considered to be done for sure
            // use -99 as special mark for preload complete status.
            // this is a insurance to make sure the complete mark is set.
            // besides, when partitioning, each partition does not know when
     // is its last data chunk. so the following block serves as the
     // overall preload status complete reporting.
            System.out.println("Overall preload status complete -> record
         preload status. map = " + this.ivBackingMapName + ",
         preloadStatusKey = " + preloadStatusKey + ", mark complete =" +
        preloadCompleteMark);
            session.begin();
            preloadStatusMap.get(preloadStatusKey);
            // a put follow a get will become update if teh get return an object,
            // otherwise, it will be insert.
            preloadStatusMap.put(preloadStatusKey, new Integer(preloadCompleteMark));
            session.commit();

            ivMap = preloadMap;
        } catch (Throwable e) {
            e.printStackTrace();
            throw new LoaderException("preload failed with exception: " + e, e);
        }
    }
  }
}
```

Mappa dello stato di precaricamento

È possibile utilizzare una mappa dello stato di precaricamento per supportare l'implementazione dell'interfaccia ReplicaPreloadController Il metodo preloadMap dovrebbe sempre prima verificare lo stato di precaricamento memorizzato nella mappa dello stato di precaricamento e aggiornare lo stato di precaricamento nella mappa dello stato di precaricamento ogni volta che memorizza i dati nella cache. Il metodo checkPreloadStatus può recuperare lo stato di precaricamento dalla mappa dello stato di precaricamento, determinare lo stato di precaricamento e restituire lo stato al suo chiamante. La mappa dello stato di precaricamento dovrebbe essere nella stessa mapSet delle altre mappe che dispongono di programmi di caricamento del controller di precaricamento della replica.

Plug-in per la gestione degli eventi del ciclo di vita della transazione

Utilizzare il plug-in TransactionCallback per personalizzare le operazioni di controllo versioni e confronto degli oggetti di cache quando si utilizza la strategia di blocco ottimistico.

È possibile fornire un oggetto callback ottimistico collegabile che implementi l'interfaccia com.ibm.websphere.objectgrid.plugins.OptimisticCallback. Per le mappe di entità, viene configurato automaticamente un plug-in OptimisticCallback ad elevate prestazioni.

Scopo

Utilizzare l'interfaccia OptimisticCallback per fornire operazioni di confronto ottimistico per i valori di una mappa. Quando si utilizza la strategia di blocco ottimistico, un'implementazione OptimisticCallback è obbligatoria. WebSphere eXtreme Scale fornisce un'implementazione OptimisticCallback predefinita. Tuttavia, di solito l'applicazione deve collegare la propria implementazione dell'interfaccia OptimisticCallback. Consultare le informazioni relative alle strategie di blocco in *Panoramica sul prodotto* per ulteriori informazioni.

Implementazione predefinita

Il framework eXtreme Scale fornisce un'implementazione predefinita dell'interfaccia OptimisticCallback utilizzata se l'applicazione non collega un oggetto OptimisticCallback proprio, come mostrato nella sezione precedente. L'implementazione predefinita restituisce sempre un valore speciale NULL_OPTIMISTIC_VERSION come oggetto versione per il valore e non aggiorna mai l'oggetto versione. Questa azione rende il confronto ottimistico una funzione "no operation". In molti casi, si preferisce che la funzione "no operation" non si verifichi quando si utilizza la strategia di blocco ottimistico. Dovranno essere le proprie applicazioni ad implementare l'interfaccia OptimisticCallback e collegare le proprie implementazioni OptimisticCallback in modo da non utilizzare le implementazioni predefinite. Tuttavia, esiste almeno uno scenario in cui l'implementazione predefinita OptimisticCallback torna utile. Considerare la seguente situazione:

- Viene collegato un programma di caricamento per la mappa di backup.
- Il programma di caricamento sa come effettuare il confronto ottimistico senza l'aiuto di un plug-in OptimisticCallback.

Come può un programma di caricamento sapere come gestire il controllo versioni ottimistico senza l'aiuto di un oggetto OptimisticCallback? Il programma di caricamento è a conoscenza dell'oggetto classe del valore e del campo dell'oggetto

valore che viene utilizzato come valore del controllo versioni ottimistico. Ad esempio, si supponga di utilizzare la seguente l'interfaccia come oggetto valore per la mappa employee:

```
public interface Employee
{
    // Sequential sequence number used for optimistic versioning.
    public long getSequenceNumber();
    public void setSequenceNumber(long newSequenceNumber);
    // Other get/set methods for other fields of Employee object.
}
```

In questo caso, il programma di caricamento sa di poter utilizzare il metodo getSequenceNumber per acquisire le informazioni di versione correnti per un oggetto valore Employee. Il programma di caricamento incrementa il valore restituito per generare un nuovo numero versione prima di aggiornare la memoria persistente con il nuovo valore Employee. Per un programma di caricamento JDBC (Java database connectivity), viene utilizzato il numero sequenza corrente nella clausola where di un'istruzione di aggiornamento SQL sovraqualificata ed esso utilizza il numero sequenza appena generato per impostare la colonna del numero sequenza con il nuovo valore del numero sequenza.

Un'altra possibilità consiste nell'utilizzo da parte del programma di caricamento di alcune funzioni fornite dal backend che aggiornano automaticamente una colonna nascosta da utilizzare per il controllo versioni ottimistico. In alcuni casi, è possibile utilizzare una procedura memorizzata o trigger per ottenere assistenza nella gestione di una colonna contenente le informazioni del controllo versioni. Se il programma di caricamento utilizza una di queste tecniche per la gestione delle informazioni del controllo versioni ottimistico, non sarà necessario per l'applicazione fornire un'implementazione OptimisticCallback. Sarà quindi possibile utilizzare l'implementazione OptimisticCallback predefinita poiché il programma di caricamento sarà in grado di gestire il controllo versioni ottimistico senza alcuna assistenza da parte dell'oggetto OptimisticCallback.

Impostazione predefinita delle entità

Le entità sono memorizzate nell'ObjectGrid utilizzando gli oggetti tupla. L'implementazione OptimisticCallback ha un comportamento simile a quello tenuto nei confronti delle mappe non-entità. Tuttavia, il campo versione nell'entità viene identificato utilizzando l'annotazione @Version oppure l'attributo versione nel file XML descrittore entità.

L'attributo versione può essere del seguente tipo: int, Integer, short, Short, long, Long o java.sql.Timestamp. Un'entità può avere definito un unico attributo versione. L'attributo versione deve essere impostato solo durante la costruzione. Dopo la persistenza dell'entità, l'attributo versione non deve essere modificato.

Se l'attributo versione non viene configurato e si utilizza la strategia di blocco ottimistico, l'intera tupla viene fornita di versione in modo implicito, utilizzando lo stato della tupla.

Nel seguente esempio l'entità Employee ha un attributo versione long denominato SequenceNumber:

```
@Entity
public class Employee
{
private long sequence;
    // Sequential sequence number used for optimistic versioning.
```

```
    @Version
    public long getSequenceNumber() {
        return sequence;
    }
    public void setSequenceNumber(long newSequenceNumber) {
        this.sequence = newSequenceNumber;
    }
    // Other get/set methods for other fields of Employee object.
}
```

Scrittura di un'implementazione OptimisticCallback

Un'implementazione OptimisticCallback deve implementare l'interfaccia OptimisticCallback e seguire le convenzioni comuni del plug-in ObjectGrid

Il seguente elenco fornisce una descrizione o considerazione per ciascun metodo nell'interfaccia OptimisticCallback:

NULL_OPTIMISTIC_VERSION

Questo valore speciale viene restituito dal metodo getVersionedObjectForValue se si utilizza l'implementazione OptimisticCallback predefinita invece di quella fornita dall'applicazione.

Metodo getVersionedObjectForValue

Il metodo getVersionedObjectForValue può restituire una copia del valore o un suo attributo da poter utilizzare ai fini del controllo versioni. Questo metodo viene chiamato ogni volta che si associa un oggetto ad una transazione. Quando nella mappa di backup non è impostato alcun programma di caricamento, essa utilizza questo valore al momento del commit per eseguire un confronto della versione ottimistico. Il confronto della versione ottimistico viene utilizzato dalla mappa di backup per assicurarsi che la versione non sia cambiata dal momento in cui questa transazione ha acceduto per la prima volta la voce della mappa da essa modificata. Se un'altra transazione ha già modificato la versione di questa voce della mappa, il confronto della versione non riesce e la mappa di backup mostra un'eccezione OptimisticCollisionException per forzare il rollback della transazione. Se si collega un programma di caricamento, la mappa di backup non utilizza le informazioni del controllo versioni ottimistico. Il programma di caricamento, invece, è responsabile dell'esecuzione del confronto del controllo versioni ottimistico e dell'aggiornamento delle informazioni del controllo versioni quando necessario. Il programma di caricamento solitamente riceve l'oggetto di controllo versioni iniziale dal LogElement passato al metodo batchUpdate nel programma di caricamento, che viene chiamato quando si verifica un'operazione di flush o quando si esegue il commit di una 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 richiesto dal programma di caricamento, il che implica che la persona che ha scritto il programma di caricamento sia la stessa che ha scritto l'implementazione EmployeeOptimisticCallbackImpl o che abbia lavorato a stretto contatto con la persona che l'ha implementata. Ad esempio, queste persone hanno concordato sul valore restituito dal metodo getVersionedObjectForValue. Come descritto in precedenza, l'implementazione predefinita di OptimisticCallback restituisce il valore speciale NULL_OPTIMISTIC_VERSION come oggetto versione.

Metodo updateVersionedObjectForValue

Il metodo updateVersionedObjectForValue viene chiamato quando una transazione ha aggiornato un valore e si necessita di un nuovo oggetto fornito di versione. Se il metodo getVersionedObjectForValue restituisce un attributo del valore, solitamente esso aggiorna il valore dell'attributo con un nuovo oggetto versione. Se il metodo getVersionedObjectForValue restituisce una copia del valore, solitamente questo metodo non si aggiorna. L'OptimisticCallback predefinito non si aggiorna poiché l'implementazione predefinita del metodo getVersionedObjectForValue restituisce sempre il valore speciale NULL_OPTIMISTIC_VERSION come oggetto della versione. Il seguente esempio mostra l'implementazione utilizzata dall'oggetto EmployeeOptimisticCallbackImpl che viene utilizzato nella sezione OptimisticCallback:

```
public void updateVersionedObjectForValue(Object value)
{
    if ( value != null )
    {
        Employee emp = (Employee) value;
        long next = emp.getSequenceNumber() + 1;
        emp.updateSequenceNumber( next );
    }
}
```

Come dimostrato nell'esempio precedente, l'attributo sequenceNumber viene incrementato di un'unità in modo che, quando il metodo getVersionedObjectForValue viene successivamente chiamato, il valore java.lang.Long restituito abbia un valore long corrispondente al valore del numero sequenza originale. Ad esempio, più uno è il valore di versione successivo per questa istanza Employee. Come in precedenza, questo esempio suppone che la persona che ha scritto il programma di caricamento sia la stessa che ha scritto l'implementazione EmployeeOptimisticCallbackImpl o che abbia lavorato a stretto contatto con la persona che l'ha implementata.

Metodo serializeVersionedValue

Questo metodo scrive il valore fornito di versione nel flusso specificato. A seconda dell'implementazione, il valore fornito di versione può essere utilizzato per identificare le collisioni dell'aggiornamento ottimistico. In alcune implementazioni, il valore fornito di versione è una copia del valore originale. Altre implementazioni potrebbero avere un numero sequenza o un altro oggetto per indicare la versione del valore. Poiché l'implementazione effettiva non è nota, questo metodo viene fornito per eseguire la corretta serializzazione. L'implementazione predefinita chiama il metodo writeObject.

Metodo inflateVersionedValue

Questo metodo acquisisce la versione serializzata del valore fornito di versione e restituisce l'effettivo oggetto valore fornito di versione. A seconda dell'implementazione, il valore fornito di versione può essere utilizzato per identificare le collisioni dell'aggiornamento ottimistico. In alcune implementazioni, il valore fornito di versione è una copia del valore originale. Altre implementazioni potrebbero avere un numero sequenza o un altro oggetto per indicare la versione del valore. Poiché l'implementazione effettiva non è nota, questo metodo viene fornito per eseguire la corretta deserializzazione. L'implementazione predefinita chiama il metodo readObject.

Utilizzo dell'implementazione OptimisticCallback fornita dall'applicazione

Esistono due approcci per aggiungere l'OptimisticCallback fornita dall'applicazione in una configurazione della mappa di backup: la configurazione programmatica e la configurazione XML.

Plug in programmatico di una implementazione OptimisticCallback

Il seguente esempio dimostra come un'applicazione può effettuare il plug-in programmatico di un oggetto OptimisticCallback per la mappa di backup Employee nell'istanza grid1 di 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 );
```

Approccio con configurazione XML per il plug-in di un'implementazione OptimisticCallback

L'oggetto EmployeeOptimisticCallbackImpl nell'esempio precedente deve implementare l'interfaccia OptimisticCallback. L'applicazione può anche utilizzare un file XML per effettuare il plug-in del proprio oggetto OptimisticCallback, come mostrato nell'esempio seguente:

```
<?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>
```

Panoramica sull'elaborazione della transazione

Introduzione agli slot del plug-in

Uno slot del plug-in rappresenta uno spazio di storage transazionale riservato ai plug-in che condividono il contesto transazionale. Questi slot consentono ai plug-in

di eXtreme Scale di comunicare reciprocamente, condividere il contesto transazionale ed assicurano l'utilizzo corretto e coerente delle risorse transazionali nella transazione.

Un plug-in può memorizzare contesto transazionale, come una connessione ad un database, una connessione Java Message Service (JMS) e così via, in uno slot del plug-in. Il contesto transazionale memorizzato può essere recuperato da qualsiasi plug-in a conoscenza del numero dello slot del plug-in, che serve da chiave per recuperare il contesto transazionale.

Utilizzo degli slot del plug-in

Gli slot del plug-in sono parte dell'interfaccia TxID. Consultare la Documentazione API per ulteriori informazioni sull'interfaccia.Gli slot sono voci su un array ArrayList. I plug-in possono riservare una voce nell'array ArrayList chiamando il metodo ObjectGrid.reserveSlot ed indicando che esso richiede uno slot su tutti gli oggetti TxID. Dopo che gli slot sono stati riservati, i plug-in possono inserire il contesto transazionale negli slot di ciascun oggetto TxID e recuperare il contesto successivamente. Le operazioni di put e get sono coordinate in base ai numeri degli slot restituiti dal metodo ObjectGrid.reserveSlot.

Un plug-in ha solitamente un ciclo di vita. L'utilizzo degli slot del plug-in deve adattarsi al ciclo di vita del plug-in. Di solito, il plug-in deve riservare gli slot del plug-in durante la fase di inizializzazione ed ottenere i numeri degli slot per ciascuno slot. Durante il normale runtime, il plug-in inserisce al momento appropriato il contesto transazionale nello slot riservato nell'oggetto TxID. Il momento appropriato è solitamente all'inizio della transazione. Il plug-in o altri plug-in potranno quindi ricevere il contesto transazionale memorizzato con il numero dello slot dal TxID all'interno della transazione.

Il plug-in solitamente effettua una ripulitura, rimuovendo il contesto transazionale e gli slot. Il seguente frammento di codice illustra come utilizzare gli slot del plug-in in un plug-in TransactionCallback:

```
public class DatabaseTransactionCallback implements TransactionCallback {
    int connectionSlot;
    int autoCommitConnectionSlot;
    int psCacheSlot;
    Properties ivProperties = new Properties();

    public void initialize(ObjectGrid objectGrid) throws TransactionCallbackException {
        // In initialization stage, reserve desired plug-in slots by calling the
        // reserveSlot method of ObjectGrid and
        // passing in the designated slot name, TxID.SLOT_NAME.
        // Note: you have to pass in this TxID.SLOT_NAME that is designated
        // for application.
        try {
            // cache the returned slot numbers
            connectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
            psCacheSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
            autoCommitConnectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
        } catch (Exception e) {
        }
    }

    public void begin(TxID tx) throws TransactionCallbackException {
        // put transactional contexts into the reserved slots at the
        // beginning of the transaction.
        try {
            Connection conn = null;
            conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
            tx.putSlot(connectionSlot, conn);
            conn = DriverManager.getConnection(ivDriverUrl, ivProperties);
            conn.setAutoCommit(true);
            tx.putSlot(autoCommitConnectionSlot, conn);
            tx.putSlot(psCacheSlot, new HashMap());
        } catch (SQLException e) {
            SQLException ex = getLastSQLException(e);
            throw new TransactionCallbackException("unable to get connection", ex);
        }
    }
```

```
    public void commit(TxID id) throws TransactionCallbackException {
        // get the stored transactional contexts and use them
        // then, clean up all transactional resources.
        try {
            Connection conn = (Connection) id.getSlot(connectionSlot);
            conn.commit();
            cleanUpSlots(id);
        } catch (SQLException e) {
            SQLException ex = getLastSQLException(e);
            throw new TransactionCallbackException("commit failure", ex);
        }
    }

    void cleanUpSlots(TxID tx) throws TransactionCallbackException {
        closePreparedStatements((Map) tx.getSlot(psCacheSlot));
        closeConnection((Connection) tx.getSlot(connectionSlot));
        closeConnection((Connection) tx.getSlot(autoCommitConnectionSlot));
    }

    /**
     * @param map
     */
    private void closePreparedStatements(Map psCache) {
        try {
            Collection statements = psCache.values();
            Iterator iter = statements.iterator();
            while (iter.hasNext()) {
                PreparedStatement stmt = (PreparedStatement) iter.next();
                stmt.close();
            }
        } catch (Throwable e) {
        }

    }

    /**
     * Close connection and swallow any Throwable that occurs.
     *
     * @param connection
     */
    private void closeConnection(Connection connection) {
        try {
            connection.close();
        } catch (Throwable e1) {
        }
    }

    public void rollback(TxID id) throws TransactionCallbackException
        // get the stored transactional contexts and use them
        // then, clean up all transactional resources.
        try {
            Connection conn = (Connection) id.getSlot(connectionSlot);
            conn.rollback();
            cleanUpSlots(id);
        } catch (SQLException e) {
        }
    }

    public boolean isExternalTransactionActive(Session session) {
        return false;
    }

    // Getter methods for the slot numbers, other plug-in can obtain the slot numbers
    // from these getter methods.

    public int getConnectionSlot() {
        return connectionSlot;
    }
    public int getAutoCommitConnectionSlot() {
        return autoCommitConnectionSlot;
    }
    public int getPreparedStatementSlot() {
        return psCacheSlot;
    }
```

Il seguente frammento di codice illustra come un programma di caricamento può ricevere il contesto transazionale memorizzato, precedentemente inserito dall'esempio del plug-in TransactionCallback:

```
public class DatabaseLoader implements Loader
{
    DatabaseTransactionCallback tcb;
    public void preloadMap(Session session, BackingMap backingMap) throws LoaderException
    {
        // The preload method is the initialization method of the Loader.
        // Obtain interested plug-in from Session or ObjectGrid instance.
        tcb =
   (DatabaseTransactionCallback)session.getObjectGrid().getTransactionCallback();
```

```
    }
    public List get(TxID txid, List keyList, boolean forUpdate) throws LoaderException
    {
        // get the stored transactional contexts that is put by tcb's begin method.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implement get here
        return null;
    }
    public void batchUpdate(TxID txid, LogSequence sequence) throws LoaderException,
   OptimisticCollisionException
    {
        // get the stored transactional contexts that is put by tcb's begin method.
        Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
        // implement batch update here ...
    }
}
```

Gestori delle transazioni esterne

Generalmente, le transazioni eXtreme Scale cominciano con il metodo Session.begin e terminano con il metodo Session.commit. Tuttavia, quando è integrato un ObjectGrid, un coordinatore transazione esterne può avviare e terminare le transazioni. In questo caso, non è necessario richiamare i metodi begin o commit.

Coordinamento transazione esterna

Il plug-in TransactionCallback viene esteso con il metodo isExternalTransactionActive(Session session) che associa la sessione eXtreme Scale ad una transazione esterna. Di seguito è riportato il metodo Header:

```
public synchronized boolean isExternalTransactionActive(Session session)
```

Ad esempio, eXtreme Scale può essere impostato per integrarsi con WebSphere Application Server and WebSphere Extended Deployment.

Anche eXtreme Scale fornisce un plug-in incorporato denominato WebSphere “Plug-in per la gestione degli eventi del ciclo di vita della transazione” a pagina 274, che descrive come eseguire il build del plug-in per gli ambienti WebSphere Application Server, ma è possibile adattare il plug-in ad altri framework.

La chiave di questa integrazione seamless è l'utilizzazione dell'API ExtendedJTATransaction nella Versione 5.x WebSphere Application Server e Versione 6.x. Tuttavia, se si sta utilizzando la Versione 6.0.2 WebSphere Application Server, è necessario applicare l'APAR PK07848 per supportare questo metodo. Utilizzare il codice di esempio di seguito riportato per associare una sessione ObjectGrid ad un ID della transazione WebSphere Application Server:

```
/**
* This method is required to associate an objectGrid session with a WebSphere
* Application Server transaction ID.
*/
Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
    // remember that this localid means this session is saved for later.
    localIdToSession.put(new Integer(jta.getLocalId()), session);
    return true;
}
```

Recupero di una transazione esterna

Qualche volta, si potrebbe avere la necessità di recuperare un oggetto servizio della transazione esterna per il plug-in TransactionCallback da utilizzare. Sul server WebSphere Application Server, ricercare l'oggetto ExtendedJTATransaction dal suo spazio dei nomi nell'esempio riportato di seguito:

```
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 altri prodotti, è possibile utilizzare un approccio simile per recuperare l'oggetto servizio della transazione.

Commit di controllo mediante callback esterno

Il plug-in TransactionCallback deve ricevere un segnale esterno per eseguire il commit o il rollback nella sessione eXtreme Scale. Per ricevere questo segnale esterno, utilizzare il callback da un servizio della transazione esterna. Implementare l'interfaccia di callback estrerno e registrarlo con il servizio della transazione esterna. Ad esempio, con WebSphere Application Server, si implementa l'interfaccia SynchronizationCallback, come mostrato nell'esempio di seguito riportato:

```
public class J2EETransactionCallback implements
 com.ibm.websphere.objectgrid.plugins.TransactionCallback, SynchronizationCallback {
   public J2EETransactionCallback() {
      super();
      String lookupName="java:comp/websphere/ExtendedJTATransaction";
      localIdToSession = new HashMap();
      try {
         InitialContext ic = new InitialContext();
         jta = (ExtendedJTATransaction)ic.lookup(lookupName);
         jta.registerSynchronizationCallback(this);
      } catch(NotSupportedException e) {
         throw new RuntimeException("Cannot register jta callback", e);
      }
      catch(NamingException e) {
         throw new RuntimeException("Cannot get transaction object");
      }
   }

   public synchronized void afterCompletion(int localId, byte[] arg1,boolean didCommit) {
      Integer lid = new Integer(localId);
      // find the Session for the localId
      Session session = (Session)localIdToSession.get(lid);
      if(session != null) {
          try {
            // if WebSphere Application Server is committed when
            // hardening the transaction to backingMap.
            // We already did a flush in beforeCompletion
            if(didCommit) {
               session.commit();
            } else {
               // otherwise rollback
               session.rollback();
            }
         } catch(NoActiveTransactionException e) {
            // impossible in theory
         } catch(TransactionException e) {
            // given that we already did a flush, this should not fail
         } finally {
            // always clear the session from the mapping map.
            localIdToSession.remove(lid);
         }
      }
   }

   public synchronized void beforeCompletion(int localId, byte[] arg1) {
```

```
      Session session = (Session)localIdToSession.get(new Integer(localId));
      if(session != null) {
         try {
            session.flush();
         } catch(TransactionException e) {
            // WebSphere Application Server  does not formally define
            // a way to signal the
            // transaction has failed so do this
            throw new RuntimeException("Cache flush failed", e);
         }
      }
   }
}
```

Utilizzare le API eXtreme Scale con il plug-in TransactionCallback

Il plug-in TransactionCallback plug-in disabilita l'autocommit eXtreme Scale. Di seguito è riportato l'utilizzo normale del pattern per una eXtreme Scale:

```
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
ogSession.begin();
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
ogSession.commit();
```

Quando è in uso il plug-in TransactionCallback, eXtreme Scale presuppone che l'applicazione utilizzi eXtreme Scale quando è presente una transazione gestita dal contenitore. Il precedente frammento di codice modifica il codice riportato di seguito in questi ambienti:

```
public void myMethod() {
   UserTransaction tx = ...;
   tx.begin();
   Session ogSession = ...;
   ObjectMap myMap = ogSession.getMap("MyMap");
   yObject v = myMap.get("key");
   v.setAttribute("newValue");
   myMap.update("key", v);
   tx.commit();
}
```

Il metodo myMethod è simile allo scenario dell'applicazione Web. L'applicazione utilizza la normale interfaccia UserTransaction per cominciare, eseguire il commit e il rollback delle transazioni. Il eXtreme Scale avvia 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 rimuovere il riferimento UserTransaction e le chiamate per incominciare ed eseguire il commit delle transazioni e il metodo funziona allo stesso modo. In questo caso il contenitore è responsabile dell'avvio e della fine della transazione.

Plug-in WebSphereTransactionCallback

Quando si utilizza il plug-in WebSphereTransactionCallback, le applicazioni enterprise che sono in esecuzione in un ambiente WebSphere Application Server saranno in grado di gestire le transazioni ObjectGrid.

Quando si utilizza una sessione ObjectGrid all'interno di un metodo configurato per l'utilizzo di transazioni gestite dal contenitore, il contenitore enterprise esegue automaticamente l'avvio, il commit o il rollback della transazione ObjectGrid. Quando si utilizzano gli oggetti UserTransaction JTA (Java Transaction API), la transazione ObjectGrid viene gestita automaticamente dall'oggetto UserTransaction.

Per una dettagliata disamina dell'implementazione di questo plug-in, consultare “Gestori delle transazioni esterne” a pagina 281.

Nota: ObjectGrid non supporta transazioni a due fasi o XA. Questo plug-in non impegna la transazione ObjectGrid con il gestore transazioni. Per cui, se il commit di ObjectGrid non riesce, non sarà possibile effettuare il rollback di qualsiasi altra risorsa gestita dalla transazione XA.

Abilitazione del plug-in WebSphereTransactionCallback

È possibile abilitare WebSphereTransactionCallback nella configurazione di ObjectGrid tramite la configurazione programmatica o quella XML.

Approccio con configurazione XML per il plug-in dell'oggetto WebSphereTransactionCallback

La seguente configurazione XML crea l'oggetto WebSphereTransactionCallback e lo aggiunge ad un ObjectGrid. Il seguente testo deve essere inserito 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="TransactionCallback" className=
 "com.ibm.websphere.objectgrid.plugins.builtins.WebSphereTransactionCallback" />

        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

Plug-in programmatico dell'oggetto WebSphereTransactionCallback

Il seguente frammento di codice crea l'oggetto WebSphereTransactionCallback e lo aggiunge ad un ObjectGrid:

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
WebSphereTransactionCallback wsTxCallback= new WebSphereTransactionCallback ();
myGrid.setTransactionCallback(wsTxCallback);
```

Capitolo 6. Programmazione di attività di gestione

In aggiunta alle API (Application Programming Interface) di sistema, WebSphere eXtreme Scale include anche delle API di gestione che consentono alle applicazioni di eseguire il monitoraggio e la gestione dei server e dei client.

API server incorporate

WebSphere eXtreme Scale include API (application programming interface) e le interfacce di programmazione di sistema per incorporare i server ed i client eXtreme Scale all'interno delle proprie applicazioni Java. Il seguente argomento descrive le API del server incorporate disponibili.

Creazione di un'istanza del server eXtreme Scale

È possibile utilizzare diverse proprietà per configurare l'istanza del server eXtreme Scale, che è possibile recuperare dal metodo ServerFactory.getServerProperties. L'oggetto ServerProperties è un singleton, quindi ciascuna chiamata al metodo getServerProperties recupera la stessa istanza.

È possibile creare un nuovo server con il seguente codice.

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

Tutte le proprietà impostate prima della prima chiamata getInstance sono utilizzate per inizializzare il server.

Impostazione delle proprietà del server

È possibile impostare le proprietà del server fino a quando ServerFactory.getInstance non viene chiamata per la prima volta. La prima chiamata del metodo getInstance crea un'istanza del server eXtreme Scale e legge tutte le proprietà configurate. L'impostazione delle proprietà dopo la creazione non ha alcun effetto. L'esempio seguente mostra come impostare le proprietà prima di creare un'istanza Server.

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

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

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

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

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

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

// Turn off all MBeans for this process.
serverProperties.setMBeansEnabled(false);

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

Inserimento del servizio catalogo

Qualsiasi impostazione JVM su cui è stato definito un indicatore da parte del metodo CatalogServerProperties.setCatalogServer può ospitare il servizio catalogo per eXtreme Scale. Questo metodo indica al runtime del server eXtreme Scale di creare un'istanza del servizio catalogo all'avvio del server. Il seguente codice mostra come creare un'istanza del server di catalogo di eXtreme Scale.

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

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

Inserimento del contenitore eXtreme Scale

Immettere il metodo Server.createContainer per consentire a qualsiasi JVM di ospitare più contenitori eXtreme Scale. Il seguente codice mostra come creare un'istanza ad un contenitore eXtreme Scale:

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

Processo server autonomo

È possibile avviare tutti i servizi contemporaneamente, il che è utile per lo sviluppo pratico in produzione. Avviando tutti i servizi contemporaneamente, un singolo processo effettuerà quanto segue: avvia il servizio catalogo, avvia una serie di contenitori ed esegue la logica di connessione del client. L'avvio dei servizi in questa modalità risolverà i problemi di programmazione prima della distribuzione in un ambiente distribuito. Il seguente codice mostra come creare un'istanza ad un server eXtreme Scale autonomo:

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

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

Inserimento di un eXtreme Scale in WebSphere Application Server

La configurazione di eXtreme Scale è impostata automaticamente al momento dell'installazione di WebSphere Extended Deployment DataGrid in un ambiente WebSphere Application Server. Non è necessario impostare alcuna proprietà prima di accedere il server per creare un contenitore. Il seguente codice mostra come creare un'istanza ad un server eXtreme Scale in WebSphere Application Server:

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

Per un esempio dettagliato dell'avvio di un servizio catalogo incorporato ed di un contenitore in modo programmatico, consultare “Utilizzo delle API server incorporate”.

Utilizzo delle API server incorporate

Con WebSphere eXtreme Scale, è possibile utilizzare un'API programmatica per gestire il ciclo di vita di contenitori e server incorporati. È possibile configurare in modo programmatico il server con le opzioni che è possibile configurare anche con le opzioni della riga comandi o le proprietà del server basate su file. È possibile configurare il server incorporato come server contenitore, servizio catalogo o entrambi.

Prima di iniziare

È necessario disporre di un metodo per l'esecuzione del codice da una Java virtual machine già esistente. Le classi eXtreme Scale devono essere disponibili nella struttura ad albero del programma di caricamento classe.

Informazioni su questa attività

Con l'API Administration, è possibile eseguire diverse attività di gestione. Un utilizzo comune dell'API è quello di un server interno per la memorizzazione dello stato dell'applicazione Web. Il server Web può avviare un WebSphere eXtreme Scale incorporato, indicare il server contenitore al servizio catalogo ed il server viene aggiunto come membro di una griglia distribuita di dimensioni maggiori. Questo utilizzo può fornire scalabilità e HA (high availability) ad un archivio di dati altrimenti volatile.

È possibile controllare in modo programmatico l'intero ciclo di vita di un server eXtreme Scale incorporato. Gli esempi sono molto generici e mostrano solo esempi di codice diretto per le fasi indicate.

Procedura

1. Ottenere l'oggetto ServerProperties dalla classe ServerFactory e configurare tutte le opzioni necessarie.

   Ciascun server eXtreme Scale dispone di una serie di proprietà configurabili. Quando un server viene avviato dalla riga comandi, tali proprietà sono impostate sui relativi valori predefiniti, ma è possibile sostituire diverse proprietà fornendo un file o un'origine esterni. Nell'ambito incorporato, è possibile impostare direttamente le proprietà con un oggetto ServerProperties. È necessario impostare tali proprietà prima di ottenere un'istanza del server dalla classe ServerFactory. Il frammento di esempio riportato di seguito ottiene un oggetto ServerProperties, imposta il campo CatalogServiceBootStrap ed inizializza diverse altre impostazioni facoltative del server. Consultare la documentazione relativa all'API per un elenco delle impostazioni configurabili.

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

2. Se si desidera che il server sia un servizio catalogo, richiedere l'oggetto CatalogServerProperties.

   Qualsiasi server incorporato può essere un servizio catalogo, un server contenitore o entrambi. Nell'esempio riportato di seguito, viene richiesto l'oggetto CatalogServerProperties, viene abilitata l'opzione del servizio catalogo e vengono configurate diverse impostazioni del servizio catalogo.

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

3. Ottenere un'istanza Server dalla classe ServerFactory. L'istanza Server è un singleton nell'ambito del processo responsabile della gestione dell'appartenenza nella griglia. Una volta creata questa istanza, il processo è connesso ed è altamente disponibile con gli altri server nella griglia. L'esempio riportato di seguito illustra come creare l'istanza Server:

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

   Esaminando l'esempio precedente, la classe ServerFactory fornisce un metodo statico che restituisce un'istanza Server. La classe ServerFactory deve essere l'unica interfaccia per ottenere un'istanza Server. Per questo motivo, la classe garantisce che l'istanza sia un singleton oppure un'istanza per ciascuna JVM o programma di caricamento classe isolato. Il metodo getInstance inizializza l'istanza Server. Prima di inizializzare l'istanza, è necessario configurare tutte le proprietà del server. La classe Server è responsabile della creazione di nuove istanze Container. È possibile utilizzare le classi ServerFactory e Server per gestire il ciclo di vita dell'istanza Server incorporata.

4. Avviare un'istanza Container utilizzando l'istanza Server.

   Prima che sia possibile posizionare i frammenti su un server incorporato, è necessario creare un contenitore sul server. L'interfaccia Server dispone di un metodo createContainer che utilizza un argomento DeploymentPolicy. L'esempio riportato di seguito utilizza l'istanza server ottenuta per creare un contenitore utilizzando un file DeploymentPolicy creato. Notare che Containers richiede un programma di caricamento classe con i binari dell'applicazione disponibili per la serializzazione. È possibile rendere disponibili tali binari richiamando il metodo createContainer con il programma di caricamento classe del contesto Thread impostato sul programma di caricamento classe che si desidera utilizzare.

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

5. Rimuovere e ripulire un contenitore.

   È possibile rimuovere e ripulire un server contenitore eseguendo il metodo teardown sull'istanza Container ottenuta. L'esecuzione del metodo teardown su un contenitore ripulisce il contenitore e lo rimuove dal server incorporato.

   Il processo di ripulitura del contenitore include lo spostamento e e l'eliminazione di tutti i frammenti posizionati all'interno di tale contenitore. Ciascun server può contenere molti contenitori e frammenti. La ripulitura di un contenitore non influisce sul ciclo di vita dell'istanza Server parent. L'esempio riportato di seguito illustra come eseguire il metodo teardown su un server. Il metodo teardown è reso disponibile mediante l'interfaccia ContainerMBean. Utilizzando l'interfaccia ContainerMBean, se non si dispone più di accesso programmatico a tale contenitore, è tuttavia possibile rimuovere e ripulire il contenitore con i relativi MBean. Sull'interfaccia Container è disponibile anche il metodo terminate, che non deve essere utilizzato se non assolutamente necessario. Tale metodo è più potente e non coordina in modo appropriato la ripulitura e lo spostamento del frammento.

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

6. Arrestare il server incorporato.

   Quando un server incorporato viene arrestato, vengono arrestati anche tutti i contenitori ed i frammenti in esecuzione sul server. Quando viene arrestato un server incorporato, è necessario ripulire tutte le connessioni aperte e spostare o

eliminare tutti i frammenti. L'esempio riportato di seguito illustra come arrestare un server e come utilizzare il metodo waitFor sull'interfaccia Server per verificare che l'istanza Server venga chiusa completamente. Come per l'esempio relativo al contenitore, il metodo stopServer è reso disponibile mediante l'interfaccia ServerMBean. Con tale interfaccia, è possibile arrestare un server con il bean gestito (MBean) corrispondente.

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

Di seguito è riportato l'esempio di codice completo:

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

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

public class ServerFactoryTest {

    public static void main(String[] args) {

        try {

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

            /*
             * In most cases, the server will serve as a container server only
             * and will connect to an external catalog service. This is a more
             * highly available way of doing things. The commented code excerpt
             * below will enable this Server to be a catalog service.
             *
             *
             * CatalogServerProperties catalogProps =
             * ServerFactory.getCatalogProperties();
             * catalogProps.setCatalogServer(true); // enable catalog service
             * catalogProps.setQuorum(true); // enable quorum
             */

            Server server = ServerFactory.getInstance();

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

            /*
             * Shard will now be placed on this container if the deployment requirements are met.
             * This encompasses embedded server and container creation.
             *
             * The lines below will simply demonstrate calling the cleanup methods
             */

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

        } catch (ObjectGridException e) {
            // Container failed to initialize
        } catch (MalformedURLException e2) {
            // invalid url to xml file(s)
        }

    }

}
```

Monitoraggio con l'API delle statistiche

L'API delle statistiche è l'interfaccia diretta alla struttura ad albero delle statistiche interne. Per impostazione predefinita, le statistiche sono disabilitate ma possono essere abilitate impostando un'interfaccia StatsSpec. Un'interfaccia StatsSpec definisce il modo in cui WebSphere eXtreme Scale deve monitorare le statistiche.

Informazioni su questa attività

È possibile utilizzare l'API StatsAccessor locale per eseguire query sui dati ed accedere alle statistiche su una qualunque istanza ObjectGrid che si trava nella stessa Java virtual machine (JVM) del codice di esecuzione. Per ulteriori informazioni sulle interfacce specifiche, consultare la documentazione API. Utilizzare i seguenti passi per abilitare il monitoraggio della struttura ad albero delle statistiche interne.

Procedura

1. Richiamare l'oggetto StatsAccessor. L'interfaccia StatsAccessor segue il pattern singleton. Per cui, a parte i problemi correlati a classloader, deve esistere un'istanza StatsAccessor per ogni JVM. Questa classe serve come interfaccia principale per tutte le operazioni di statistiche in locale. Il seguente codice rappresenta un esempio di come richiamare la classe accessor. Chiamare questa operazione prima di qualunque altra chiamata ObjectGrid.

```
public class LocalClient
{

   public static void main(String[] args) {

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

   }

}
```

2. Impostare l'interfaccia StatsSpec della griglia. Impostare questo JVM in modo che raccolga tutte le statistiche solo a livello ObjectGrid. Bisogna assicurarsi che un'applicazione abiliti tutte le statistiche che potrebbero essere necessarie prima di iniziare qualunque transazione. Il seguente esempio imposta l'interfaccia StatsSpec utilizzando sia un campo statico costante che una Stringa spec. L'uso di un campo statico costante è più semplice poiché il campo definisce la specifica. Tuttavia, utilizzando una stringa spec è possibile abilitare una qualunque combinazione delle statistiche richieste.

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

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

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

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

}
```

3. Inviare le transazioni alla griglia per forzare i dati da raccogliere per il monitoraggio. Per raccogliere dati utili per le statistiche, bisogna inviare alla griglia le transazioni. Il seguente codice excerpt inserisce un record nella MapA,

che è in ObjectGridA. Poiché le statistiche si trovano al livello di ObjectGrid, qualunque mappa all'interno di ObjectGrid produce gli stessi risultati.

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

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

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

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

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

4. Eseguire una query di StatsFact utilizzando l'API StatsAccessor. Il percorso di qualunque statistica è associato ad un'interfaccia StatsFact. L'interfaccia StatsFact è un segnaposto generico che viene utilizzato per organizzare e contenere un ogetto StatsModule. Prima di poter accedere al modulo attuale delle statistiche, deve essere richiamato l'oggetto StatsFact.

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

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

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

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

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

      // Retrieve StatsFact

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

}
```

5. Interazione con l'oggetto StatsModule. L'oggetto StatsModule è contenuto all'interno dellinterfaccia StatsFact. Si può ottenere un riferimento al modulo mediante l'interfaccia StatsFact. Poiché l'interfaccia StatsFact è un'interfaccia generica, bisogna eseguire il cast del modulo restituito sul tipo StatsModule previsto. Poiché questa attività raccoglie statistiche eXtreme Scale, viene eseguito il cast dell'oggetto StatsModule restituito su un tipo OGStatsModule. Dopo aver eseguito il cast del modulo, si ha accesso a tutte le statistiche disponibili.

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

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

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

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

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

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

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

}
```

Monitoraggio con WebSphere Application Server PMI

WebSphere eXtreme Scale supporta PMI (Performance Monitoring Infrastructure) quando viene eseguito in un server delle applicazioni WebSphere Application Server o WebSphere Extended Deployment. PMI raccoglie dati delle prestazioni relativi ad applicazioni runtime e fornisce interfacce che supportano applicazioni esterne per il monitoraggio dei dati delle prestazioni. Per accedere ai dati di monitoraggio, è possibile utilizzare la console di gestione oppure lo strumento wsadmin.

Prima di iniziare

È possibile utilizzare PMI per il monitoraggio dell'ambiente quando si utilizza WebSphere eXtreme Scale in combinazione con WebSphere Application Server.

Informazioni su questa attività

WebSphere eXtreme Scale utilizza la funzione PMI personalizzato di WebSphere Application Server per aggiungere la propria strumentazione PMI. Con questo approccio è possibile abilitare e disabilitare PMI di WebSphere eXtreme Scale con la console di gestione o con le interfacce JMX (Java Management Extensions nello strumento wsadmin. Inoltre, è possibile accedere alle statistiche WebSphere eXtreme Scale con le interfacce PMI e JMX standard utilizzate dagli strumenti di monitoraggio, incluso Tivoli Performance Viewer.

Procedura

1. Abilitare PMI di eXtreme Scale. È necessario abilitare PMI per visualizzare le statistiche PMI. Per ulteriori informazioni, consultare la sezione "Abilitazione di PMI" a pagina 293.

2. Richiamare le statistiche PMI di eXtreme Scale. Visualizzare le prestazioni delle applicazioni eXtreme Scale con Tivoli Performance Viewer. Per ulteriori informazioni, consultare la sezione “Recupero statistiche PMI” a pagina 295.

Operazioni successive

Per ulteriori informazioni sullo strumento wsadmin, consultare la sezione “Accesso a MBeans utilizzando lo strumento wsadmin” a pagina 303.

Abilitazione di PMI

È possibile utilizzare WebSphere Application Server PMI (Performance Monitoring Infrastructure) per abilitare statistiche a qualunque livello. Ad esempio, è possibile scegliere di abilitare la statistica della velocità di immissione mappa per una mappa particolare ma non il numero di statistica della voce o la statistica del tempo di aggiornamento in batch del programma di caricamento. È possibile abilitare PMI nella console di gestione o con la programmazione script.

Prima di iniziare

Il proprio server delle applicazioni deve essere avviato e deve avere installata un'applicazione eXtreme Scale abilitata. Per abilitare PMI con la programmazione script, bisogna anche essere in grado di collegarsi e di utilizzare lo strumento wsadmin. Per ulteriori informazioni sullo strumento wsadmin, consultare l'argomento Strumento wsadmin di strumento nel centro informazioni WebSphere Application Server.

Informazioni su questa attività

Utilizzare PMI di WebSphere Application Server per fornire un meccanismo granulare con il quale abilitare o disabilitare le statistiche a qualunque livello. Ad esempio, è possibile scegliere di abilitare le statistiche della velocità di immissione mappa per una mappa particolare ma non il numero della voce o le statistiche del tempo di aggiornamento in batch del programma di caricamento. Questa sezione mostra come utilizzare la console di gestione e gli script wsadmin per abilitare PMI ObjectGrid.

Procedura

- **Abilitare PMI nella console di gestione.**
  1. Nella console di gestione, fare clic su **Monitoring and Tuning** → **Performance Monitoring Infrastructure** → *nome_server*.
  2. Verificare che sia selezionata l'abilitazione PMI (Performance Monitoring Infrastructure). Come impostazione predefinita, è abilitata. Se non lo è, selezionare la casella di spunta e riavviare il server.
  3. Fare clic su **Custom** per personalizzare. Nella struttura ad albero della configurazione, selezionare il modulo delle mappe di mObjectGrid e ObjectGrid. Per ogni modulo, abilitare le statistiche.

  La categoria del tipo di transazione per ObjectGrid statistics viene creata al runtime. Si possono vedere solo le sottocategorie delle statistiche di mappa e di ObjectGrid nella scheda **Runtime**.
- **Abilitare PMI con la programmazione script.**
  1. Aprire un prompt di riga comandi. Navigare fino alla directory install_root/bin. Immettere wsadmin per avviare lo strumento di riga comandi wsadmin.

2. Modificare la configurazione runtime di eXtreme Scale PMI. Verificare che PMI sia abilitato per il server utilizzando 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 abilitato, eseguire i seguenti comandi per abilitare PMI:

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

   Se è necessario abilitare PMI, riavviare il server.

3. Impostare le variabili per cambiare la serie statistica in una serie personalizzata utilizzando i seguenti 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 statistica su personalizzata utilizzando il seguente comando:

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

5. Impostare le variabili per abilitare la statistica PMI di objectGridModule utilizzando i seguenti 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 delle statistiche utilizzando il seguente comando:

```
wsadmin>set params2 [java::new {java.lang.Object[]} 2]
wsadmin>$params2 set 0 [java::new java.lang.String mapModule=*]
wsadmin>$params2 set 1 [java::new java.lang.Boolean false]
wsadmin>set sigs2 [java::new {java.lang.String[]} 2]
wsadmin>$sigs2 set 0 java.lang.String
wsadmin>$sigs2 set 1 java.lang.Boolean
```

7. Impostare la stringa delle statistiche utilizzando il seguente comando:

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

Questi passi abilitano eXtreme Scale runtime PMI, ma non modificano la configurazione di PMI. Se si riavvia il server delle applicazioni, le impostazioni PMI vengono perse tranne l'abilitazione PMI.

Esempio

È possibile eseguire i passi di seguito riportati per abilitare le statistiche PMI per l'applicazione di esempio.

1. Lanciare l'applicazione utilizzando l'indirizzo Web di `http://host:port/ObjectGridSample`, dove host e porta sono il nome dell'host e il numero della porta HTTP del server su cui è installato l'esempio.
2. Nella stessa applicazione, fare clic su ObjectGridCreationServlet e poi sui pulsanti 1, 2, 3, 4 e 5 per generare azioni in ObjectGrid e nelle mappe. Non chiudere ora la pagina del servlet.

3. Nella console di gestione, fare clic su **Monitoring and Tuning → Performance Monitoring Infrastructure → *nome_server*** Fare clic sulla scheda **Runtime**.
4. Fare clic sul pulsante **Custom**.
5. Espandere il modulo ObjectGrid Maps nella struttura ad albero del runtime e poi fare clic sul link clusterObjectGrid. Nel gruppo delle mappe di ObjectGrid, esiste un'istanza ObjectGrid denominata clusterObjectGrid e sotto il gruppo clusterObjectGrid esistono quattro mappe: sportelli, impiegati, uffici e siti. Nell'istanza ObjectGrids, esiste l'istanza clusterObjectGrid e sotto quell'istanza c'è un tipo di transazione denominata DEFAULT.
6. È possibile abilitare le statistiche di proprio interesse. Ad esempio, è possibile abilitare il numero di voci delle mappe per la mappa impiegati ed il tempo di risposta della transazione per il tipo di transazione DEFAULT.

Operazioni successive

Dopo aver abilitato PMI, è possibile visualizzare le statistiche PMI con la console di gestione o mediante la programmazione script.

Recupero statistiche PMI

Recuperando le statistiche PMI, è possibile vedere la prestazione delle proprie applicazioni eXtreme Scale.

Prima di iniziare

- Abilitare la tracciatura delle statistiche PMI per il proprio ambiente. Consultare "Abilitazione di PMI" a pagina 293 per ulteriori informazioni.
- I percorsi in questa attività presumono che si stanno recuperando le statistiche per la stessa applicazione, ma è possibile utilizzare queste statistiche per qualsiasi altra applicazione con passi simili.
- Se si sta utilizzando la console di gestione per recuperare le statistiche, si deve essere in grado di effettuare il log nella console di gestione. Se si sta utilizzando la programmazione script, è necessario effettuare il log in wsadmin.

Informazioni su questa attività

È possibile recuperare le statistiche PMI per visualizzare in Tivoli Performance Viewer completando i passi nella console di gestione o con la programmazione script.

- Passi console di gestione
- Passi programmazione script

Per ulteriori informazioni sulle statistiche possono essere recuperate, consultare "Moduli PMI" a pagina 296.

Procedura

- Recuperare le statistiche PMI nella console di gestione.
  1. Nella console di gestione, fare clic su **Monitoring and tuning → Performance viewer → Current activity**
  2. Selezionare il server che si desidera monitorare utilizzando Tivoli Performance Viewer, quindi abilitare il monitoraggio.
  3. Fare clic sul server per visualizzare la pagina di Performance viewer.

4. Espandere la struttura ad albero della configurazione. Fare clic su **Mappe ObjectGrid** → **clusterObjectGrid** selezionare **dipendenti**. Espandere **ObjectGrids** → **clusterObjectGrid** e selezionare **DEFAULT**.
5. Nell'applicazione di esempio ObjectGrid, andare sul servlet ObjectGridCreationServlet, fare clic sul pulsante 1, quindi popolare le mappe. È possibile visualizzare le statistiche nel viewer.

- Recuperare le statistiche PMI con la programmazione script.
  1. Da un prompt della riga comandi, navigare nella directory `install_root/bin`. Immettere `wsadmin` per avviare lo strumento wsadmin.
  2. Impostare le variabili per l'ambiente utilizzando i seguenti 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 mapModule utilizzando i seguenti 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. Ottenere le statistiche mapModule utilizzando il seguente comando:

     ```
     wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params $sigs
     ```
  5. Impostare le variabili per ottenere le statistiche objectGridModule utilizzando i seguenti 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. Ottenere le statistiche objectGridModule utilizzando il seguente comando:

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

Risultati

È possibile visualizzare le statistiche in Tivoli Performance Viewer.

Moduli PMI

È possibile monitorare le prestazioni delle applicazioni con moduli PMI (Performance Monitoring Infrastructure).

objectGridModule

objectGridModule contiene una statistica temporale: il tempo di risposta delle transazioni. Una transazione viene definita come la durata tra la chiamata di metodo Session.begin e quella Session.commit. Viene tenuta traccia di questa durata come tempo di risposta delle transazioni. L'elemento root di objectGridModule, "root", serve come punto di ingresso nelle statistiche WebSphere eXtreme Scale. Questo elemento root dispone degli ObjectGrid come elementi

child, che dispongono di tipi di transazione come relativi elementi child. La statistica del tempo di risposta è associata ad ogni tipo di transazione.

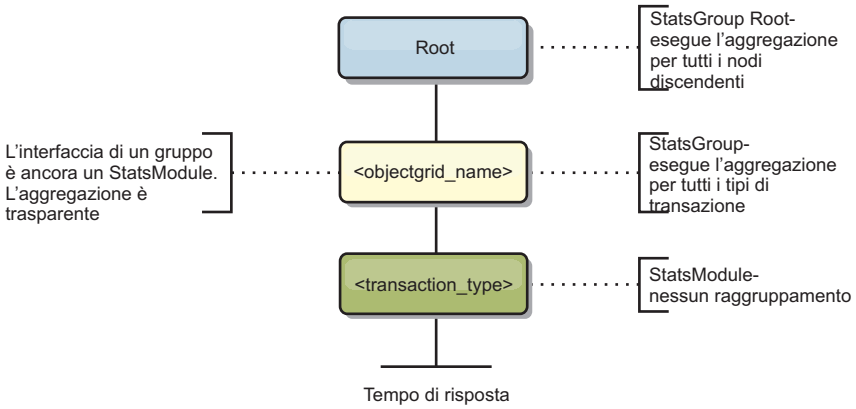


Figura 4. Struttura del modulo ObjectGridModule

Il seguente diagramma mostra un esempio della struttura ObjectGridModule. In questo esempio, esistono due istanze ObjectGrid nel sistema: ObjectGrid A e ObjectGrid B. L'istanza ObjectGrid A ha due tipi di transazioni: A e predefinita. L'istanza ObjectGrid B ha solo il tipo di transazione predefinito.

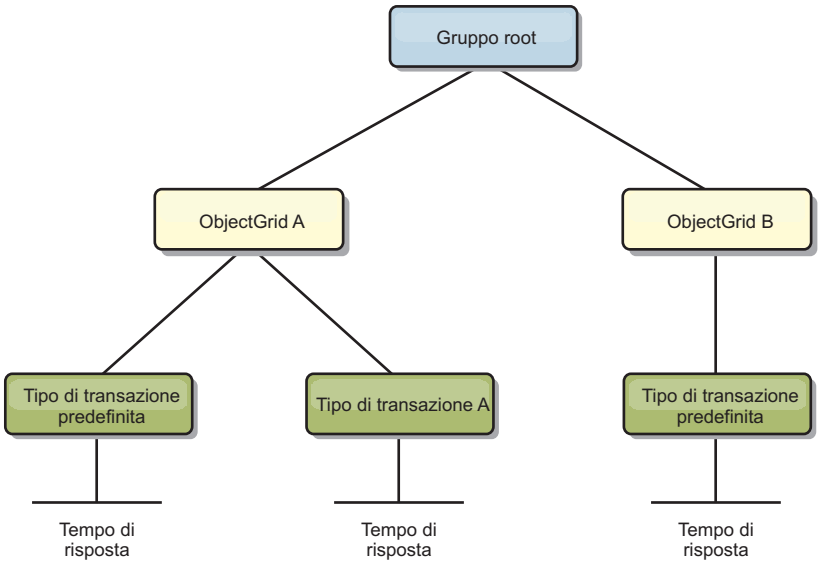


Figura 5. Esempio della struttura del modulo ObjectGridModule

I tipi di transazione vengono definiti dagli sviluppatori dell'applicazione poiché sanno quali tipi di transazioni vengono utilizzati dalle applicazioni. Il tipo di transazione viene impostato utilizzando il seguente metodo Session.setTransactionType(String):

```
/**
* Sets the transaction type for future transactions.
*
* After this method is called, all of the future transactions have the
* same type until another transaction type is set. If  no transaction
* type is set, the default  TRANSACTION_TYPE_DEFAULT transaction type
* is used.
*
* Transaction types are used mainly for statistical data tracking purpose.
```

```
* Users can predefine types of transactions that run in an
* application. The idea is to categorize transactions with the same characteristics
* to one category (type), so one transaction response time statistic can be
* used to track each transaction type.
*
* This tracking is useful when your application has different types of
* transactions.
* Among them, some types of transactions, such as update transactions, process
* longer than other transactions, such as read-only transactions. By using the
* transaction type, different transactions are tracked by different statistics,
* so the statistics can be more useful.
*
* @param tranType the transaction type for future transactions.
*/
void setTransactionType(String tranType);
```

Nel seguente esempio il tipo di transazione viene impostato su `updatePrice`:

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

La prima riga indica che il tipo di transazione successivo è updatePrice. Una statistica updatePrice esiste al di sotto dell'istanza ObjectGrid che corrisponde alla sessione nell'esempio. Utilizzando le interfacce JMX (Java Management Extension) è possibile ottenere il tempo di risposta delle transazioni per le transazioni updatePrice. È possibile anche ottenere la statistica aggregata per tutti i tipi di transazioni nell'istanza ObjectGrid specificata.

mapModule

mapModule contiene tre statistiche correlate alle mappe eXtreme Scale:

- **Percentuale di corrispondenza mappa** - *StatisticaIntervalloAssociato*: tiene traccia della percentuale di corrispondenza di una mappa. La percentuale di corrispondenza è un valore float compreso tra 0 e 100, che rappresenta la percentuale delle immissioni mappa in relazione alle operazioni get della mappa.
- **Numero di voci**-*StatisticaNumero*: tiene traccia del numero di voci nella mappa.
- **Tempo di risposta per l'aggiornamento batch del programma di caricamento**-*StatisticaTempo*: tiene traccia del tempo di risposta utilizzato per l'operazione di aggiornamento batch del programma di caricamento.

L'elemento root di mapModule, "root", serve come punto di ingresso nelle statistiche della mappa ObjectGrid. Questo elemento root dispone degli ObjectGrid come elementi child, che dispongono di mappe come relativi elementi child. Per ogni istanza della mappa vengono elencate tre statistiche. Nel seguente diagramma viene illustrata la struttura mapModule:

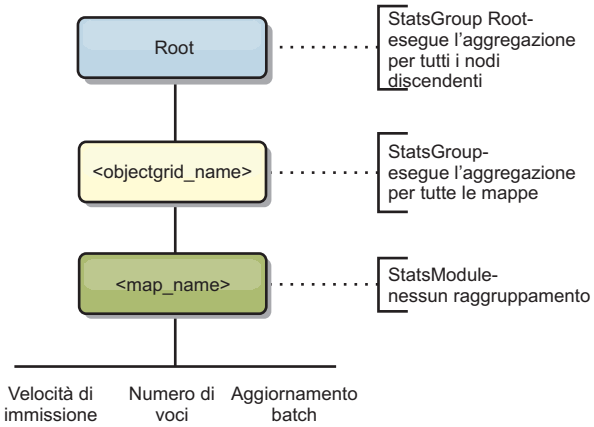
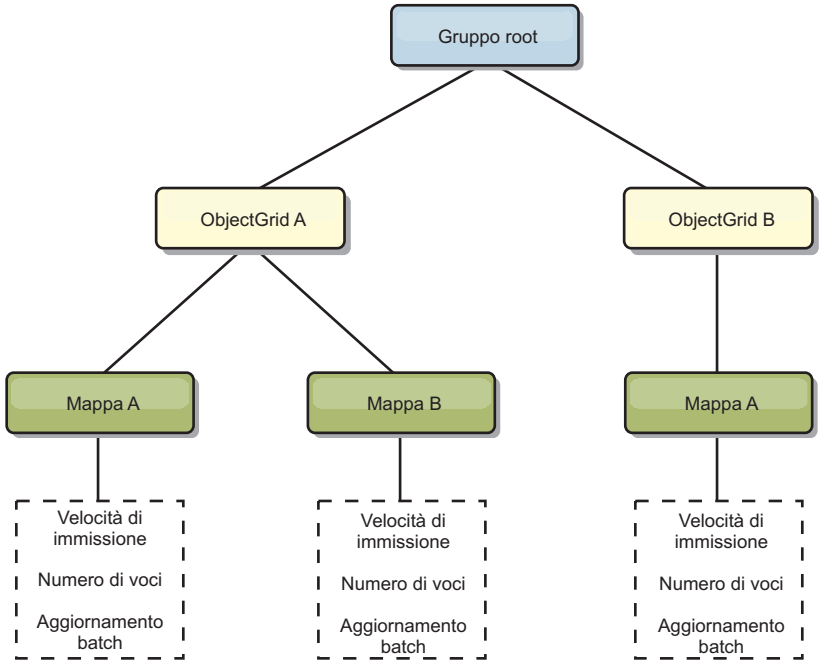


Figura 6. Struttura mapModule

Il seguente diagramma mostra un esempio della struttura mapModule:

Figura 7. Esempio della struttura del modulo mapModule



hashIndexModule

hashIndexModule contiene le seguenti statistiche relative agli indici a livello di mappa:

- **Conteggio rilevamento**-*StatisticaNumero*: il numero di richiami per l'operazione di ricerca dell'indice.
- **Conteggio collisioni**-*StatisticaNumero*: il numero di collisioni per l'operazione di ricerca.
- **Conteggio errori**-*StatisticaNumero*: il numero di errori per l'operazione di ricerca.
- **Conteggio risultati**-*StatisticaNumero*: il numero di chiavi restituite dall'operazione di ricerca.

- **Conteggio BatchUpdate**-*StatisticaNumero*: il numero di aggiornamenti batch eseguiti su questo indice. Quando la mappa corrispondente viene modificata in qualsiasi modo, verrà richiamato il metodo doBatchUpdate() dell'indice. Questa statistica indicherà la frequenza con cui l'indice viene modificato o aggiornato.
- **Tempo di durata dell'operazione di ricerca**-*StatisticaTempo*: la quantità di tempo impiegata dall'operazione di ricerca per il completamento

L'elemento root di hashIndexModule, "root", server come punto di ingresso nelle statistiche HashIndex. L'elemento root dispone degli ObjectGrid come elementi child, gli ObjectGrid dispongono di mappe come elementi child, che infine dispongono di HashIndexe come elementi child e nodi foglia della struttura ad albero. Per ogni istanza HashIndex sono elencate tre statistiche. Nel seguente diagramma viene illustrata la struttura hashIndexModule:

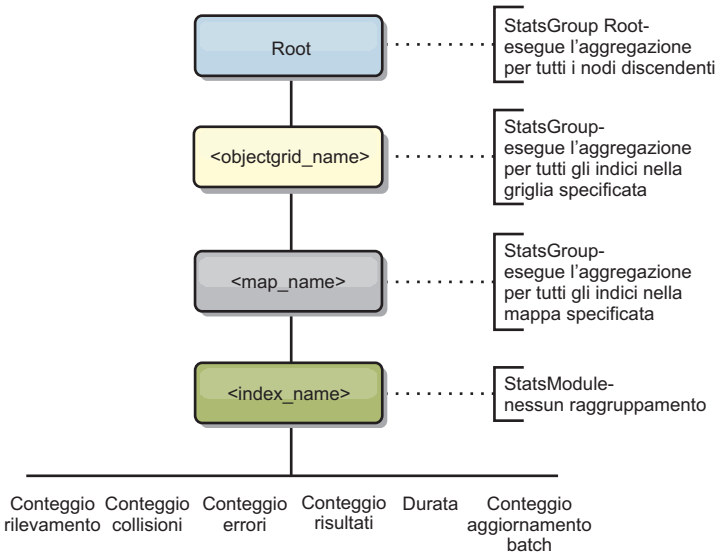


Figura 8. Struttura del modulo hashIndexModule

Il seguente diagramma mostra un esempio della struttura hashIndexModule:

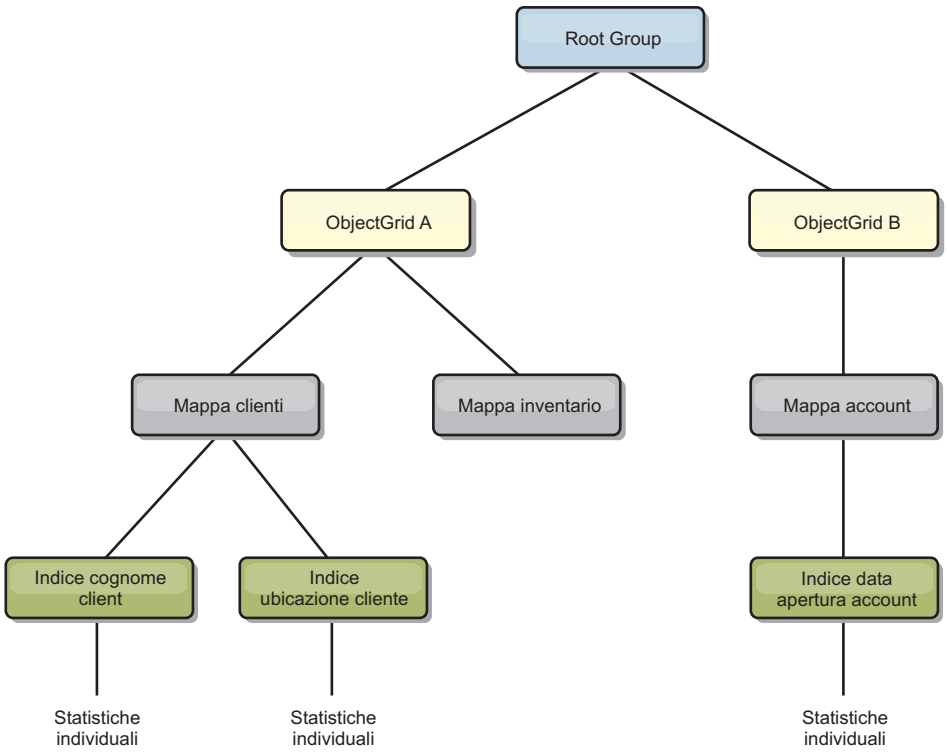


Figura 9. Esempio della struttura del modulo hashIndexModule

agentManagerModule

agentManagerModule contiene le statistiche relative agli agent a livello di mappa:

- **Tempo riduzione**: *StatisticaTempo* - la quantità di tempo per l'agent per terminare l'operazione di riduzione.
- **Tempo di durata totale**: *StatisticaTempo* - la quantità di tempo totale per l'agent per completare tutte le operazioni.
- **Tempo di serializzazione agent**: *StatisticaTempo* - la quantità di tempo per serializzare l'agent.
- **Tempo deserializzazione agent**: *StatisticaTempo* - la quantità di tempo impiegata per deserializzare l'agent sul server.
- **Tempo di serializzazione risultati**: *StatisticaTempo* - la quantità di tempo per serializzare i risultati dall'agent.
- **Tempo di deserializzazione dei risultati**: *StatisticaTempo* - la quantità di tempo per deserializzare i risultati dall'agent.
- **Conteggio errori**: *StatisticaNumero* - il numero di volte che l'agent non funziona correttamente.
- **Conteggio richiamo**: *StatisticaNumero* - il numero di volte che AgentManager è stato richiamato.
- **Conteggio partizioni**: *StatisticaNumero* - il numero di partizioni al quale viene inviato l'agent.

L'elemento root di agentManagerModule, "root", server come punto di ingresso nelle statistiche AgentManager. Questo elemento root dispone degli ObjectGrid come elementi child, gli ObjectGrid dispongono di mappe come elementi child, che infine dispongono di istanze AgentManager come elementi child e nodi foglia della struttura ad albero. Per ogni istanza AgentManager vengono elencate tre

statistiche.

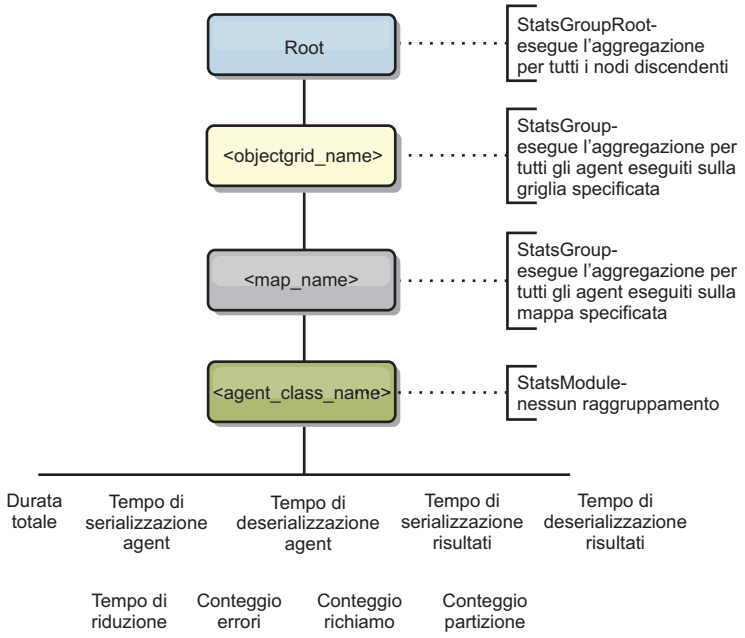


Figura 10. Struttura agentManagerModule

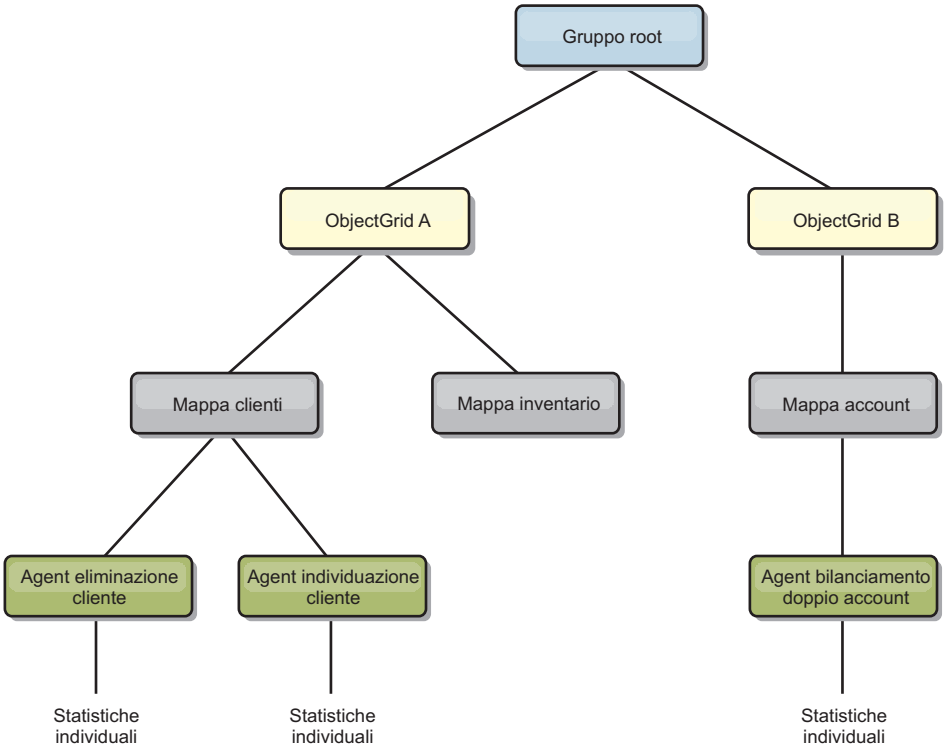


Figura 11. Esempio della struttura agentManagerModule

queryModule

queryModule contiene le statistiche relative alle query eXtreme Scale:

- **Tempo di creazione piano**: *StatisticaTempo* - la quantità di tempo per creare il piano di query.

- **Tempo di esecuzione**: *StatisticaTempo* - la quantità di tempo per eseguire la query.
- **Conteggio esecuzioni**: *StatisticaNumero* - il numero di volte che la query è stata eseguita.
- **Conteggio risultati**: *StatisticaNumero* - il conteggio di ogni serie di risultati di ogni query eseguita.
- **Conteggio errori**: *StatisticaNumero* - il numero di volte in cui la query non è riuscita correttamente.

L'elemento root di queryModule, "root", serve come punto di ingresso delle statistiche Query. Questo elemento root dispone degli ObjectGrid come elementi child, che dispongono di oggetti Query come elementi child e nodi fogli nella struttura ad albero. Per ogni istanza Query sono elencate tre statistiche.

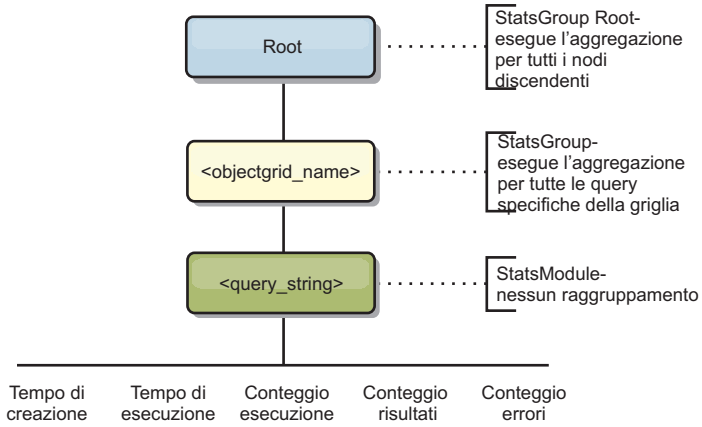


Figura 12. Struttura queryModule

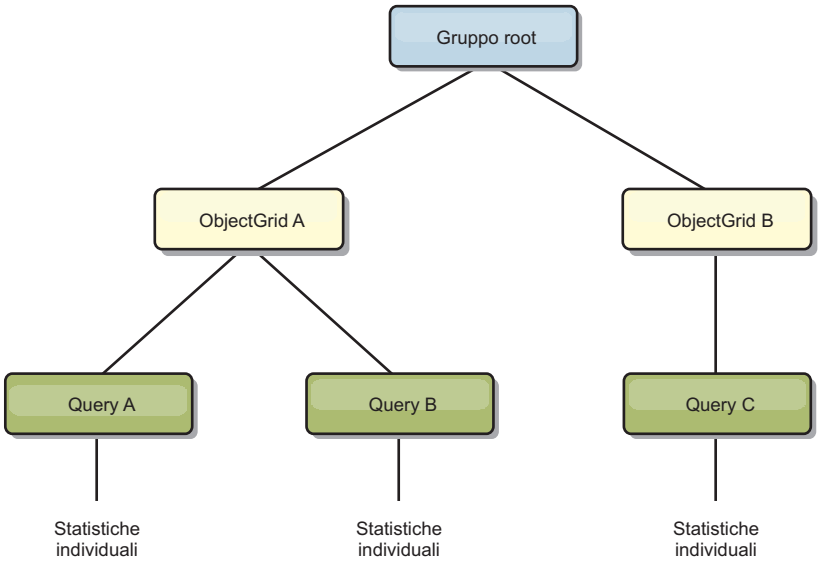


Figura 13. Esempio della struttura queryModule QueryStats.jpg

Accesso a MBeans utilizzando lo strumento wsadmin

È possibile utilizzare l'utilità wsadmin fornita in WebSphere Application Server per accedere alle informazioni MBean.

Eseguire lo strumento wsadmin dalla directory bin nell'installazione di WebSphere Application Server. Il seguente esempio richiama una vista del posizionamento corrente del frammento in eXtreme Scale dinamico. È possibile eseguire wsadmin da qualsiasi installazione in cui sia in esecuzione eXtreme Scale. Non è necessario eseguire wsadmin nel servizio catalogo.

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

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

Capitolo 7. Programmazione per l'integrazione JPA

JPA (Java Persistence API) è una specifica che consente l'associazione di oggetti Java a database relazionali. JPA contiene una specifica ORM (Object-Relational Mapping) completa che utilizza descrittori XML, annotazioni metadati Java o entrambi per definire l'associazione tra gli oggetti Java e un database relazionale. Sono disponibili varie implementazioni open-source e commerciali.

Per utilizzare JPA, è necessario disporre di un provider JPA supportato, come ad esempio OpenJPA o Hibernate, di file JAR e di un file `META-INF/persistence.xml` nel percorso di classe.

Programmi di caricamento JPA

JPA (Java Persistence API) è una specifica che consente l'associazione di oggetti Java a database relazionali. JPA contiene una specifica ORM (Object-Relational Mapping) completa che utilizza descrittori XML, annotazioni metadati Java o entrambi per definire l'associazione tra gli oggetti Java e un database relazionale. Sono disponibili varie implementazioni open-source e commerciali.

È possibile utilizzare un'implementazione del plug-in del programma di caricamento JPA (Java Persistence API) con eXtreme Scale per interagire con qualunque database supportato dal programma di caricamento scelto. Per utilizzare JPA, è necessario disporre di un provider JPA supportato, come ad esempio OpenJPA o Hibernate, di file JAR e di un file `META-INF/persistence.xml` nel percorso di classe.

I plug-in JPALoader com.ibm.websphere.objectgrid.jpa.JPALoader e JPAEntityLoader com.ibm.websphere.objectgrid.jpa.JPAEntityLoader sono due plug-in incorporati del programma di caricamento JPA che vengono utilizzati per sincronizzare le mappe ObjectGrid con un database. Per utilizzare questa funzione, bisogna avere un'implementazione JPA come Hibernate o OpenJPA. Il database può essere qualsiasi backend supportato dal provider JPA scelto.

È possibile utilizzare il plug-in JPALoader durante la memorizzazione di dati utilizzando l'API ObjectMap. Utilizzare il plug-in JPAEntityLoader quando si stanno memorizzando dati utilizzando l'API EntityManager.

Architettura del programma di caricamento JPA

Il programma di caricamento JPA viene utilizzato per le mappe eXtreme Scale che memorizzano POJO Java (plain old Java objects).

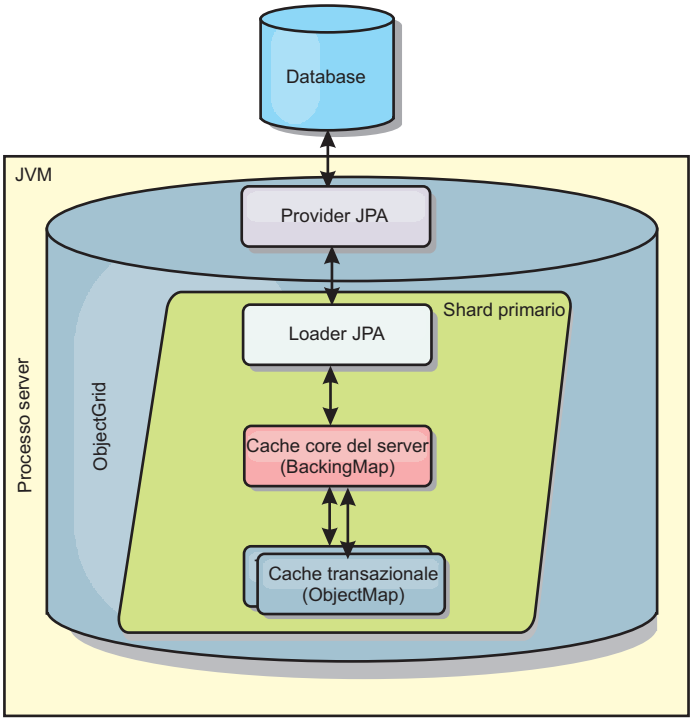


Figura 14. Architettura del programma di caricamento JPA

Quando viene chiamato un metodo ObjectMap.get(Object key), il runtime eXtreme Scale controlla prima se la voce è contenuta nel livello ObjectMap. Se non lo è, il runtime delega la richiesta al programma di caricamento JPA. Su richiesta di caricamento della chiave, JPALoader chiama il metodo EntityManager.find(Object key) JPA per cercare i dati dal livello JPA. Se i dati sono contenuti nel gestore entità JPA, vengono restituiti; altrimenti il provider JPA interagisce con il database per ottenere il valore.

Quando si verifica un aggiornamento di ObjectMap, ad esempio utilizzando il metodo ObjectMap.update(Object key, Object value), il runtime eXtreme Scale crea un LogElement per questo aggiornamento e lo invia a JPALoader. JPALoader chiama il metodo EntityManager.merge(Object value) JPA per aggiornare il valore nel database.

Per JPAEntityLoader, vengono coinvolti tutti e quattro i livelli. Tuttavia, poiché il plug-in JPAEntityLoader viene utilizzato per mappe che memorizzano le entità eXtreme Scalele relazioni tra entità potrebbero complicare gli scenari di utilizzo. Un'entità eXtreme Scale è distinta dall'entità JPA. Per ulteriori informazioni, consultare “Plug-in JPAEntityLoader” a pagina 262.

Metodi

I programmi di caricamento forniscono tre metodi principali:

1. get: restituisce un elenco di valori che corrisponde all'elenco di chiavi che vengono passate mediante il richiamo dei dati che utilizzano JPA. Il metodo utilizza JPA per cercare le entità nel database. Per il plug-in JPALoader, l'elenco restituito contiene un elenco di entità JPA direttamente dall'operazione di ricerca. Per il plug-in JPAEntityLoader, l'elenco restituito contiene tuple di valori dell'entità eXtreme Scale convertite dall'entità JPA.

2. batchUpdate: scrive i dati dalle mappe ObjectGrid nel database. A seconda dei diversi tipi di operazione (insert, update o delete), il programma di caricamento utilizza le operazioni JPA persist, merge e remove per aggiornare i dati nel database. Per JPALoader, gli oggetti nella mappa vengono utilizzati direttamente come entità JPA. Per JPAEntityLoader, le tuple delle entità nella mappa vengono convertite in oggetti che vengono utilizzati come entità JPA.
3. preloadMap: precarica la mappa utilizzando il metodo di caricamento del client ClientLoader.load. Per le mappe partizionate, il metodo preloadMap viene chiamato in una sola partizione. Viene specificata la partizione, la proprietà preloadPartition della classe JPALoader o JPAEntityLoader. Se il valore preloadPartition viene impostato su un valore minore di zero o maggiore di (*numero_totale_di_partizioni* - 1), viene disabilitato il precaricamento.

Entrambi i plug-in JPALoader e JPAEntityLoader funzionano con la classe JPATxCallback per coordinare le transazioni eXtreme Scale e le transazioni JPA. Per poter utilizzare questi due programmi di caricamento, JPATxCallback deve essere configurato nell'istanza ObjectGrid.

Panoramica sul programma di utilità di precaricamento JPA basato su client

Il programma di utilità di precaricamento JPA (Java Persistence API) basato su client carica i dati nelle mappe di backup eXtreme Scale utilizzando una connessione client all'ObjectGrid.

Questa capability può semplificare il caricamento delle mappe eXtreme Scale quando le query eseguite sul database non possono essere suddivise in partizioni. È possibile utilizzare anche un programma di caricamento, come ad esempio un programma di caricamento JPA e tale programma è l'ideale quando i dati possono essere caricati in parallelo.

Il programma di utilità di precaricamento JPA basato su client può utilizzare le implementazioni JPA OpenJPA o Hibernate per caricare l'ObjectGrid da un database. Poiché WebSphere eXtreme Scale non interagisce direttamente con il database o con JDBC (Java Database Connectivity), per caricare l'ObjectGrid è possibile utilizzare qualunque database supportato da OpenJPA o Hibernate.

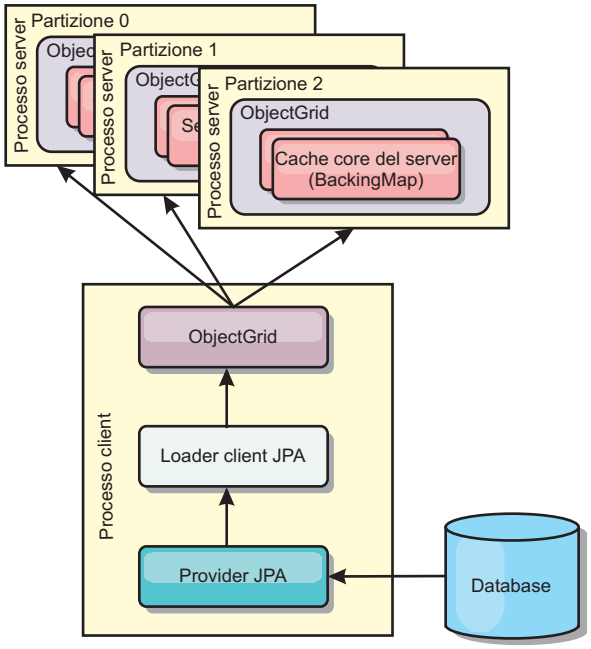


Figura 15. Programma di caricamento del client che utilizza l'implementazione JPA per caricare l'ObjectGrid

Tipicamente, un'applicazione utente fornisce al programma di caricamento del client il nome di un'unità di persistenza, il nome di una classe entità e una query JPA. Il programma di caricamento del client richiama il gestore entità JPA in base al nome dell'unità di persistenza, utilizza il gestore entità per eseguire query sui dati del database con la query JPA e la classe entità fornite e infine carica i dati nelle mappe ObjectGrid distribuite. Quando nella query sono coinvolte relazioni su più livelli, è possibile utilizzare una stringa di query personalizzata per ottimizzare le prestazioni. Facoltativamente, è possibile fornire una mappa di proprietà di persistenza per sostituire le proprietà di persistenza configurate.

Un programma di caricamento del client può caricare i dati in due modi diversi, come illustrato nella seguente tabella:

Tabella 12. Modi del programma di caricamento del client

Modo	**Descrizione**
Precaricamento	Cancella e carica tutte le voci nella mappa di backup. Se la mappa è una mappa di entità, verranno cancellate anche tutte le mappe di entità correlate se è abilitata l'opzione ObjectGrid CascadeType.REMOVE.
Ricaricamento	La query JPA viene eseguita sull'ObjectGrid per invalidare tutte le entità presenti nella mappa che soddisfano la query. Se la mappa è una mappa di entità, verranno cancellate anche tutte le mappe di entità correlate se è abilitata l'opzione ObjectGrid CascadeType.INVALIDATE.

In entrambi i casi, viene utilizzata una query JPA per selezionare e caricare dal database le entità desiderate e memorizzarle nelle mappe ObjectGrid. Se la mappa ObjectGrid è una mappa non di entità, le entità JPA verranno scollegate e memorizzate direttamente. Se la mappa ObjectGrid è una mappa di entità, le entità

JPA vengono memorizzate come tuple di entità ObjectGrid. È possibile fornire una query JPA oppure utilizzare la query predefinita `select o from EntityName o`.

Per ulteriori informazioni sulla configurazione del programma di utilità di precaricamento JPA basato su client, consultare la sezione relativa in *Guida alla programmazione*

Programmazione utilità di precaricamento JPA basata su client

Il programma di utilità di precaricamento JPA (Java Persistence API) basato su client carica i dati in mappe di backup eXtreme Scale utilizzando una connessione client all'ObjectGrid. È possibile implementare il precaricamento e il ricaricamento dati nella propria applicazione.

Utilizzo dell'interfaccia StateManager

Utilizzare il metodo setObjectGridState dell'interfaccia StateManager per impostare lo stato ObjectGrid su uno dei seguenti valori: OFFLINE, ONLINE, QUIESCE o PRELOAD. L'interfaccia StateManager impedisce ad altri client l'accesso all'ObjectGrid quando non è ancora in linea.

Ad esempio, impostare lo stato ObjectGrid su PRELOAD prima di caricare le mappe con dati. Al termine del caricamento dei dati, impostare lo stato ObjectGrid nuovamente su ONLINE. Per ulteriori informazioni, consultare le informazioni sull'impostazione della disponibilità di un ObjectGrid in *Guida alla gestione*.

Quando si esegue il precaricamento di mappe diverse in un ObjectGrid, impostare una volta lo stato ObjectGrid su `PRELOAD` e ripristinare l'impostazione del valore su `ONLINE` dopo che tutte le mappe hanno terminato il caricamento dei dati. Tale coordinamento può essere effettuato dall'interfaccia ClientLoadCallback. Impostare lo stato ObjectGrid su `PRELOAD` dopo la prima notifica preStart dell'istanza ObjectGrid e ripristinare l'impostazione su `ONLINE` dopo l'ultima notifica postFinish.

Se è necessario precaricare le mappe da diverse macchine virtuali Java, occorre coordinare più macchine virtuali Java. Impostare una volta lo stato ObjectGrid su `PRELOAD` prima che venga precaricata la prima mappa su una delle macchine virtuali Java e ripristinare il valore su `ONLINE` dopo che tutte le mappe hanno terminato il caricamento dei dati su tutte le macchine virtuali Java.

Esempio di precaricamento basato su client

Di seguito è riportato il flusso di precaricamento dei dati:

1. Cancellare la mappa da precaricare. Nel caso di una mappa entità, se una qualsiasi relazione viene configurata come cascade-remove, anche le relative mappe vengono cancellate.
2. Eseguire la query su JPA per le entità in un batch. La dimensione batch è 1000.
3. Per ogni batch, generare un elenco di chiavi e un elenco di valori per ogni partizione.
4. Per ciascuna partizione, richiamare l'agent della griglia dati per inserire o aggiornare i dati direttamente sul lato server come se si trattasse di un client eXtreme Scale. Se la griglia è un'istanza locale, inserire o aggiornare direttamente i dati nelle mappe ObjectGrid.

Il seguente frammento di codice di esempio mostra un semplice caricamento client.

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

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

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Load the data
c.load(objectGrid, "CUSTOMER", "custPU", null,
    null, null, null, true, null);

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

In questo esempio, la mappa CUSTOMER viene configurata come mappa entità. La classe entità Customer, che è configurata nel file XML descrittore di metadati dell'entità ObjectGrid, ha una relazione una-a-molti con entità Order. L'entità Customer ha l'opzione CascadeType.ALL abilitata sulla relazione con l'entità Order.

Prima di richiamare il metodo ClientLoader.load, lo stato ObjectGrid viene impostato su PRELOAD.

I parametri utilizzati nel metodo ClientLoader.load sono riportati di seguito:

1. **objectGrid** : l'istanza ObjectGrid. È un'istanza ObjectGrid lato client.
2. **"CUSTOMER"** : la mappa da caricare. Poichè Customer ha una relazione cascade-all con le entità Order, anche le entità Order verranno caricate.
3. **"custPU"** : il nome dell'unità di persistenza JPA per le entità Customer e Order.
4. **null** : la mappa persistenceProps è nulla, indicando che verranno utilizzate le proprietà di persistenza predefinite configurate nel file persistence.xml.
5. **null** : l'elemento entityClass è configurato come nullo. Verrà impostato sulla classe entità configurata nel file XML descrittore di metadati dell'entità ObjectGrid per la mappa "CUSTOMER"; in questo caso, Customer.class.
6. **null** : l'elemento loadSql è nullo, indicando che verrà utilizzata l'istruzione "select o from CUSTOMER o" predefinita per eseguire la query delle entità JPA.
7. **null** : la mappa del parametro query è nulla.
8. **true** : ciò indica che la modalità di caricamento dei dati è preload. Pertanto, verranno richiamate le operazioni di cancellazione per entrambe le mappe CUSTOMER e ORDER per cancellare tutti i dati prima del caricamento dovuto alla relazione cascade-remove tra esse.
9. **null** : ClientLoaderCallback è nullo.

Per ulteriori informazioni sui parametri richiesti, consultare l'API ClientLoader nella documentazione API.

Esempio di ricaricamento

L'operazione di ricaricamento di una mappa è identica all'operazione di precaricamento di una mappa ad eccezione che l'argomento isPreload è impostato su false nel metodo ClientLoader.load.

Nella modalità di ricaricamento, il caricamento del flusso di dati è riportato di seguito:

1. Eseguire la query fornita sulla mappa ObjectGrid e invalidare tutti i risultati. Nel caso di una mappa entità, se una qualsiasi relazione viene configurata con l'opzione CascadeType.INVALIDATE, anche le entità associate vengono invalidate dalle loro mappe.
2. Eseguire la query fornita a JPA per le entità JPA nel batch. La dimensione batch è 1000.
3. Per ogni batch, generare un elenco di chiavi e un elenco di valori per ogni partizione.
4. Per ciascuna partizione, richiamare l'agent della griglia dati per inserire o aggiornare dati direttamente sul lato server come se si trattasse di un client eXtreme Scale. Se la griglia è una configurazione eXtreme Scale locale, inserire o aggiornare direttamente i dati nelle mappe ObjectGrid.

Di seguito è riportato un esempio di ricaricamento:

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

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

ClientLoader c = ClientLoaderFactory.getClientLoader();

// Load the data
String loadSql = "select c from CUSTOMER c
    where c.custId >= :startCustId and c.custId < :endCustId ";
Map<String, Long> params = new HashMap<String, Long>();
params.put("startCustId", 1000L);
params.put("endCustId", 2000L);

c.load(objectGrid, "CUSTOMER", "customerPU", null, null,
    loadSql, params, false, null);

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

Rispetto all'esempio di precaricamento, la differenze principale consiste nella consegna di parametri e un loadSql. Questo esempio ricarica solo i dati Customer con id tra 1000 e 2000.

Si noti che questa stringa query è conforme alla sintassi query JPA e alla sintassi query dell'entità eXtreme Scale. Questa stringa query è importante perché viene eseguita due volte, una in ObjectGrid per invalidare le entità ObjectGrid corrispondenti e infine in JPA per caricare le entità JPA corrispondenti.

Richiamo di un programma di caricamento client in un'implementazione Loader

Nell'interfaccia Loader, vi è un metodo preload:

```
void preloadMap(Session session, BackingMap backingMap) throws
LoaderException;
```

Questo metodo segnala al programma di caricamento di precaricare i dati nella mappa. Un'implementazione del programma di caricamento può utilizzare un programma di caricamento client per precaricare i dati in tutte le sue partizioni. Ad esempio, il programma di caricamento JPA utilizza il programma di caricamento client per precaricare i dati nella mappa.

Per ulteriori informazioni, consultare l'argomento relativo alla panoramica sui programmi di caricamento JPA nella sezione *Panoramica sul prodotto*.

Di seguito è riportato un esempio che descrive il modo in cui precaricare la mappa utilizzando il programma di caricamento client nel metodo preloadMap. L'esempio controlla innanzitutto se il numero di partizione corrente è lo stesso del numero di partizione di precaricamento. Se i numeri non sono corrispondenti, non si verifica alcuna azione. Se i numeri di partizione sono corrispondenti, il programma di caricamento client viene richiamato per caricare i dati nelle mappe. È importante richiamare il programma di caricamento client in una sola e unica partizione.

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

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

    ClientLoader loader = ClientLoaderFactory.getClientLoader();

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

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

Nota: Configurare l'attributo backingMap "preloadMode" su true, affinché il metodo di precaricamento venga eseguito in modo asincrono. Altrimenti, il metodo di precaricamento bloccherà l'attivazione dell'istanza ObjectGrid.

Caricamento client manuale

Il metodo ClientLoader.load fornisce una funzione di caricamento client che soddisfa la maggior parte degli scenari. Tuttavia, se si desidera caricare i dati senza il metodo ClientLoader.load, è possibile implementare il proprio precaricamento.

Di seguito è riportato un template di caricamento client manuale:

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

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


// Load the data
...

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

Se si caricano i dati dal lato client, il caricamento dati mediante un agent DataGrid potrebbe aumentare le prestazioni. Mediante l'utilizzo di un agent DataGrid, tutte le operazioni di lettura e scrittura dati avvengono nel processo del server. È

possibile inoltre progettare le proprie applicazioni per essere certi che gli agent DataGrid in più partizioni vengano eseguiti in parallelo per migliorare ulteriormente le prestazioni.

Per implementare il precaricamento dei dati con un agent DataGrid, vedere il seguente esempio.

Dopo aver creato l'implementazione di precaricamento dei dati, è possibile creare un Loader generico per completare le seguenti attività:

1. Eseguire una query dei dati del database nei batch.
2. Per ciascuna partizione, generare un elenco di chiavi e un elenco di valori.
3. Per ciascuna partizione, chiamare il metodo agentMgr.callReduceAgent(agent, aKey) per eseguire l'agent nel server in un thread. Con l'esecuzione in un thread, è possibile eseguire gli agent simultaneamente su più partizioni.

Esempio: precaricamento dati con un agent DataGrid

Se si caricano i dati dal lato client, il caricamento dati mediante un agent DataGrid potrebbe aumentare le prestazioni. Mediante l'utilizzo di un agent DataGrid, tutte le operazioni di lettura e scrittura dati avvengono nel processo del server. È possibile inoltre progettare le proprie applicazioni per essere certi che gli agent DataGrid in più partizioni vengano eseguiti in parallelo per migliorare ulteriormente le prestazioni.

Di seguito è riportato un esempio che descrive come caricare i dati con un agent DataGrid:

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

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

public class InsertAgent implements ReduceGridAgent, Externalizable {

    private static final long serialVersionUID = 6568906743945108310L;

    private List keys = null;

    private List vals = null;

    protected boolean isEntityMap;

    public InsertAgent() {
    }

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

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

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

}

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

        if (!isEntityMap) {
            // In the POJO case, it is very straightforward,
            // we can just put everything in the
            // map using insert
            insert(m);
        } else {
            // 2. Entity case.
            // In the Entity case, we can persist the entities
            EntityManager em = s.getEntityManager();
            persistEntities(em);

        }
        return Boolean.TRUE;
    } catch (ObjectGridRuntimeException e) {
        throw e;
    } catch (ObjectGridException e) {
        throw new ObjectGridRuntimeException(e);
    } catch (Throwable t) {
        throw new ObjectGridRuntimeException(t);
    }

}

/**
 * Basically this is fresh load.
 * @param s
 * @param m
 * @throws ObjectGridException
```

```
 */
protected void insert(ObjectMap m) throws ObjectGridException {

    int size = keys.size();

    for (int i = 0; i < size; i++) {
        m.insert(keys.get(i), vals.get(i));
    }

}


protected void persistEntities(EntityManager em) {
    Iterator<Object> iter = vals.iterator();

    while (iter.hasNext()) {
        Object value = iter.next();
        em.persist(value);
    }
}

public Object reduceResults(Collection arg0) {
    return arg0;
}

public void readExternal(ObjectInput in)
  throws IOException, ClassNotFoundException {
    int v = in.readByte();
    isEntityMap = in.readBoolean();
    vals = readList(in);
    if (!isEntityMap) {
        keys = readList(in);
    }
}

public void writeExternal(ObjectOutput out) throws IOException {
    out.write(1);
    out.writeBoolean(isEntityMap);

    writeList(out, vals);
    if (!isEntityMap) {
        writeList(out, keys);
    }

}

public void setData(List ks, List vs) {
    vals = vs;
    if (!isEntityMap) {
        keys = ks;
    }
}

/**
 * @return Returns the isEntityMap.
 */
public boolean isEntityMap() {
    return isEntityMap;
}

static public void writeList(ObjectOutput oo, Collection l)
  throws IOException {
    int size = l == null ? -1 : l.size();
    oo.writeInt(size);
    if (size > 0) {
        Iterator iter = l.iterator();
        while (iter.hasNext()) {
```

```
                Object o = iter.next();
                oo.writeObject(o);
            }
        }
    }

    public static List readList(ObjectInput oi)
      throws IOException, ClassNotFoundException {
        int size = oi.readInt();
        if (size == -1) {
            return null;
        }


        ArrayList list = new ArrayList(size);
        for (int i = 0; i < size; ++i) {
            Object o = oi.readObject();
            list.add(o);
        }
        return list;
    }
}
```

Programma di aggiornamento dati JPA basato sull'orario

Un programma di aggiornamento database JPA (Java Persistence API) basato sull'orario aggiorna le mappe ObjectGrid con le modifiche più recenti presenti nel database.

Quando le modifiche vengono apportate direttamente su un database utilizzato da WebSphere eXtreme Scale, queste non vengono contemporaneamente riportate nella griglia eXtreme Scale. Per implementare correttamente eXtreme Scale come uno spazio di elaborazione del database in memoria, prendere in considerazione il fatto che la griglia può evitare la sincronizzazione con il database. Il programma di aggiornamento database basato sull'orario utilizza la capability SCN (System Change Number) in Oracle 10g e la funzione di data/ora di modifica della riga in DB2 9.5 per il monitoraggio delle modifiche presenti nel database per l'invalidazione e l'aggiornamento. Il programma di aggiornamento, inoltre, consente alle applicazioni di disporre di un campo definito dall'utente per lo stesso scopo.

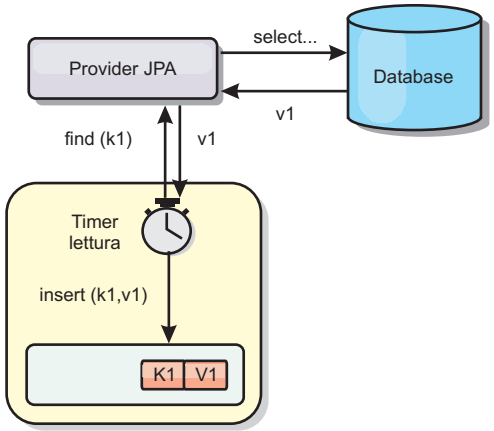


Figura 16. Aggiornamento periodico

Il programma di aggiornamento database basato sull'orario esegue periodicamente query sul database utilizzando le interfacce JPA per ottenere le entità JPA che

rappresentano i record appena inseriti e aggiornati nel database. Per aggiornare periodicamente i record, ciascun record nel database dovrebbe disporre di un identificativo di data/ora per l'orario o la sequenza in cui il record è stato aggiornato o inserito l'ultima volta. Non è necessario che la data/ora sia in un formato data/ora. Il valore di data/ora può essere nel formato intero o esteso, se genera un valore univoco crescente.

Vari database commerciali forniscono questa capability.

Ad esempio, in DB2 9.5, è possibile definire una colonna utilizzando il formato ROW CHANGE TIMESTAMP come di seguito riportato:

```
ROWCHGTS TIMESTAMP NOT NULL
      GENERATED ALWAYS
      FOR EACH ROW ON UPDATE AS
      ROW CHANGE TIMESTAMP
```

In Oracle, è possibile utilizzare la pseudo colonna **ora_rowscn**, che rappresenta il numero SCN (System Change Number) del record.

Il programma di aggiornamento database basato sull'orario aggiorna le mappe ObjectGrid in tre modi differenti:

1. INVALIDATE_ONLY. Le voci nella mappa ObjectGrid vengono invalidata se i record corrispondenti nel database sono stati modificati.
2. UPDATE_ONLY. Le voci nella mappa ObjectGrid vengono aggiornate se i record corrispondenti nel database sono stati modificati. Tuttavia, tutti i record appena inseriti nel database vengono ignorati.
3. INSERT_UPDATE. Le voci esistenti nella mappa ObjectGrid vengono aggiornate con i valori più recenti del database. Inoltre, tutti i record appena inseriti nel database vengono inseriti nella mappa ObjectGrid.

Per ulteriori informazioni sulla configurazione del programma di aggiornamento dati JPA basato sull'orario, consultare le informazioni contenute in *Guida alla gestione*.

Avvio del programma di aggiornamento JPA basato sul tempo

Quando si avvia il programma di aggiornamento JPA basato sul tempo (Java Persistence API), le mappe ObjectGrid vengono aggiornate nel database con le ultime modifiche.

Prima di iniziare

Configurazione del programma di aggiornamento basato sul tempo. Consultare le informazioni sulla configurazione di un programma di aggiornamento dati JPA basato sul tempo in *Guida alla gestione*.

Informazioni su questa attività

Per ulteriori informazioni sul funzionamento del programma di aggiornamento dati basato sul tempo di JPA (Java Persistence API), consultare “Programma di aggiornamento dati JPA basato sull'orario” a pagina 316.

Procedura

- Avviare un programma di aggiornamento database basato sul tempo.
  - **Automaticamente per eXtreme Scale distribuiti:**

Se si crea la configurazione timeBasedDBUpdate per la mappa di backup, il programma di aggiornamento database basato sul tempo viene avviato automaticamente quando viene attivato un frammento primario ObjectGrid distribuito. Per un ObjectGrid con più partizioni, il programma di aggiornamento database basato sul tempo si avvia solo alla partizione 0.

– **Automaticamente per eXtreme Scale:**

  Se si crea la configurazione timeBasedDBUpdate per la mappa di backup, il programma di aggiornamento database basato sul tempo viene avviato automaticamente quando viene attivata la mappa locale.

– **Manualmente:**

  È possibile anche avviare o arrestare manualmente un programma di aggiornamento database basato sul tempo utilizzando l'API TimeBasedDBUpdater API.

```
public synchronized void startDBUpdate(ObjectGrid objectGrid, String mapName,
 String punitName, Class entityClass, String timestampField, DBUpdateMode mode) {
```

  1. **ObjectGrid**: l'istanza ObjectGrid (locale o client).
  2. **mapName**: il nome della mappa di backup per il quale viene avviato il programma di aggiornamentodatabase basato sul tempo.
  3. **punitName**: il nome dell'unità di persistenza JPA per la creazione di una factory del gestore entità JPA definita nel file `persistence.xml`.
  4. **entityClass**: il nome classe dell'entità utilizzata per interagire con il provider JPA (Java Persistence);il nome classe dell'entità viene utilizzata per richiamare le entità JPA utilizzando query di entità.
  5. **timestampField**: un campo timestamp di una classe entità per identificare orario o sequenza in cui è stato aggiornato o inserito un record di back-end del database.
  6. **mode**: la modalità di aggiornamento del database basata sull'orario; un tipo INVALIDATE_ONLY ne comporta l'invalidazione delle voci nella mappa ObjectGrid map se sono cambiati i record corrispondenti nel database; un tipo UPDATE_ONLY indica di aggiornare le voci esistenti nella mappa ObjectGrid con i valori più recenti derivanti dal database; tuttavia, tutti i record appena inseriti nel database vengono ignorati; un tipo INSERT_UPDATE indica di aggiornare le voci esistenti nella mappa ObjectGrid con i valori più recenti derivanti dal database; inoltre, tutti i record appena inseriti nel database vengono inseriti nella mappa ObjectGrid.

  Se si desidera interrompere il programma di aggiornamentodatabase basato sul tempo, è possibile chiamare il seguente metodo per arrestare il programma di aggiornamento:

```
public synchronized void stopDBUpdate(ObjectGrid objectGrid, String mapName)
```

  I parametri ObjectGrid e mapName devono essere uguali a quelli passati nel metodo startDBUpdate.

• Creare il campo timestamp nel proprio database.

  – **DB2**

    Come parte della funzione di blocco ottimistico, DB2 9.5 fornisce una funzione timestamp di modifica della riga. È possibile definire una colonna ROWCHGTS utilizzando il formato ROW CHANGE TIMESTAMP come segue:

```
ROWCHGTS TIMESTAMP NOT NULL
     GENERATED ALWAYS
     FOR EACH ROW ON UPDATE AS
     ROW CHANGE TIMESTAMP
```

Quindi, è possibile indicare il campo entità che corrisponde a questa colonna come campo timestamp mediante annotazione o configurazione. Segue un esempio:

```
@Entity(name = "USER_DB2")
@Table(name = "USER1")
public class User_DB2 implements Serializable {

    private static final long serialVersionUID = 1L;

    public User_DB2() {
    }

    public User_DB2(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ROWCHGTS", updatable = false, insertable = false)
    public Timestamp rowChgTs;
}
```

– **Oracle**

In Oracle, è presente una pseudo colonna ora_rowscn per la modifica di sistema del numero del record. Questa colonna può essere utilizzata per lo stesso scopo. Segue un esempio dell'entità che utilizza il campo ora_rowscn come il campo timestamp di aggiornamento del database basato sul tempo:

```
@Entity(name = "USER_ORA")
@Table(name = "USER1")
public class User_ORA implements Serializable  {

    private static final long serialVersionUID = 1L;

    public User_ORA() {
    }

    public User_ORA(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;
```

```
    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ora_rowscn", updatable = false, insertable = false)
    public long rowChgTs;
}
```

– **Altri database**

  Per altri tipi di database, è possibile creare una colonna di tabella per tener traccia delle modifiche. I valori della colonna devono essere gestiti manualmente dall'applicazione che aggiorna la tabella.

  Prendere come esempio un database Apache Derby: è possibile creare una colonna "ROWCHGTS" per tener traccia dei numeri modificati. Inoltre, viene tenuta traccia dell'ultimo numero modificato in questa tabella. Ogni volta che un record viene inserito o aggiornato, si incrementa il numero di modifica più recente per la tabella ed il valore della colonna ROWCHGTS per il record viene aggiornato con questo numero incrementato.

```
@Entity(name = "USER_DER")
@Table(name = "USER1")
public class User_DER implements Serializable {

    private static final long serialVersionUID = 1L;

    public User_DER() {
    }

    public User_DER(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @Column(name = "ID")
    public int id;

    @Column(name = "FIRSTNAME")
    public String firstName;

    @Column(name = "LASTNAME")
    public String lastName;

    @com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp
    @Column(name = "ROWCHGTS", updatable = true, insertable = true)
    public long rowChgTs;
}
```

Capitolo 8. Programmazione per l'integrazione di Spring

Informazioni sull'integrazione delle applicazioni eXtreme Scale con Spring Framework.

Panoramica sull'integrazione Spring framework

Spring è un noto framework per lo sviluppo di applicazioni Java. WebSphere eXtreme Scale fornisce il supporto per consentire a Spring la gestione delle transazioni eXtreme Scale e la configurazione dei client e dei server che compongono la griglia di dati in memoria distribuita.

Transazioni native gestite da Spring

Spring fornisce transazioni gestite da contenitori che sono simili ad un server delle applicazioni Java Platform, Enterprise Edition. Tuttavia, il meccanismo Spring può collegare implementazioni diverse. WebSphere eXtreme Scale fornisce l'integrazione del gestire transazioni che consente a Spring di gestire i cicli di vita delle transazioni ObjectGrid. Per i dettagli consultare le informazioni relative alle transazioni native in *Guida alla programmazione*.

Bean di estensione gestiti da Spring e supporto spazio dei nomi

eXtreme Scale si integra con Spring per consentire i bean in stile Spring definiti per i punti di estensione o i plug-in. Questa funzione fornisce configurazioni più sofisticate e e una maggiore flessibilità per la configurazione dei punti di estensione.

In aggiunta ai bean di estensione gestiti da Spring, eXtreme Scale fornisce uno spazio dei nomi Spring denominato "objectgrid". I bean e le implementazioni integrate vengono predefiniti in questo spazio dei nomi e questo comporta che è più facile per gli utenti configurare eXtreme Scale. Per ulteriori dettagli su questi argomenti ed un esempio di come avviare un server contenitore eXtreme Scale utilizzando le configurazioni Spring, consultare Bean di estensione Spring e supporto spazio dei nomi.

Supporto dell'ambito shard (frammento)

Con la configurazione Spring in stile tradizionale, un bean ObjectGrid può essere un tipo singleton o un tipo prototipo. ObjectGrid supporta anche un nuovo ambito denominato ambito "shard" (frammento). Se è definito un bean come ambito shard, viene creato un unico bean per frammento. Tutte le richieste di bean con un ID o più ID che corrispondono a quella definizione di bean nello stesso frammento avranno come risultato un'istanza di quel bean specifico restituita dal contenitore Spring.

Il seguente esempio mostra che è definito un bean com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl con l'ambito impostato su shard. Quindi, vene creata una sola istanza della classe JPAPropFactoryImpl per frammento.

```
<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard" />
```

Spring Web Flow

Spring Web Flow memorizza lo stato della sessione nella sessione HTTP per impostazione predefinita. Se un'applicazione Web è configurata per utilizzare eXtreme Scale per la gestione delle sessioni, viene utilizzata automaticamente da Spring per memorizzare lo stato e viene resa tollerante agli errori nello stesso modo della sessione.

Creazione package

Le estensioni eXtreme Scale Spring sono nel file `ogspring.jar`. Questo file JAR (Java archive) deve essere nel percorso di classe, perché il supporto Spring funzioni. Se un'applicazione JEE in esecuzione in un WebSphere Extended Deployment ha potenziato WebSphere Application Server Network Deployment, l'applicazione deve posizionare il file `spring.jar` ed i relativi file associati nei moduli EAR (Enterprise Archive). È necessario inoltre posizionare il file `ogspring.jar` nella stessa ubicazione.

Transazioni gestite da Spring

Spring è un noto framework per lo sviluppo di applicazioni Java. WebSphere eXtreme Scale fornisce supporto per consentire a Spring la gestione delle transazioni eXtreme Scale e la configurazione dei client e dei server eXtreme Scale.

Transazioni native con WebSphere eXtreme Scale

Spring fornisce transazioni gestite dal contenitore con lo stile di un server delle applicazioni Java Platform, Enterprise Edition, ma c'è il vantaggio che con il meccanismo Spring possono essere collegate differenti implementazioni. In questa sezione viene descritto un gestore eXtreme Scale Platform Transaction che può essere utilizzato con Spring. In tal modo si consente ai programmatori di annotare i POJO (Plain Old Java Object) e poi si consente che Spring acquisisca automaticamente le Session da eXtreme Scale e inizi, esegua il commit, il rollback, sospenda e riprenda le transazioni eXtreme Scale. Le transazioni Spring sono descritte con maggiore completezza nel capitolo 10 della documentazione di riferimento Spring ufficiale. Di seguito vengono fornite informazioni sulla modalità di creazione di un gestore transazioni eXtreme Scale e sul suo utilizzo con POJO annotati. Inoltre illustra la modalità di utilizzo di questo approccio con eXtreme Scale locale o client oltre ad un'applicazione dello stile Data Grid collocato.

Creazione di un gestore transazioni

eXtreme Scale fornisce un'implementazione di un PlatformTransactionManager di Spring. Questo gestore può fornire sessioni gestite eXtreme Scale a POJO gestiti da Spring. Tramite l'utilizzo delle annotazioni, Spring gestisce queste sessioni per i POJO in termini di ciclo di vita delle transazioni. Il seguente frammento snippet mostra la modalità di creazione di un gestore transazioni:

```
<aop:aspectj-autoproxy/>
 <tx:annotation-driven transaction-manager="transactionManager"/>

 <bean id="ObjectGridManager"
    class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
    factory-method="getObjectGridManager"/>

 <bean id="ObjectGrid"
    factory-bean="ObjectGridManager"
    factory-method="createObjectGrid"/>

 <bean id="transactionManager"
    class="com.ibm.websphere.objectgrid.spring.ObjectGridSpringFactory"
    factory-method="getLocalPlatformTransactionManager"/>
```

```
</bean>

<bean id="Service" class="com.ibm.websphere.objectgrid.spring.test.TestService">
 <property name="txManager" ref+"transactionManager"/>
</bean>
```

Questo frammento mostra il bean transactionManager che viene dichiarato e collegato in un bean Service che utilizzerà transazioni Spring. Ciò verrà dimostrato con l'utilizzo di annotazioni e questo è il motivo per cui vi è la clausola tx:annotation all'inizio.

Acquisizione di una sessione ObjectGrid per la transazione Spring corrente

Un POJO che possiede metodi gestiti da Spring ora può acquisire la sessione ObjectGrid per la transazione corrente utilizzando

```
Session s = txManager.getSession();
```

Viene restituita la sessione da utilizzare per POJO. I bean che partecipano alla stessa transazione riceveranno la stessa sessione quando richiamano questo metodo. Spring gestirà automaticamente l'inizio della Session e richiamerà automaticamente il commit o il rollback quando necessario. È possibile anche ottenere un'EntityManager ObjectGrid anche semplicemente richimando getEntityManager per l'oggetto Session.

Un esempio POJO utilizzando le annotazioni

Di seguito viene riportato un POJO che utilizza le annotazioni per dichiarare le proprie intenzioni transazionali a Spring. Si può notare che la classe dispone di un'annotazione a livello di classe la quale indica che tutti i metodi devono utilizzare le semantiche di transazione REQUIRED per impostazione predefinita. La classe implementa un'interfaccia con un metodo per tutti i metodi nella classe. Questo è necessario per il funzionamento dell'AOP Spring quando non può eseguire la compilazione in bytecode. La classe dispone di una variabile di istanza txManager che viene collegata al gestore transazioni ObjectGrid mediante il file xml di Spring. Ogni metodo richiama semplicemente il metodo txManager.getSession per ottenere la Session da utilizzare per il metodo. Il metodo queryNewTx viene annotato per indicare una semantica REQUIRES_NEW. Ciò significa che qualsiasi transazione esistente verrà sospesa e verrà creata una nuova transazione indipendente per questo metodo.

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

    public TestService()
    {

    }

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

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

```
    public String query()
        throws ObjectGridException
    {
        Session s = txManager.getSession();
        System.out.println("Query using " + s);
        ObjectMap m = s.getMap("TEST");
        return (String)m.get("Hello");
    }

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

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

    public SpringLocalTxManager getTxManager() {
        return txManager;
    }

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

Impostazione dell'istanza ObjectGrid per un thread

Una singola JVM (Java Virtual Machine) può ospitare molte istanze ObjectGrid. Ogni frammento primario collocato in una JVM possiede la propria istanza ObjectGrid. Una JVM che agisce come un client su un ObjectGrid remoto utilizza un'istanza ObjectGrid restituita dal ClientClusterContext del metodo di connessione per interagire con quella griglia. Prima di richiamare un metodo su un POJO che utilizza transazioni Spring per ObjectGrid, il thread deve essere preparato con l'istanza ObjectGrid da utilizzare. L'istanza TransactionManager possiede un metodo che consente di specificare un'istanza ObjectGrid specifica. Una volta specificata, qualsiasi chiamata txManager.getSession successiva restituirà sessioni per quella istanza ObjectGrid.

Semplice avvio di test

Nel seguente esempio viene mostrato un elemento root di esempio per esercitarsi con questa capability:

```
ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext(new String[]
            {"applicationContext.xml"});
    SpringLocalTxManager txManager = (SpringLocalTxManager)ctx.getBean("transactionManager");
    txManager.setObjectGridForThread(og);

    ITestService s = (ITestService)ctx.getBean("Service");
    s.initialize();
    assertEquals(s.query(), "Billy");
    s.update("Bobby");
    assertEquals(s.query(), "Bobby");
    System.out.println("Requires new test");
    s.testRequiresNew(s);
    assertEquals(s.query(), "1");
```

In questo esempio viene utilizzato un ApplicationContext di Spring. ApplicationContext viene utilizzato per ottenere un riferimento al txManager e specificare un ObjectGrid da utilizzare su questo thread. Il codice quindi ottiene un riferimento al servizio e vi richiama i metodi. Ogni chiamata di metodo a questo livello fa in modo che Spring crei una Session e inizi le chiamate/ne esegua il commit sulla chiamata di metodo. Qualsiasi eccezione provocherà un rollback.

Interfaccia SpringLocalTxManager

L'interfaccia SpringLocalTxManager è implementata da ObjectGrid Platform Transaction Manager e dispone di tutte interfacce pubbliche. I metodi su questa interfaccia sono per la selezione dell'istanza ObjectGrid da utilizzare su un thread e per ottenere una Session per il thread. Qualsiasi POJO che utilizza le transazioni ObjectGrid locali deve essere inserito con un riferimento a questa istanza del gestore e deve essere creata solo una singola istanza, cioè, il suo ambito deve essere singleton. Questa istanza viene creata utilizzando un metodo statico su ObjectGridSpringFactory. getLocalPlatformTransactionManager().

eXtreme Scale per JTA e transazioni globali

eXtreme Scale non supporta JTA o un commit a 2 fasi per vari motivi, principalmente a causa della scalabilità. Pertanto, tranne che per un ultimo partecipante di una fase singola, ObjectGrid non interagisce in transazioni globali di tipo XA o JTA. Questo gestore transazioni è progettato per rendere l'utilizzo delle transazioni ObjectGrid locali il più semplice possibile per gli sviluppatori Spring.

Bean di estensione gestito Spring

È possibile dichiarare i POJO da utilizzare come punti di estensione nel file `objectgrid.xml`. Se si denominano i bean e poi si specificano i nomi classe, eXtreme Scale normalmente crea le istanze della classe specificata e utilizza tali istanze come plug-in. eXtreme Scale può ora delegare Spring ad agire come bean factory per ottenere le istanze di questi oggetti plug-in.

Se un'applicazione utilizza Spring, in genere tali POJO necessitano di essere collegati al resto dell'applicazione.

Un'applicazione può registrare un'istanza Bean Factory di Spring da utilizzare per un ObjectGrid denominato specifico. L'applicazione deve creare un'istanza di BeanFactory o un contesto dell'applicazione Spring e quindi registrarlo con ObjectGrid utilizzando il seguente metodo statico:

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

Questo metodo specifica che se ObjectGrid individua un bean di estensione (come, ad esempio, ObjectTransformer, Loader, TransactionCallback, etc...) il cui className inizia con il prefisso {spring}, utilizzare la parte restante del nome come un nome Spring Bean e ottenere l'istanza bean utilizzando il Bean Factory di Spring. ObjectGrid può anche creare un bean factory di Spring da un file di configurazione xml Spring predefinito. Se non è stato registrato alcun bean factory per un determinato ObjectGrid, ObjectGrid cerca di trovare un file xml denominato 'ObjectGridName'_spring.xml. Ad esempio, se la griglia viene denominata GRID il file xml viene denominato '/GRID_spring.xml' e deve trovarsi nello stesso percorso di classe del package principale. Se questo file viene trovato, ObjectGrid costruisce un ApplicationContext che utilizza quel file e costruisce bean dal quel bean factory. Ad esempio, il nome classe sarà:

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

Ciò indurrà ObjectGrid a richiedere a Spring un bean denominato "MyLoaderBean". Questo approccio può essere utilizzato per specificare POJO gestiti da Spring per qualsiasi punto di estensione in ObjectGrid purché il bean factory sia stato registrato fin dall'inizio. Le estensioni spring ObjectGrid si trovano nel file ogspring.jar. Questo file jar deve trovarsi nello stesso percorso di classe affinché il supporto spring funzioni come dovuto. Se l'applicazione JavaEE che viene eseguita in un ND ingrandito con XD, l'applicazione deve posizionare il file spring.jar e i relativi file associati nei moduli EAR. Anche ogspring.jar deve essere posizionato nella stessa ubicazione.

Bean di estensione Spring e supporto spazio dei nomi

WebSphere eXtreme Scale fornisce una funzione per dichiarare i POJO (Plain Old Java Object) da utilizzare come punti di estensione nel file `objectgrid.xml` e un modo per denominare i bean e specificare il nome classe. In genere, vengono create istanze della classe specificata e quegli oggetti vengono utilizzati come plug-in. Ora eXtreme Scale può delegare Spring per l'ottenimento di istanze di quegli oggetti plug-in. Se un'applicazione utilizza Spring, in genere quei POJO necessitano di essere collegati al resto dell'applicazione.

In alcuni casi è necessario utilizzare Spring per configurare determinati oggetti plug-in. Prendere da esempio la seguente configurazione:

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
        <property name="persistenceUnitName" type="java.lang.String"  value="employeePU" />
    </bean>
    ...
</objectGrid>
```

L'implementazione integrata TransactionCallback, classe com.ibm.websphere.objectgrid.jpa.JPATxCallback, è configurata come classe TransactionCallback. Questa classe è configurata con la proprietà persistenceUnitName come mostrato nell'esempio precedente. Anche la classe JPATxCallback ha l'attributo JPAPropertyFactory, che è di tipo java.lang.Object. La configurazione dell'XML ObjectGrid non può supportare questo tipo di configurazione.

L'integrazione di eXtreme Scale Spring risolve questo problema delegando la creazione dei bean al framework Spring. Segue la configurazione corretta:

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="{spring}jpaTxCallback"/>
    ...
</objectGrid>
```

Il file spring per l'oggetto "Grid" contiene le seguenti informazioni:

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.
JPAPropFactoryImpl" scope="shard">
</bean>
```

Qui TransactionCallback viene specificato come `{spring}jpaTxCallback`, e i bean jpaTxCallback e jpaPropFactory vengono configurati nel file Spring come mostrato nell'esempio precedente. La configurazione di Spring rende possibile configurare un bean JPAPropertyFactory come parametro dell'oggetto JPATxCallback.

Bean factory di Spring predefinito

Quando eXtreme Scale trova un plug-in o un bean di estensione (ad esempio ObjectTransformer, Loader, TransactionCallback e così via) con il valore di classname che inizia con il prefisso `{spring}`, eXtreme Scale utilizza la parte restante del nome come nome Spring Bean ed ottiene l'istanza bean utilizzando il Bean Factory di Spring.

Per impostazione predefinita, se non è stato registrato un bean factory per un determinato ObjectGrid, prova a cercare un file `ObjectGridName_spring.xml`. Ad esempio, se la griglia è denominata "Grid" il file XML è denominato `/Grid_spring.xml`. Questo file deve essere nel percorso di classe o in una directory `META-INF` contenuta nel percorso di classe. Se il file viene individuato, eXtreme Scale costruisce un ApplicationContext utilizzando quel file e crea i bean da quel bean factory.

Bean factory di Spring personalizzato

WebSphere eXtreme Scale fornisce anche un'API ObjectGridSpringFactory per registrare un'istanza Bean Factory di Spring da utilizzare per un ObjectGrid denominato specifico. Questa API registra un'istanza di BeanFactory con eXtreme Scale utilizzando il seguente metodo statico:

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

Supporto spazio dei nomi

Fin dalla versione 2.0, Spring ha un meccanismo per le estensioni basate su schemi al formato XML Spring di base per definire e configurare i bean. ObjectGrid utilizza questa nuova funzione per definire e configurare i bean ObjectGrid. Con l'estensione dello schema XML Spring, alcune delle implementazioni integrate di plug-in di eXtreme Scale ed alcuni dei bean ObjectGrid sono definiti in precedenza nello spazio dei nomi "objectgrid". Quando si scrivono file di configurazione Spring, non è necessario specificare il nome classe completo delle implementazioni incorporate. È possibile invece fare riferimento ai bean definiti in precedenza.

Inoltre, con gli attributi dei bean definiti nello schema XML, è meno probabile che venga fornito un nome attributo non corretto. La convalida XML basata sullo schema XML può rilevare questo tipo di errori all'inizio del ciclo di sviluppo.

Tali bean definiti nelle estensioni schema XML sono:

- transactionManager
- register
- server
- catalog
- catalogServerProperties
- container
- JPALoader
- JPATxCallback
- JPAEntityLoader
- LRUEvictor
- LFUEvictor

- HashIndex

Questi bean sono definiti nello schema XML objectgrid.xsd. Questo file XSD viene incluso come file `com/ibm/ws/objectgrid/spring/namespace/objectgrid.xsd` nel file `ogspring.jar`. Per descrizioni dettagliate del file XSD e dei bean definiti nel file XSD, vedere le informazioni relative al file descrittore Spring in *Guida alla gestione*.

Utilizzare ancora l'esempio di JPATxCallback della sezione precedente. Nella sezione precedente, il bean JPATxCallback è configurato nel modo seguente:

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard">
</bean>
```

Utilizzando questa funzione spazio dei nomi, la configurazione XML spring può essere scritta nel modo seguente:

```
<objectgrid:JPATxCallback id="jpaTxCallback" persistenceUnitName="employeeEMPU"
 jpaPropertyFactory="jpaPropFactory" />

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl"
 scope="shard">
</bean>
```

Notare che qui invece di specificare la classe "com.ibm.websphere.objectgrid.jpa.JPATxCallback" come nell'esempio precedente, viene utilizzato direttamente il bean "objectgrid:JPATxCallback" definito in precedenza. Come si può vedere, questa configurazione è meno dettagliata e più agevole per la verifica degli errori.

Avvio del server contenitore con bean di estensione Spring

In questo esempio viene mostrato come avviare un server ObjectGrid utilizzando bean di estensione Spring gestiti di ObjectGrid ed il supporto spazio dei nomi.

File XML ObjectGrid

Innanzitutto definire un file XML ObjectGrid molto semplice che contiene un "Grid" ObjectGrid ed una mappa "Test". L'ObjectGrid ha un plug-in ObjectGridEventListener denominato "partitionListener", e la mappa "Test" ha un plug-in Evictor collegato denominato "testLRUEvictor". Notare che entrambi i plug-in ObjectGridEventListener e Evictor sono configurati utilizzando Spring poiché i loro nomi contengono "{spring}".

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
    <objectGrids>
        <objectGrid name="Grid">
     <bean id="ObjectGridEventListener" className="{spring}partitionListener" />
            <backingMap name="Test" pluginCollectionRef="test" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="test">
            <bean id="Evictor" className="{spring}testLRUEvictor"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

File XML di distribuzione ObjectGrid

Ora, creare un file XML di distribuzione ObjectGrid semplice nel modo seguente. Esegue la suddivisione dell'ObjectGrid in 5 partizioni e non è richiesta alcuna replica.

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numInitialContainers="1" numberOfPartitions="5" minSyncReplicas="0"
            maxSyncReplicas="1" maxAsyncReplicas="0">
            <map ref="Test"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

File XML Spring ObjectGrid

Ora viene utilizzata sia la funzione dei bean di estensione gestiti Spring ObjectGrid che quella del supporto spazio dei nomi per configurare i bean ObjectGrid. Il file XML Spring è denominato "Grid_spring.xml". Notare che nel file XML sono inclusi due schemi: spring-beans-2.0.xsd per utilizzare i bean gestiti Spring e objectgrid.xsd per utilizzare i bean definiti in precedenza nello spazio dei nomi objectgrid.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
       xsi:schemaLocation="
       http://www.ibm.com/schema/objectgrid
   http://www.ibm.com/schema/objectgrid/objectgrid.xsd
       http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <objectgrid:register id="ogregister" gridname="Grid"/>

    <objectgrid:server id="server" isCatalog="true" name="server">
        <objectgrid:catalog host="localhost" port="2809"/>
    </objectgrid:server>

    <objectgrid:container id="container"
   objectgridxml="com/ibm/ws/objectgrid/test/springshard/objectgrid.xml"
       deploymentxml="com/ibm/ws/objectgrid/test/springshard/deployment.xml"
   server="server"/>

    <objectgrid:LRUEvictor id="testLRUEvictor" numberOfLRUQueues="31"/>

    <bean id="partitionListener"
   class="com.ibm.websphere.objectgrid.springshard.ShardListener" scope="shard"/>
</beans>
```

Sono stati definiti 6 bean in questo file XML Spring:

1. *objectgrid:register*: registra il bean factory predefinito per il "Grid" ObjectGrid.
2. *objectgrid:server*: definisce un server ObjectGrid con nome "server". Questo server fornirà anche il servizio catalogo dal momento che contiene un bean objectgrid:catalog nidificato.
3. *objectgrid:catalog*: definisce un endpoint del servizio catalogo ObjectGrid, che è impostato su "localhost:2809".
4. *objectgrid:container*: definisce un contenitore ObjectGrid con il file XML objectgrid e il file XML di distribuzione specificati, come trattato in precedenza. La proprietà server specifica in quale server è ospitato il contenitore.

5. *objectgrid:LRUEvictor*: definisce un LRUEvictor con il numero di code LRU da utilizzare impostato su 31.
6. *bean partitionListener*: definisce un plug-in ShardListener. È necessario fornire un'implementazione per questo plug-in, quindi non può utilizzare i bean definiti in precedenza. Inoltre questo ambito del bean è impostato su "shard", ossia vi è una sola istanza di questo ShardListener per frammento ObjectGrid.

Avvio del server

Il frammento riportato di seguito avvia il server ObjectGrid, che ospita sia il servizio contenitore che il servizio catalogo. Come si può vedere, l'unico metodo che è necessario richiamare per avviare il server è ottenere un bean "container" da un bean factory. Questo semplifica la complessità della programmazione spostando la gran parte della logica nella configurazione di Spring.

```
public class ShardServer extends TestCase
{
    Container container;
    org.springframework.beans.factory.BeanFactory bf;

    public void startServer(String cep)
    {
        try
        {
            bf = new org.springframework.context.support.ClassPathXmlApplicationContext(
                "/com/ibm/ws/objectgrid/test/springshard/Grid_spring.xml", ShardServer.class);
            container = (Container)bf.getBean("container");
        }
        catch(Exception e)
        {
            throw new ObjectGridRuntimeException("Cannot start OG container", e);
        }
    }

    public void stopServer()
    {
        if(container != null)
            container.teardown();
    }
}
```

Capitolo 9. Programmazione per la sicurezza

Utilizzo delle interfacce di programmazione per la gestione di vari aspetti della sicurezza in un ambiente eXtremeScale.

API di sicurezza

WebSphere eXtreme Scale utilizza un'architettura di sicurezza aperta. Esso fornisce un framework di sicurezza base per l'autenticazione, l'autorizzazione e la sicurezza di trasporto, e richiede che gli utenti implementino i plugin per completare l'infrastruttura di sicurezza.

Il seguente immagine mostra il flusso di base dell'autorizzazione e autenticazione client per un server eXtreme Scale.

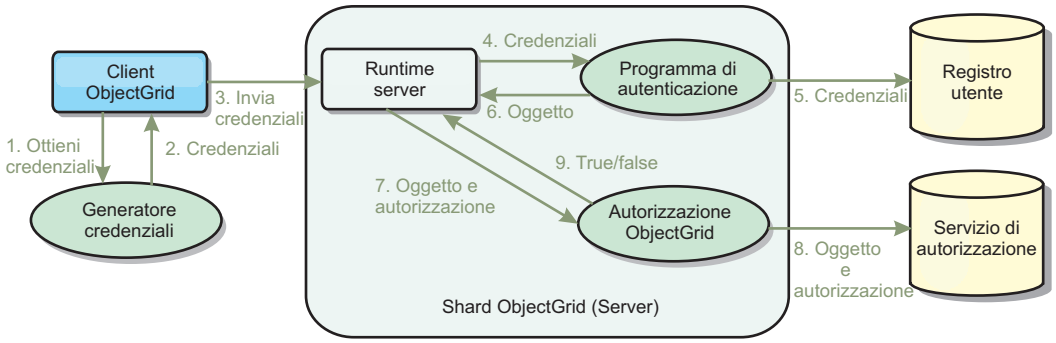


Figura 17. Flusso di autorizzazione e autenticazione client

I flussi di autenticazione e di autorizzazione sono descritti di seguito.

Flusso di autenticazione

1. Il flusso di autenticazione inizia con l'acquisizione di una credenziale da parte del client eXtreme Scale. Tale operazione viene eseguita dal plugin com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator.
2. Un oggetto CredentialGenerator è in grado di generare credenziali client valide, ad esempio, un ID utente e password, un ticket Kerberos e così via. Le credenziali generate vengono restituite al client.
3. Dopo che il client richiama l'oggetto Credential utilizzando l'oggetto CredentialGenerator, l'oggetto richiamato viene inviato insieme alla richiesta eXtreme Scale al server eXtreme Scale server.
4. Il server eXtreme Scale esegue l'autenticazione dell'oggetto Credential prima di elaborare la richiesta eXtreme Scale. Infine, il server utilizza il plugin Authenticator per autenticare l'oggetto Credential.
5. Il plugin Authenticator rappresenta un'interfaccia per il registro utente, ad esempio, un registro utente del sistema operativo o del server LDAP (Lightweight Directory Access Protocol). Il plug-in Authenticator consulta il registro utente e assume decisioni di autenticazione.
6. Se l'autenticazione ha esito positivo, viene restituito un oggetto Subject per rappresentare questo client.

Flusso di autorizzazione

WebSphere eXtreme Scale utilizza un meccanismo basato sull'autorizzazione con diverse categorie di autorizzazione rappresentate da classi di autorizzazione differenti. Ad esempio, un oggetto com.ibm.websphere.objectgrid.security.MapPermission rappresenta le autorizzazioni per leggere, scrivere, inserire, invalidare e rimuovere le voci di dati da un elemento ObjectMap. Poiché WebSphere eXtreme Scale supporta l'autorizzazione JAAS (Java Authentication and Authorization Service) pronta all'utilizzo, è possibile utilizzare JAAS per gestire l'autorizzazione mediante le politiche di autorizzazione.

Inoltre, eXtreme Scale supporta autorizzazioni personalizzate. Le autorizzazioni personalizzate vengono attivate tramite il plugin com.ibm.websphere.objectgrid.security.plugins.ObjectGridAuthorization. Il flusso di autorizzazione del cliente è descritto di seguito.

7. Il runtime del server invia l'oggetto Subject e l'autorizzazione richiesta al plug-in di autorizzazione.
8. Il plug-in di autorizzazione consulta il servizio di autorizzazione e assume una decisione di autorizzazione. Se viene concessa l'autorizzazione per questo oggetto Subject, viene restituito un valore `true`, in caso contrario viene restituito `false`.
9. La decisione di autorizzazione (true o false) viene restituita al runtime del server.

Implementazione della sicurezza

Gli argomenti riportati in questa sezione descrivono la modalità di programmazione di una distribuzione WebSphere eXtreme Scale sicura e la modalità di programmazione delle implementazioni plug-in. La sezione è strutturata in base alle svariate funzioni di sicurezza. In ogni argomento secondario, si comprenderà l'utilizzo dei plugin pertinenti e la relativa modalità di implementazione. Nella sezione di autenticazione, sarà possibile capire come connettersi a un ambiente di distribuzione WebSphere eXtreme Scale protetto.

Autenticazione client: l'argomento di autenticazione client descrive il modo in cui un client WebSphere eXtreme Scale ottiene le credenziali e un server autentica il client. Viene inoltre descritto il modo in cui un client WebSphere eXtreme Scale si connette a un server WebSphere eXtreme Scale protetto.

Autorizzazione: l'argomento di autorizzazione descrive in che modo utilizzare ObjectGridAuthorization per eseguire autorizzazioni personalizzate oltre all'autorizzazione JAAS.

Autenticazione griglia: l'argomento di autenticazione griglia descrive la modalità di utilizzo di SecureTokenManager per trasportare con sicurezza dati riservati del server.

Programmazione JMX (Java Management Extensions): quando il server WebSphere eXtreme Scale viene protetto, il client JMX potrebbe dover inviare credenziali JMX al server.

Programmazione dell'autenticazione del client

Per l'autenticazione, WebSphere eXtreme Scale fornisce un runtime per l'invio delle credenziali dal client al server e quindi chiama il plug-in Authenticator per l'autenticazione degli utenti.

WebSphere eXtreme Scale richiede l'implementazione dei seguenti plug-in per completare l'autenticazione.

- Credential: un Credential rappresenta le credenziali di un utente, quali il binomio ID utente e password.
- CredentialGenerator: un CredentialGenerator rappresenta la factory delle credenziali per la loro generazione.
- Authenticator: un Authenticator autentica le credenziali di un client e ne recupera le informazioni.

Plug-in Credential e CredentialGenerator

Quando un client eXtreme Scale si collega ad un server che richiede l'autenticazione, al client viene richiesto di fornire le proprie credenziali. Le credenziali di un client sono rappresentate da un'interfaccia com.ibm.websphere.objectgrid.security.plugins.Credential. Le credenziali di un client possono essere rappresentate da un binomio nome utente e password, da un ticket Kerberos, da un certificato client o da dati in qualsiasi formato su cui concordano il client e il server. Consultare le informazioni relative alle API Credential nella documentazione delle API per ulteriori informazioni. Questa interfaccia definisce esplicitamente i metodi equals(Object) e hashCode. Questi due metodi sono importanti in quanto gli oggetti Subject autenticati vengono memorizzati nella cache utilizzando l'oggetto Credential come la chiave lato server. WebSphere eXtreme Scale fornisce inoltre un plug-in per la generazione delle credenziali. Questo plug-in viene rappresentato dall'interfaccia com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator ed è utile quando le credenziali possono scadere. In questo caso, il metodo getCredential viene chiamato per rinnovare le credenziali.

L'interfaccia Credential definisce esplicitamente i metodi equals(Object) e hashCode. Questi due metodi sono importanti in quanto gli oggetti Subject autenticati vengono memorizzati nella cache utilizzando l'oggetto Credential come la chiave lato server.

È inoltre possibile utilizzare il plug-in fornito per generare le credenziali. Questo plug-in viene rappresentato dall'interfaccia com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator ed è utile quando le credenziali possono scadere. In questo caso, il metodo getCredential viene chiamato per rinnovare le credenziali. Consultare la documentazione API per ulteriori informazioni.

Vengono fornite tre implementazioni predefinite per le interfacce Credential:

- L'implementazione com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential, che contiene un binomio ID utente e password.
- L'implementazione com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential, che contiene token di autenticazione ed autorizzazione specifici di WebSphere Application Server. Questi token possono essere utilizzati per propagare gli attributi di sicurezza tra i server delle applicazioni nello stesso dominio di sicurezza.

WebSphere eXtreme Scale fornisce inoltre un plug-in per la generazione delle credenziali. Questo plug-in è rappresentato dall'interfaccia com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator . WebSphere eXtreme Scale fornisce due implementazioni incorporate predefinite:

- Il costruttore com.ibm.websphere.objectgrid.security.plugins.builtins. UserPasswordCredentialGenerator acquisisce un ID utente ed una password. Quando viene chiamato il metodo getCredential, esso restituisce un oggetto UserPasswordCredential che contiene l'ID utente e la password.
- Il com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator rappresenta un generatore di credenziali (token di sicurezza) quando è in esecuzione in WebSphere Application Server. Quando viene chiamato il metodo getCredential, viene recuperato il Subject associato al thread corrente. Quindi le informazioni di sicurezza contenute in questo oggetto Subject vengono convertite in un oggetto WSTokenCredential. È possibile specificare se recuperare un soggetto runAs o un soggetto caller dal thread utilizzando la costante WSTokenCredentialGenerator.RUN_AS_SUBJECT o WSTokenCredentialGenerator.CALLER_SUBJECT.

UserPasswordCredential e UserPasswordCredentialGenerator

WebSphere eXtreme Scale fornisce le seguenti implementazioni di plug-in per scopi di test:

1. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential`
2. `com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator`

Le credenziali della password dell'utente memorizzano un ID utente ed una password. Il generatore di credenziali della password dell'utente quindi conterrà questo ID utente e password.

Il seguente codice di esempio mostra come implementare questi due plug-in.

UserPasswordCredential.java

```
// This sample program is provided AS IS and may be used, executed, copied and modified
// without royalty payment by customer
// (a) for its own instruction and study,
// (b) in order to develop applications designed to run with an IBM WebSphere product,
// either for customer's own internal use or for redistribution by customer, as part of such an
// application, in customer's own products.
// Licensed Materials - Property of IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import com.ibm.websphere.objectgrid.security.plugins.Credential;

/**
 * This class represents a credential containing a user ID and password.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Credential
 * @see UserPasswordCredentialGenerator#getCredential()
 */
public class UserPasswordCredential implements Credential {

    private static final long serialVersionUID = 1409044825541007228L;

    private String ivUserName;

    private String ivPassword;

    /**
     * Creates a UserPasswordCredential with the specified user name and
     * password.
     *
     * @param userName the user name for this credential
     * @param password the password for this credential
     *
     * @throws IllegalArgumentException if userName or password is <code>null</code>
     */
    public UserPasswordCredential(String userName, String password) {
        super();
        if (userName == null || password == null) {
            throw new IllegalArgumentException("User name and password cannot be null.");
        }
        this.ivUserName = userName;
```

```
    this.ivPassword = password;
}

/**
 * Gets the user name for this credential.
 *
 * @return   the user name argument that was passed to the constructor
 *           or the <code>setUserName(String)</code>
 *           method of this class
 *
 * @see #setUserName(String)
 */
public String getUserName() {
    return ivUserName;
}

/**
 * Sets the user name for this credential.
 *
 * @param userName the user name to set.
 *
 * @throws IllegalArgumentException if userName is <code>null</code>
 */
public void setUserName(String userName) {
    if (userName == null) {
        throw new IllegalArgumentException("User name cannot be null.");
    }
    this.ivUserName = userName;
}

/**
 * Gets the password for this credential.
 *
 * @return   the password argument that was passed to the constructor
 *           or the <code>setPassword(String)</code>
 *           method of this class
 *
 * @see #setPassword(String)
 */
public String getPassword() {
    return ivPassword;
}

/**
 * Sets the password for this credential.
 *
 * @param password the password to set.
 *
 * @throws IllegalArgumentException if password is <code>null</code>
 */
public void setPassword(String password) {
    if (password == null) {
        throw new IllegalArgumentException("Password cannot be null.");
    }
    this.ivPassword = password;
}

/**
 * Checks two UserPasswordCredential objects for equality.
 * <p>
 * Two UserPasswordCredential objects are equal if and only if their user names
 * and passwords are equal.
 *
 * @param o the object we are testing for equality with this object.
 *
 * @return <code>true</code> if both UserPasswordCredential objects are equivalent.
 *
 * @see Credential#equals(Object)
 */
public boolean equals(Object o) {
    if (this == o) {
        return true;
    }
    if (o instanceof UserPasswordCredential) {
        UserPasswordCredential other = (UserPasswordCredential) o;
        return other.ivPassword.equals(ivPassword) && other.ivUserName.equals(ivUserName);
    }

    return false;
}

/**
 * Returns the hashcode of the UserPasswordCredential object.
 *
 * @return the hash code of this object
 *
 * @see Credential#hashCode()
 */
```

```
    public int hashCode() {
        return ivUserName.hashCode() + ivPassword.hashCode();
    }
}
```

UserPasswordCredentialGenerator.java

```
// This sample program is provided AS IS and may be used, executed, copied and modified
// without royalty payment by customer
// (a) for its own instruction and study,
// (b) in order to develop applications designed to run with an IBM WebSphere product,
// either for customer's own internal use or for redistribution by customer, as part of such an
// application, in customer's own products.
// Licensed Materials - Property of IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.util.StringTokenizer;

import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator;

/**
 * This credential generator creates <code>UserPasswordCredential</code> objects.
 * <p>
 * UserPasswordCredentialGenerator has a one to one relationship with
 * UserPasswordCredential because it can only create a UserPasswordCredential
 * representing one identity.
 *
 * @since WAS XD 6.0.1
 * @ibm-api
 *
 * @see CredentialGenerator
 * @see UserPasswordCredential
 */
public class UserPasswordCredentialGenerator implements CredentialGenerator {

    private String ivUser;

    private String ivPwd;

    /**
     * Creates a UserPasswordCredentialGenerator with no user name or password.
     *
     * @see #setProperties(String)
     */
    public UserPasswordCredentialGenerator() {
        super();
    }

    /**
     * Creates a UserPasswordCredentialGenerator with a specified user name and
     * password
     *
     * @param user the user name
     * @param pwd the password
     */
    public UserPasswordCredentialGenerator(String user, String pwd) {
        ivUser = user;
        ivPwd = pwd;
    }

    /**
     * Creates a new <code>UserPasswordCredential</code> object using this
     * object's user name and password.
     *
     * @return a new <code>UserPasswordCredential</code> instance
     *
     * @see CredentialGenerator#getCredential()
     * @see UserPasswordCredential
     */
    public Credential getCredential() {
        return new UserPasswordCredential(ivUser, ivPwd);
    }

    /**
     * Gets the password for this credential generator.
     *
     * @return   the password argument that was passed to the constructor
     */
    public String getPassword() {
        return ivPwd;
    }

    /**
     * Gets the user name for this credential.
     *
     * @return   the user argument that was passed to the constructor
     *           of this class
     */
    public String getUserName() {
        return ivUser;
    }
```

```
    /**
     * Sets additional properties namely a user name and password.
     *
     * @param properties a properties string  with a user name and
     *                   a password separated by a blank.
     *
     * @throws IllegalArgumentException if the format is not valid
     */
    public void setProperties(String properties) {
        StringTokenizer token = new StringTokenizer(properties, " ");
        if (token.countTokens() != 2) {
            throw new IllegalArgumentException(
                "The properties should have a user name and password and separated by a blank.");
        }

        ivUser = token.nextToken();
        ivPwd = token.nextToken();
    }
    /**
     * Checks two UserPasswordCredentialGenerator objects for equality.
     * <p>
     * Two UserPasswordCredentialGenerator objects are equal if and only if
     * their user names and passwords are equal.
     *
     * @param obj the object we are testing for equality with this object.
     *
     * @return <code>true</code> if both UserPasswordCredentialGenerator objects
     *         are equivalent.
     */
    public boolean equals(Object obj) {
        if (obj == this) {
            return true;
        }

        if (obj != null && obj instanceof UserPasswordCredentialGenerator) {
            UserPasswordCredentialGenerator other = (UserPasswordCredentialGenerator) obj;

            boolean bothUserNull = false;
            boolean bothPwdNull = false;

            if (ivUser == null) {
                if (other.ivUser == null) {
                    bothUserNull = true;
                } else {
                    return false;
                }
            }

            if (ivPwd == null) {
                if (other.ivPwd == null) {
                    bothPwdNull = true;
                } else {
                    return false;
                }
            }

            return (bothUserNull || ivUser.equals(other.ivUser)) && (bothPwdNull || ivPwd.equals(other.ivPwd));
        }

        return false;
    }

    /**
     * Returns the hashcode of the UserPasswordCredentialGenerator object.
     *
     * @return the hash code of this object
     */
    public int hashCode() {

        return ivUser.hashCode() + ivPwd.hashCode();
    }

}
```

La classe UserPasswordCredential contiene due attributi: nome utente e password. UserPasswordCredentialGenerator funge da factory contenente gli oggetti UserPasswordCredential.

WSTokenCredential e WSTokenCredentialGenerator

Quando tutti i client ed i server WebSphere eXtreme Scale sono stati distribuiti in WebSphere Application Server, l'applicazione client potrà utilizzare queste due implementazioni incorporate se sono soddisfatte le seguenti condizioni:

1. La sicurezza globale di WebSphere Application Server è attiva.

2. Tutti i client ed i server di WebSphere eXtreme Scale sono in esecuzione in WebSphere Application Server Java virtual machine.
3. I server delle applicazioni sono nello stesso dominio di sicurezza.
4. Il client è già autenticato in WebSphere Application Server.

In questa situazione, il client può utilizzare la classe com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator per generare le credenziali. Il server utilizza la classe di implementazione WSAuthenticator per autenticare le credenziali.

Questo scenario sfrutta l'avvenuta autenticazione del client eXtreme Scale. Poiché i server delle applicazioni che ospitano i server sono nello stesso dominio di sicurezza dei server delle applicazioni che ospitano i client, i token di sicurezza possono essere propagati dal client al server in modo da non dover autenticare nuovamente lo stesso registro utente.

Nota: non presupporre che un CredentialGenerator generi sempre le stesse credenziali. Per le credenziali che scadono e possono essere aggiornate, il CredentialGenerator deve essere in grado di generare le credenziali valide più recenti per essere sicuro che l'autenticazione vada a buon fine. Un esempio è rappresentato dall'utilizzo del ticket Kerberos come oggetto Credential. Quando il ticket Kerberos viene aggiornato, il CredentialGenerator deve recuperare il ticket aggiornato quando viene chiamato CredentialGenerator.getCredential.

Plug-in Authenticator

Dopo che il client eXtreme Scale recupera l'oggetto Credential tramite l'oggetto CredentialGenerator, tale oggetto Credential del client viene inviato insieme alla richiesta client al server eXtreme Scale. Il server autentica l'oggetto Credential prima di elaborare la richiesta. Se l'oggetto Credential viene autenticato correttamente, un oggetto Subject viene restituito per rappresentare questo client.

Questo oggetto Subject viene quindi memorizzato nella cache e scade dopo che il suo ciclo di vita raggiunge il valore di timeout della sessione. Il valore di timeout dell sessione di login può essere impostato utilizzando la proprietà loginSessionExpirationTime nel file XML del cluster. Ad esempio, l'impostazione di `loginSessionExpirationTime="300"` fa scadere l'oggetto Subject in 300 secondi.

L'oggetto Subject viene quindi utilizzato per autorizzare la richiesta, che viene mostrata successivamente. Un server eXtreme Scale utilizza il plug-in Authenticator per autenticare l'oggetto Credential. Consultare le informazioni relative all'Authenticator nella documentazione API per ulteriori informazioni.

Il plug-in Authenticator si trova dove il runtime di eXtreme Scale autentica l'oggetto Credential proveniente dal registro utente del client, ad esempio, il server LDAP (Lightweight Directory Access Protocol).

WebSphere eXtreme Scale non fornisce una configurazione del registro utente immediatamente disponibile. La configurazione e la gestione del registro utente sono state lasciate fuori da WebSphere eXtreme Scale per motivi di semplicità e flessibilità. Questo plug-in implementa la connessione e l'autenticazione al registro utente. Ad esempio, un'implementazione del programma di autenticazione estrae l'ID utente e la password dalle credenziali, li utilizza per collegarsi ed eseguire la convalida su un server LDAP e crea un oggetto Subject come risultato

dell'autenticazione. L'implementazione potrebbe utilizzare i moduli di login JAAS. Viene restituito un oggetto Subject come risultato dell'autenticazione.

Si noti che questo metodo crea due eccezioni: InvalidCredentialException e ExpiredCredentialException. L'eccezione InvalidCredentialException indica che le credenziali non sono valide. L'eccezione ExpiredCredentialException indica che le credenziali sono scadute. Se dal metodo di autenticazione risulta una di queste due eccezioni, le eccezioni vengono rispedite al client. Tuttavia, il runtime del client gestisce queste due eccezioni in modo diverso:

- Se l'errore è un'eccezione InvalidCredentialException, il runtime del client visualizza questa eccezione. La propria applicazione dovrà gestire l'eccezione. Si potrà correggere il CredentialGenerator, ad esempio e riprovare l'operazione.
- Se l'errore è un'eccezione ExpiredCredentialException ed il conteggio dei tentativi non è 0, il runtime del client chiama nuovamente il metodo CredentialGenerator.getCredential ed invia il nuovo oggetto Credential al server. Se la nuova autenticazione delle credenziali ha esito positivo, il server elabora la richiesta. Se invece non riesce, l'eccezione viene rispedita al client. Se i tentativi di autenticazione raggiungono il valore supportato ed il client continua a ricevere un'eccezione ExpiredCredentialException, ne risulterà un'eccezione ExpiredCredentialException. La propria applicazione dovrà gestire l'errore.

L'interfaccia Authenticator garantisce un'alta flessibilità. L'interfaccia Authenticator può essere implementata come si preferisce. Ad esempio, la si potrà implementare in modo che supporti due diversi registri utente.

WebSphere eXtreme Scale fornisce delle implementazioni del plug-in Authenticator di esempio. Tutte le implementazioni sono solo esempi a scopo di test, tranne quella del plug-in Authenticator di WebSphere Application Server.

KeyStoreLoginAuthenticator

Questo esempio utilizza un'implementazione eXtreme Scale incorporata: KeyStoreLoginAuthenticator a scopo di esempio e test (un keystore è un semplice registro utente e non dovrebbe essere utilizzato in un ambiente di produzione). La classe viene visualizzata per dimostrare ulteriormente come implementare l'Authenticator.

KeyStoreLoginAuthenticator.java

```
// This sample program is provided AS IS and may be used, executed, copied and modified
// without royalty payment by customer
// (a) for its own instruction and study,
// (b) in order to develop applications designed to run with an IBM WebSphere product,
// either for customer's own internal use or for redistribution by customer, as part of such an
// application, in customer's own products.
// Licensed Materials - Property of IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007

package com.ibm.websphere.objectgrid.security.plugins.builtins;

import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

import com.ibm.websphere.objectgrid.security.plugins.Authenticator;
import com.ibm.websphere.objectgrid.security.plugins.Credential;
import com.ibm.websphere.objectgrid.security.plugins.ExpiredCredentialException;
import com.ibm.websphere.objectgrid.security.plugins.InvalidCredentialException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.security.auth.callback.UserPasswordCallbackHandlerImpl;

/**
 * This class is an implementation of the <code>Authenticator</code> interface
 * when a user name and password are used as a credential.
 * <p>
 * When user ID and password authentication is used, the credential passed to the
 * <code>authenticate(Credential)</code> method is a UserPasswordCredential object.
 * <p>
```

```
 * This implementation will use a <code>KeyStoreLoginModule</code> to authenticate
 * the user into the key store using the JAAS login module "KeyStoreLogin". The key
 * store can be configured as an option to the <code>KeyStoreLoginModule</code>
 * class. Please see the <code>KeyStoreLoginModule</code> class for more details
 * about how to set up the JAAS login configuration file.
 * <p>
 * This class is only for sample and quick testing purpose. Users should
 * write your own Authenticator implementation which can fit better into
 * the environment.
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see Authenticator
 * @see KeyStoreLoginModule
 * @see UserPasswordCredential
 */
public class KeyStoreLoginAuthenticator implements Authenticator {

    /**
     * Creates a new KeyStoreLoginAuthenticator.
     */
    public KeyStoreLoginAuthenticator() {
        super();
    }

    /**
     * Authenticates a <code>UserPasswordCredential</code>.
     * <p>
     * Uses the user name and password from the specified UserPasswordCredential
     * to login to the KeyStoreLoginModule named "KeyStoreLogin".
     *
     * @throws InvalidCredentialException if credential isn't a
     *         UserPasswordCredential or some error occurs during processing
     *         of the supplied UserPasswordCredential
     *
     * @throws ExpiredCredentialException if credential is expired.  This exception
     *         is not used by this implementation
     *
     * @see Authenticator#authenticate(Credential)
     * @see KeyStoreLoginModule
     */
    public Subject authenticate(Credential credential) throws InvalidCredentialException,
   ExpiredCredentialException {

        if (credential == null) {
            throw new InvalidCredentialException("Supplied credential is null");
        }

        if (! (credential instanceof UserPasswordCredential) ) {
            throw new InvalidCredentialException("Supplied credential is not a UserPasswordCredential");
        }

        UserPasswordCredential cred = (UserPasswordCredential) credential;
        LoginContext lc = null;
        try {
            lc = new LoginContext("KeyStoreLogin",
                    new UserPasswordCallbackHandlerImpl(cred.getUserName(), cred.getPassword().toCharArray()));

            lc.login();

            Subject subject = lc.getSubject();

            return subject;
        }
        catch (LoginException le) {
            throw new InvalidCredentialException(le);
        }
        catch (IllegalArgumentException ile) {
            throw new InvalidCredentialException(ile);
        }
    }
}
```

KeyStoreLoginModule.java

```
// This sample program is provided AS IS and may be used, executed, copied and modified
// without royalty payment by customer
// (a) for its own instruction and study,
// (b) in order to develop applications designed to run with an IBM WebSphere product,
// either for customer's own internal use or for redistribution by customer, as part of such an
// application, in customer's own products.
// Licensed Materials - Property of IBM
// 5724-J34 © COPYRIGHT International Business Machines Corp. 2007
package com.ibm.websphere.objectgrid.security.plugins.builtins;

import java.io.File;
import java.io.FileInputStream;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.UnrecoverableKeyException;
```

```
import java.security.cert.Certificate;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.Arrays;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import javax.security.auth.x500.X500Principal;
import javax.security.auth.x500.X500PrivateCredential;

import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.ws.objectgrid.Constants;
import com.ibm.ws.objectgrid.ObjectGridManagerImpl;
import com.ibm.ws.objectgrid.util.ObjectGridUtil;

/**
 * A KeyStoreLoginModule is keystore authentication login module based on
 * JAAS authentication.
 * <p>
 * A login configuration should provide an option "<code>keyStoreFile</code>" to
 * indicate where the keystore file is located. If the <code>keyStoreFile</code>
 * value contains a system property in the form, <code>${system.property}</code>,
 * it will be expanded to the value of the system property.
 * <p>
 * If an option "<code>keyStoreFile</code>" is not provided, the default keystore
 * file name is <code>"${java.home}${/}.keystore"</code>.
 * <p>
 * Here is a Login module configuration example:
 * <pre><code>
 *    KeyStoreLogin {
 *        com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule required
 *            keyStoreFile="${user.dir}${/}security${/}.keystore";
 *    };
 * </code></pre>
 *
 * @ibm-api
 * @since WAS XD 6.0.1
 *
 * @see LoginModule
 */
public class KeyStoreLoginModule implements LoginModule {

    private static final String CLASS_NAME = KeyStoreLoginModule.class.getName();

    /**
     * Key store file property name
     */
    public static final String KEY_STORE_FILE_PROPERTY_NAME = "keyStoreFile";

    /**
     * Key store type. Only JKS is supported
     */
    public static final String KEYSTORE_TYPE = "JKS";

    /**
     * The default key store file name
     */
    public static final String DEFAULT_KEY_STORE_FILE = "${java.home}${/}.keystore";

    private CallbackHandler handler;

    private Subject subject;

    private boolean debug = false;

    private Set principals = new HashSet();

    private Set publicCreds = new HashSet();

    private Set privateCreds = new HashSet();

    protected KeyStore keyStore;

    /**
     * Creates a new KeyStoreLoginModule.
     */
    public KeyStoreLoginModule() {
    }

    /**
     * Initializes the login module.
     *
```

```
 * @see LoginModule#initialize(Subject, CallbackHandler, Map, Map)
 */
public void initialize(Subject sub, CallbackHandler callbackHandler,
        Map mapSharedState, Map mapOptions) {

    // initialize any configured options
    debug = "true".equalsIgnoreCase((String) mapOptions.get("debug"));

    if (sub == null)
        throw new IllegalArgumentException("Subject is not specified");

    if (callbackHandler == null)
        throw new IllegalArgumentException(
        "CallbackHander is not specified");

    // Get the key store path
    String sKeyStorePath = (String) mapOptions
        .get(KEY_STORE_FILE_PROPERTY_NAME);

    // If there is no key store path, the default one is the .keystore
    // file in the java home directory
    if (sKeyStorePath == null) {
        sKeyStorePath = DEFAULT_KEY_STORE_FILE;
    }

    // Replace the system enviroment variable
    sKeyStorePath = ObjectGridUtil.replaceVar(sKeyStorePath);

    File fileKeyStore = new File(sKeyStorePath);

    try {
        KeyStore store = KeyStore.getInstance("JKS");
        store.load(new FileInputStream(fileKeyStore), null);

        // Save the key store
        keyStore = store;

        if (debug) {
            System.out.println("[KeyStoreLoginModule] initialize: Successfully loaded key store");
        }
    }
    catch (Exception e) {
        ObjectGridRuntimeException re = new ObjectGridRuntimeException(
                "Failed to load keystore: " + fileKeyStore.getAbsolutePath());
        re.initCause(e);
        if (debug) {
            System.out.println("[KeyStoreLoginModule] initialize: Key store loading failed with exception "
                    + e.getMessage());
        }
    }

    this.subject = sub;
    this.handler = callbackHandler;
}

/**
 * Authenticates a user based on the keystore file.
 *
 * @see LoginModule#login()
 */
public boolean login() throws LoginException {

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: entry");
    }

    String name = null;
    char pwd[] = null;

    if (keyStore == null || subject == null || handler == null) {
        throw new LoginException("Module initialization failed");
    }

    NameCallback nameCallback = new NameCallback("Username:");
    PasswordCallback pwdCallback = new PasswordCallback("Password:", false);

    try {
        handler.handle(new Callback[] { nameCallback, pwdCallback });
    }
    catch (Exception e) {
        throw new LoginException("Callback failed: " + e);
    }

    name = nameCallback.getName();
    char[] tempPwd = pwdCallback.getPassword();

    if (tempPwd == null) {
        // treat a NULL password as an empty password
        tempPwd = new char[0];
    }
    pwd = new char[tempPwd.length];
```

```
    System.arraycopy(tempPwd, 0, pwd, 0, tempPwd.length);

    pwdCallback.clearPassword();

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: "
                + "user entered user name: " + name);
    }

    // Validate the user name and password
    try {
        validate(name, pwd);
    }
    catch (SecurityException se) {
        principals.clear();
        publicCreds.clear();
        privateCreds.clear();
        LoginException le = new LoginException(
        "Exception encountered during login");
        le.initCause(se);

        throw le;
    }

    if (debug) {
        System.out.println("[KeyStoreLoginModule] login: exit");
    }
    return true;
}

/**
 * Indicates the user is accepted.
 * <p>
 * This method is called only if the user is authenticated by all modules in
 * the login configuration file. The principal objects will be added to the
 * stored subject.
 *
 * @return false if for some reason the principals cannot be added; true
 *         otherwise
 *
 * @exception LoginException
 *                LoginException is thrown if the subject is readonly or if
 *                any unrecoverable exceptions is encountered.
 *
 * @see LoginModule#commit()
 */
public boolean commit() throws LoginException {
    if (debug) {
        System.out.println("[KeyStoreLoginModule] commit: entry");
    }

    if (principals.isEmpty()) {
        throw new IllegalStateException("Commit is called out of sequence");
    }

    if (subject.isReadOnly()) {
        throw new LoginException("Subject is Readonly");
    }

    subject.getPrincipals().addAll(principals);
    subject.getPublicCredentials().addAll(publicCreds);
    subject.getPrivateCredentials().addAll(privateCreds);

    principals.clear();
    publicCreds.clear();
    privateCreds.clear();

    if (debug) {
        System.out.println("[KeyStoreLoginModule] commit: exit");
    }
    return true;
}

/**
 * Indicates the user is not accepted
 *
 * @see LoginModule#abort()
 */
public boolean abort() throws LoginException {
    boolean b = logout();
    return b;
}

/**
 * Logs the user out. Clear all the maps.
 *
 * @see LoginModule#logout()
 */
public boolean logout() throws LoginException {
```

```
    // Clear the instance variables
    principals.clear();
    publicCreds.clear();
    privateCreds.clear();

    // clear maps in the subject
    if (!subject.isReadOnly()) {
        if (subject.getPrincipals() != null) {
            subject.getPrincipals().clear();
        }

        if (subject.getPublicCredentials() != null) {
            subject.getPublicCredentials().clear();
        }

        if (subject.getPrivateCredentials() != null) {
            subject.getPrivateCredentials().clear();
        }
    }
    return true;
}

/**
 * Validates the user name and password based on the keystore.
 *
 * @param userName user name
 * @param password password
 * @throws SecurityException if any exceptions encountered
 */
private void validate(String userName, char password[])
    throws SecurityException {

    PrivateKey privateKey = null;

    // Get the private key from the keystore
    try {
        privateKey = (PrivateKey) keyStore.getKey(userName, password);
    }
    catch (NoSuchAlgorithmException nsae) {
        SecurityException se = new SecurityException();
        se.initCause(nsae);
        throw se;
    }
    catch (KeyStoreException kse) {
        SecurityException se = new SecurityException();
        se.initCause(kse);
        throw se;
    }
    catch (UnrecoverableKeyException uke) {
        SecurityException se = new SecurityException();
        se.initCause(uke);
        throw se;
    }

    if (privateKey == null) {
        throw new SecurityException("Invalid name: " + userName);
    }

    // Check the certificats
    Certificate certs[] = null;
    try {
        certs = keyStore.getCertificateChain(userName);
    }
    catch (KeyStoreException kse) {
        SecurityException se = new SecurityException();
        se.initCause(kse);
        throw se;
    }

    if (debug) {
        System.out.println("  Print out the certificates:");
        for (int i = 0; i < certs.length; i++) {
            System.out.println("  certificate " + i);
            System.out.println("    " + certs[i]);
        }
    }

    if (certs != null && certs.length > 0) {

        // If the first certificate is an X509Certificate
        if (certs[0] instanceof X509Certificate) {
            try {
                // Get the first certificate which represents the user
                X509Certificate certX509 = (X509Certificate) certs[0];

                // Create a principal
                X500Principal principal = new X500Principal(certX509
                        .getIssuerDN()
                        .getName());
                principals.add(principal);
```

```
                    if (debug) {
                        System.out.println("  Principal added: " + principal);
                    }
                    // Create the certification path object and add it to the
                    // public credential set
                    CertificateFactory factory = CertificateFactory
                        .getInstance("X.509");
                    java.security.cert.CertPath certPath = factory
                        .generateCertPath(Arrays.asList(certs));
                    publicCreds.add(certPath);

                    // Add the private credential to the private credential set
                    privateCreds.add(new X500PrivateCredential(certX509,
                            privateKey, userName));

                }
                catch (CertificateException ce) {
                    SecurityException se = new SecurityException();
                    se.initCause(ce);
                    throw se;
                }
            }
            else {
                // The first certificate is not an X509Certificate
                // We just add the certificate to the public credential set
                // and the private key to the private credential set.
                publicCreds.add(certs[0]);
                privateCreds.add(privateKey);
            }
        }
    }
}
```

Utilizzo del plug-in authenticator LDAP

Si è muniti dell'implementazione predefinita com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator per la gestione dell'autenticazione del nome utente e della password su un server LDAP. Questa implementazione utilizza il modulo di login LDAPLogin per collegare l'utente al server LDAP (Lightweight Directory Access Protocol). Il seguente frammento 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);
    }
}
```

A questo scopo eXtreme Scale fornisce anche un modulo di login com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule. 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 factory di contesto LDAP

Il modulo LDAPLoginModule chiama il metodo com.ibm.websphere.objectgrid.security.plugins.builtins.

LDAPAuthenticationHelper.authenticate. Il seguente frammento di codice mostra come implementare il metodo di autenticazione del LDAPAuthenticationHelper.

```
/**
* Authenticate the user to the LDAP directory.
* @param user the user ID, e.g., uid=xxxxxx,c=us,ou=bluepages,o=ibm.com
* @param pwd the password
*
* @throws NamingException
*/
public String[] authenticate(String user, String pwd)
throws NamingException {
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, factoryClass);
    env.put(Context.PROVIDER_URL, providerURL);
    env.put(Context.SECURITY_PRINCIPAL, user);
    env.put(Context.SECURITY_CREDENTIALS, pwd);
    env.put(Context.SECURITY_AUTHENTICATION, "simple");

    InitialContext initialContext = new InitialContext(env);

    // Look up for the user
    DirContext dirCtx = (DirContext) initialContext.lookup(user);

    String uid = null;
    int iComma = user.indexOf(",");
    int iEqual = user.indexOf("=");
    if (iComma > 0 && iComma > 0) {
        uid = user.substring(iEqual + 1, iComma);
    }
    else {
        uid = user;
    }

    Attributes attributes = dirCtx.getAttributes("");

    // Check the UID
    String thisUID = (String) (attributes.get(UID).get());

    String thisDept = (String) (attributes.get(HR_DEPT).get());

    if (thisUID.equals(uid)) {
        return new String[] { thisUID, thisDept };
    }
    else {
        return null;
    }
}
```

Se l'autenticazione riesce, l'ID e la password sono considerati validi. Quindi il modulo di login riceve le informazioni dell'ID e del reparto da questo metodo di autenticazione. Il modulo di login crea due principal: SimpleUserPrincipal e SimpleDeptPrincipal. È possibile utilizzare il soggetto autenticato per l'autorizzazione di gruppo (in questo caso il reparto è il gruppo) e per l'autorizzazione individuale.

L'esempio che segue mostra la configurazione di un modulo di login utilizzato per collegarsi ad un 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 configurazione precedente, il server LDAP punta a `ldap://directory.acme.com:389/server`. Cambiare questa impostazione con quella del

proprio server LDAP. Questo modulo di login utilizza l'ID e la password forniti per collegarsi al server LDAP. Questa implementazione è unicamente a scopo di test.

Utilizzo del plug-in Authenticator di WebSphere Application Server

eXtreme Scale fornisce inoltre l'implementazione incorporata com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator per utilizzare l'infrastruttura di sicurezza di WebSphere Application Server. Questa implementazione incorporata può essere utilizzata quando si realizzano le seguenti condizioni.

1. La sicurezza globale di WebSphere Application Server è attiva.
2. Tutti i client ed i server eXtreme Scale sono avviati nelle JVM di WebSphere Application Server.
3. Questi server delle applicazioni sono nello stesso dominio di sicurezza.
4. Il client eXtreme Scale è già autenticato in WebSphere Application Server.

Il client può utilizzare la classe com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator per generare delle credenziali. Il server utilizza questa classe di implementazione Authenticator per autenticare le credenziali. Se l'autenticazione del token riesce, viene restituito un oggetto Subject.

Questo scenario sfrutta l'avvenuta autenticazione del client. Poiché i server delle applicazioni che ospitano i server sono nello stesso dominio di sicurezza dei server delle applicazioni che ospitano i client, i token di sicurezza possono essere propagati dal client al server in modo da non dover autenticare nuovamente lo stesso registro utente.

Utilizzo del plug-in Authenticator di Tivoli Access Manager

Tivoli Access Manager viene è molto diffuso come server di sicurezza. È possibile implementare Authenticator utilizzando anche i moduli di login forniti da Tivoli Access Manager.

Per autenticare un utente per Tivoli Access Manager, applicare il modulo di login com.tivoli.mts.PDLoginModule, che richiede che l'applicazione di chiamata fornisca le seguenti informazioni:

1. Un nome principal, specificato come nome breve o come nome X.500 (DN)
2. Una password

Il modulo di login, autentica il principal e restituisce le credenziali Tivoli Access Manager. Il modulo di login prevede che applicazione di chiamata fornisce le seguenti informazioni:

1. Il nome utente, attraverso un oggetto javax.security.auth.callback.NameCallback.
2. La password, attraverso un oggetto javax.security.auth.callback.PasswordCallback.

Quando le credenziali di Tivoli Access Manager vengono recuperate con successo, il JAAS LoginModule crea un Subject ed un PDPrincipal. Non è fornita alcuna autenticazione incorporata per Tivoli Access Manager, poiché essa viene fornita unicamente con il modulo PDLoginModule. Consultare IBM Tivoli Access Manager Authorization Java Classes Developer Reference per ulteriori dettagli.

Connessione sicura a WebSphere eXtreme Scale

Per collegare un client eXtreme Scale ad un server in modo sicuro, è possibile utilizzare qualsiasi metodo di collegamento nell'interfaccia ObjectGridManager che acquisisca un oggetto ClientSecurityConfiguration. Il seguente è un breve esempio.

```
public ClientClusterContext connect(String catalogServerAddresses,
   ClientSecurityConfiguration securityProps,
     URL overRideObjectGridXml) throws ConnectException;
```

Questo metodo acquisisce un parametro del tipo ClientSecurityConfiguration, che è un'interfaccia che rappresenta una configurazione della sicurezza del client. È possibile utilizzare l'API pubblica com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory per creare un'istanza con valori predefiniti o creare un'istanza passando il file delle proprietà del client WebSphere eXtreme Scale. Questo file contiene le seguenti proprietà relative all'autenticazione. Il valore contrassegnato con un segno più (+) rappresenta l'impostazione predefinita.

- `securityEnabled (true, false+):` questa proprietà indica se la sicurezza è abilitata. Quando un client si collega ad un server, i valori securityEnabled lato client e server devono essere entrambi impostati su `true` o su `false`. Ad esempio se la sicurezza del server connesso è abilitata, il client deve impostare questa proprietà su true per potersi collegare al server.
- authenticationRetryCount (un valore intero, 0+): questa proprietà determina quanti tentativi vengono effettuati se le credenziali sono scadute. Se il valore è 0, non vengono effettuati tentativi. Il tentativo di autenticazione si applica unicamente nel caso di credenziali scadute. Se le credenziali sono scadute, non viene effettuato alcun tentativo. Sarà la propria applicazione responsabile dell'esecuzione della ripetizione dell'operazione.

Dopo aver creato un oggetto com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration, impostare l'oggetto credentialGenerator sul client utilizzando il seguente metodo:

```
/**
* Set the {@link CredentialGenerator} object for this client.
* @param generator the CredentialGenerator object associated with this client
*/
void setCredentialGenerator(CredentialGenerator generator);
```

È possibile impostare l'oggetto CredentialGenerator anche nel file delle proprietà del client WebSphere eXtreme Scale come riportato di seguito.

- credentialGeneratorClass: il nome dell'implementazione della classe per l'oggetto CredentialGenerator. Deve avere un costruttore predefinito.
- credentialGeneratorProps: le proprietà della classe CredentialGenerator. Se il valore non è nullo, esso viene impostato nell'oggetto CredentialGenerator costruito utilizzando il metodo setProperties(String).

Di seguito viene riportato un esempio per creare un'istanza ClientSecurityConfiguration e per poi utilizzarla per collegarsi al server.

```
/**
* Get a secure ClientClusterContext
* @return a secure ClientClusterContext object
*/
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 chiamata una connessione, il client WebSphere eXtreme Scale chiama il metodo CredentialGenerator.getCredential per ottenere le credenziali del client. Queste credenziali vengono inviate assieme alla richiesta di connessione al server per effettuarne l'autenticazione.

Utilizzo un'istanze di CredentialGenerator diversa per sessione

In alcuni casi, un client WebSphere eXtreme Scale rappresenta solo l'identità di un singolo client, ma in altri casi, esso può rappresentare più identità. Di seguito viene riportato uno scenario per quest'ultimo caso: un client WebSphere eXtreme Scale viene creato e condiviso in un server Web. Tutti i servlet in questo server Web utilizzano questo unico client WebSphere eXtreme Scale. Poiché ciascun servlet rappresenta un diverso client Web, utilizzare credenziali diverse quando si inviano richieste ai server WebSphere eXtreme Scale.

WebSphere eXtreme Scale consente di modificare le credenziali a livello di sessione. Ciascuna sessione utilizza un oggetto CredentialGenerator differente. Per cui, gli scenari precedenti possono essere implementati consentendo al servlet di ottenere una sessione con un diverso oggetto CredentialGenerator. Il seguente esempio illustra il metodo ObjectGrid.getSession(CredentialGenerator) nell'interfaccia ObjectGridManager.

```
/**
     * Get a session using a <code>CredentialGenerator</code>.
     * <p>
     * This method can only be called by the ObjectGrid client in an ObjectGrid
     * client server environment. If ObjectGrid is used in a local model, that is,
     * within the same JVM with no client or server existing, <code>getSession(Subject)</code>
     * or the <code>SubjectSource</code> plugin should be used to secure the ObjectGrid.
     *
     * <p>If the <code>initialize()</code> method has not been invoked prior to
     * the first <code>getSession</code> invocation, an implicit initialization
     * will occur.  This ensures that all of the configuration is complete
     * before any runtime usage is required.</p>
     *
     * @param credGen A <code>CredentialGenerator</code> for generating a credential
     *                for the session returned.
     *
     * @return An instance of <code>Session</code>
     *
     * @throws ObjectGridException if an error occurs during processing
     * @throws TransactionCallbackException if the <code>TransactionCallback</code>
     *         throws an exception
     * @throws IllegalStateException if this method is called after the
     *         <code>destroy()</code> method is called.
     *
     * @see #destroy()
     * @see #initialize()
     * @see CredentialGenerator
     * @see Session
     * @since WAS XD 6.0.1
 */
Session getSession(CredentialGenerator credGen) throws
ObjectGridException, TransactionCallbackException;
```

Il seguente è un esempio:

```
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();

CredentialGenerator credGenManager = new UserPasswordCredentialGenerator("manager", "xxxxxx");
CredentialGenerator credGenEmployee = new UserPasswordCredentialGenerator("employee", "xxxxxx");

ObjectGrid og = ogManager.getObjectGrid(ctx, "accounting");

// Get a session with CredentialGenerator;
Session session = og.getSession(credGenManager );

// Get the employee map
```

```
ObjectMap om = session.getMap("employee");

// start a transaction.
session.begin();

Object rec1 = map.get("xxxxxx");

session.commit();

// Get another session with a different CredentialGenerator;
session = og.getSession(credGenEmployee );

// Get the employee map
om = session.getMap("employee");

// start a transaction.
session.begin();

Object rec2 = map.get("xxxxx");

session.commit();
```

Se si utilizza il metodo ObjectGird.getSession per ottenere un oggetto Session, la sessione utilizza l'oggetto CredentialGenerator impostato sull'oggetto ClientConfigurationSecurity. Il metodo ObjectGrid.getSession(CredentialGenerator) sostituisce il CredentialGenerator impostato nell'oggetto ClientSecurityConfiguration.

Se si può riutilizzare l'oggetto Sessione, si otterrà un miglioramento delle prestazioni. Tuttavia, la chiamata del metodo ObjectGrid.getSession(CredentialGenerator) non è molto dispendiosa. Il maggior sovraccarico è rappresentato dall'aumento del tempo di raccolta dei dati obsoleti degli oggetti. Assicurarsi di effettuare il rilascio dei riferimenti dopo aver terminato con gli oggetti Session. Generalmente, se i propri oggetti Session possono condividere l'identità, è bene provare a riutilizzarli. In alternativa, utilizzare il metodo ObjectGrid.getSession(CredentialGenerator).

Programmazione dell'autorizzazione del client

WebSphere eXtreme Scale supporta le autorizzazioni JAAS (Java Authentication and Authorization Service) senza ulteriori configurazioni e supporta inoltre le autorizzazioni personalizzate con l'interfaccia ObjectGridAuthorization.

Il plug-in ObjectGridAuthorization viene utilizzato per autorizzare in modo personalizzato gli accessi ObjectGrid, ObjectMap e JavaMap ai Principal rappresentati da un oggetto Subject. Un'implementazione tipica di questo plug-in è quella che consente di recuperare, da un oggetto Subject, i Principal e quindi di verificare se gli sono state concesse le autorizzazioni.

Un autorizzazione passata al metodo checkPermission(Subject, Permission) può essere una delle seguenti:

- MapPermission
- ObjectGridPermission
- ServerMapPermission
- AgentPermission

Fare riferimento alla documentazione API di ObjectGridAuthorization per ulteriori dettagli.

MapPermission

La classe public com.ibm.websphere.objectgrid.security.MapPermission rappresenta l'autorizzazione alle risorse ObjectGrid, nello specifico, i metodi delle interfacce

ObjectMap o JavaMap. WebSphere eXtreme Scale definisce la seguente stringa di autorizzazione per accedere i metodi di ObjectMap e JavaMap:

- **read**: autorizzazione a leggere i dati dalla mappa. La costante intera è definita come `MapPermission.READ`.
- **write**: autorizzazione ad aggiornare i dati nella mappa. La costante intera è definita come `MapPermission.WRITE`.
- **insert**: autorizzazione ad inserire i dati nella mappa. La costante intera è definita come `MapPermission.INSERT`.
- **remove**: autorizzazione a rimuovere i dati dalla mappa. La costante intera è definita come `MapPermission.REMOVE`.
- **invalidate**: autorizzazione ad invalidare i dati dalla mappa. La costante intera è definita come `MapPermission.INVALIDATE`.
- **all**: tutte le autorizzazioni precedenti: read, write, insert, remote e invalidate. La costante intera è definita come `MapPermission.ALL`.

Fare riferimento alla documentazione API di MapPermission per ulteriori dettagli.

È possibile costruire un oggetto MapPermission passando il nome completo della mappa ObjectGrid (nel formato [ObjectGrid_name].[ObjectMap_name]) e la stringa di autorizzazione o il valore intero. Un stringa di autorizzazione può essere una stringa delimitata da virgole delle stringhe di autorizzazione precedenti quali read, insert, o all. Un valore intero dell'autorizzazione può essere una qualsiasi delle costanti di autorizzazione intere precedentemente menzionate o un valore matematico di più costanti di autorizzazioni intere, quale MapPermission.READ|MapPermission.WRITE.

L'autorizzazione avviene quando viene chiamato un metodo ObjectMap o JavaMap. Il runtime di eXtreme Scale verifica diverse autorizzazioni per diversi metodi. Se le autorizzazioni richieste non vengono concesse al client, ne risulterà un'eccezione AccessControlException.

Tabella 13. Elenco dei metodi e delle MapPermission richieste

Autorizzazione	**ObjectMap/JavaMap**
read	boolean containsKey(Object)
	boolean equals(Object)
	Object get(Object)
	Object get(Object, Serializable)
	List getAll(List)
	List getAll(List keyList, Serializable)
	List getAllForUpdate(List)
	List getAllForUpdate(List, Serializable)
	Object getForUpdate(Object)
	Object getForUpdate(Object, Serializable)
	public Object getNextKey(long)

Tabella 13. Elenco dei metodi e delle MapPermission richieste (Continua)

Autorizzazione	ObjectMap/JavaMap
write	Object put(Object key, Object value)
	void put(Object, Object, Serializable)
	void putAll(Map)
	void putAll(Map, Serializable)
	void update(Object, Object)
	void update(Object, Object, Serializable)
insert	public void insert (Object, Object)
	void insert(Object, Object, Serializable)
remove	Object remove (Object)
	void removeAll(Collection)
	void clear()
invalidate	public void invalidate (Object, boolean)
	void invalidateAll(Collection, boolean)
	void invalidateUsingKeyword(Serializable)
	int setTimeToLive(int)

L'autorizzazione si basa unicamente sul metodo utilizzato, piuttosto che su ciò che esso effettivamente fa. Ad esempio un metodo put può inserire o aggiornare un record a seconda che questo esista già o meno. Tuttavia, i casi inserimento o aggiornamento non sono distinti.

Un tipo di operazione può essere realizzata attraverso la combinazione di altri tipi. Ad esempio, un aggiornamento può essere realizzato tramite una rimozione e quindi un inserimento. Tenere in considerazione queste combinazioni nella progettazione delle politiche di autorizzazione.

ObjectGridPermission

Una com.ibm.websphere.objectgrid.security.ObjectGridPermission rappresenta le autorizzazioni su un ObjectGrid:

- Query: autorizzazione a creare una query dell'oggetto o dell'entità. La costante intera è definita come ObjectGridPermission.QUERY.
- Dynamic map: autorizzazione a creare una mappa dinamica basata sul template della mappa. La costante intera è definita come ObjectGridPermission.DYNAMIC_MAP.

Fare riferimento alla documentazione API di ObjectGridPermission per ulteriori dettagli.

La seguente tabella riepiloga i metodi e le ObjectGridPermission richieste:

Tabella 14. Elenco dei metodi e delle ObjectGridPermission richieste

Azione di autorizzazione	Metodi
query	com.ibm.websphere.objectgrid.Session.createObjectQuery(String)
query	com.ibm.websphere.objectgrid.em.EntityManager.createQuery(String)
dynamicmap	com.ibm.websphere.objectgrid.Session.getMap(String)

ServerMapPermission

Un ServerMapPermission rappresenta le autorizzazioni per un ObjectMap ospitata su un server. Il nome dell'autorizzazione è il nome completo della mappa ObjectGrid. Essa ha le seguenti azioni:

- replicate: autorizzazione a replicare una mappa del server in una cache vicina.
- dynamicIndex: autorizzazione per un client a creare o rimuovere un indice dinamico su un server

Fare riferimento alla documentazione API di ServerMapPermission per ulteriori dettagli. I metodi dettagliati, che richiedono diverse ServerMapPermission, sono elencati nella seguente tabella:

Tabella 15. Autorizzazioni su ObjectMap ospitata su un server

Azione di autorizzazione	Metodi
replicate	com.ibm.websphere.objectgrid.ClientReplicableMap.enableClientReplication(Mode, int[], ReplicationMapListener)
dynamicIndex	com.ibm.websphere.objectgrid.BackingMap.createDynamicIndex(String, boolean, String, DynamicIndexCallback)
dynamicIndex	com.ibm.websphere.objectgrid.BackingMap.removeDynamicIndex(String)

AgentPermission

Un'AgentPermission rappresenta le autorizzazioni per gli agenti del datagrid. Il nome dell'autorizzazione è il nome completo della mappa ObjectGrid e l'azione è una stringa delimitata da virgole dei nomi della classe di implementazione dell'agent o dei nomi del package.

Fare riferimento alla documentazione API di AgentPermission per ulteriori dettagli.

I seguenti metodi nella classe com.ibm.websphere.objectgrid.datagrid.AgentManager richiedono AgentPermission.

```
com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent, Collection)
com.ibm.websphere.objectgrid.datagrid.AgentManager#callMapAgent(MapGridAgent)
com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)
com.ibm.websphere.objectgrid.datagrid.AgentManager#callReduceAgent(ReduceGridAgent, Collection)
```

Meccanismi di autorizzazione

WebSphere eXtreme Scale supporta due tipi di meccanismi di autorizzazione: autorizzazione JAAS (Java Authentication and Authorization Service) e l'autorizzazione personalizzata. Questi meccanismi si applicano a tutte le autorizzazioni. Le autorizzazioni JAAS ampliano le politiche di sicurezza di Java con dei controlli di accesso incentrati sull'utente. Le autorizzazioni possono essere concesse non solo in base al codice in esecuzione, ma anche in base a chi lo esegue. Le autorizzazioni JAAS sono parte di SDK Versione 1.4 e successive.

Inoltre, WebSphere eXtreme Scale supporta anche le autorizzazioni personalizzate con il seguente plug-in:

- ObjectGridAuthorization: personalizzazione delle autorizzazioni ad accedere a tutte le risorse.

È possibile implementare il proprio meccanismo di autorizzazione se non si vogliono utilizzare le autorizzazioni JAAS. Utilizzando un meccanismo di

autorizzazione personalizzato, è possibile utilizzare il database delle politiche, il server delle politiche oppure Tivoli Access Manager per gestire le autorizzazioni.

È possibile configurare il meccanismo di autorizzazione in due modi:

- configurazione XML

  È possibile utilizzare il file XML ObjectGrid per definire un ObjectGrid ed impostare il meccanismo di autorizzazione per AUTHORIZATION_MECHANISM_JAAS o AUTHORIZATION_MECHANISM_CUSTOM. Di seguito è riportato il file secure-objectgrid-definition.xml utilizzato nell'applicazione enterprise ObjectGridSample:

  ```
  <objectGrids>
   <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
    authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
    <bean id="TransactionCallback"
  classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
  ...
  </objectGrids>
  ```

- Configurazione programmatica

  Se s'intende creare un ObjectGrid utilizzando il metodo ObjectGrid.setAuthorizationMechanism(int), è possibile chiamare il seguente metodo per impostare il meccanismo di autorizzazione. Questo metodo può essere chiamato unicamente dal modello di programmazione di WebSphere eXtreme Scale locale quando si crea direttamente l'istanza ObjectGrid:

  ```
  /**
   * Set the authorization Mechanism. The default is
   * com.ibm.websphere.objectgrid.security.SecurityConstants.
   * AUTHORIZATION_MECHANISM_JAAS.
   * @param authMechanism the map authorization mechanism
  */
  void setAuthorizationMechanism(int authMechanism);
  ```

Autorizzazione JAAS

Un oggetto javax.security.auth.Subject rappresenta un utente autenticato. Un Subject è composto di una serie di principal e ciascun Principal rappresenta un'identità per tale utente. Ad esempio, un Subject può avere un principal nome, ad esempio Joe Smith ed un principal gruppo, ad esempio manager.

Utilizzando le politiche di autorizzazione JAAS, le autorizzazioni possono essere concesse a Principal specifici. WebSphere eXtreme Scale associa il Subject con il contesto del controllo accessi corrente. Per ciascuna chiamata al metodo ObjectMap o Javamap, il runtime di Java determina automaticamente se la politica concede le autorizzazioni richieste unicamente ad un Principal specifico, l'operazione viene consentita solo se il Subject associato al contesto del controllo accessi contiene il Principal designato.

È necessario avere familiarità con la sintassi della politica del file delle politiche. Per una descrizione dettagliata delle autorizzazioni JAAS, fare riferimento a JAAS Reference Guide.

WebSphere eXtreme Scale ha un codebase speciale che viene utilizzato per la verifica delle autorizzazioni JAAS per le chiamate di metodo ObjectMap e JavaMap. Questo codebase speciale è `http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction`. Utilizzare questo codebase quando si concedono le

autorizzazioni ObjectMap o JavaMap ai principal. Questo codice speciale è stato creato perché al file JAR (Java archive) per eXtreme Scale vengano concesse tutte le autorizzazioni.

Il template 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 Principal è simile a quello riportato nell'esempio seguente:

```
principal Principal_class "principal_name"
```

In questa politica, ad un determinato principal sono concesse unicamente le autorizzazioni insert e read per queste quattro mappe. L'altro file delle politiche, fullAccessAuth.policy, concede tutte le autorizzazioni a queste mappe ad un principal. Prima di eseguire l'applicazione, modificare le classi principal_name e principal con i valori appropriati. Il valore di principal_name dipende dal registro utente. Ad esempio, se si utilizza l'OS locale come registro utente, il nome macchina è MACH1, l'ID utente è user1 ed il principal_name è MACH1/user1.

La politica di autorizzazione JAAS può essere inserita direttamente nel file della politica di Java oppure in un file di autorizzazione JAAS separato e quindi impostato in uno dei seguenti due modi:

- Utilizzando il seguente argomento di JVM:

  ```
  -Djava.security.auth.policy=file:[JAAS_AUTH_POLICY_FILE]
  ```

- Utilizzando la seguente proprietà nel file java.security:

  ```
  -Dauth.policy.url.x=file:[JAAS_AUTH_POLICY_FILE]
  ```

Autorizzazione ObjectGrid personalizzata

ObjectGridAuthorization viene utilizzato per autorizzare in modo personalizzato gli accessi ObjectGrid, ObjectMap e JavaMap ai Principal rappresentati da un oggetto Subject. Un'implementazione tipica di questo plug-in è quella che consente di recuperare, da un oggetto Subject, i Principal e quindi di verificare se gli sono state concesse le autorizzazioni.

Un autorizzazione passata al metodo checkPermission(Subject, Permission) potrebbe essere una delle seguenti:

- MapPermission
- ObjectGridPermission
- AgentPermission
- ServerMapPermission

Fare riferimento alla documentazione API di ObjectGridAuthorization per ulteriori dettagli.

Il plug-in ObjectGridAuthorization può essere configurato nei seguenti modi:

- configurazione XML

È possibile utilizzare il file XML ObjectGrid per definire il plug-in ObjectAuthorization. Di seguito è riportato un esempio:

```
<objectGrids>
 <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
  authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
...
  <bean id="ObjectGridAuthorization"
classNameName="com.acme.ObjectGridAuthorizationImpl" />
</objectGrids>
```

- Configurazione programmatica

  Se s'intende creare un ObjectGrid utilizzando il metodo API ObjectGrid.setObjectGridAuthorization(ObjectGridAuthorization), è possibile chiamare il seguente metodo per impostare il plug-in di autorizzazione. Questo metodo è applicabile solo al modello di programmazione eXtreme Scale locale quando si crea direttamente l'istanza ObjectGrid.

```
/**
     * Sets the <code>ObjectGridAuthorization</code> for this ObjectGrid instance.
     * <p>
     * Passing <code>null</code> to this method removes a previously set
     * <code>ObjectGridAuthorization</code> object from an earlier invocation of this method
     * and indicates that this <code>ObjectGrid</code> is not associated with a
     * <code>ObjectGridAuthorization</code> object.
     * <p>
     * This method should only be used when ObjectGrid security is enabled. If
     * the ObjectGrid security is disabled, the provided <code>ObjectGridAuthorization</code> object
     * will not be used.
     * <p>
     * A <code>ObjectGridAuthorization</code> plugin can be used to authorize
     * access to the ObjectGrid and maps. Please refer to <code>ObjectGridAuthorization</code> for more details.
     *
     * <p>
     * As of XD 6.1, the <code>setMapAuthorization</code> is deprecated and
     * <code>setObjectGridAuthorization</code> is recommended for use. However,
     * if both <code>MapAuthorization</code> plugin and <code>ObjectGridAuthorization</code> plugin
     * are used, ObjectGrid will use the provided <code>MapAuthorization</code> to authorize map accesses,
     * even though it is deprecated.
     * <p>
     * Note, to avoid an <code>IllegalStateException</code>, this method must be
     * called prior to the <code>initialize()</code> method.  Also, keep in mind
     * that the <code>getSession</code> methods implicitly call the
     * <code>initialize()</code> method if it has yet to be called by the
     * application.
     *
     * @param ogAuthorization the <code>ObjectGridAuthorization</code> plugin
     *
     * @throws IllegalStateException if this method is called after the
     *         <code>initialize()</code> method is called.
     *
     * @see #initialize()
     * @see ObjectGridAuthorization
     * @since WAS XD 6.1
     */
    void setObjectGridAuthorization(ObjectGridAuthorization ogAuthorization);
```

Implementazione di ObjectGridAuthorization

Il metodo booleano checkPermission(Subject subject, Permission permission) dell'interfaccia ObjectGridAuthorization viene chiamato dal runtime di WebSphere eXtreme Scale per verificare se l'oggetto Subject inserito ha l'autorizzazione inserita. L'implementazione dell'interfaccia ObjectGridAuthorization restituisce true se l'oggetto ha l'autorizzazione e false se non la ha.

Un'implementazione tipica di questo plug-in è quella che consente di recuperare, da un oggetto Subject, i Principal e quindi di verificare se agli sono state concesse le autorizzazioni, consultando le politiche specifiche. Queste politiche sono definite dagli utenti. Le politiche possono essere definite in un database, in un file ordinario oppure in un server delle politiche Tivoli Access Manager.

Ad esempio si potrà utilizzare il server delle politiche Tivoli Access Manager per gestire le politiche di autorizzazione ed utilizzare le sue API per autorizzare

l'accesso. Per informazioni sull'utilizzo delle API di Tivoli Access Manager Authorization, fare riferimento a IBM Tivoli Access Manager Authorization Java Classes Developer Reference.

La seguente implementazione di esempio ha i seguenti presupposti:

- Verifica unicamente l'autorizzazione MapPermission. Per altre autorizzazioni restituisce sempre `true`.
- L'oggetto Subject contiene un principal com.tivoli.mts.PDPrincipal.
- Il server delle politiche Tivoli Access Manager ha definite le seguenti autorizzazioni per l'oggetto nome ObjectMap o JavaMap. L'oggetto definito nel server delle politiche deve avere lo stesso nome di ObjectMap o JavaMap nel formato [ObjectGrid_name].[ObjectMap_name]. L'autorizzazione è rappresentata dal primo carattere delle stringhe autorizzazioni definite nell'autorizzazione MapPermission. Ad esempio, l'autorizzazione "r" definita nel server delle politiche rappresenta l'autorizzazione read per la mappa ObjectMap.

Il seguente frammento di codice descrive 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,
 Permission p) {

  // For non-MapPermission, we always authorize.
  if (!(p instanceof MapPermission)){
    return true;
  }

  MapPermission permission = (MapPermission) p;

  String[] str = permission.getParsedNames();

  StringBuffer pdPermissionStr = new StringBuffer(5);
  for (int i=0; i<str.length; i++) {
    pdPermissionStr.append(str[i].substring(0,1));
  }

  PDPermission pdPerm = new PDPermission(permission.getName(),
  pdPermissionStr.toString());

  Set principals = subject.getPrincipals();

  Iterator iter= principals.iterator();
  while(iter.hasNext()) {
    try {
      PDPrincipal principal = (PDPrincipal) iter.next();
      if (principal.implies(pdPerm)) {
        return true;
      }
    }
    catch (ClassCastException cce) {
      // Handle exception
    }
  }
  return false;
}
```

Autenticazione griglia

È possibile utilizzare il plug-in Secure TokenManager per abilitare l'autenticazione server-a-server, che richiede l'implementazione dell'interfaccia SecureTokenManager.

Il metodo generateToken(Object) rende protetto un oggetto e poi genera un token che non può essere compreso da altri. Il metodo verifyTokens(byte[]) esegue il processo inverso: riconverte il token nell'oggetto originale.

Un'implementazione SecureTokenManager semplice utilizza un algoritmo di codifica semplice, come ad esempio un algoritmo XOR, per codificare l'oggetto in formato serializzato e poi utilizzare il corrispondente algoritmo di decodifica per decodificare il token. Questa implementazione non è sicura ed è facile da interrompere.

Implementazione predefinita di WebSphere eXtreme Scale

WebSphere eXtreme Scale fornisce un'implementazione immediatamente disponibile per questa interfaccia. Questa implementazione predefinita utilizza una coppia di chiavi per la firma e la verifica della firma e utilizza una chiave segreta per crittografare il contenuto. Ogni server dispone di un keystore di tipo JCKES per memorizzare la coppia di chiavi, di una chiave privata e una chiave pubblica e di una chiave segreta. Il keystore deve essere di tipo JCKES per memorizzare le chiavi segrete. Queste chiavi vengono utilizzate per la crittografia e la firma oppure per la verifica della stringa segreta al termine dell'invio. Inoltre, il token è associato ad un orario di scadenza. Al termine della ricezione, i dati vengono verificati, decrittografati e confrontati con la stringa segreta del ricevitore. I protocolli di comunicazione SSL (Secure Sockets Layer) non sono necessari tra una coppia di server per l'autenticazione poiché le chiavi private e le chiavi pubbliche servono allo stesso scopo. Tuttavia, se la comunicazione del server non è crittografata, i dati possono essere sottratti esaminando la comunicazione. Poiché il token ha una scadenza a breve, la minaccia di attacco della risposta è ridotta al minimo. Questa possibilità diminuisce in modo significativo se tutti i server sono distribuiti dietro un firewall.

Lo svantaggio di questo approccio è che gli amministratori WebSphere eXtreme Scale devono generare chiavi e trasportarle su tutti i server, il che può causare una violazione della sicurezza durante il trasporto.

Sicureza locale

WebSphere eXtreme Scale fornisce diversi endpoint di sicurezza per consentire di integrare meccanismi personalizzati. Nel modello di programmazione locale, la principale funzione di sicurezza è l'autorizzazione e non ha il supporto di autenticazione. È necessario autenticare al di fuori di WebSphere Application Server. Tuttavia, esistono dei plug-in che sono forniti per ottenere e convalidare oggetti Subject.

Abilitazione sicurezza

L'elenco che segue fornisce i due metodi per abilitare la sicurezza locale:

- **Configurazione XML** È possibile utilizzare il file XML ObjectGrid XML per definire un ObjectGrid e abilitare la sicurezza per quel ObjectGrid. Il file seguente è il file secure-objectgrid-definition.xml che viene utilizzato

nell'esempio di applicazione enterprise ObjectGridSample. In questo file XML, la sicurezza viene abilitata impostando l'attributo securityEnabled su true.

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

- **Programmazione** Se si desidera creare un ObjectGrid utilizzando il metodo API ObjectGrid.setSecurityEnabled(), richiamare il seguente metodo nell'interfaccia ObjectGrid per abilitare la sicurezza.

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

Autenticazione

Nel modello di programmazione locale eXtreme Scale non fornisce alcun meccanismo di autenticazione, ma confida sull'ambiente, sia sui server delle applicazioni che sulle applicazioni per l'autenticazione. Quando viene utilizzato eXtreme Scale nel WebSphere Application Server or WebSphere Extended Deployment, le applicazioni possono utilizzare il meccanismo di autenticazione per la sicurezza di WebSphere Application Server Quando eXtreme Scale è in esecuzione in un ambiente J2SE (Java 2 Platform, Standard Edition), l'applicazione deve gestire le autenticazioni utilizzando il meccanismo di autenticazione di JAAS (Java Authentication and Authorization Service) o altri meccanismi di autenticazione. Per ulteriori informazioni relative all'utilizzo dell'autenticazione JAAS, consultare JAAS reference guide. Il contratto tra un'applicazione e un'istanza ObjectGrid è l'oggetto javax.security.auth.Subject Dopo che il client è autenticato da un server delle applicazioni o dall'applicazione, l'applicazione può recuperare l'oggetto autenticato javax.security.auth.Subject ed utilizzare questo oggetto Subject per ottenere una sessione dall'istanza ObjectGrid richiamando il metodo ObjectGrid.getSession(Subject). Questo oggetto Subject viene utilizzato per autorizzare gli accessi ai dati della mappa. Questo contratto è denominato meccanismo di passaggio dell'oggetto Subject. L'esempio che segue illustra l'API ObjectGrid.getSession(Subject).

```
/**
 * This API allows the cache to use a specific subject rather than the one
 * configured on the ObjectGrid to get a session.
 * @param subject
 * @return An instance of Session
 * @throws ObjectGridException
 * @throws TransactionCallbackException
 * @throws InvalidSubjectException the subject passed in is not valid based
 * on the SubjectValidation mechanism.
 */
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;
```

Il metodo ObjectGrid.getSession() nell'interfaccia ObjectGrid può anche essere utilizzato per ottenere un oggetto Session:

```
/**
 * This method returns a Session object that can be used by a single thread at a time.
 * You cannot share this Session object between threads without placing a
 * critical section around it. While the core framework allows the object to move
 * between threads, the TransactionCallback and Loader might prevent this usage,
 * especially in J2EE environments. When security is enabled, this method uses the
 * SubjectSource to get a Subject object.
 *
 * If the initialize method has not been invoked prior to the first
 * getSession invocation, then an implicit initialization occurs.  This
```

```
 * initialization ensures that all of the configuration is complete before
 * any runtime usage is required.
 *
 * @see #initialize()
 * @return An instance of Session
 * @throws ObjectGridException
 * @throws TransactionCallbackException
 * @throws IllegalStateException if this method is called after the
 *         destroy() method is called.
 */
public Session getSession()
throws ObjectGridException, TransactionCallbackException;
```

Come specifica la documentazione API, quando la sicurezza è abilitata, questo metodo utilizza il plug-in SubjectSource per ottenere l'oggetto Subject. Il plug-in SubjectSource è uno dei plug-in della sicurezza definito in eXtreme Scale per supportare la propagazione degli oggetti Subject. Consultare i plug-in relativi alla sicurezza per ulteriori informazioni. Il metodo getSession(Subject) può essere richiamato solo nell'istanza ObjectGrid locale. Se si richiama il metodo getSession(Subject) dal lato client in una configurazione distribuita eXtreme Scale, risulterà un'eccezione IllegalStateException.

Plug-in Security

WebSphere eXtreme Scalefornisce due plug-in Security che sono relativi al meccanismo di passaggio dell'oggetto Subject: plug-in SubjectSource e SubjectValidation.

Plug-in SubjectSource

Il plug-in SubjectSource, rappresentato dall'interfaccia com.ibm.websphere.objectgrid.security.plugins.SubjectSource, è un plug-in che è utilizzato per ottenere un oggetto Subject da un ambiente di esecuzione eXtreme Scale. Questo ambiente può essere un'applicazione che utilizza ObjectGrid o un server delle applicazioni che ospita l'applicazione. Considerare il plug-in SubjectSource un'alternativa al meccanismo di passaggio dell'oggetto Subject. Utilizzando il meccanismo di passaggio dell'oggetto Subject, l'applicazione recupera l'oggetto Subject e lo utilizza per ottenere l'oggetto Session ObjectGrid. Con il plug-in SubjectSource, il runtime eXtreme Scale recupera l'oggetto Subject e lo utilizza per ottenere l'oggetto Session. Il meccanismo di passaggio dell'oggetto Subject fornisce il controllo degli oggetti Subject alle applicazioni mentre il meccanismo di plug-in SubjectSource non richiede alle applicazioni di recuperare l'oggetto Subject. È possibile utilizzare il plug-in SubjectSource per ottenere l'oggetto Subject che rappresenta un client eXtreme Scale che viene utilizzato per l'autorizzazione. Quando il metodo ObjectGrid.getSession viene richiamato, il getSubject Subject genera un'eccezione ObjectGridSecurityException se la sicurezza è abilitata. WebSphere eXtreme Scale fornisce un'implementazione predefinita di questo plug-in: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl. Questa implementazione può essere utilizzata per recuperare un soggetto chiamante o un soggetto RunAs dal thread quando l'applicazione è in esecuzione in WebSphere Application Server. È possibile configurare questa classe nel file XML del descrittore ObjectGrid come la classe di implementazione SubjectSource quando si utilizza eXtreme Scale in WebSphere Application Server. Il frammento di codice che segue illustra il principale flusso 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 ulteriori dettagli, fare riferimento alla documentazione sull'API per il plug-in SubjectSource e l'implementazione WSSubjectSourceImpl.

Plug-in SubjectValidation

Il plug-in SubjectValidation che è 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 convalidare che un javax.security.auth.Subject, sia passato a ObjectGrid oppure recuperato dal plug-in SubjectSource, che è un valido Subject che non è stato manomesso.

Il metodo SubjectValidation.validateSubject(Subject) nell'interfaccia SubjectValidation prende un oggetto Subject e restituisce un oggetto Subject. Se un oggetto Subject è considerato valido e viene restituito quale oggetto Subject sono tutti validi per l'implementazione. Se l'oggetto Subject non è valido , risulterà un'eccezione InvalidSubjectException.

È possibile utilizzare questo plug-in se non si confida nell'oggetto Subject che viene passato a questo metodo. Questo caso è raro considerando che si confida negli sviluppatori dell'applicazione che sviluppano il codice per recuperare l'oggetto Subject.

Un'implementazione di questo plug-in necessita del supporto di chi ha creato l'oggetto Subject perché solo chi lo ha creato conosce se l'oggetto Subject è stato manomesso. Tuttavia, potrebbe accadere che anche chi ha creato il soggetto non conosca se il Subject è stato manomesso. In questo caso, questo plug-in non è utile.

WebSphere eXtreme Scale fornisce un'implementazione predefinita di SubjectValidation: com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl. È possibile utilizzare questa implementazione per convalidare il soggetto autenticato WebSphere Application Server. È possibile configurare questa classe come la classe di implementazione SubjectValidation quando si utilizza eXtreme Scale in WebSphere Application Server. L'implementazione WSSubjectValidationImpl considera un oggetto Subject valido solo se il token della credenziale che è associato a questo Subject non è stato manomesso. È possibile modificare altre parti dell'oggetto Subject. L'implementazione WSSubjectValidationImpl richiede WebSphere Application Server per il Subject originale corrispondente al token della credenziale e restituisce l'oggetto Subject originale come l'oggetto Subject convalidato. Perciò, le modifiche apportate al contenuto del Subject oltre che sul token della credenziale non hanno effetto. Il seguente frammento di codice illustra il flusso di base di WSSubjectValidationImpl.validateSubject(Subject).

```
// Create a LoginContext with scheme WSLogin and
// pass a Callback handler.
LoginContext lc = new LoginContext("WSLogin",
```

```
new WSCredTokenCallbackHandlerImpl(subject));

// When this method is called, the callback handler methods
// will be called to log the user in.
lc.login();

// Get the subject from the LoginContext
return lc.getSubject();
```

Nel precedente frammento di codice un oggetto gestore callback del token della credenziale, WSCredTokenCallbackHandlerImpl, viene creato con l'oggetto Subject da convalidare. Successivamente un oggetto LoginContext viene creato con lo schema di login WSLogin. Quando viene richiamato il metodo lc.login, la sicurezza WebSphere Application Server recupera il token della credenziale dall'oggetto Subject e poi restituisce il corrispondente Subject come un oggetto Subject convalidato.

Per ulteriori dettagli, fare riferimento alle API Java dell'implementazione SubjectValidation e WSSubjectValidationImpl.

Configurazione del plug-in

È possibile configurare il plug-in SubjectValidation e il plug-in SubjectSource in due modi:

- **Configurazione XML**È possibile utilizzare il file XML ObjectGrid per definire un ObjectGrid e impostare questi due plug-in. Di seguito è riportato un esempio in cui la classe WSSubjectSourceImpl viene configurata come il plug-in SubjectSource e la classe WSSubjectValidation viene configurata come il 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>
  ```

- **Programamazione** Se si desidera creare un ObjectGrid mediante le API è possibile richiamare i seguenti metodi per impostare i plug-in SubjectSource o SubjectValidation.

  ```
  **
  * Set the SubjectValidation plug-in for this ObjectGrid instance. A
  * SubjectValidation plug-in can be used to validate the Subject object
  * passed in as a valid Subject. Refer to {@link SubjectValidation}
  * for more details.
  * @param subjectValidation the SubjectValidation plug-in
  */
  void setSubjectValidation(SubjectValidation subjectValidation);


  /**
  * Set the SubjectSource plug-in. A SubjectSource plug-in can be used
  * to get a Subject object from the environment to represent the
  * ObjectGrid client.
  *
  ```

```
 * @param source the SubjectSource plug-in
 */
 void setSubjectSource(SubjectSource source);
```

Scrittura del proprio codice di autenticazione JAAS

È possibile scrivere il proprio codice di autenticazione Java Authentication and Authorization Service (JAAS) per gestire l'autenticazione È necessario scrivere i propri moduli di login e successivamente configurare i moduli di login per il proprio modulo di autenticazione.

Il modulo di login riceve le informazioni relative ad un utente e autentica l'utente. Questa informazione può essere qualsiasi cosa che possa identificare l'utente. Ad esempio, l'informazione può riguardare l'ID e la password, il certificato del client e così via. Dopo aver ricevuto l'informazione, il modulo di login verifica che l'informazione rappresenti un soggetto valido e successivamente crea un oggetto Subject. Attualmente, diverse implementazioni di moduli di login sono disponibili al pubblico.

Dopo che un modulo di login sia stato scritto, configurare il modulo di login per il runtime da utilizzare. È necessario configurare un modulo di login JAAS. Questo modulo di login contiene il modulo di login e il suo 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 convalidare questo login o l'intero schema avrà esito negativo.

L'impostazione del file di configurazione del modulo di login JAAS può essere fatta in uno dei seguenti modi:

- impostare il file di configurazione del modulo di login JAAS nella proprietà 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 **-(Djava.security.auth.login.config** JVM (Java virtual machine), for example, `-Djava.security.auth.login.config ==$JAVA_HOME/lib/security/file.login`

Se il codice è in esecuzione su WebSphere Application Server, è necessario configurare il login JAAS nella console di gestione e memorizzare questa configurazione di login nella configurazione del server delle applicazioni. Vedere la configurazione di login per i dettagli su Java Authentication and Authorization Service.

Capitolo 10. Considerazioni sulle prestazioni per gli sviluppatori di applicazione

Per migliorare le prestazioni relative alla propria griglia di dati in memoria o allo spazio di elaborazione del database, è possibile tener presente diversi fattori come l'ottimizzazione dell'impostazione della propria Java virtual e l'utilizzo delle migliori pratiche per le funzioni del prodotto come il blocco, la serializzazione e le prestazioni della query.

Ottimizzazione JVM

È necessario tenere in considerazione svariati aspetti specifici relativi all'ottimizzazione di JVM (Java virtual machine) per ottenere le migliori prestazioni di WebSphere eXtreme Scale.

Si consigliano heap da 1 a 2Gb con una JVM ogni 4 core. La dimensione heap dipende dalla natura degli oggetti memorizzati nei server, discussa in seguito in questo documento.

Suggerimenti sulla dimensione heap e sulla raccolta dati obsoleti

La dimensione heap ottimale dipende da tre fattori:

1. Il numero di oggetti vivi nell'heap.
2. La complessità degli oggetti vivi nell'heap.
3. Numero di core disponibili per la JVM.

Ad esempio, un'applicazione che memorizzi array di byte di 10K potrà eseguire un'heap più grande di un'applicazione che utilizzi grafici complessi di POJO.

Al giorno d'oggi tutte le più moderne JVM utilizzano algoritmi di raccolta dati obsoleti paralleli, il che significa che utilizzando più core possono ridurre le pause nella raccolta dati obsoleti. Quindi, unità ad 8-core possono raccogliere più velocemente delle unità a 4-core.

Utilizzo della memoria reale in confronto alle specifiche dell'heap

Un JVM con un heap di 1Gb utilizza approssimativamente 1,3 Gb di memoria reale. Nei nostri laboratori, non siamo riusciti ad eseguire dieci JVM da 1Gb con un'unità con 16Gb di RAM. Una volta che le heap della JVM si sono riempite fino ad 800 e più MB, l'unità ha iniziato la paginazione.

Raccolta dati obsoleti

Per le JVM di IBM, utilizzare il programma di raccolta avgoptpause per scenari ad alto aggiornamento (il 100% delle transazioni modificano le voci). Il programma di raccolta gencon lavora molto meglio di quello avgoptpause per scenari dove i dati non sono frequentemente aggiornati (10% delle volte o meno). Provare entrambi i programma di raccolta per verificare quello che lavora meglio nel proprio scenario. Se si denotano problemi di prestazioni, eseguire la raccolta dati obsoleti interattiva per controllare quanto tempo in percentuale viene impiegato nella raccolta dati obsoleti. Si sono verificati dei casi in cui l'80% del tempo veniva impiegato nella

raccolta dati obsoleti finché la ottimizzazione non ha risolto il problema.

Prestazioni JVM

WebSphere eXtreme Scale può essere eseguito su diverse versioni di Java 2 Platform, Standard Edition (J2SE). ObjectGrid Version 6.1 supporta J2SE Versione 1.4.2 e successive. Per ottenere un miglioramento della produttività e delle prestazioni di Developer utilizzare J2SE 5 o successive per sfruttare le annotazioni e la raccolta dati obsoleti migliorata. ObjectGrid lavora sia su JVM a 32 bit che a 64 bit.

I client ObjectGrid Versione 6.0.2 si possono allegare ad una griglia ObjectGrid Versione 6.1. Utilizzare client ObjectGrid Versione 6.1 per i client J2SE Versione 1.4.2 o migliori. L'unico motivo per utilizzare un client ObjectGrid Versione 6.0.2 e per il supporto di J2SE Versione 1.3.

WebSphere eXtreme Scale è stato verificato con un sottoinsieme delle macchine virtuali disponibili, tuttavia, l'elenco supportato non è esclusivo. È possibile eseguire WebSphere eXtreme Scale su qualsiasi Versione 1.4.2 o successiva, ma se si verifica un errore sulla JVM, si dovrà contattare il fornitore JVM per supporto. Ove possibile, utilizzare la JVM fornita con il runtime di WebSphere su qualsiasi piattaforma supportata da WebSphere Application Server.

Java Platform, Standard Edition 6 è la migliore JVM. Le prestazioni di Java 2 Platform, Standard Edition, v 1.4 sono scarse soprattutto negli scenari dove il programma di raccolta gencon fa la differenza. Le prestazioni di Java Platform Standard Edition 5 sono buone, ma quelle di Java Platform, Standard Edition 6 sono migliori.

Ottimizzazione di orb.properties

Si consiglia di utilizzare il seguente file orb.properties nell'ambiente di produzione. Nei nostri laboratori abbiamo utilizzato questo file su griglie con fino a 1500 JVM. Il file orb.properties si trova nella cartella lib del JRE in uso.

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

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

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

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

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

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

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

Conteggio thread

Il conteggio thread dipende da pochi fattori. Esiste un limite di thread gestibili da un singolo frammento. Con più frammenti per ciascuna JVM vi potrà essere un numero maggiore di JVM ed una maggiore contemporaneità. Ciascun frammento

aggiuntivo fornisce un numero maggiore di percorsi contemporanei per i dati. Ciascun frammento è contemporaneo per quanto possibile, ma malgrado ciò un limite esiste.

Migliori pratiche per CopyMode

WebSphere eXtreme Scale esegue una copia del valore sulla base delle sei impostazioni CopyMode disponibili. Determinare quale impostazione meglio si adatta alle proprie esigenze di distribuzione.

È possibile utilizzare il metodo setCopyMode(CopyMode, valueInterfaceClass) dell'API BackingMap per impostare la modalità di copia con uno dei seguenti campi statici finali definiti nella classe com.ibm.websphere.objectgrid.CopyMode.

Quando l'applicazione utilizza l'interfaccia ObjectMap per ottenere un riferimento ad una voce della mappa, esso deve essere utilizzato solo all'interno della transazione WebSphere eXtreme Scale che lo ha ottenuto. L'utilizzo del riferimento in una transazione diversa potrebbe generare errori. Ad esempio, se si utilizza una strategia di blocco pessimistico per la BackingMap, una chiamata del metodo get o getForUpdate acquisisce un blocco S (condiviso) o U (aggiorna), a seconda della transazione. Il metodo get restituisce un riferimento al valore ed il blocco ottenuto viene rilasciato al completamento della transazione. La transazione deve chiamare il metodo get o getForUpdate per bloccare la voce della mappa in una transazione diversa. Ciascuna transazione deve ottenere il proprio riferimento ad un valore chiamando il metodo get o getForUpdate invece di riutilizzare lo stesso riferimento al valore in più transazioni.

CopyMode per le mappe di entità

Quando si utilizza una mappa associata con un'entità API EntityManager, la mappa restituisce sempre l'oggetto Tupla dell'entità in modo diretto, senza effettuare una copia, a meno che non si stia utilizzando la modalità di copia COPY_TO_BYTES. È importante che il CopyMode sia aggiornato o che la Tupla venga adeguatamente copiata quando si effettuano modifiche.

COPY_ON_READ_AND_COMMIT

La modalità COPY_ON_READ_AND_COMMIT è quella predefinita. Quando si utilizza questa modalità l'argomento valueInterfaceClass verrà ignorato. Questa modalità assicura che un'applicazione non contenga un riferimento ad un oggetto valore presente nella BackingMap. Al contrario, l'applicazione lavora sempre con una copia del valore presente nella BackingMap. La modalità COPY_ON_READ_AND_COMMIT assicura che l'applicazione non danneggi inavvertitamente i dati memorizzati nella cache nella BackingMap. Quando una transazione dell'applicazione chiama un metodo ObjectMap.get per una determinata chiave e si tratta del primo accesso alla voce dell'ObjectMap, viene restituita una copia del valore. Quando viene eseguito il commit della transazione, tutte le modifiche di cui l'applicazione ha eseguito il commit vengono copiate nella BackingMap per assicurare che l'applicazione non abbia un riferimento al valore di cui si è eseguito il commit nella BackingMap.

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 della transazione. Quando si utilizza questa modalità

l'argomento valueInterfaceClass verrà ignorato. Per preservare l'integrità dei dati della BackingMap, l'applicazione assicura che qualsiasi riferimento in essa contenuto ad una voce venga distrutto dopo l'esecuzione del commit della transazione. Con questa modalità, il metodo ObjectMap.get restituisce una copia del valore, invece di un riferimento ad esso, in modo da assicurare che le modifiche effettuate al valore dall'applicazione non influenzino il valore nella BackingMap fino al commit della transazione. Tuttavia, dopo il commit della transazione, non viene eseguita una copia delle modifiche. Nella BackingMap verrà, invece, memorizzato il riferimento alla copia restituito dal metodo ObjectMap.get. L'applicazione distruggerà i riferimenti alla voce della mappa dopo il commit della transazione. Se l'applicazione non distrugge i riferimenti alla voce della mappa, si potrà verificare il danneggiamento dei dati memorizzati nella cache BackingMap. Se un'applicazione utilizza questa modalità e si manifestano dei problemi, passare alla modalità COPY_ON_READ_AND_COMMIT per verificare se i problemi persistono. Se i problemi scompaiono, significa che l'applicazione non riesce a distruggere tutti i suoi riferimenti dopo il commit della transazione.

COPY_ON_WRITE

La modalità COPY_ON_WRITE migliora le prestazioni rispetto alla modalità COPY_ON_READ_AND_COMMIT eliminando la copia che si verifica quando viene chiamato il metodo ObjectMap.get per la prima volta da una transazione per una determinata chiave. Il metodo ObjectMap.get restituisce un proxy al valore invece di un riferimento diretto all'oggetto valore. Il proxy assicura che non venga effettuata una copia del valore a meno che l'applicazione non chiami un metodo set sull'interfaccia del valore specificato dall'argomento valueInterfaceClass. Il proxy fornisce una copia sull'implementazione write. Quando avviene il commit della transazione, la BackingMap esamina il proxy per determinare se è stata effettuata una copia causata da una chiamata al metodo set. Se si è effettuata una copia, nella BackingMap verrà memorizzato il riferimento a tale copia. Il grande vantaggio di questo metodo è che il valore non viene mai copiato durante una lettura o un commit quando la transazione non effettua una chiamata ad un metodo set per cambiare il valore.

Le modalità COPY_ON_READ_AND_COMMIT e COPY_ON_READ effettuano sempre una copia completa quando un valore viene recuperato dall'ObjectMap. Questa modalità non è quella ottimale quando un'applicazione aggiorna solo alcuni dei valori recuperati in una transazione. La modalità COPY_ON_WRITE supporta questo comportamento in maniera efficiente, ma richiede che l'applicazione utilizzi un pattern semplice. Per supportare un'interfaccia, sono richiesti gli oggetti valore. L'applicazione deve utilizzare i metodi su questa interfaccia quando interagisce con il valore in una sessione eXtreme Scale. In questo caso eXtreme Scale crea delle deleghe per i valori che sono restituite all'applicazione. Il proxy ha un riferimento al valore reale. Se l'applicazione esegue solo operazioni di lettura, esse vengono sempre eseguite sulla copia reale. Se l'applicazione modifica un attributo sull'oggetto, il proxy effettua una copia dell'oggetto reale e quindi esegue la modifica sulla copia. Da quel momento in poi il proxy utilizzerà la copia. L'utilizzo della copia consente di evitare completamente l'operazione di copia per gli oggetti che vengono solo letti dall'applicazione. Tutte le operazioni di modifica devono cominciare con il prefisso impostato. Gli Enterprise JavaBeans sono solitamente codificati in modo da utilizzare questo stile di denominazione del metodo per i metodi che modificano gli attributi degli oggetti. Bisogna seguire questa convenzione. Qualsiasi oggetto modificato dall'applicazione viene copiato in quel momento. Questo scenario di lettura e scrittura è quello più efficiente supportato da eXtreme Scale. Per configurare una mappa in modo che utilizzi la modalità COPY_ON_WRITE, utilizzare l'esempio

che segue. In questo esempio, l'applicazione memorizza gli oggetti Person a cui è stata assegnata una chiave utilizzando il nome nella mappa. L'oggetto Person viene rappresentato nel 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 recuperati da un'ObjectMap. Modificare l'oggetto in modo che utilizzi un'interfaccia come nell'esempio seguente.

```
interface IPerson
{
    void setName(String n);
    String getName();
    void setAge(int a);
    int getAge();
}
// Modify Person to implement IPerson interface
class Person implements IPerson {
    ...
}
```

L'applicazione quindi dovrà configurare la BackingMap in modo che utilizzi la modalità COPY_ON_WRITE, come nel seguente esempio:

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// use COPY_ON_WRITE for this Map with
// IPerson as the valueProxyInfo Class
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// The application should then use the following
// pattern when using the PERSON Map.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// the application casts the returned value to IPerson and not Person
IPerson p = (IPerson)person.get("Billy");
p.setAge(p.getAge()+1);
...
// make a new Person and add to Map
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// the following snippet WON'T WORK. Will result in ClassCastException
sess.begin();
```

```
// the mistake here is that Person is used rather than
// IPerson
Person a = (Person)person.get("Bobby");
sess.commit();
```

La prima sezione mostra l'applicazione che recupera un valore denominato Billy nella mappa. L'applicazione esegue il cast dei valori restituiti in un oggetto IPerson, non ad un oggetto Person poiché il proxy restituito implementa due interfacce:

- L'interfaccia specificata nella chiamata di metodo BackingMap.setCopyMode
- L'interfaccia com.ibm.websphere.objectgrid.ValueProxyInfo

È possibile eseguire il cast del proxy in due tipi. La parte finale del precedente frammento di codice dimostra cosa non è consentito nella modalità COPY_ON_WRITE. L'applicazione recupera il record Bobby e cerca di eseguire il cast del record in un oggetto Person. Questa azione fallisce con un'eccezione class cast poiché il proxy restituito non è un oggetto Person. Il proxy restituito implementa un oggetto IPerson e ValueProxyInfo.

Interfaccia ValueProxyInfo e supporto aggiornamento parziale: quest'interfaccia consente ad un'applicazione il recupo dei valori in sola lettura di cui si è effettuato il commit ed a cui il proxy fa riferimento o la serie di attributi modificati durante la transazione.

```
public interface ValueProxyInfo {
    List /**/ ibmGetDirtyAttributes();
    Object ibmGetRealValue();
}
```

Il metodo ibmGetRealValue restituisce una copia dell'oggetto in sola lettura. L'applicazione non deve modificare questo valore. Il metodo ibmGetDirtyAttributes restituisce un elenco di stringhe che rappresentano gli attributi modificati dall'applicazione durante questa transazione. Lo scenario di utilizzo dell'elemento primario di ibmGetDirtyAttributes è un JDBC Java database connectivity) oppure un programma di caricamento basato su CMP. Solo gli attributi presenti nell'elenco devono essere aggiornati nell'istruzione SQL o nell'oggetto associato alla tabella, il che consente al programma di caricamento di generare SQL più efficienti. Quando si esegue il commit di una copia durante una transazione write e se è collegato un programma di caricamento, per ottenere questa informazione, il programma di caricamento potrà effettuare il cast dei valori degli oggetti modificati sull'interfaccia ValueProxyInfo.

Gestione del metodo equals quando si utilizza COPY_ON_WRITE o le deleghe: ad esempio, il codice seguente costruisce un oggetto Person e lo inserisce in un ObjectMap. Successivamente, esso recupera lo stesso oggetto utilizzando il metodo ObjectMap.get. Viene eseguito il cast del valore sull'interfaccia. Se si esegue il cast del valore sull'interfaccia Person, ne risulterà un'eccezione ClassCastException, poiché il valore restituito rappresenta un proxy che implementa l'interfaccia IPerson e non l'oggetto Person. La verifica di uguaglianza non riesce quando si utilizza l'operazione == poiché non si tratta di oggetti uguali.

```
session.begin();
// new the Person object
Person p = new Person(...);
personMap.insert(p.getName, p);
// retrieve it again, remember to use the interface for the cast
IPerson p2 = personMap.get(p.getName());
if(p2 == p) {
```

```
    // they are the same
} else {
    // they are not
}
```

Un'altra considerazione riguarda la necessità di sostituire 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 ed eseguire il cast dell'argomento in modo che sia un IPerson. Poiché l'argomento potrebbe essere un proxy che implementa l'interfaccia IPerson, quando si confrontano delle istanza per verificarne l'uguaglianza è necessario utilizzare i metodi getAge e getName.

```
{
    if ( obj == null ) return false;
    if ( obj instanceof IPerson ) {
        IPerson x = (IPerson) obj;
        return ( age.equals( x.getAge() ) && name.equals( x.getName() ) )
    }
    return false;
}
```

Requisiti delle configurazioni ObjectQuery e HashIndex: quando si utilizza COPY_ON_WRITE con un ObjectQuery o con un plug-in HashIndex, è importante configurare lo schema ObjectQuery ed il plug-in HashIndex in modo che accedano agli oggetti utilizzando i metodi delle proprietà, che rappresenta l'impostazione predefinita. Se sono invece configurati in modo da utilizzare l'accesso al campo, l'indice ed il motore di query cercheranno di accedere ai campi nell'oggetto proxy, operazione che restituirà sempre un valore nullo oppure 0, poiché l'istanza dell'oggetto sarà un proxy.

NO_COPY

La modalità NO_COPY consente di assicurarsi che un'applicazione non modifichi mai un oggetto valore ottenuto utilizzando il metodo ObjectMap.get in cambio di un miglioramento delle prestazioni. Quando si utilizza questa modalità l'argomento valueInterfaceClass verrà ignorato. Se si utilizza questo metodo non verrà effettuata mai alcuna copia del valore. Se l'applicazione modifica dei valori, i dati presenti nella BackingMap saranno danneggiati. La modalità NO_COPY è utile principalmente per le mappe in sola lettura, dove i dati non sono mai modificati dall'applicazione. Se l'applicazione utilizza questa modalità e si manifestano dei problemi, passare alla modalità COPY_ON_READ_AND_COMMIT per vedere se i problemi persistono. Se i problemi scompaiono, significa che l'applicazione sta modificando il valore restituito dal metodo ObjectMap.set durante la transazione o dopo che ne sia stato effettuato il commit. Tutte le mappe associate con le entità API EntityManager utilizzano automaticamente questa modalità indipendentemente da ciò che è stato specificato nella configurazione di eXtreme Scale.

Tutte le mappe associate con le entità API EntityManager utilizzano automaticamente questa modalità indipendentemente da ciò che è stato specificato nella configurazione di eXtreme Scale.

COPY_TO_BYTES

È possibile memorizzare gli oggetti in un formato serializzato invece che in un formato POJO. Utilizzando le impostazioni COPY_TO_BYTES, è possibile diminuire lo spazio occupato in memoria da un grafico di Oggetti di grandi dimensioni. Consultare “Mappe array di byte” a pagina 40 per ulteriori informazioni.

Uso errato di CopyMode

Quando un'applicazione tenta di migliorare le prestazioni utilizzando le modalità di copia COPY_ON_READ, COPY_ON_WRITE o NO_COPY si verificano errori, come descritto in precedenza. Gli errori intermittenti non si verificano quando si cambia la modalità di copia in COPY_ON_READ_AND_COMMIT.

Problema

Il problema potrebbe essere causato da dati danneggiati nella mappa ObjectGrid, il che è dovuto alla violazione da parte dell'applicazione del contratto di programmazione della modalità di copia in uso. Il danneggiamento dei dati può causare errori imprevedibili che si presentano in modo intermittente o in maniera inspiegabile o inaspettata.

Soluzione

L'applicazione deve essere conforme al contratto di programmazione definito per la modalità di copia in uso. Per le modalità di copia COPY_ON_READ e COPY_ON_WRITE, l'applicazione utilizza un riferimento ad un oggetto valore esterno all'ambito della transazione da cui è stato ottenuto il riferimento al valore. Per utilizzare queste modalità, l'applicazione deve cancellare il riferimento all'oggetto valore dopo il completamento della transazione ed ottenere un nuovo riferimento in ciascuna transazione che accede l'oggetto valore. Per la modalità NO_COPY, l'applicazione non deve mai modificare l'oggetto valore. In questo caso, scrivere l'applicazione in modo che non modifichi l'oggetto valore oppure impostarla in modo che utilizzi una modalità di copia differente.

Mappe array di byte

Le coppie chiave-valore possono essere memorizzate nelle proprie mappe in un array di byte invece che nel modulo POJO, riducendo lo spazio occupato in memoria utilizzabile da un grande grafico di oggetti.

Vantaggi

Il quantitativo di memoria che viene consumato aumenta con il numero di oggetti in un grafico di oggetti. Riducendo un grafico complicato di oggetti in un array di byte, viene gestito solo un oggetto nell'heap invece di svariati oggetti. Con questa riduzione del numero di oggetti nell'heap, il runtime Java ha meno oggetti da ricercare durante la raccolta dati obsoleti.

Il meccanismo predefinito di copia utilizzato da WebSphere eXtreme Scale è la serializzazione, che è dispendiosa. Ad esempio, utilizzando la modalità predefinita di copia `COPY_ON_READ_AND_COMMIT`, viene fatta una copia sia al momento della lettura che al momento del richiamo. Invece di fare una copia al momento della lettura, con array di byte, il valore viene deserializzato dai byte, ed invece di fare una copia al momento del commit, il valore viene serializzato in byte. L'utilizzo di array di byte produce come risultato una equivalente coerenza di dati nelle impostazioni predefinite con una riduzione dell'utilizzo della memoria.

Quando si utilizzano array di byte, tener presente che disporre di un meccanismo di serializzazione ottimizzato è cruciale per poter osservare una riduzione nel consumo della memoria. Per ulteriori informazioni, consultare “Prestazione della serializzazione” a pagina 232.

Configurazione di mappe array di byte

È possibile abilitare mappe array di byte con il file XML ObjectGrid modificando l'attributo CopyMode utilizzato da una mappa sull'impostazione COPY_TO_BYTES, mostrato nel seguente esempio:

```
<backingMap name="byteMap" copyMode="COPY_TO_BYTES" />
```

Per ulteriori informazioni,

Per ulteriori informazioni, consultare l'argomento nel file descrittore XML ObjectGrid nella *Guida alla gestione*.

Considerazioni

Bisogna ponderare se utilizzare o meno le Mappe array di byte in un dato scenario. Sebbene si possa ridurre l'utilizzo della memoria, quando si utilizzano array di byte può aumentare l'uso del processore.

L'elenco di seguito riportato sottolinea diversi fattori da tenere in considerazione prima di scegliere di utilizzare la funzione della mappa array di byte.

Tipo di oggetto

Comparativamente, la riduzione della memoria può non essere possibile quando si utilizzano mappe array di byte per gli stessi tipi di oggetto. Di conseguenza, esistono diversi tipi di oggetti per i quali non bisognerebbe usare le mappe array di byte. Se come valori si stanno utilizzando alcuni dei wrapper originali Java o un POJO che non contiene riferimenti ad altri oggetti (solo campi primitivi di memorizzazione), il numero di Oggetti Java è già quanto più basso possibile – ce n'è solo uno. Poiché il quantitativo di memoria utilizzato dall'oggetto è già ottimizzato, non si consiglia l'utilizzo di una mappa array di byte per questi tipi di oggetti. Le mappe array di byte si adattano maggiormente a tipi di oggetti che contengono altri oggetti o raccolte di oggetti dove il numero totale di oggetti POJO è maggiore di uno.

Ad esempio, se si ha un oggetto Customer con un indirizzo dell'ufficio ed un indirizzo di casa nonché una raccolta di ordini, il numero degli oggetti nell'heap ed il numero di byte utilizzato da quegli oggetti può essere ridotto con le mappe array di byte.

Accesso locale

Quando si utilizzano altre modalità di copia, le applicazioni possono essere ottimizzate quando vengono fatte le copie se gli oggetti sono duplicabili con l'ObjectTransformer predefinito oppure quando un ObjectTransformer personalizzato viene fornito di un metodo copyValue ottimizzato. Confrontato con altri metodi di copia, le operazioni di copia in lettura, scrittura o di commit avranno costi aggiuntivi quando si accede agli oggetti in locale. Ad esempio, se si ha una cache locale in una topologia distribuita o se si accede direttamente ad un'istanza ObjectGrid server locale, il tempo di accesso e di commit aumenterà quando si utilizzano mappe array di byte a causa dei costi di serializzazione. Un costo simile sarà riscontrabile in una topologia distribuita se si usano agenti di griglie di dati o se si accede al server primario quando si utilizza il plug-in ObjectGridEventGroup.ShardEvents.

Interazioni di plug-in

Con le mappe array di byte, gli oggetti non vengono deserializzati quando comunicano da un client verso un server a meno che il server non abbia bisogno del modulo POJO. I Plugin che interagiscono con il valore mappa sperimenteranno una riduzione delle prestazioni a causa del requisito di deserializzazione del valore.

Eventuali plug-in che dovessero utilizzare LogElement.getCacheEntry o LogElement.getCurrentValue constateranno questo costo aggiuntivo. Se si vuole richiamare la chiave, è possibile utilizzare LogElement.getKey, che evita il sovraccarico aggiuntivo associato al metodo LogElement.getCacheEntry().getKey. Nelle seguenti sezioni vengono trattati i plug-in alla luce dell'utilizzo degli array di byte.

Indici e query

Quando gli oggetti vengono memorizzati in formato POJO, il costo dell'indicizzazione e delle query è minimo in quanto l'oggetto non ha bisogno di essere deserializzato. Quando si utilizza una mappa array di byte, si avrà un costo aggiuntivo di deserializzazione dell'oggetto. In generale, se la propria applicazione utilizza indici o query, non si consiglia l'utilizzo di mappe array di byte a meno che non vengano eseguite solo query su attributi chiave.

Blocco ottimistico

Quando si utilizza la strategia di blocco ottimistico, si avranno costi aggiuntivi durante gli aggiornamenti e le operazioni di invalidazione. Questo deriva dall'aver deserializzato il valore sul server per richiamare il valore della versione per eseguire il controllo ottimistico di collisione. Se si sta eseguendo il blocco ottimistico per garantire le operazioni fetch e non si ha bisogno del controllo ottimistico di collisione, è possibile utilizzare com.ibm.websphere.objectgrid.plugins.builtins.NoVersioningOptimisticCallback per disabilitare il controllo della versione.

Programma di caricamento

Con un programma di caricamento, si avrà anche il costo nel runtime eXtreme Scale derivante dalla serializzazione e riserializzazione del valore quando viene utilizzato dal programma di caricamento. È sempre possibile utilizzare le mappe array di byte con i programmi di caricamento, ma occorre considerare il costo derivante dall'apportare modifiche al valore in un tale scenario. Ad esempio, è possibile utilizzare la funzione di array di byte nel contesto di una cache prevalentemente in lettura. In questo caso, il beneficio derivante dall'avere meno oggetti nell'heap e meno memoria utilizzata supererà il costo nel quale si incorre dall'utilizzo degli array di byte nelle operazioni di inserimento e di aggiornamento.

ObjectGridEventListener

Quando si utilizza il metodo transactionEnd nel plug-in ObjectGridEventListener, si avrà un costo aggiuntivo dal lato server per le richieste remote durante l'accesso alla CacheEntry di LogElement o al valore corrente. Se l'implementazione del metodo non accede a questi campi, non si avranno costi aggiuntivi.

Migliori pratiche per le prestazioni del plug-in Evictor

Se si utilizzano programmi di eliminazione di plug-in, questi non saranno attivi finché non vengono creati e associati ad una mappa di backup. Le migliori pratiche di seguito riportate aumenteranno le prestazioni dei programmi di eliminazione LFU (least frequently used) e LRU (least recently used).

Programma di eliminazione LFU (Least frequently used)

Il concetto di programma di eliminazione LFU è quello di rimuovere dalla mappa le voci che vengono utilizzate con poca frequenza. Le voci della mappa vengono diffuse su un quantitativo impostato di heap binari. Via via che cresce l'utilizzo di una voce particolare della cache, diventa ordinata ad un livello più alto nell'heap. Quando il programma di eliminazione tenta una serie di eliminazioni, rimuove solo le voci della cache posizionate più in basso rispetto ad un punto specifico sull'heap binario. Come risultato, vengono eliminate le voci utilizzate meno frequentemente.

Programma di eliminazione LRU (Least recently used)

Il programma di eliminazione LRU segue lo stesso concetto del programma di eliminazione LFU con alcune differenze. La differenza principale consiste nel fatto che LRU utilizza una coda FIFO (first in, first out) invece di una serie di heap binari. Ogni volta che si accede ad una voce della cache, la voce si sposta in testa alla coda. Di conseguenza, l'inizio della coda contiene le voci della mappa utilizzate più di recente e la fine della coda diventa la parte con le voci della mappa utilizzate meno di recente. Ad esempio, la voce A della cache viene utilizzata 50 volte e la voce B della cache viene utilizzata solo una volta proprio dopo la voce A della cache. In questa situazione, la voce B della cache si trova all'inizio della coda in quanto è stata usata più di recente e la voce A della cache si trova alla fine della coda. Il programma di eliminazione LRU espelle le voci della cache che si trovano alla fine della coda e che sono le voci della mappa utilizzate meno di recente.

Proprietà LFU e LRU e le migliori pratiche per migliorare le prestazioni

Numero di heap

Quando si utilizza il programma di eliminazione LFU, tutte le voci della cache di una mappa particolare vengono ordinate sul numero di heap che è stato specificato, migliorando drasticamente le prestazioni e prevenendo che tutte le eliminazioni vengano sincronizzate su un heap binario che contiene tutto l'ordinamento della mappa. Un quantitativo maggiore di heap velocizza inoltre il tempo richiesto per il riordinamento dell'heap in quanto ogni heap ha un numero minore di voci. Impostare il numero di heap sul 10% del numero di voci presenti nella propria BaseMap.

Numero di code

Quando si utilizza il programma di eliminazione LRU, tutte le voci della cache di una mappa particolare vengono ordinate sul numero di code LRU che viene specificato, migliorando drasticamente le prestazioni e prevenendo che tutte le eliminazioni vengano sincronizzate su una coda che contiene tutto l'ordinamento della mappa. Impostare il numero delle code sul 10% del numero di voci presenti nella propria BaseMap.

Proprietà MaxSize

Quando un programma di eliminazione LFU o LRU inizia ad eliminare le voci, utilizza le proprietà del programma di eliminazione MaxSize per determinare quanti heap binari o elementi di coda LRU eliminare. Ad esempio, si supponga di impostare il numero di heap o di code in modo che abbia circa dieci voci di mappa in ciascuna coda della mappa. Se la propria proprietà MaxSize è impostata su 7, il programma di eliminazione elimina 3 voci da ogni oggetto coda o heap in modo da portare la dimensione di ogni heap o coda indietro su 7. Il programma di eliminazione elimina voci di mappa da un'heap o coda solo quando quell'heap o coda ha un valore più alto della proprietà MaxSize di elementi in essa contenuti. Impostare MaxSize al 70% della dimensione della propria coda o heap. Per questo esempio, il valore è impostato su 7. È possibile ottenere una dimensione approssimativa di ogni heap o coda dividendo il numero di voci di BaseMap per il numero di heap o di code che sono utilizzate.

Proprietà SleepTime

Un programma di eliminazione non rimuove costantemente voci dalla propria mappa. Infatti è inattivo per un certo quantitativo di tempo, controllando solo la mappa ogni n numero di secondi, dove n si riferisce alla proprietà SleepTime. Questa proprietà influisce positivamente anche sulle prestazioni: l'esecuzione troppo frequente di una curva di eliminazione riduce le prestazioni a causa delle risorse che sono necessarie per l'elaborazione. Tuttavia, il non utilizzo del programma di eliminazione può generare una mappa con voci che non sono necessarie. Una mappa piena di voci non necessarie può influire in modo negativo sia sui requisiti della memoria che sulle risorse di elaborazione richieste per la propria mappa. Un intervallo della curva di eliminazione su quindici secondi è una buona impostazione per la maggior parte delle mappe. Se la mappa è scritta frequentemente e viene utilizzata ad un'elevata velocità di transazione, impostare il valore su un tempo più basso. Se alla mappa si accede poco di frequente, è possibile impostare il tempo su un valore più alto.

Esempio

L'esempio di seguito riportato definisce una mappa, crea un nuovo programma di eliminazione LFU, imposta le proprietà del programma di eliminazione ed imposta la mappa in modo che utilizzi il programma:

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

//Set properties assuming 50,000 map entries
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

L'utilizzo del programma di eliminazione LRU è molto simile all'uso del programma di eliminazione LFU. Segue un esempio:

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");

//Set properties assuming 50,000 map entries
LRUEvictor someEvictor = new LRUEvictor();
```

```
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Notare che solo due righe sono differenti dall'esempio di LFUEvictor.

Migliori pratiche per la prestazione del blocco

Le strategie di blocco e le impostazioni di isolamento della transazione influenzano le prestazioni dell'applicazione.

Recupero di un'istanza della cache

per ulteriori informazioni, econsultare le informazioni relative al blocco della voce della mappa nella sezione *Guida alla gestione*.

Strategia di blocco pessimistico

Utilizzare la strategia di blocco pessimistico per operazioni di lettura e scrittura della mappa in cui le chiavi spesso sono in conflitto. La strategia di blocco pessimistico ha il maggiore impatto sulle prestazioni.

L'isolamento della transazione read committed e read uncommitted

Quando si utilizza una strategia di blocco pessimistico, impostare il livello di isolamento della transazione utilizzando il metodo Session.setTransactionIsolation. Per l'isolamento read committed o read uncommitted, utilizzare gli argomenti Session.TRANSACTION_READ_COMMITTED o Session.TRANSACTION_READ_UNCOMMITTED a seconda dell'isolamento. Per reimpostare il livello di isolamento della transazione per il comportamento di blocco pessimistico predefinito, utilizzare il metodo Session.setTransactionIsolation con l'argomento Session.REPEATABLE_READ.

L'isolamento read committed riduce la durata di blocchi condivisi il che può migliorare la concorrenza e ridurre la possibilità di deadlock. Questo livello di isolamento dovrebbe essere utilizzato quando una transazione non necessita di garantire che i valori in lettura restino non modificati per la durata della transazione.

Utilizzare un isolamento uncommitted read quando la transazione non necessita di vedere i dati sui quali è stato eseguito il commit.

Strategia di blocco ottimistico

La configurazione predefinita è il blocco ottimistico. Questa strategia migliora sia le prestazioni che la scalabilità rispetto alla strategia pessimistica. Utilizzare questa strategia quando le proprie applicazioni possono tollerare che alcuni aggiornamenti ottimistici non riescano pur essendo ancora le prestazioni migliori rispetto alla strategia pessimistica. Questa strategia è eccellente per le operazioni in lettura e per applicazioni di aggiornamento non frequenti.

Plug-in OptimisticCallback

La strategia di blocco ottimistico effettua una copia delle voci di cache e le confronta in base a quanto necessario. Questa operazione può essere onerosa

perché copiare la voce potrebbe coinvolgere una clonazione o serializzazione. Per implementare le prestazioni in modo che siano quanto più veloci possibili, implementare il plug-in personalizzato per le mappe non entità.

Per ulteriori informazioni, vedere Per ulteriori informazioni, consultare le informazioni relative al plug-in OptimisticCallback in *Panoramica sul prodotto*.

Utilizzare i campi versione per le entità

Quando si utilizza un blocco ottimistico con le entità, utilizzare l'annotazione @Version o l'attributo equivalente nel file descrittore dei metadati dell'entità. L'annotazione della versione fornisce a ObjectGrid uno metodo molto efficace per tenere traccia della versione di un oggetto. Se l'entità non dispone di un campo versione ed è utilizzato il blocco ottimistico per l'entità, allora l'intera entità deve essere copiata e confrontata.

Strategia di blocco None

Utilizzare la strategia di blocco None per le applicazioni che sono solo in lettura. La strategia di blocco None non ottiene dei blocchi o utilizza il gestore blocco. Perciò, questa strategia offre più concorrenza, prestazioni e scalabilità.

Prestazione della serializzazione

WebSphere eXtreme Scale utilizza più processi Java per contenere i dati. Questi processi serializzano i dati: convertono, cioè, i dati (che si trovano nel modulo delle istanze dell'oggetto Java) in byte e poi di nuovo in oggetti secondo quanto necessario per spostare i dati tra i processi del client e del server. L'esecuzione del marshalling dei dati è l'operazione più dispendiosa e deve essere indirizzata dallo sviluppatore dell'applicazione durante la progettazione dello schema, configurando la griglia e interagendo con le API di accesso ai dati.

Le routine predefinite di copia e serializzazione Java sono relativamente lente e possono consumare il 60 - 70 per cento del processore in una configurazione tipica. Le seguenti sezioni rappresentano delle scelte per il miglioramento della prestazione della serializzazione.

Scrittura di un ObjectTransformer per ogni BackingMap

Un ObjectTransformer può essere associato ad una BackingMap. La propria applicazione può avere una classe che implementa l'interfaccia di ObjectTransformer e che fornisce implementazioni per le operazioni di seguito riportate:

- Copia di valori
- Chiavi di serializzazione e deserializzazione verso e dai flussi
- Valori di serializzazione e deserializzazione verso e dai flussi.

L'applicazione non ha bisogno di copiare le chiavi poiché le chiavi sono considerate non mutabili.

Per ulteriori informazioni, consultare Plug-in per la serializzazione e la copia di oggetti memorizzati nella cache e Migliori prassi per l'interfaccia ObjectTransformer.

Nota: ObjectTransformer è l'unico richiamato quando ObjectGrid è informato sui dati in fase di trasformazione. Ad esempio, quando vengono utilizzati gli agent dell'API DataGrid, gli stessi agent così come i dati dell'istanza dell'agent o i dati restituiti dall'agent, devono essere ottimizzati utilizzando le tecniche personalizzate di serializzazione. ObjectTransformer non viene richiamato per gli agent delle API DataGrid.

Utilizzo delle entità

Quando si utilizzano le API EntityManager con entità, l'ObjectGrid non memorizza gli oggetti dell'entità direttamente nelle BackingMaps. L'API EntityManager converte l'oggetto dell'entità in oggetti Tupla. Per ulteriori informazioni, consultare Per ulteriori informazioni, consultare l'argomento sull'utilizzo di un programma di caricamento con mappe di entità e tuple in *Guida alla programmazione*. Le mappe dell'entità sono associate automaticamente ad un ObjectTransformer altamente ottimizzato. Ogni volta che viene utilizzata l'API ObjectMap o l'API EntityManager per interagire con le mappe dell'entità, viene richiamata l'entità ObjectTransformer.

Personalizzazione della serializzazione

Esistono alcuni casi in cui gli oggetti devono essere modificati per utilizzare la serializzazione personalizzata, come l'implementazione dell'interfaccia java.io.Externalizable o l'implementazione dei metodi writeObject e readObject per classi che implementano l'interfaccia java.io.Serializable. Le tecniche di serializzazione personalizzata devono essere impiegate quando vengono serializzati gli oggetti utilizzando meccanismi diversi dai metodi delle API ObjectGrid o EntityManager.

Ad esempio, quando oggetti o entità vengono memorizzati come dati dell'istanza in un agent dell'API DataGrid oppure quando l'agent restituisce oggetti o entità, quegli oggetti non vengono trasformati utilizzando un ObjectTransformer. L'agent, tuttavia, utilizzerà automaticamente l'ObjectTransformer quando utilizza l'interfaccia `EntityMixin`. Per ulteriori dettagli, consultare Agent DataGrid e Mappe basate su entità.

Array di byte

Quando si utilizzano le API ObjectMap o DataGrid, gli oggetti chiave e valore vengono serializzati ogni volta che il client interagisce con la griglia e quando viene replicato l'oggetto. Per evitare il sovraccarico della serializzazione, utilizzare gli array di byte invece degli oggetti Java. Gli array di byte sono molto più semplici da inserire in memoria poiché JDK ha un numero inferiore di oggetti da cercare per la raccolta dati obsoleti e possono essere deserializzati solo all'occorrenza. Gli array di byte devono essere utilizzati solo se non è necessario accedere agli oggetti usando query o indici. Poiché i dati vengono memorizzati come byte, è possibile accedere ai dati solo attraverso chiavi.

WebSphere eXtreme Scale può memorizzare dati automaticamente come array di byte utilizzando l'opzione di configurazione della mappa CopyMode.COPY_TO_BYTES oppure può essre gestito manualmente dal client. Questa opzione memorizzerà in modo efficiente i dati in memoria e può anche deserializzare gli oggetti all'interno dell'array di byte per l'uso mediante query e indici on demand.

Migliori pratiche per l'interfaccia ObjectTransformer

L'interfaccia ObjectTransformer utilizza i callback all'applicazione per fornire implementazioni personalizzate di operazioni comuni e dispendiose quali la serializzazione degli oggetti e le copie complete sugli oggetti.

Panoramica

Per i dettagli relativi all'interfaccia ObjectTransformer, consultare “Plug-in ObjectTransformer” a pagina 228. Da un punto di vista delle prestazioni, e dalle informazioni relative al metodo CopyMode contenute nell'argomento Migliori pratiche del metodo CopyMode, eXtreme Scale chiaramente copia i valori per tutti i casi tranne quando viene utilizzata la modalità NO_COPY. Il meccanismo di copia predefinito impiegato in eXtreme Scale è la serializzazione, nota come un'operazione dispendiosa. In questa situazione viene utilizzata l'interfaccia ObjectTransformer. L'interfaccia ObjectTransformer utilizza i callback all'applicazione per fornire un'implementazione personalizzata di operazioni comuni e dispendiose, come la serializzazione degli oggetti e le copie complete sugli oggetti.

Un'applicazione può fornire un'implementazione dell'interfaccia ObjectTransformer ad una mappa, e eXtreme Scale quindi delega ai metodi su questo oggetto e si affida all'applicazione per fornire una versione ottimizzata di ciascun 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 possibile associare un'interfaccia ObjectTransformer ad una BackingMap utilizzando il seguente codice di esempio:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Ottimizzazione della serializzazione e deserializzazione degli oggetti

La serializzazione degli oggetti è in genere la considerazione più importante sulle prestazioni con eXtreme Scale, che utilizza il meccanismo serializzabile predefinito se l'applicazione non fornisce un plug-in ObjectTransformer. Un'applicazione può fornire le implementazioni readObject e writeObject serializzabili o può lasciare che siano gli oggetti ad implementare l'interfaccia esternalizzabile, che è circa dieci volte più rapido. Se gli oggetti nella mappa non possono essere modificati, un'applicazione può associare un'interfaccia ObjectTransformer all'ObjectMap. I metodi di serializzazione e deserializzazione vengono forniti per consentire all'applicazione di fornire codice personalizzato per ottimizzare queste operazioni, dato il loro vasto impatto sulle prestazioni nel sistema. Il metodo di serializzazione serializza l'oggetto nel flusso fornito. Il metodo di deserializzazione fornisce il flusso di input e si aspetta che l'applicazione crei l'oggetto, lo deserializzi utilizzando i dati contenuti nel flusso e restituisce l'oggetto. Le implementazioni dei metodi di serializzazione e deserializzazione devono riflettersi reciprocamente.

Ottimizzazione delle operazioni di copia completa

Dopo che l'applicazione ha ricevuto un oggetto da una ObjectMap, eXtreme Scale esegue una copia completa sul valore dell'oggetto per assicurarsi che la copia nella mappa BaseMap mantenga l'integrità dei dati. L'applicazione può quindi modificare tranquillamente il valore dell'oggetto. Quando viene eseguito il commit della transazione, la copia del valore dell'oggetto nella mappa BaseMap viene aggiornata al nuovo valore modificato e l'applicazione si arresta utilizzando il valore da quel punto in poi. Nella fase del commit si sarebbe potuto copiare l'oggetto di nuovo per effettuare una copia privata. Tuttavia, in questo caso il costo delle prestazioni di questa azione è stato contraccambiato richiedendo al programmatore dell'applicazione di non utilizzare il valore dopo il commit della transazione. L'ObjectTransformer predefinito cerca di utilizzare un clone o una coppia serializzazione-deserializzazione per generare una copia. La coppia serializzazione-deserializzazione è il peggior caso di scenario delle prestazioni. Se l'analisi del profilo rivela che la serializzazione e la deserializzazione è un problema per l'applicazione, scrivere un metodo clone appropriato per creare una copia completa. Se non è possibile alterare la classe, creare un plug-in ObjectTransformer personalizzato ed implementare metodi copyValue e copyKey più efficienti.

Capitolo 11. Risoluzione dei problemi

Oltre ai file di log e alla traccia, ai messaggi e alle note sulla release di cui si è trattato in questa sezione, è possibile utilizzare strumenti di monitoraggio per chiarire aspetti quali l'ubicazione dei dati nell'ambiente, la disponibilità dei server nella griglia e così via. Se si sta utilizzando un ambiente WebSphere Application Server, è possibile utilizzare lo strumento PMI (Performance Monitoring Infrastructure). Se si sta utilizzando un ambiente autonomo, è possibile utilizzare uno strumento di monitoraggio del fornitore, come ad esempio CA Wily Introscope o Hyperic HQ. È inoltre possibile utilizzare e personalizzare il programma di utilità di esempio xsAdmin per visualizzare informazioni di testo relative all'ambiente.

Log e traccia

È possibile utilizzare le funzioni di log e traccia per monitorare e risolvere i problemi nel proprio ambiente. I file di log si trovano in ubicazioni diverse a seconda della configurazione utilizzata. Potrebbe essere necessario fornire la funzione di traccia per un server quando si utilizza il supporto IBM.

File di log con WebSphere Application Server

Per ulteriori informazioni, consultare il WebSphere Application Server Centro informazioni.

File di log con WebSphere eXtreme Scale in un ambiente autonomo

Con i server contenitore e di catalogo autonomi, l'utente può impostare l'ubicazione dei file di log ed eventuali specifiche sulla traccia. I file di log del server di catalogo si trovano nell'ubicazione in cui è stato eseguito il comando di avvio del server.

Impostazione dell'ubicazione file di log per server contenitore

Per impostazione predefinita, i file di log di un contenitore si trovano nella directory in cui è stato eseguito il comando del server. Se si avviano i server nella directory *<home_eXtremeScale>*/bin, i file di log e di traccia si trovano nelle directory logs/*<nome_server>* della directory bin. Per specificare un'ubicazione alternativa dei file di log di un server contenitore, creare un file delle proprietà, come ad esempio il file server.properties, con il seguente contenuto:

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

La proprietà workingDirectory è la directory principale per i file di log e il file di traccia opzionale. WebSphere eXtreme Scale crea una directory con il nome del server contenitore con un file SystemOut.log, un file SystemErr.log e un file di traccia se la traccia è stata abilitata con l'opzione traceSpec. Per utilizzare un file delle proprietà durante l'avvio del contenitore, utilizzare l'opzione **-serverProps** e fornire l'ubicazione del file delle proprietà del server.

I messaggi informativi comuni che vanno controllati nel file SystemOut.log sono messaggi di conferma dell'avvio. Per ulteriori informazioni su uno specifico messaggio, consultare “Messaggi” a pagina 388.

Funzione di traccia con WebSphere Application Server

Per ulteriori informazioni, consultare il WebSphere Application Server Centro informazioni.

Funzione di traccia sul servizio catalogo

È possibile impostare la funzione di traccia su un servizio catalogo utilizzando i parametri **-traceSpec** e **-traceFile** durante l'avvio del servizio catalogo. Ad esempio:

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

Se si avvia il servizio catalogo nella directory *<home_eXtremeScale>*/bin, i file di log e di traccia si trovano in una directory logs/*<nome_servizio_catalogo>* all'interno della directory bin. Vedere le informazioni sull'avvio del processo del servizio catalogo in un ambiente autonomo in *Guida alla gestione*.

Funzione di traccia su un server contenitore autonomo

È possibile abilitare la funzione di traccia su un server contenitore in due modi. È possibile creare un file delle proprietà del server come descritto nella sezione dei file di log oppure è possibile abilitare la traccia utilizzando la riga comandi all'avvio. Per abilitare la traccia del contenitore con un file delle proprietà del server, aggiornare la proprietà **traceSpec** con la specifica di traccia richiesta. Per abilitare la traccia del contenitore utilizzando i parametri di avvio, utilizzare i parametri **-traceSpec**e **-traceFile**. Ad esempio:

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

Se si avvia il server dalla directory *<home_eXtremeScale>*/bin, i file di log e di traccia si trovano nella directory logs/*<nome_server>* all'interno della directory bin

Per ulteriori informazioni, consultare

Funzione di traccia con l'interfaccia ObjectGridManager

Un'altra opzione consiste nel impostare la traccia durante il runtime su un'interfaccia ObjectGridManager. L'impostazione della traccia su un'interfaccia ObjectGridManager può essere utilizzata per ottenere la traccia su un client eXtreme Scale mentre si collega a un eXtreme Scale ed esegue il commit delle transazioni. Per impostare la traccia su un'interfaccia ObjectGridManager, fornire una specifica di traccia e un file di log della traccia.

```
ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
...
manager.setTraceEnabled(true);
manager.setTraceFileName("logs/myClient.log");
manager.setTraceSpecification("ObjectGridReplication=all=enabled");
```

Abilitazione della traccia con il programma di utilità xsadmin

Per abilitare la traccia con il programma di utilità xsadmin, utilizzare l'opzione **setTraceSpec**. Utilizzare il programma di utilità xsadmin per abilitare la traccia in un ambiente autonomo durante il runtime anziché durante l'avvio. È possibile

abilitare la traccia su tutti i server e servizi di catalogo o filtrare i server in base al nome ObjectGrid e così via. Ad esempio, per eseguire la traccia ObjectGridReplication con l'accesso al server per il servizio catalogo, eseguire:

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

È possibile inoltre disabilitare la traccia impostando la specifica di traccia su `*=all=disabled`.

Per ulteriori dettagli, consultare le informazioni sul programma di utilità di avvio xsAdmin in *Guida alla gestione*.

File e directory ffdc

I file FFDC vengono utilizzati dal supporto IBM per assistenza durante il debug. Questi file potrebbero essere richiesti dal supporto IBM se si verifica un problema.

I file si trovano in una directory con nome `ffdc` e contengono file simili al seguente:

```
server2_exception.log
server2_20802080_07.03.05_10.52.18_0.txt
```

Opzioni per la traccia

È possibile abilitare la traccia per fornire al supporto IBM informazioni relative al proprio ambiente.

Informazioni sulla traccia

La traccia di WebSphere eXtreme Scale è divisa in vari componenti diversi. Analogamente alla traccia di WebSphere Application Server, è possibile specificare il livello di traccia da utilizzare. I livelli di traccia comuni includono: all, debug, entryExit e event.

Di seguito è riportato un esempio di stringa di traccia:

```
ObjectGridComponent=level=enabled
```

Le stringhe di traccia possono essere concatenate. Utilizzare il simbolo `*` (asterisco) per specificare un valore jolly, ad esempio `ObjectGrid*=all=enabled`. Se è necessario fornire una traccia al supporto IBM, è richiesta una stringa di traccia specifica. Ad esempio, se si verifica un problema relativo alla replica, potrebbe essere richiesta la traccia `ObjectGridReplication=debug=enabled`.

Specifica della traccia

ObjectGrid
Motore della cache principale generale.

ObjectGridCatalogServer
Servizio catalogo generale.

ObjectGridChannel
Comunicazioni di topologia di distribuzione statica.

7.1+ **ObjectGridClientInfo**
Informazioni sul client DB2.

7.1+ **ObjectGridClientInfoUser**
Informazioni sull'utente DB2.

ObjectgridCORBA
Comunicazioni di topologia di distribuzione dinamica.

ObjectGridDataGrid
API AgentManager.

ObjectGridDynaCache
Provider della cache dinamica di WebSphere eXtreme Scale.

ObjectGridEntityManager
API EntityManager. Utilizzare questa opzione con l'opzione Projector.

ObjectGridEvictors
Programmi di eliminazione ObjectGrid incorporati.

ObjectGridJPA
Programmi di caricamento JPA (Java Persistence API).

ObjectGridJPACache
Plug-in cache JPA.

ObjectGridLocking
Gestore blocco voci della cache ObjectGrid.

ObjectGridMBean
Bean di gestione.

7.1+ **ObjectGridMonitor**
Infrastruttura monitoraggio cronologico.

ObjectGridPlacement
Servizio di posizionamento frammenti del server di catalogo.

ObjectGridQuery
Query ObjectGrid.

=ObjectGridReplication
Servizio di replica.

ObjectGridRouting
Dettagli di instradamento client/server.

ObjectGridSecurity
Traccia della sicurezza.

ObjectGridStats
Statistiche ObjectGrid.

ObjectGridStreamQuery
API Stream Query.

ObjectGridWriteBehind
Write-behind ObjectGrid.

Projector
Motore all'interno dell'API EntityManager.

QueryEngine
Il motore di query per l'API Object Query e l'API EntityManager Query.

QueryEnginePlan
Diagnostica del piano di query.

IBM Support Assistant per WebSphere eXtreme Scale

È possibile utilizzare IBM Support Assistant per raccogliere dati, analizzare sintomi e accedere alle informazioni del prodotto.

IBM Support Assistant Lite

IBM Support Assistant Lite per WebSphere eXtreme Scale fornisce la raccolta automatica dei dati e supporto per l'analisi dei sintomi nei casi di determinazione dei problemi.

IBM Support Assistant Lite riduce la quantità di tempo necessaria a riprodurre un problema con adeguate affidabilità disponibilità e supportabilità tracciandone l'impostazione dei livelli (i livelli di traccia vengono impostati automaticamente dallo strumento) per semplificare la determinazione del problema. Se è necessaria ulteriore assistenza, IBM Support Assistant Lite riduce anche lo sforzo richiesto per inviare le appropriate informazioni del log al Supporto IBM.

IBM Support Assistant Lite viene incluso in ciascuna installazione di WebSphere eXtreme Scale Versione 7.1.0

IBM Support Assistant

ISA (IBM® Support Assistant) fornisce accesso rapido al prodotto, alla formazione e alle risorse di supporto che possono guidare l'utente a rispondere alle domande e a risolvere i problemi con i prodotti software IBM senza la necessità di contattare il Supporto IBM. Differenti plug-in per specifici prodotti consentono di personalizzare IBM Support Assistant per particolari prodotti che sono stati installati. IBM Support Assistant può anche raccogliere i dati di sistema, i file di log e altre informazioni per guidare il Supporto IBM a determinare la causa di un particolare problema.

IBM Support Assistant è un programma di utilità da installare sulla propria stazione di lavoro, non direttamente sul sistema server stesso WebSphere eXtreme Scale I requisiti di memoria e di risorsa per Assistant potrebbero influire negativamente sulle prestazioni del sistema server WebSphere eXtreme Scale. I componenti di diagnostica inclusi sono progettati per produrre il minimo impatto sulle normali operazioni di un server.

È possibile utilizzare IBM Support Assistant per guidare l'utente nei seguenti modi:

- per ricercare attraverso knowledge IBM e non-IBM e le fonti di informazioni tra più prodotti IBM la risposta a domande o la soluzione di un problema.
- Per ottenere ulteriori informazioni attraverso le risorse Web di specifici prodotti incluso le home page del prodotto e del supporto, i newsgroup del cliente e i forum , le risorse di formazione e addestramento e le informazioni relative alla risoluzione dei problemi e alle domande comuni.
- Per estendere la possibilità di diagnosticare problemi per prodotti specifici utilizzando strumenti di diagnostica mirati disponibili in Support Assistant
- Per semplificare la raccolta dei dati di diagnostica per guidare l'utente e l'IBM a risolvere i problemi (raccogliendo sia dati generali che dati specifici per prodotto/o sintomo)
- Per guidate il Supporto IBM a redigere i report dei problemi mediante un'interfaccia in linea personalizzata, che includa la possibilità di allegare i dati di diagnostica indicati sopra o qualsiasi altra informazione relativa a problemi nuovi o esistenti.

Alla fine è possibile utilizzare la funzione incorporata Updater per ottenere il supporto per altri prodotti software e capability appena essi divengono disponibili. Per impostare IBM Support Assistant per l'utilizzo con WebSphere eXtreme Scale, installare prima IBM Support Assistant utilizzando i file forniti nell'immagine scaricata dalla pagina Web per una panoramica sul supporto IBM nel sito http://www-947.ibm.com/support/entry/portal/Overview/Software/Other_Software/IBM_Support_Assistant. Successivamente, utilizzare IBM Support Assistant per individuare ed installare qualsiasi aggiornamento di prodotto. È possibile anche scegliere di installare i plug-in disponibili per altri software IBM nel proprio ambiente. Ulteriori informazioni e la versione più recente di IBM Support Assistant sono disponibili alla pagina Web IBM Support Assistant http://www.ibm.com/software/support/isa/.

Messaggi

Quando si incontra un messaggio in un file di log o in altre parti dell'interfaccia del prodotto, è possibile ricercarlo tramite prefisso del componente per ricevere maggiori informazioni.

Ricerca di messaggi

Quando si rileva un messaggio in un file di log, copiare il numero del messaggio con relativo prefisso in lettere e numero e ricercarlo nel centro informazioni (ad esempio, `CWOBJ1526I`). Quando si ricerca un messaggio, è possibile trovare un'ulteriore spiegazione del messaggio e le possibili azioni che si possono intraprendere per risolvere il problema.

Per ottenere un indice dei messaggi del prodotto, consultare il centro informazioni.

Note sulla release

Sono forniti collegamenti al sito Web del supporto per il prodotto, alla documentazione del prodotto e agli aggiornamenti dell'ultimo minuto, alle limitazioni e ai problemi noti relativi al prodotto.

- “Accesso agli aggiornamenti dell'ultimo minuto, alle limitazioni e ai problemi noti”
- “Accesso ai requisiti software e di sistema” a pagina 389
- “Accesso alla documentazione del prodotto” a pagina 389
- “Accesso al sito Web del supporto per il prodotto” a pagina 389
- “Contattare il supporto software IBM.” a pagina 389

Accesso agli aggiornamenti dell'ultimo minuto, alle limitazioni e ai problemi noti

Le note sulla release sono disponibili come note tecniche nel sito del supporto per il prodotto. Per visualizzare un elenco di tutte le note tecniche per WebSphere eXtreme Scale, andare alla pagina Web del Supporto. Facendo clic sui collegamenti forniti qui, inizierà una ricerca della pagina Web del supporto per le note sulla release rilevanti, che verrà restituita sotto forma di elenco.

- 7.1+ Per visualizzare un elenco delle note sulla release per la Versione 7.1, andare alla pagina Web del supporto http://www-01.ibm.com/support/search.wss?rs=3023&tc=SSPPLQ&q=v71xsrnotes.

- Per visualizzare un elenco delle note sulla release per la Versione 7.0, andare alla pagina Web del Supporto.
- Per visualizzare un elenco delle note sulla release per la Versione 6.1, andare alla pagina wiki delle note sulla release .

Accesso ai requisiti software e di sistema

I requisiti hardware e software sono documentati nelle seguenti pagine:

- Requisiti di sistema dettagliati

Accesso alla documentazione del prodotto

Per l'intera serie informazioni, andare alla pagina Library.

Accesso al sito Web del supporto per il prodotto

Per cercare le note tecniche, i download e le fix più recenti e altre informazioni relative al supporto, andare alla pagina del Supporto.

Contattare il supporto software IBM.

Se si verifica un problema relativo al prodotto, in primo luogo provare ad effettuare queste operazioni:

- Seguire le procedure descritte nella documentazione del prodotto
- Cercare la documentazione correlata nella guida in linea
- Cercare i messaggi di errore nei riferimenti ai messaggi

Se con i metodi precedentemente indicati non è possibile risolvere il problema, contattare il supporto tecnico IBM.

Informazioni particolari

Riferimenti in questa documentazione a prodotti, programmi o servizi IBM non implica che IBM intenda renderli disponibili in tutti i paesi in cui IBM opera. Qualsiasi riferimento ad un prodotto, programma o servizio IBM non implica o non intende dichiarare che possa essere utilizzato solo quel prodotto, programma o servizio IBM. In sostituzione a quelli forniti da IBM possono essere utilizzati prodotti, programmi o servizi funzionalmente equivalenti che non comportino la violazione dei diritti di proprietà intellettuale IBM. È responsabilità dell'utente valutare e verificare il funzionamento con altri prodotti, ad eccezione di quelli progettati espressamente da IBM.

IBM può avere brevetti o domande di brevetto in corso relativi a quanto trattato nel presente documento. Il rilascio di questo documento non fornisce alcuna licenza a questi brevetti. È possibile inviare per iscritto richieste di licenze a:

IBM Director of Commercial Relations
IBM Europe
Schoenaicher Str. 220
D-7030 Boeblingen Deutschland

Coloro che detengono la licenza su questo programma e desiderano avere informazioni su di esso allo scopo di consentire (i) uno scambio di informazioni tra programmi indipendenti ed altri (compreso questo) e (ii) l'uso reciproco di tali informazioni, dovrebbero rivolgersi a:

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Tali informazioni possono essere disponibili, in base ad appropriate clausole e condizioni, includendo in alcuni casi, il pagamento di un corrispettivo.

Marchi

I seguenti termini sono marchi di IBM Corporation negli Stati Uniti e/o in altri paesi:

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java e tutti i marchi basati su Java sono marchi di Sun Microsystems, Inc. negli Stati Uniti e/o in altri paesi.

LINUX è un marchio di Linus Torvalds negli Stati Uniti e/o in altri paesi.

Microsoft, Windows®, Windows NT®, e il logo Windows sono marchi di Microsoft Corporation negli Stati Uniti e/o in altri paesi.

UNIX® è un marchio registrato di The Open Group negli Stati Uniti e in altri paesi.

Nomi di altri prodotti, società e servizi possono essere marchi di altre società.

Indice analitico

Stampato in Italia