WebSphere® WebSphere Extended Deployment Version 6.0

Handbuch zum Programmiermodell von ObjectGrid

Hinweis

Vor Verwendung dieser Informationen müssen die allgemeinen Informationen im Abschnitt „Bemerkungen" auf Seite 223 gelesen werden.

- Die IBM Homepage finden Sie im Internet unter: **ibm.com**
- IBM und das IBM Logo sind eingetragene Marken der International Business Machines Corporation.
- Das e-business-Symbol ist eine Marke der International Business Machines Corporation.
- Infoprint ist eine eingetragene Marke der IBM.
- ActionMedia, LANDesk, MMX, Pentium und ProShare sind Marken der Intel Corporation in den USA und/oder anderen Ländern.
- C-bus ist eine Marke der Corollary, Inc. in den USA und/oder anderen Ländern.
- Java und alle auf Java basierenden Marken und Logos sind Marken der Sun Microsystems, Inc. in den USA und/oder anderen Ländern.
- Microsoft Windows, Windows NT und das Windows-Logo sind Marken der Microsoft Corporation in den USA und/oder anderen Ländern.
- PC Direct ist eine Marke der Ziff Communications Company in den USA und/oder anderen Ländern.
- SET und das SET-Logo sind Marken der SET Secure Electronic Transaction LLC.
- UNIX ist eine eingetragene Marke der Open Group in den USA und/oder anderen Ländern.
- Marken anderer Unternehmen/Hersteller werden anerkannt.

Erste Ausgabe (Oktober 2005)

Diese Veröffentlichung ist eine Übersetzung des Handbuchs
WebSphere Extended Deployment Version 6.0 ObjectGrid programming model guide,
herausgegeben von International Business Machines Corporation, USA

Informationen, die nur für bestimmte Länder Gültigkeit haben und für Deutschland, Österreich und die Schweiz nicht zutreffen, wurden in dieser Veröffentlichung im Originaltext übernommen.

Möglicherweise sind nicht alle in dieser Übersetzung aufgeführten Produkte in Deutschland angekündigt und verfügbar; vor Entscheidungen empfiehlt sich der Kontakt mit der zuständigen IBM Geschäftsstelle.

Änderung des Textes bleibt vorbehalten.

Herausgegeben von:
SW TSC Germany
Kst. 2877
Oktober 2005

Inhaltsverzeichnis

Hinweise zu Rückmeldungen

Ihre Rückmeldungen sind wichtig, damit eine bestmögliche Qualität der Informationen geliefert werden kann.

- Für Rückmeldungen zu Artikeln im WebSphere Extended Deployment Information Center unter der Adresse `http://www.ibm.com/software/webservers/appserv/extend/library/` gehen Sie folgendermaßen vor:
  1. Rufen Sie im Web-Browser den Artikel auf und blättern Sie an das Ende des Artikels.
  2. Verwenden Sie für Ihre Rückmeldung den Link **Feedback** am Ende des Artikels und senden Sie anschließend Ihre Rückmeldung ab.
- Bitte senden Sie Kommentare zu diesem oder anderen Handbüchern im PDF-Format per E-Mail an folgende Adresse: **wasdoc@us.ibm.com**.

  Geben Sie unbedingt den Namen und Nummer des Dokuments sowie ggf. die Seite, Tabelle oder Abbildung an.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Kapitel 1. Erste Schritte mit ObjectGrid

Dieser Abschnitt enthält eine Einführung in ObjectGrid, ein Framework für verteilte Datenverarbeitung, das Objekte verschiedenen Anwendungen zur Verfügung stellt.

WebSphere Extended Deployment Version 6.0 und WebSphere Application Server Version 6.0.2 oder höher müssen auf mindestens einer Maschine in Ihrer Umgebung installiert sein.

Einschränkung: Sie können ObjectGrid in einer Umgebung mit WebSphere Extended Deployment Version 6.0 einsetzen. ObjectGrid kann auch in Umgebungen mit Java 2 Platform, Standard Edition (J2SE) Version 1.4.2 und höher oder Umgebungen mit WebSphere Application Server Version 6.02 und höher verwendet werden, sofern bestimmte zusätzliche Lizenzvereinbarungen eingehalten werden. Nähere Einzelheiten hierzu können Sie bei Ihrem Vertriebsbeauftragten erfragen.

Wenn Sie ObjectGrid-Anwendungen entwickeln möchten, ohne auf Servermaschinen zuzugreifen, auf denen WebSphere Extended Deployment installiert ist, können Sie diese auf Ihrer lokalen Maschine ausführen. Auf der lokalen Maschine muss ein IBM Software Developer Kit (SDK) oder Eclipse installiert sein.

Wenn Sie ObjectGrid-Anwendungen auf Ihrer lokalen Maschine entwickeln möchten, kopieren Sie die Datei
<Installationsstammverzeichnis>/optionalLibraries/ObjectGrid/objectgridSA.jar
und die Datei
<Installationsstammverzeichnis>/optionalLibraries/ObjectGrid/objectgridSamples.jar
vom Server mit WebSphere Extended Deployment in ein Arbeitsverzeichnis auf Ihrer lokalen Maschine.

Verwenden Sie diese Task, um die ObjectGrid-Beispielanwendungen auszuführen und schrittweise durchzugehen. Sie können die in dieser Task beschriebenen Anwendungen in einer Java-Befehlszeile, in einer Eclipse- oder einer J2EE-Umgebung (Java 2 Platform, Enterprise Edition) ausführen.

- Informationen zum Ausführen einer ObjectGrid-Beispielanwendung in der Befehlszeile finden Sie im Abschnitt ″ObjectGrid-Beispielanwendung in der Befehlszeile ausführen″.
- Informationen zum Ausführen einer ObjectGrid-Beispielanwendung in der Eclipse-Umgebung finden Sie im Abschnitt ″ObjectGrid-Beispielanwendung in Eclipse importieren und verwenden″.
- Informationen zum Ausführen einer ObjectGrid-Beispielanwendung in der Umgebung von WebSphere Extended Deployment finden Sie im Abschnitt ″ObjectGrid-Beispielanwendung mit WebSphere Extended Deployment laden und ausführen″.

Jetzt können Sie anfangen, mit ObjectGrid zu arbeiten, indem Sie die Beispielanwendung ausführen und in Ihre Entwicklungsumgebung laden.

Zugehörige Konzepte

Kapitel 2, „ObjectGrid”, auf Seite 9
ObjectGrid ist ein erweiterbares, transaktionsorientiertes Framework für das

Zwischenspeichern (Caching) von Objekten für J2SE- (Java 2 Platform, Standard Edition) und J2EE-Anwendungen (Java 2 Platform, Enterprise Edition).

ObjectGrid-Beispielanwendung in der Befehlszeile ausführen

Dieser Artikel beschreibt, wie Sie ObjectGrid-Anwendungen in einer Java-Befehlszeile ausführen und Ihre ObjectGrid-Konfiguration testen.

Sie müssen ein Software Development Kit (SDK) installiert haben. Außerdem müssen Sie Zugriff auf die ObjectGrid-Beispielanwendungen haben. Nähere Informationen hierzu finden Sie im Artikel ″Erste Schritte mit ObjectGrid″.

Verwenden Sie diese Task, um einen schnellen Einstieg in die Ausführung von Anwendungen zu finden, die ObjectGrid unterstützen.

1. Überprüfen Sie Ihre SDK-Version. Für ObjectGrid benötigen Sie ein IBM SDK 1.4.2 oder höher. Führen Sie die folgenden Schritte aus, um Ihre Java-Umgebung zu testen, bevor Sie die ObjectGrid-Beispielanwendung ausführen:
   a. Öffnen Sie eine Eingabeaufforderung.
   b. Geben Sie den folgenden Befehl ein:

      ```
      java -version
      ```

      Wenn der Befehl ordnungsgemäß ausgeführt wird, erscheint ein Text wie der folgende:

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

      Sollte ein Fehler angezeigt werden, prüfen Sie, ob das SDK installiert und im Klassenpfad enthalten ist.
2. Führen Sie die ObjectGrid-Beispielanwendung aus. Die Beispielanwendung veranschaulicht einen einfachen Fall mit Mitarbeitern (employees), Niederlassungen (offices) und Arbeitsplätzen (work locations). Die Beispielanwendung erstellt eine ObjectGrid-Instanz mit Maps für jeden Objekttyp. In jede Map werden Einträge eingefügt und dort bearbeitet, um die Caching-Funktion von ObjectGrid zu veranschaulichen.
   a. Öffnen Sie eine Befehlszeile und navigieren Sie in das Arbeitsverzeichnis. Wenn Sie auf einer lokalen Maschine arbeiten, kopieren Sie die Dateien mit der Erweiterung `.jar` in ein Arbeitsverzeichnis. Wenn Sie WebSphere Extended Deployment verwenden, können Sie in das Verzeichnis <Installationsstammverzeichnis>/optionalLibraries/ObjectGrid navigieren.
   b. Setzen Sie den folgenden Befehl ab:

      ```
      cd Arbeitsverzeichnis
      java -cp "objectgridSA.jar;objectgridSamples.jar"
       com.ibm.websphere.samples.objectgrid.basic.ObjectGridSample
      ```

      Das System gibt Text wie den folgenden aus. Die Ausgabe wurde für diese Veröffentlichung gekürzt:

      ```
      Initializing ObjectGridSample ...
      resourcePath: META-INF/objectgrid-definition.xml
      objectgridUrl:
       jar:file:/C:/temp/objg/objectgridSample.jar!/
       META-INF/objectgrid-definition.xml
      EmployeeOptimisticCallback returning version object for employee
       = Perry Cheng, version = 0
      EmployeeOptimisticCallback returning version object for employee =
      ```

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

Sie können die ObjectGrid-Beispielanwendung in einer Java-Befehlszeile ausführen, um die Funktionalität von ObjectGrid zu testen.

Den Quellcode für diese Beispielanwendung finden Sie in der Datei objectgridSamples.jar und dort in der Datei com\ibm\websphere\samples\objectgrid\basic\ObjectGridSample.java.

ObjectGrid-Beispielanwendung in die Eclipse-Umgebung importieren und verwenden

Verwenden Sie diese Task, um die ObjectGrid-Beispielanwendung in die Eclipse-Umgebung zu importieren und auszuführen.

Verwenden Sie Eclipse Version 3.0 oder höher, um diese Beispielanwendung zu importieren und auszuführen. Eclipse ist im Application Server Toolkit, das mit

WebSphere Application Server bereitgestellt wird, und in Rational Application Developer enthalten. Sie können Eclipse aber auch direkt von der Website Eclipse-.org herunterladen.

Mit Eclipse können Sie Ihre Anwendungen auf einfache Weise testen und die Beispielanwendung schrittweise durchgehen.

1. Importieren Sie das Projekt in Eclipse:
   a. Führen Sie das Programm Eclipse aus. Verwenden Sie dazu die Datei `eclipse.exe` im Installationsverzeichnis von Eclipse.
   b. Erstellen Sie mit Eclipse ein neues Projekt.
      1) Klicken Sie auf **Datei > Neu > Projekt > Java > Java-Projekt**. Klicken Sie auf **Weiter**.
      2) Geben Sie einen Projektnamen ein, z. B. `ObjectGridSamples`.
      3) Wählen Sie **Projekt im Arbeitsbereich erstellen** aus.
      4) Klicken Sie im Abschnitt "Projektlayout" auf **Standardwerte konfigurieren**.
      5) Wählen Sie unter "Quellen- und Ausgabeordner" **Projekt** aus und klicken Sie auf **OK**.
      6) Klicken Sie auf **Weiter**.
      7) Klicken Sie auf das Register **Bibliotheken**.
      8) Klicken Sie auf **Externe JARs hinzufügen**.
      9) Navigieren Sie in den Ordner `<Installationsstammverzeichnis>/optionalLibraries/ObjectGrid` und wählen Sie die Datei **ObjectGridSA.jar** aus. Klicken Sie im Assistenten **JAR-Auswahl** auf **Öffnen**.
      10) Klicken Sie auf **Fertig stellen**.
2. Importieren Sie die Datei `objectgridSamples.jar` in das Java-Projekt.
   a. Klicken Sie mit der rechten Maustaste auf das Java-Projekt und wählen Sie **Importieren** aus.
   b. Wählen Sie unter "Importquelle auswählen" den Eintrag **Komprimierte Datei (ZIP)** aus.
   c. Klicken Sie auf **Weiter**.
   d. Klicken Sie auf **Durchsuchen**, um den Assistenten "Aus ZIP-Datei importieren" zu öffnen.
   e. Navigieren Sie in das Verzeichnis `<Installationsstammverzeichnis>/optionalLibraries/ObjectGrid`. Wählen Sie die Datei **objectgridSamples.jar** aus und klicken Sie auf **Öffnen**.
   f. Vergewissern Sie sich, dass das Markierungsfeld für den Stammdateibaum ausgewählt ist.
   g. Vergewissern Sie sich, dass im Feld **In Ordner** das Java-Projekt angezeigt wird, das Sie im vorherigen Schritt erstellt haben, z. B. `ObjectGridSamples`.
   h. Klicken Sie auf **Fertig stellen**.
3. Überprüfen Sie die Merkmale des Java-Projekts.
   a. Öffnen Sie die Java-Perspektive. Klicken Sie auf **Fenster > Perspektive öffnen > Java**.
   b. Wechseln Sie in die Konsolsicht. Klicken Sie auf **Fenster > Sicht anzeigen > Konsole**.
   c. Vergewissern Sie sich, dass die Sicht "Paket-Explorer" verfügbar und ausgewählt ist. Klicken Sie auf **Fenster > Sicht anzeigen > Paket-Explorer**.

d. Klicken Sie mit der rechten Maustaste auf das Java-Projekt und wählen Sie **Eigenschaften** aus.
e. Klicken Sie in der linken Anzeige auf **Java-Erstellungspfad**.
f. Klicken Sie in der rechten Anzeige auf das Register **Quelle**.
g. Vergewissern Sie sich, dass unter "Quellenordner im Erstellungspfad" das Projektstammverzeichnis aufgelistet ist.
h. Klicken Sie in der rechten Anzeige auf **Bibliotheken**.
i. Vergewissern Sie sich, dass unter "JARs und Klassenordner im Erstellungspfad" die Datei `objectgridSA.jar` und eine JRE-Systembibliothek aufgelistet sind.
j. Klicken Sie auf **OK**.

4. Führen Sie das ObjectGrid-Beispiel aus.
   a. Erweitern Sie in der Sicht "Paket-Explorer" das Java-Projekt.
   b. Erweitern Sie das Paket com.ibm.websphere.samples.objectgrid.basic.
   c. Klicken Sie mit der rechten Maustaste auf die Datei **ObjectGridSample.java**. Klicken Sie auf **Ausführen > Java-Anwendung**.
   d. In der Konsole erscheint eine ähnliche Ausgabe wie bei der Ausführung der Anwendung in der Java-Befehlszeile. Eine Beispielausgabe finden Sie im Abschnitt ObjectGrid-Beispielanwendung in der Befehlszeile ausführen.

Die Schritte zum Laden des Projekts in die Eclipse-Umgebung und zum Ausführen des Debugger für die Datei `objectGridSamples.jar` sind auch in der Datei `SamplesGuide.htm` beschrieben. Sie finden die Datei `SamplesGuide.htm` im Verzeichnis `<Installationsstammverzeichnis>/optionalLibraries/ObjectGrid/doc`.

ObjectGrid-Beispielanwendung mit WebSphere Extended Deployment laden und ausführen

Verwenden Sie diese Task, um die ObjectGrid-J2EE-Beispielanwendung in WebSphere Extended Deployment zu laden und auszuführen.

WebSphere Application Server und WebSphere Extended Deployment müssen installiert sein.

Verwenden Sie diese, um sich mit der Integration von ObjectGrid in WebSphere Extended Deployment vertraut zu machen und diese zu testen. Nähere Informationen finden Sie in Kapitel 6, „ObjectGrid mit WebSphere Application Server integrieren", auf Seite 145.

1. Installieren Sie die Datei `ObjectGridSample.ear`. Sie können die EAR-Datei (Enterprise Archive) in einem einzelnen Anwendungsserver oder in einem Cluster installieren. Führen Sie zum Installieren der Datei `ObjectGridSample.ear` in der Administrationskonsole die folgenden Schritte aus:
   a. Klicken Sie in der Administrationskonsole auf **Anwendungen > Neue Anwendung installieren**.
   b. Geben Sie auf der Seite **Vorbereitung der Anwendungsinstallation** die Position der ObjectGrid-Beispielanwendung ein. Navigieren Sie beispielsweise zur Datei `<Installationsstammverzeichnis>/installableApps/ObjectGridSample.ear`. Klicken Sie auf **Weiter**.
   c. Übernehmen Sie auf der zweiten Seite **Anwendungsinstallation vorbereiten** die Standardeinstellungen und klicken Sie auf **Weiter**.

d. Übernehmen Sie auf der Seite **Installationsoptionen auswählen** die Standardeinstellungen und klicken Sie auf **Weiter**.

e. Geben Sie auf der Seite **Servern Module zuordnen** die Implementierungsziele an, in denen Sie die in Ihrer Anwendung enthaltenen Module installieren möchten. Wählen Sie in der Liste **Cluster und Server** für jedes Modul einen Zielserver bzw. Cluster aus. Wählen Sie das Markierungsfeld **Modul** aus, wenn Sie alle Anwendungsmodule auswählen möchten, oder wählen Sie einzelne Module aus.

f. Übernehmen Sie auf der folgenden Seite die Standardwerte und klicken Sie anschließend auf **Fertig stellen**.

g. Klicken Sie nach Abschluss der Anwendungsinstallation auf **In Master-Konfiguration speichern**.

h. Klicken Sie auf die Option **Änderungen mit Knoten synchronisieren**. Klicken Sie auf der Seite **Enterprise-Anwendungen > Speichern** auf **Speichern**.

i. Klicken Sie auf **OK**.

2. Wählen Sie den HTTP-Port des virtuellen Hosts default_host für die Server aus und fügen Sie einen Hostalias hinzu. Standardmäßig werden Webmodule an den virtuellen Host default_host gebunden, sofern Sie den Hostnamen während der Installation nicht ändern. Wenn Sie die Anwendung in einem Cluster installieren, müssen Sie für jedes Cluster-Member mindestens einen Hostalias für den HTTP-Port konfigurieren. Außerdem müssen Sie den HTTP-Port von default_host für jedes Cluster-Member auswählen und in der Administrationskonsole den entsprechenden Hostalias der Liste der Hostaliasnamen hinzufügen. Führen Sie zum Auswählen des HTTP-Port für den virtuellen Host default_host eines Servers die folgenden Schritte aus:

   a. Klicken Sie in der Administrationskonsole auf **Server > Anwendungsserver > *Servername***.

   b. Erweitern Sie im Abschnitt ″Kommunikation″ den Abschnitt ″Ports″. Der Port **WC_defaulthost** ist der Port des virtuellen Hosts default_host.

   Führen Sie zum Hinzufügen eines Hostalias die folgenden Schritte aus:

   a. Klicken Sie in der Administrationskonsole auf **Umgebung > Virtuelle Hosts > default_host > Hostaliasnamen > Neu**.

   b. Verwenden Sie den Standardwert für den Hostnamen und geben Sie den Port an.

   c. Klicken Sie auf **OK**.

3. Starten Sie die ObjectGrid-Beispielanwendung.

   - Zum Starten der Anwendung in einem Server klicken Sie auf **Server > Anwendungsserver**. Wählen Sie den Server aus, in dem die Datei `ObjectGridSample.ear` installiert ist. Klicken Sie auf **Starten**.
   - Zum Starten der Anwendung in einem Cluster klicken Sie auf **Server > Cluster**. Wählen Sie den Cluster aus, in dem die Datei `ObjectGridSample.ear` installiert ist. Klicken Sie auf **Starten**.

   Nachdem Sie die Anwendung in einem Server oder Cluster gestartet haben, können Sie die Anwendung unabhängig vom Hostserver oder Cluster starten und stoppen. Führen Sie zum Stoppen oder Starten der ObjectGrid-Beispielanwendung die folgenden Schritte aus:

   a. Klicken Sie in der Administrationskonsole auf **Anwendungen > Enterprise-Anwendungen**.

   b. Wählen Sie die ObjectGrid-Beispielanwendung aus.

   c. Klicken Sie auf **Starten** bzw. **Stoppen**.

4. Rufen Sie die ObjectGrid-Beispielanwendung auf. Nachdem Sie die Datei `ObjectGridSample.ear` in einem Server oder Cluster installiert und die Anwendung gestartet haben, können Sie die ObjectGrid-Beispielanwendung mit der folgenden Webadresse aufrufen:

   `http://`*`Hostname`*`:`*`Port`*`/ObjectGridSample`

   Wenn Ihr Host beispielsweise den Namen `localhost` und den Port `9080` hat, verwenden Sie die Webadresse `http://localhost:9080/ObjectGridSample`.

Sie haben die ObjectGrid-Beispielanwendung in WebSphere Extended Deployment installiert und konfiguriert.

Nachdem Sie die Anwendung in einem Server oder Cluster installiert haben, können Sie nach dem Starten der Anwendung mit der folgenden Webadresse auf die Dokumentation zur Beispielanwendung zugreifen:

`http://`*`Hostname`*`:`*`Port`*`/ObjectGridSample/docs/introduction.html`

Wenn Ihr Host beispielsweise den Namen **localhost** und den Port **9080** hat, verwenden Sie die Webadresse `http://localhost:9080/ObjectGridSample/docs/introduction.html`.

Kapitel 2. ObjectGrid

ObjectGrid ist ein erweiterbares, transaktionsorientiertes Framework für das Zwischenspeichern (Caching) von Objekten für J2SE- (Java 2 Platform, Standard Edition) und J2EE-Anwendungen (Java 2 Platform, Enterprise Edition).

Sie können die ObjectGrid-API bei der Entwicklung Ihrer Anwendungen einsetzen, um Objekte im ObjectGrid-Framework abzurufen, zu speichern, zu löschen und zu aktualisieren. Außerdem können Sie angepasste Plug-ins implementieren, die Aktualisierungen des Cache überwachen, Daten aus externen Datenquellen abrufen und dort speichern, das Löschen von Einträgen aus dem Cache verwalten oder Cache-Funktionen für Ihre eigene ObjectGrid-Anwendungsumgebung im Hintergrund ausführen.

Map-basierte API

ObjectGrid stellt eine API bereit, die auf dem Java-Interface Map basiert. Die erweiterte API unterstützt die Gruppierung von Operationen zu transaktionsorientierten Blöcken. Sie können einer Gruppe von Schlüsselwörtern einen Schlüssel zuordnen, und alle Objekte, denen ein Schlüsselwort zugeordnet ist, können jederzeit aus der Map-Instanz gelöscht werden. Dieses Interface ist ein Superset des Interface Map und bietet zusätzlich Unterstützung für Stapeloperationen, Invalidierung, Zuordnung von Schlüsselwörtern sowie explizites Einfügen und Aktualisieren. Der Semantik des Java-Interface Map wurden Erweiterungspunkte hinzugefügt, damit Sie die folgenden funktionalen Erweiterungen implementieren können:

- Cache-Bereinigungsfunktionen (Evictor), mit denen die Lebensdauer von Einträgen im Cache optimiert werden kann,
- Callback-Interfaces für Transaktionen, mit denen die Transaktionsverwaltung sorgfältig gesteuert werden kann,
- Implementierungen von Ladeprogrammen, die Daten aus fernen Datenquellen abrufen und dort speichern können,
- Listener-Interfaces, die Informationen über alle festgeschriebenen Transaktionen liefern, wenn diese stattfinden und auf das ObjectGrid-Framework oder bestimmte Map-Instanzen angewendet werden.

ObjectGrid-Umgebung

Sie können das ObjectGrid-Framework mit oder ohne WebSphere Application Server und WebSphere Extended Deployment verwenden. ObjectGrid kann in einer Umgebung mit J2SE Version 1.4.2 und höher eingesetzt werden. Außerdem wird die Verwendung von ObjectGrid in einer Umgebung mit WebSphere Application Server Version 6.0.2 und höher unterstützt. Wenn Sie die Unterstützung von WebSphere Application Server verwenden, können weitere Unterstützungsfunktionen für verteilte Transaktionen genutzt werden. Die Unterstützungsfunktionen für verteilte Transaktionen können in jeder Umgebung verwendet werden, die ein zuverlässiges Publish/Subscribe-Messaging-System wie beispielsweise einen JMS-Provider (Java Message Service) besitzt.

Einschränkung: Sie können ObjectGrid in einer Umgebung mit WebSphere Extended Deployment Version 6.0 einsetzen. ObjectGrid kann auch in Umgebungen mit Java 2 Platform, Standard Edition (J2SE) Version 1.4.2 und höher oder Umgebungen mit WebSphere Application

Server Version 6.02 und höher verwendet werden, sofern bestimmte zusätzliche Lizenzvereinbarungen eingehalten werden. Nähere Einzelheiten hierzu können Sie bei Ihrem Vertriebsbeauftragten erfragen.

Einfache Installation

ObjectGrid lässt sich in wenigen einfachen Schritten installieren und konfigurieren. Zu diesen Schritten gehören das Kopieren der JAR-Dateien (Java-Archiv) in Ihren Klassenpfad und das Definieren einiger Konfigurationsanweisungen.

Transaktionsorientierte Änderungen

Alle Änderungen werden im Kontext einer Transaktion vorgenommen, um eine stabile Programmierschnittstelle zu bieten. Die Transaktion kann explizit innerhalb der Anwendung gesteuert werden, oder die Anwendung kann den Programmiermodus für automatische Festschreibung verwenden. Diese transaktionsorientierten Änderungen können zur Synchronisation der ObjectGrid-Peers auch an andere Anwendungsserver weitergegeben werden.

Sie können ObjectGrid von einem einfachen Grid in einer einzelnen Java Virtual Machine (JVM) zu einem Grid mehrerer Java Virtual Machines skalieren, die eine Gruppe von Objekten gemeinsam nutzen, und die Änderungen an den Objekten überwachen, die von den Peer-JVMs vorgenommen werden. Für die Synchronisation von ObjectGrid in den verschiedenen Instanzen von WebSphere Application Server können Sie die Unterstützungsfunktionen für verteilte Transaktionen verwenden oder eine ähnliche Lösung für die Verteilung von Nachrichten wie beispielsweise Java Message Service (JMS) implementieren.

Wenn Sie die Unterstützungsfunktionen für verteilte Transaktionen verwenden, können die Peers mit dem Messaging-Protokoll über Änderungen informiert werden. Sie können die Java Virtual Machines auch so konfigurieren, dass die von Peer-Maschinen geänderten Objekte ungültig gemacht werden oder eine lokale Kopie durch die Kopie mit den Änderungen, die auf einer fernen Maschine vorgenommen wurden, ersetzt wird. Nähere Informationen finden Sie im Abschnitt „Weitergabe von verteilten Transaktionen in ObjectGrid" auf Seite 156.

Mit dem Injection-Container kompatible APIs

Sie können ObjectGrid mit einer einfachen XML-Datei oder über das Programm mit Java-APIs konfigurieren. Die Java-APIs sind so konzipiert, dass sie auch in Umgebungen funktionieren, in denen Sie Injection-basierte Frameworks für die Konfiguration Ihrer Anwendungen verwenden. Die APIs und Interfaces der ObjectGrid-Objekte können auch von einem IoC-Container (Inversion of Control) aufgerufen werden. Anschließend können Referenzen auf ObjectGrid-Schlüsselobjekte in die Anwendung eingespeist (engl. inject) werden.

Verteilter Cache

ObjectGrid unterstützt in der ersten Version ein einfaches Push-basiertes Modell für die Weitergabe von Änderungen. Wenn eine Transaktion in ObjectGrid festgeschrieben wird, kann diese Gruppe von Änderungen mit einem Messaging-Protokoll an ObjectGrid-Peers weitergegeben werden. Die Änderungen können als Nachricht des Typs Invalidierung weitergegeben. Es ist aber auch möglich, die neuen Werte im Push-Verfahren an die Peer-Grids zu verteilen. ObjectGrid kann so konfiguriert werden, dass nur die Änderungen für bestimmte Maps verteilt wer-

den. Als Messaging-Protokoll kann der High Availability Manager von WebSphere Application Server Version 6.x oder jedes andere Messaging-Protokoll verwendet werden, das dem Benutzer in anderen Umgebungen zur Verfügung steht. Diese Aktualisierungen können auf Peer-Caches angewendet oder sie können bedingt angewendet werden, um zu verhindern, dass eine aktuellere Kopie in einem Peer-Cache überschrieben oder ungültig gemacht wird.

Erweiterbare Architektur

Die meisten Elemente des ObjectGrid-Framework können mit Plug-ins erweitert werden. Sie können das ObjectGrid-Framework so optimieren, dass eine Anwendung bei ihren Entscheidungen zwischen Konsistenz und Leistung abwägen kann. Mit angepasstem Plug-in-Code können auch die folgenden anwendungsspezifischen Verhalten unterstützt werden:

- Ereignisse bezüglich Initialisierung, Transaktionsbeginn, Transaktionsende und Löschvorgängen in ObjectGrid-Instanzen überwachen,
- COMMIT-Operationen für Transaktionen mit Listener-Interfaces für die einzelnen Map-Instanzen überwachen,
- Callback-Interfaces für Transaktionen aufrufen, um eine transaktionsspezifische Verarbeitung zu aktivieren,
- spezielle Transaktions-Policys für generische ObjectGrid-Transaktionen implementieren,
- Ladeprogramme für transparente und allgemeine Eingangs- und Ausgangspunkte für externe Datenspeicher und andere Informations-Repositorys verwenden,
- nicht serialisierbare Objekte mit ObjectTransformer-Interfaces auf eine spezielle Weise verarbeiten.

Sie können jedes dieser Verhalten implementieren, ohne die Verwendung der grundlegenden API-Interfaces für ObjectGrid-Caches zu beeinträchtigen. Mit dieser Transparenz ist es möglich, die Datenspeicher- und Transaktionsverarbeitung von Anwendungen, die die Cache-Infrastruktur verwenden, ohne Beeinträchtigung in großem Umfang zu ändern.

ObjectGrid als primäre API oder Cache der zweiten Ebene verwenden

Die ObjectGrid-APIs können von der Anwendung direkt als Lookaside-Cache oder Durchschreib-Cache verwendet werden. Im Durchschreibmodus implementiert die Anwendung ein Loader-Objekt, so dass die ObjectGrid-Instanz transparent für die Anwendung Änderungen anwenden und Daten direkt abrufen kann. ObjectGrid kann für gängige objektrelationale Zuordnungsfunktionen (Mapper) auch als Cache der zweiten Ebene verwendet werden. Hierfür müssen Sie einen Adapter schreiben. In diesem Modus ist der Cache für die Anwendung nicht sichtbar, weil die Anwendung die APIs des objektrelationalen Mapper als primäre API für den Zugriff auf die Daten verwendet.

Der Artikel Einführung in ObjectGrid enthält Informationen zur Verwendung und Entwicklung von ObjectGrid-Anwendungen.

Zugehörige Tasks

Erste Schritte mit ObjectGrid
Dieser Abschnitt enthält eine Einführung in ObjectGrid, ein Framework für verteilte Datenverarbeitung, das Objekte verschiedenen Anwendungen zur Verfügung stellt.

Kapitel 3. Lerntext zu ObjectGrid: Modell für Anwendungsprogrammierung

Verwenden Sie diese Task, um sich mit dem ObjectGrid-Modell für die Anwendungsprogrammierung vertraut zu machen.

Bereiten Sie Ihre Umgebung für die Ausführung von ObjectGrid-Anwendungen vor. In Kapitel 1, „Erste Schritte mit ObjectGrid", auf Seite 1 wird beschrieben, wo Sie die JAR-Dateien finden, welche Java-Voraussetzungen erfüllt sein müssen und wie Sie durch Ausführen einer einfachen Datei prüfen können, ob Ihre Umgebung ordnungsgemäß konfiguriert ist.

Legen Sie fest, welche Programmierumgebung Sie für diese Task verwenden möchten. Sie können eine integrierte Entwicklungsumgebung (IDE, Integrated Development Environment) wie Eclipse, aber auch die Java-Befehlszeile verwenden. Wenn Sie ein wenig vertrauter mit ObjectGrid sind, binden Sie ObjectGrid in Enterprise-Beans und Servlets ein. Die Beispiele im Lerntext setzen keine bestimmte Java-Umgebung voraus. Sie können also jede Umgebung verwenden, mit der Sie vertraut sind.

In der Grunddefinition ist ObjectGrid ein Cache und ein im Hauptspeicher befindliches Repository für Objekte. Die Verwendung von ObjectGrid ist mit der Verwendung einer Map des Typs java.util.Map zum Speichern von und Zugreifen auf Objekte vergleichbar. ObjectGrid ist jedoch mehr als nur ein Cache. Wenn Sie die in dieser Task beschriebenen Features und Plug-ins ausprobieren, werden Sie feststellen, dass ObjectGrid in hohem Maße erweiterbar und flexibel ist. Sie können ObjectGrid als einfachen *Lookaside*-Cache oder komplexeren Cache, der von einem Ressourcenmanager gestützt wird, einsetzen.

Die Beispiele in diesem Lerntext sind keine vollständigen Programme. Importoperationen, Ausnahmebehandlung und selbst einige Variablen sind nicht in jedem Beispiel vollständig deklariert. Sie können diese Beispiele als Vorlage für das Schreiben eigener Programme verwenden.

Verwenden Sie diese Task, um ObjectGrid in einem Java-Programm zu verwenden.

1. Suchen Sie die ObjectGrid-APIs und -Ausnahmen. Alle allgemein zugänglichen (public) ObjectGrid-APIs und -Ausnahmen sind im Paket `com.ibm.websphere.objectgrid` enthalten. Wenn Sie weiterführende Informationen zum System und zur Konfiguration benötigten, sehen Sie sich die zusätzlichen APIs und Ausnahmen im Paket `com.ibm.websphere.objectgrid.plugins` an. Werden Plug-in-Implementierungen bereitgestellt, finden Sie die zugehörigen Klassen im Paket `com.ibm.websphere.objectgrid.plugins.builtins`. Die Sicherheits-Features von ObjectGrid finden Sie in den Paketen, die **security** in ihrem Namen enthalten, wie z. B. `com.ibm.websphere.objectgrid.security`, `com.ibm.websphere.objectgrid.security.plugins` und `com.ibm.websphere.objectgrid.security.plugins.builtins`.

   Diese Task konzentriert sich auf die APIs, die im Paket `com.ibm.websphere.objectgrid` enthalten sind. Die vollständige JavaDoc zu ObjectGrid finden Sie im Pfad `<Installationsstammverzeichnis>/web/xd/apidocs`.

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

2. Rufen Sie eine ObjectGrid-Instanz ab oder erstellen Sie eine solche. Verwenden Sie ObjectGridManagerFactory zum Abrufen einer Singleton-Instanz von ObjectGridManager. Erstellen Sie anschließend mit den folgenden Anweisungen eine ObjectGrid-Instanz:

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

   Das Interface ObjectGridManager besitzt verschiedene Methoden für das Erstellen, Abrufen und Entfernen von ObjectGrid-Instanzen. Lesen Sie den Abschnitt „Interface ObjectGridManager" auf Seite 37, um eine Variante für Ihre spezielle Situation auszuwählen. Mit dem Interface ObjectGridManager können auch Trace-Einstellungen festgelegt werden. Wenn Sie mit WebSphere Extended Deployment oder WebSphere Application Server arbeiten, sind diese Methoden nicht erforderlich, weil der Trace von den in diesen Produkten integrierten Funktionen verwaltet wird. Diese Methoden können hilfreich sein, wenn Sie nicht mit WebSphere Application Server arbeiten. Ausführliche Informationen zu diesen Methoden finden Sie im Abschnitt „Trace für ObjectGrid konfigurieren" auf Seite 44.

3. Initialisieren Sie ObjectGrid.
   a. Legen Sie einen Namen für die ObjectGrid-Instanz fest, falls Sie den Namen nicht mit den `create`-Methoden angegeben haben.
   b. Definieren Sie die BackingMap-Instanzen. Verwenden Sie dazu für Ihre ersten Anwendungen die Standardkonfiguration für eine BackingMap.
   c. Nachdem Sie Ihre BackingMaps definiert haben, initialisieren das ObjectGrid. Mit der Initialisierung des ObjectGrid signalisieren Sie, dass die Konfiguration abgeschlossen ist und Sie mit der Verwendung von ObjectGrid beginnen möchten.
   d. Rufen Sie nach der Initialisierung des ObjectGrid ein Session-Objekt ab. Nähere Informationen finden Sie im Abschnitt „Interface ObjectGrid" auf Seite 44 und in der JavaDoc.

   Verwenden Sie in diesem Schritt das folgende Beispiel als Anleitung:

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

4. Verwenden Sie Sitzungen für die Verwaltung transaktionsorientierter Operationen. Alle Zugriffe auf ein ObjectGrid sind transaktionsorientiert. Wenn Sie mehrere Zugriffe, Einfüge-, Aktualisierungs- oder Entfernungsoperationen für Objekte im Cache durchführen, sind diese in einer Arbeitseinheit, einer so genannten Sitzung (oder Session) enthalten. Am Ende einer Sitzung können Sie alle Änderungen, die innerhalb dieser Arbeitseinheit vorgenommen wurden, festschreiben (COMMIT) oder rückgängig machen (ROLLBACK) bzw. verwerfen.

Für einzelne autarke Cache-Operationen können Sie auch mit automatischer Festschreibung arbeiten. Wenn kein aktiver Sitzungskontext vorhanden ist, werden einzelne Zugriffe auf den Cache-Inhalt in eigene automatisch festgeschriebene Sitzungen eingeschlossen.

Ein weiterer wichtiger Aspekt des Interface Session ist der, dass mit dem Interface ObjectMap eine transaktionsorientierter Zugriff auf die BackingMap erfolgen kann. Mit der Methode getMap können Sie ein ObjectMap-Handle für eine vordefinierte BackingMap erstellen. Alle Operationen, die mit dem Cache ausgeführt werden, wie z. B. Einfüge-, Aktualisierungs- oder Löschoperationen, werden mit der ObjectMap-Instanz ausgeführt. Nähere Informationen finden Sie im Abschnitt „Interface Session" auf Seite 51. Verwenden Sie das folgende Beispiel, um eine Sitzung abzurufen und zu verwalten:

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

5. Verwenden Sie das Interface ObjectMap, um auf den Cache zuzugreifen und diesen zu aktualisieren. Wenn Sie sich mit dem Interface ObjectMap beschäftigen, schauen Sie sich die verschiedenen Methoden für den Zugriff und die Aktualisierung des Cache an. Das Interface ObjectMap ist wie ein Map-Interface aufgebaut. Es enthält jedoch neue geprüfte Ausnahmen, die Sie bei der Entwicklung von ObjectGrid-Anwendungen mit einer IDE wie Eclipse unterstützen. Wenn Sie ein Interface java.util.Map ohne geprüfte Ausnahmen verwenden möchten, können Sie die Methode getJavaMap verwenden. Nähere Informationen finden Sie im Abschnitt „Interfaces ObjectMap und JavaMap" auf Seite 54.

   Mit den expliziten Methoden insert und update können Sie die vage Operation put umgehen. Sie können die Methode put zwar weiterhin verwenden, aber mit den expliziten Methoden insert und update können Sie Ihre Absichten klarer übermitteln. Die Verwendung der Methode put kann transparenter gestaltet werden, indem eine Methode put ohne vorausgehende Operation get als Einfügeoperation definiert wird. Wenn Sie vor der Operation put eine Operation get ausführen, wird die Operation put als Einfügeoperation behandelt, wenn der Eintrag nicht vorhanden ist, bzw. als Aktualisierungsoperation, wenn der Eintrag vorhanden ist.

   Sie können die folgenden ObjectMap-Basisoperationen ausführen: get, put, insert, update, remove, touch, invalidate und containsKey. Ausführliche Informationen und Varianten finden Sie im Artikel „Übersicht über das Systemprogrammiermodell" auf Seite 16 und in der Dokumentation zur ObjectMap-API. Das folgende Beispiel veranschaulicht, wie mit dem Interface ObjectMap der Cache geändert wird:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid objectGrid =
 objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
ObjectMap objectMap = session.getMap("someMap");
// Start a transaction/session...
session.begin();
objectMap.insert("key1", "value1");
objectMap.put("key2", "value2");
session.commit();
// Verify changes did commit
```

```
String value1 = (String)objectMap.get("key1");
String value2 = (String)objectMap.get("key2");
System.out.println("key1 = " + value1 + ", key2 = " + value2);
//Start a new transaction/session...
session.begin();
objectMap.update("key2", "newValue2");
objectMap.remove("key1");
session.rollback();
// Verify changes didn't commit
String newValue1 = (String)objectMap.get("key1");
String newValue2 = (String)objectMap.get("key2");
System.out.println("key1 = " + newValue1 + ", key2 = " + newValue2);
```

Wenn Sie diesen Abschnitt gelesen und mit dem Beispielcode experimentiert haben, sind Sie mit dem grundlegenden Programmiermodell von ObjectGrid vertrauter.

Ausführliche Informationen finden Sie in Kapitel 5, „Übersicht über die Anwendungsprogrammierschnittstelle von ObjectGrid", auf Seite 37.

Übersicht über das Systemprogrammiermodell

Das Systemprogrammiermodell enthält weitere Features und Erweiterungspunkte für ObjectGrid.

Die folgende Abbildung veranschaulicht, wie das Systemprogrammiermodell die zusätzlichen Features und Erweiterungspunkte zur Verfügung stellt.

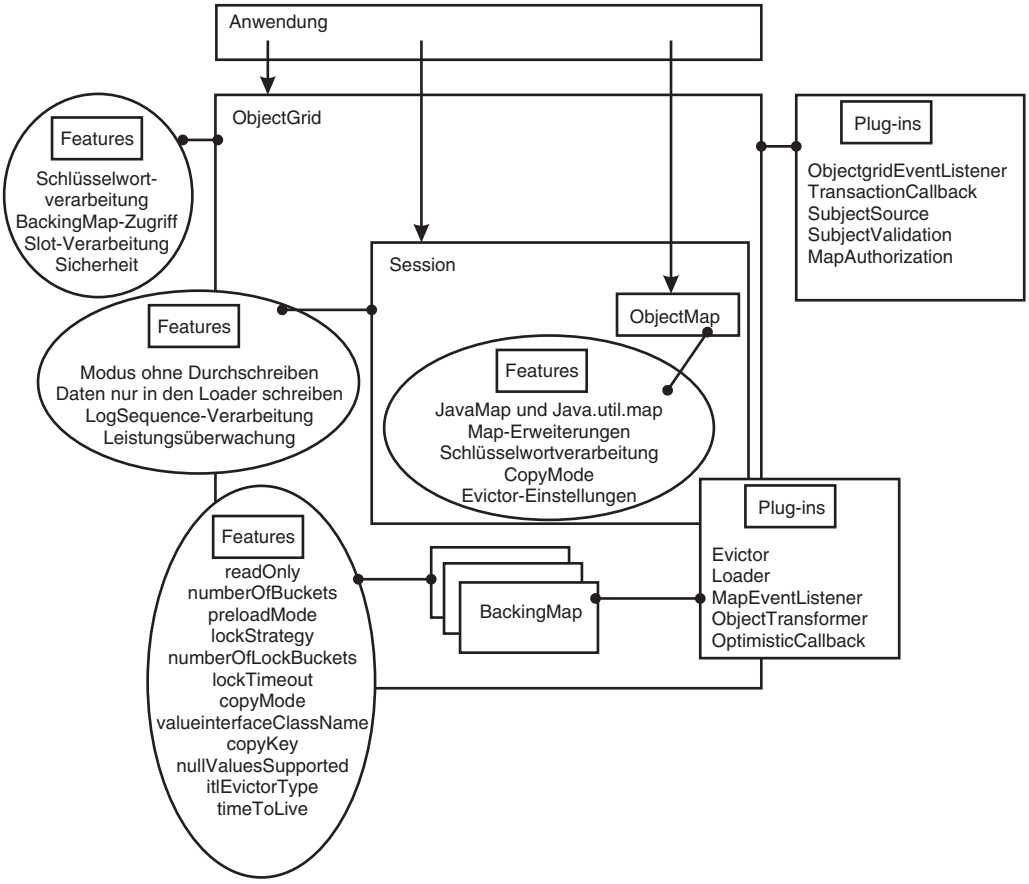


Abbildung 1. Übersicht über ObjectGrid

In ObjectGrid ist ein Plug-in eine Komponente, die den modular aufgebauten ObjektGrid-Komponenten wie ObjectGrid und BackingMap einen bestimmten Typ von Funktion bereitgestellt. Ein Feature ist eine bestimmte Funktion oder ein bestimmtes Merkmal einer ObjectGrid-Komponente wie ObjectGrid, Session, BackingMap, ObjectMap usw. Wenn ein Feature eine Funktion darstellt, kann es verwendet werden, um ein bestimmtes Datenverarbeitungsziel zu erreichen. Wenn ein Feature ein Merkmal darstellt, kann es verwendet werden, um das Verhalten der ObjectGrid-Komponenten zu optimieren.

In den folgenden Abschnitten werden einige Features und Erweiterungen beschrieben, die im vorherigen Diagramm gezeigt werden:

- „Übersicht über das Systemprogrammiermodell: Plug-in-Punkte und Features für das Interface ObjectGrid" auf Seite 18

  Das Interface ObjectGrid hat mehrere Plug-in-Punkte und Features für weitergehende Interaktionen mit dem ObjectGrid-Framework.
- „Übersicht über das Systemprogrammiermodell: Plug-in-Punkte und Features für das Interface BackingMap" auf Seite 20

  Das Interface BackingMap hat mehrere optionale Plug-in-Punkte und Features für weitergehende Interaktionen mit dem ObjectGrid-Framework.
- „Übersicht über das Systemprogrammiermodell: Features für das Interface Session" auf Seite 27

Das Interface Session hat mehrere Features für weitergehende Interaktionen mit dem ObjectGrid-Framework. Die einzelnen Unterabschnitte beschreiben diese Features und enthalten einige kurze Codefragmente für das Verwendungsszenario.

- „Übersicht über das Systemprogrammiermodell: Features für das Interface ObjectMap" auf Seite 29

  Das Interface ObjectMap hat mehrere Features für weitergehende Interaktionen mit dem ObjectGrid-Framework. Die einzelnen Unterabschnitte beschreiben diese Features und enthalten einige kurze Codefragmente für das Verwendungsszenario.

Weitere Informationen zu den einzelnen Features und Plug-ins finden Sie in Kapitel 5, „Übersicht über die Anwendungsprogrammierschnittstelle von ObjectGrid", auf Seite 37.

Übersicht über das Systemprogrammiermodell: Plug-in-Punkte und Features für das Interface ObjectGrid

Das Interface ObjectGrid hat mehrere Plug-in-Punkte und Features für weitergehende Interaktionen mit dem ObjectGrid-Framework.

Die folgenden Unterabschnitte beschreiben diese Features und enthalten einige kurze Code-Snippets für das Verwendungsszenario. An den entsprechenden Stellen wird ein XML-Snippet gezeigt, um die alternative XML-Konfiguration zu veranschaulichen. Ausführliche Informationen finden Sie in den Abschnitten „Interface ObjectGrid" auf Seite 44 und „ObjectGrid-Konfiguration" auf Seite 130.

Schlüsselwortverarbeitung

Das Interface ObjectGrid stellt einen flexiblen Invalidierungsmechanismus bereit, der auf Schlüsselwörtern basiert. Ein Schlüsselwort ist eine Instanz eines serialisierbaren Objekts, die nicht null ist. Sie können BackingMap-Einträgen Schlüsselwörter zuordnen. Der größte Teil der Schlüsselwortverarbeitung wird auf ObjectMap-Ebene durchgeführt, aber die Zuordnung eines Schlüsselwortes zu einem anderen Schlüsselwort mit dem Ziel, eine hierarchische Struktur von Schlüsselwörtern zu erstellen, wird auf ObjectGrid-Ebene durchgeführt.

Die Methode associateKeyword(java.io.Serializable parent, java.io.Serializable child) erstellt eine gerichtete Beziehung zwischen den beiden Schlüsselwörtern. Wenn ein übergeordnetes Objekt ungültig gemacht wird, wird auch das untergeordnete Objekt ungültig gemacht. Wird das untergeordnete Objekt ungültig gemacht, hat dies keine Auswirkung auf das übergeordnete Objekt. Mit der folgenden Methode wird beispielsweise der Map-Eintrag New York als untergeordneter Eintrag für den Map-Eintrag USA hinzugefügt. Wenn der Eintrag USA ungültig gemacht wird, werden auch alle New-York-Einträge ungültig gemacht. Sehen Sie sich das folgende Codebeispiel an:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
// mehreren Städten das Schlüsselwort "USA" zuordnen
objectGrid.associateKeyword("USA", "New York");
objectGrid.associateKeyword("USA", "Rochester");
objectGrid.associateKeyword("USA", "Raleigh");
:
// mehrere Einträge mit verschiedenen Schlüsselwörtern hinzufügen
objectMap.insert("key1", "value1", "New York");
objectMap.insert("key2", "value2", "Mexico");
objectMap.insert("key3", "value3", "Raleigh");
```

```
objectMap.insert("key4", "value4", "USA");
objectMap.insert("key5", "value5", "Rochester");
objectMap.insert("key6", "value6", "France");
:
// alle Einträge mit dem Schlüsselwort "USA" ungültig machen,
// aber die Einträge "key2" und "key6" beibehalten
objectMap.invalidateUsingKeyword("USA", true);
:
```

Nähere Informationen finden Sie im Abschnitt „Schlüsselwörter" auf Seite 58.

Zugriff auf BackingMap

ObjectGrid unterstützt den Zugriff auf die BackingMap-Objekte. Sie können mit den Methoden defineMap und getMap auf ein BackingMap-Objekt zugreifen. Nähere Informationen finden Sie im Abschnitt „Interface BackingMap" auf Seite 47. Das folgende Beispiel erstellt zwei BackingMap-Referenzen:

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

Slot-Verarbeitung

Sie können einen Slot reservieren, um Objekte zu speichern, die im Verlauf der Transaktion verwendet werden, wie z. B. das Objekt für die Transaktions-ID (TxID) oder ein Objekt für eine Datenbankverbindung (Connection). Diese gespeicherten Objekte werden anschließend mit einem bestimmten Index referenziert, der von der Methode reserveSlot bereitgestellt wird. Weitere Informationen zur Verwendung von Slots finden Sie in den Abschnitten „Loader" auf Seite 102 und „Plug-in TransactionCallback" auf Seite 119. Das folgende Code-Snippet veranschaulicht die Slot-Verarbeitung:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
int index = objectGrid.reserveSlot
   (com.ibm.websphere.objectgrid.TxID.SLOT_NAME);
:
// Sie verwenden den Index später, wenn Sie Objekte aus dem Objekt
// TxID speichern oder abrufen...
TxID tx = session.getTxID();
tx.putSlot(index, someObject);
:
Object theTxObject = tx.getSlot(index);
:
```

Sicherheitsverarbeitung

Maps können mit Sicherheitsmechanismen geschützt werden. Die folgenden Methoden sind in einem ObjectGrid-Framework für die Konfiguration und Verwendung der Sicherheits-Features verfügbar:

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

Nähere Informationen zu den verfügbaren Sicherheitsmechanismen finden Sie im Abschnitt „ObjectGrid-Sicherheit" auf Seite 73.

ObjectGridEventListener

Der Listener ObjectGridEventListener ermöglicht Anwendungen, beim Beginn oder Festschreiben einer Transaktion über dieses Ereignis benachrichtigt zu werden. In der ObjectGrid-Instanz kann eine Instanz eines ObjectGridEventListener definiert werden. Nähere Informationen finden Sie im Abschnitt „Listener" auf Seite 88. Das folgende Beispiel zeigt, wie Sie das Interface ObjectGridEventListener über das Programm implementieren:

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

Sie können dieselbe Konfiguration auch mit XML durchführen:

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

Plug-in TransactionCallback

Beim Aufruf von Methoden in der Sitzung werden entsprechende Ereignisse an das Plug-in TransactionCallback gesendet. Eine ObjectGrid-Instanz kann kein oder ein TransactionCallback-Plug-in haben. BackingMap-Objekte, die in einer ObjectGrid-Instanz mit einem Plug-in TransactionCallback definiert sind, müssen einen Loader haben. Nähere Informationen finden Sie im Abschnitt „Plug-in TransactionCallback" auf Seite 119. Das folgende Code-Snippet veranschaulicht, wie das Plug-in TransactionCallback über das Programm implementiert wird:

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

Sie können dieselbe Konfiguration auch mit XML durchführen:

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

Übersicht über das Systemprogrammiermodell: Plug-in-Punkte und Features für das Interface BackingMap

Das Interface BackingMap hat mehrere optionale Plug-in-Punkte für weitergehende Interaktionen mit dem ObjectGrid-Framework.

Die folgenden Unterabschnitte beschreiben diese Features und enthalten einige kurze Code-Snippets für das Verwendungsszenario. An den entsprechenden Stellen

wird ein XML-Snippet gezeigt, um die alternative XML-Konfiguration zu veranschaulichen. Weitergehende Informationen finden Sie in den Abschnitten „Interface BackingMap" auf Seite 47 und „ObjectGrid-Konfiguration" auf Seite 130 bzw. in der API-Dokumentation.

Konfigurationsattribute

BackingMaps sind verschiedene Konfigurationselemente zugeordnet:

- **ReadOnly** (Standardeinstellung ist `false`): Wenn Sie dieses Attribut auf `true` setzen, ist BackingMap schreibgeschützt. Wenn Sie das Attribut auf `false` setzen, sind Lese- und Schreibzugriffe auf BackingMap möglich. Wenn Sie keinen Wert angeben, wird der Standardwert `false` angenommen, d. h. es sind Lese- und Schreibzugriffe möglich.
- **NullValuesSupported** (Standardeinstellung ist `true`): Die Unterstützung von Nullwerten bedeutet, dass ein Nullwert in eine Map geschrieben werden kann. Wenn dieses Attribut auf `true` gesetzt ist, werden Nullwerte in der ObjectMap unterstützt. Andernfalls werden keine Nullwerte unterstützt. Falls Nullwerte unterstützt werden und eine `get`-Operation einen Nullwert zurückgibt, kann dies bedeuten, dass der Wert null ist oder dass die Map den übergebenen Schlüssel nicht enthält.
- **NumberOfBuckets** (Standardeinstellung ist `503`): Gibt die Anzahl der Buckets an, die von dieser BackingMap verwendet werden. Die BackingMap-Implementierung verwendet eine Hash-Tabelle für ihre Implementierung. Wenn in der BackingMap viele Einträge vorhanden sind, kann mit einer höheren Anzahl von Buckets eine bessere Leistung erzielt werden, weil das Risiko von Kollisionen mit zunehmender Anzahl von Buckets abnimmt. Außerdem sind bei einer höheren Anzahl von Buckets mehr gemeinsame Zugriffe möglich.
- **NumberOfLockBuckets** (Standardeinstellung `383`): Gibt die Anzahl der Sperren-Buckets an, die vom Sperrenmanager für diese BackingMap-Instanz verwendet wird. Wenn das Attribut lockStrategy auf OPTIMISTIC oder PESSIMISTIC gesetzt ist, wird ein Sperrenmanager für die BackingMap erstellt. Der Sperrenmanager verwendet eine Hash-Tabelle, um die Einträge zu verfolgen, die von einer oder mehreren Transaktionen gesperrt werden. Wenn in der Hash-Tabelle viele Einträge vorhanden sind, kann mit einer höheren Anzahl von Sperren-Buckets eine bessere Leistung erzielt werden, weil das Risiko von Kollisionen mit zunehmender Anzahl von Buckets abnimmt. Außerdem sind bei einer höheren Anzahl von Buckets mehr gemeinsame Zugriffe möglich. Wenn das Attribut lockStrategy auf NONE gesetzt ist, wird von der BackingMap kein Sperrenmanager verwendet. In diesem Fall hat die Definition des Attributs numberOfLockBuckets keine Wirkung.

Beispiel für Konfiguration über das Programm

Das folgende Beispiel konfiguriert Merkmale für eine BackingMap:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
 // Lese-/Schreibzugriff (Standardeinstellung) überschreiben
backingMap.setReadOnly(true);
// Unterstützung von Nullwerten (Standardeinstellung) überschreiben
backingMap.setNullValuesSupported(false);
// Standardeinstellung überschreiben (Primzahlen eignen sich am besten)
backingMap.setNumberOfBuckets(251);
// Standardeinstellung überschreiben (Primzahlen eignen sich am besten)
backingMap.setNumberOfLockBuckets(251);
:
```

Beispiel für Konfiguration mit XML

Das folgende XML-Konfigurationsbeispiel konfiguriert dieselben Merkmale, die im vorherigen Beispiel für die Konfiguration über das Programm beschrieben wurden.

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

Sperrstrategie

Wenn die Sperrstrategie auf OPTIMISTIC oder PESSIMISTIC eingestellt ist, wird ein Sperrenmanager für die BackingMap erstellt. Um gegenseitiges Sperren zu verhindern, arbeitet der Sperrenmanager mit einem Standardzeitlimit, das die Wartezeit für das Erteilen einer Sperre vorgibt. Bei einer Überschreitung dieses Zeitlimits wird eine Ausnahme des Typs LockTimeoutException ausgelöst. Der Standardwert von 15 Sekunden ist für die meisten Anwendungen ausreichend. Auf Systemen mit hoher Arbeitslast können jedoch Zeitlimitüberschreitungen auftreten, selbst wenn kein gegenseitiges Sperren vorliegt. In diesem Fall können Sie mit der Methode setLockTimeout das Zeitlimit für Sperren vom Standardwert auf den erforderlichen Wert erhöhen, um solche Zeitlimitüberschreitungen und die daraufhin ausgelösten Ausnahmen zu verhindern. Wenn die Sperrstrategie auf NONE eingestellt ist, wird von der BackingMap kein Sperrenmanager verwendet. In diesem Fall hat die Definition des Attributs lockTimeout keine Wirkung. Nähere Informationen finden Sie im Abschnitt „Sperren” auf Seite 64.

Beispiel für Konfiguration über das Programm

Das folgende Beispiel definiert die Sperrstrategie:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// Standardeinstellung OPTIMISTIC überschreiben
backingMap.setLockStrategy(LockStrategy.PESSIMISTIC);
backingMap.setLockTimeout(30); // setzt das Zeitlimit für Sperren auf 30 Sekunden
:
```

Beispiel für Konfiguration mit XML

Das folgende Beispiel definiert dieselbe Sperrstrategie, die im vorherigen Beispiel für die Konfiguration über das Programm definiert wurde:

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

Schlüssel und Werte kopieren

Das Kopieren von Schlüsseln und Werten kann aus Ressourcen- und Leistungsperspektive sehr kostenintensiv sein. Ohne die Möglichkeit, diese Kopien zu erstellen, können jedoch sonderbare Fehler auftreten, die nur schwer zu bestimmen und zu beheben sind. ObjectGrid bietet die Möglichkeit festzulegen, ob und wann

Kopien von Schlüsseln oder Werten erstellt werden. Normalerweise sind Schlüssel unveränderlich, und deshalb besteht keine Notwendigkeit, die Schlüsselobjekte zu kopieren. Standardmäßig werden keine Kopien von Schlüsselobjekten erstellt. Bei Wertobjekten ist die Wahrscheinlichkeit höher, dass sie von der Anwendung geändert werden. Wann eine Kopie des Wertobjekts anstelle der Referenz auf das Wertobjekt bereitgestellt werden soll, kann konfiguriert werden. Nähere Einzelheiten zu den Einstellungen copyKey und copyMode finden Sie in Kapitel 7, „Best Practices für die Leistung von ObjectGrid", auf Seite 187 und in der JavaDoc.

Beispiel für Konfiguration über das Programm

Das folgende Beispiel zeigt, wie die Einstellungen copyMode und copyKey konfiguriert werden:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.setCopyKey(true); // eine Kopie jedes neuen Schlüssels erstellen
backingMap.setCopyMode(NO_COPY); // sehr wirksam - der Anwendung vertrauen
:
```

Beispiel für Konfiguration mit XML

Das folgende Beispiel erstellt dieselbe Konfiguration wie das vorherige Beispiel für die Konfiguration über das Programm:

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

Evictor

Evictor werden verwendet, um nicht mehr benötigte Einträge in der Map in regelmäßigen Abständen zu bereinigen. Die zu entfernenden Einträge werden vom Evictor definiert. Die integrierten Evictor sind zeitbasiert, so dass die Bereinigungsstrategie auf der Lebensdauer (TTL, Time to Live) eines Eintrags in der Map basiert. Es gibt andere Bereinigungsstrategien, die auf der Verwendung, der Größe oder eine Kombination von Faktoren basieren.

- **Integrierter TTL-Evictor**: Für den integrierten TTL-Evictor können mit den Methoden setTtlEvictorType und setTimeToLive verschiedene Konfigurationseinträge in der BackingMap definiert werden. Standardmäßig ist der integrierte TTL-Evictor nicht aktiv. Sie können den Evictor aktivieren, indem Sie die Methode setTtlEvictorType mit einem der folgenden drei Werte aufrufen: CREATION_TIME, LAST_ACCESS_TIME und NONE (Standardeinstellung). Anschließend wird abhängig vom Typ des ausgewählten TTL-Evictor der Wert für die Methode setTimeToLive verwendet, um die Lebensdauer für jeden Map-Eintrag zu definieren.
- **Evictor-Plug-ins:** Zusätzlich zum integrierten TTL-Evictor kann eine Anwendung ein Plug-in mit ihrer eigenen Evictor-Implementierung bereitstellen. Sie können in regelmäßigen Abständen einen beliebigen Algorithmus ausführen, um die Map-Einträge ungültig zu machen.

Konfiguration über das Programm

Die folgende Klasse erstellt einen Evictor:

```
class MyEvictor implements com.ibm.websphere.objectgrid.plugins.Evictor { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// Zeitgeber startet, wenn der Eintrag erstellt wird
backingMap.setTtlEvictorType(CREATION_TIME);
// Jedem Map-Eintrag eine Lebensdauer von 30 zuweisen
backingMap.setTimeToLive(30);
// Integrierte und benutzerdefinierte Evictor werden aktiv
backingMap.setEvictor(new MyEvictor()); :
```

XML-Konfiguration

Der folgende XML-Code erstellt eine Konfiguration, die mit der vorherigen Konfiguration über das Programm identisch ist:

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

:
```

Nähere Informationen finden Sie im Abschnitt „Evictor” auf Seite 93.

Loader

Ein ObjectGrid-Loader (oder Ladeprogramm) ist eine Plug-in-Komponente, die es einer ObjectGrid-Map-Instanz ermöglicht, als Speicher-Cache für Daten aufzutreten, die in der Regel in einem persistenten Speicher auf demselben oder einem anderen System verwaltet werden. In der Regel wird eine Datenbank oder ein Dateisystem als persistenter Speicher verwendet. Ein Loader besitzt die Logik zum Lesen und Schreiben von Daten in einem persistenten Speicher.

Ein Loader ist ein Plug-in für eine ObjectGrid-BackingMap-Instanz. Einer BackingMap-Instanz kann jeweils nur ein Loader zugeordnet werden, und jede BackingMap hat ihre eigene Loader-Instanz. Die BackingMap-Instanz fordert alle Daten, die sie selbst nicht enthält, von ihrem Loader an. Alle Änderungen, die an der Map-Instanz vorgenommen werden, werden in den Loader geschrieben. Das Loader-Plug-in ermöglicht der BackingMap-Instanz, Daten zwischen der Map und ihrem persistenten Speicher zu verschieben.

Konfiguration über das Programm

Beispiel für eine Loader-Implementierung:

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

```
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.setLoader(myLoader);
backingMap.setPreloadMode(true);
:
```

XML-Konfiguration

Das folgende XML-Beispiel liefert dieselbe Konfiguration wie das vorherige Beispiel für die Konfiguration über das Programm:

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

Nähere Informationen finden Sie im Abschnitt „Loader” auf Seite 102.

Interface MapEventListener

Das Callback-Interface MapEventListener wird von der Anwendung implementiert, wenn sie Ereignisse für eine Map, wie z. B. die Bereinigung eines Map-Eintrags oder das abgeschlossene Vorabladen von Daten, empfangen möchte. Das folgende Codebeispiel veranschaulicht, wie Sie eine MapEventListener-Instanz für eine BackingMap-Instanz definieren:

Konfiguration über das Programm

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

XML-Konfiguration

Das folgende Beispiel erstellt dieselbe Konfiguration wie das vorherige Beispiel für die Konfiguration über das Programm:

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

Nähere Informationen finden Sie im Abschnitt „Listener” auf Seite 88.

Interface ObjectTransformer

Mit dem Interface ObjectTransformer können Sie die Schlüssel und Werte von Cache-Einträgen serialisieren, die nicht als serialisierbar definiert sind. Auf diese Weise können Sie Ihr eigenes Serialisierungsschema definieren, ohne das Interface Serializable direkt erweitern oder implementieren zu müssen. Außerdem stellt dieses Interface Methoden für das Kopieren von Schlüsseln und Werten bereit. Das folgende Beispiel ist eine Klasse, die das Interface ObjectTransformer implementiert:

Konfiguration über das Programm

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

XML-Konfiguration

Das folgende XML-Beispiel liefert dieselbe Konfiguration wie das vorherige Beispiel für die Konfiguration über das Programm:

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

Nähere Informationen finden Sie im Abschnitt „Plug-in ObjectTransformer” auf Seite 115.

OptimisticCallback

Mit dem Interface OptimisticCallback können Sie ein Versionsfeld erstellen und verarbeiten, das einem bestimmten Value-Objekt zugeordnet ist. In vielen Fällen ist es ineffizient und fehlerträchtig, das Value-Objekt direkt zu verwenden, um festzustellen, ob ein anderer Cache-Client den Wert seit dem letzten Abruf geändert hat. Alternativ können Sie ein anderes Feld definieren, das den Status des Value-Objekts darstellt. Die ist der Zweck des OptimisticCallback. Er stellt ein alternatives VersionedValue-Objekt für das Value-Objekt bereit. Im Folgenden sehen Sie eine Beispielkonfiguration für das Interface OptimisticCallback:

Konfiguration über das Programm

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

XML-Konfiguration

Das folgende Beispiel erstellt dieselbe Konfiguration wie das vorherige Beispiel für die Konfiguration über das Programm:

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

Übersicht über das Systemprogrammiermodell: Features für das Interface Session

Das Interface Session hat mehrere Features für weitergehende Interaktionen mit dem ObjectGrid-Framework. Die folgenden Unterabschnitte beschreiben diese Features und enthalten einige kurze Code-Snippets für das Verwendungsszenario.

Nähere Informationen zum Interface Session finden Sie im Abschnitt „Interface Session” auf Seite 51.

Modus ohne Durchschreiben

Manchmal möchten Anwendungen Änderungen nur auf die Basis-Map, aber nicht auf den Loader anwenden. Mit der Methode beginNoWriteThrough des Interface Session kann diese Zielsetzung realisiert werden. Sie können mit der Methode isWriteThroughEnabled des Interface Session prüfen, ob die aktuelle Sitzung in den Backend-Loader schreibt. Für andere Benutzer des Session-Objekts kann es hilfreich sein zu wissen, welcher Typ von Sitzung derzeit verarbeitet wird. Das folgende Beispiel aktiviert den Modus ohne Durchschreiben:

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
boolean isWriteThroughEnabled = session.isWriteThroughEnabled();
// Nimmt Aktualisierungen an der Map vor...
session.commit();
:
```

Daten nur in den Loader schreiben

Anwendungen können mit der Methode flush in der Sitzung vorgenommene lokale Änderungen in den Loader schreiben, ohne diese Änderungen permanent festzuschreiben. Dies wird im folgenden Beispiel veranschaulicht:

```
:
Session session = objectGrid.getSession();
session.begin();
// Änderungen vornehmen...
session.flush(); // die Änderungen in den Loader schreiben, aber noch nicht festschreiben
// weitere Änderungen vornehmen...
session.commit();
:
```

Methode processLogSequence

Die Methode processLogSequence wird verwendet, um eine LogSequence zu verarbeiten. Jedes LogElement in der LogSequence wird untersucht. Anschließend wird die entsprechende Operation, z. B. insert (Einfügen), update (Aktualisieren) oder invalidate (Ungültig machen) für die BackingMap ausgeführt, die für den Map-Name der LogSequence angegeben ist. Damit diese Methode aufgerufen werden kann, muss eine ObjectGrid-Sitzung aktiv sein. Danach ist der Aufrufende für das Absetzen der entsprechenden COMMIT- oder ROLLBACK-Aufrufe zum Beenden der Sitzung verantwortlich. Für diesen Methodenaufruf wird das automatische Festschreiben nicht unterstützt.

Diese Methode wird hauptsächlich verwendet, um eine von einer fernen JVM empfangene LogSequence zu verarbeiten. Wenn Sie beispielsweise die Unterstützung für verteiltes Festschreiben verwenden, werden die LogSequences, die einer bestimmten festgeschriebenen Sitzung zugeordnet sind, an andere empfangsbereite ObjectGrids in anderen Java Virtual Machines (JVM) verteilt. Nach dem Empfang der LogSequences in der fernen JVM kann der Listener mit der Methode beginNoWriteThrough eine Sitzung starten, die Methode processLogSequence aufrufen und anschließend die Methode commit für die Sitzung ausführen. Es folgt ein Beispiel:

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

Leistungsüberwachung

Wenn Sie mit WebSphere Application Server arbeiten, können Maps optional mit Funktionen für die Leistungsüberwachung ausgestattet werden. Mit der Methode `setTransactionType` für Sitzungen können Sie die Funktionen für die Leistungsüberwachung konfigurieren und verwenden. Nähere Informationen finden Sie im Abschnitt „ObjectGrid-Leistung mit WebSphere Application Server Performance Monitoring Infrastructure (PMI) überwachen” auf Seite 149.

Übersicht über das Systemprogrammiermodell: Features für das Interface ObjectMap

Das Interface ObjectMap hat mehrere Features für weitergehende Interaktionen mit dem ObjectGrid-Framework. Die folgenden Unterabschnitte beschreiben diese Features und enthalten einige kurze Code-Snippets für das Verwendungsszenario.

Nähere Informationen zum Interface ObjectMap finden Sie im Abschnitt „Interfaces ObjectMap und JavaMap" auf Seite 54.

Interfaces JavaMap und java.util.Map

Für Anwendungen, die das Interface java.util.Map verwenden, stellt ObjectMap die Methode getJavaMap bereit, damit diese Anwendungen die Implementierung des von ObjectMap unterstützten Interface java.util.Map abrufen können. Die zurückgegebene Map-Instanz kann anschließend in das Interface JavaMap umgesetzt werden, eine Erweiterung des Interface Map. Das Interface JavaMap hat dieselben Methodensignaturen wie ObjectMap, aber eine andere Ausnahmebehandlung. Das Interface JavaMap erweitert das Interface java.util.Map. Somit sind alle Ausnahmen Instanzen der Klasse java.lang.RuntimeException. Da das Interface JavaMap das Interface java.util.Map erweitert, kann ObjectGrid problemlos für eine vorhandene Anwendung verwendet werden, die das Interface java.util.Map für das Zwischenspeichern von Objekten verwendet. Es folgt ein Code-Snippet:

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

Map-Erweiterungen

Das Interface ObjectMap stellt zusätzlich zu den Funktionen für geprüfte Ausnahmen weitere Funktionen bereit. Beispielsweise kann ein Benutzer angeben, dass ein bestimmter Map-Eintrag mit der Methode getForUpdate aktualisiert werden soll. Damit wird der Laufzeitumgebung von ObjectGrid und dem Plug-in Loader angezeigt, dass der Eintrag bei Bedarf während der Verarbeitung gesperrt werden kann. Außerdem wird die Stapelverarbeitung mit den Methoden getAll, putAll und removeAll unterstützt. Nähere Informationen zu diesen Methoden finden Sie in der API-Dokumentation.

Schlüsselwortverarbeitung

Die meisten Map-Operationen haben verwenden die Schlüsselparameterversion wie insert, get, getForUpdate, put, remove und invalidate. Im Hinblick auf die Benutzerfreundlichkeit wird auch die Methode setDefaultKeyword unterstützt. Diese Methode ordnet Einträge Schlüsselwörtern zu, ohne die Schlüsselwortversion der Map-Operation zu verwenden. Es folgt ein Beispiel:

```
:
// Standardschlüsselwort festlegen
session.begin();
objectMap.setDefaultKeyword("New York");
Person p = (Person) objectMap.get("Billy"); // Eintrag "Billy" hat Schlüsselwort "New York"
p = (Person) objectMap.get("Bob", "Los Angeles"); // Eintrag "Bob"
//hat das Schlüsselwort "Los Angeles"
objectMap.setDefaultKeyword(null);
p = (Person) objectMap.get("Jimmy"); // Eintrag "Jimmy" hat kein Schlüsselwort
session.commit();
:
// Schlüsselwortparameterversion der Einfügeoperation
session.begin();
```

```
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person, "Rochester"); // "BillyBob" hat
//Schlüsselwort "Rochester"
session.commit();
:
```

Nähere Informationen finden Sie im Abschnitt „Schlüsselwörter” auf Seite 58.

Methode für Kopiermodus

Mit der Methode setCopyMode kann der Kopiermodus für die Map in dieser Map für diese Sitzung oder Transaktion überschrieben werden. Diese Methode ermöglicht der Anwendung, den Kopiermodus für jede einzelne Sitzung optimal an deren Anforderungen anzupassen. Der Kopiermodus kann in einer aktiven Sitzung nicht geändert werden. Es ist eine entsprechende Methode clearCopyMode vorhanden, die den Kopiermodus auf die in der BackingMap definierte Einstellung zurücksetzt. Sie können diese Methode nur aufrufen, wenn keine aktiven Sitzungen vorhanden sind. Es folgt ein Beispiel für das Einstellen des Kopiermodus:

```
:
objectMap.setCopyMode(CopyMode.COPY_ON_READ, null);
session.begin();
// objectMap ändern...
session.commit();
objectMap.clearCopyMode(); // Kopiermodus auf BackingMap-Einstellung zurücksetzen
session.begin();
// objectMap ändern...
session.commit();
:
```

Nähere Informationen finden Sie im Abschnitt „Plug-in ObjectTransformer” auf Seite 115 und in Kapitel 7, „Best Practices für die Leistung von ObjectGrid”, auf Seite 187.

Evictor-Einstellungen

Sie können den Wert des Zeitlimits TimeToLive für den integrierten Evictor TimeToLive auf ObjectMap-Ebene überschreiben. Die Methode setTimeToLive legt die Lebensdauer für jeden Cache-Eintrag in Sekunden fest. Bei einer Änderung wird der vorherige Wert von TimeToLive zurückgegeben. Der TimeToLive-Wert gibt an, wie lange ein Eintrag mindestens im Cache verweilt, bevor er für die Bereinigung berücksichtigt wird, und zeigt dem integrierten Evictor TimeToLive an, wie lange ein Eintrag nach dem letzten Zugriff im Cache verbleiben soll. Der neue TimeToLive-Wert gilt nur für ObjectMap-Einträge, auf die die Transaktion zugreift, die von dem zum Abrufen der ObjectMap-Instanz verwendeten Session-Objekt gestartet wurde. Er gilt für alle Transaktionen, die für die Sitzung aktiv sind, und für zukünftige Transaktionen, die von der Sitzung ausgeführt werden. Keine Auswirkung hat der neue TimeToLive-Wert auf die Einträge einer ObjectMap-Instanz, auf die eine Transaktion zugreift, die von einer anderen Sitzung gestartet wurde. Wenn Sie diese Methode für die ObjektMap-Instanz aufrufen, wird der vorherige Wert, der mit der Methode setTimeToLive in der BackingMap definiert wurde, für diese ObjectMap-Instanz überschrieben. Es folgt ein Beispiel:

```
:
session.begin();
int oldTTL = objectMap.setTimeToLive(60); // TTL auf 60 Sekunden setzen
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person); // Eintrag "BillyBob" hat eine TTL
//von 60 Sekunden
session.commit();
:
```

```
objectMap.setTimeToLive(oldTTL); // TTL auf ursprünglichen Wert zurücksetzen
Person person2 = new Person("Angelina", "Jolie", "somewhere");
objectMap.insert("Brad", person2); // Eintrag "Brad" verwendet den ursprünglichen Wert
:
```

Nähere Informationen finden Sie im Abschnitt „Evictor" auf Seite 93.

Kapitel 4. ObjectGrid-Beispielanwendungen

Dieses Kapitel beschreibt die ObjectGrid-Beispielanwendungen, die mit der Installation des Produkts WebSphere Extended Deployment bereitgestellt werden.

Übersicht

Verschiedene ObjectGrid-spezifische und ObjectGrid-Beispielanwendungen veranschaulichen die Integration mit J2EE-Anwendungen (Java 2 Platform, Enterprise Edition) und dem Partitionierungs-Feature (WPF). Dieses Kapitel beschreibt die einzelnen Beispielanwendungen und die Features, die das jeweilige Muster veranschaulicht. Außerdem wird beschrieben, wo Sie die Beispielanwendungen finden und in welchen Umgebungen Sie die jeweilige Beispielanwendung ausführen können.

Dieses Kapitel beschreibt die Beispielanwendungen, die mit der Installation von WebSphere Extended Deployment bereitgestellt werden. Auf der folgenden Website sind weitere Beispielanwendungen verfügbar, die sich auf die JMS-Integration (Java Message Service) und die Integration von ObjectGrid mit anderen Open-Source-Frameworks beziehen: http://www-1.ibm.com/support/docview.wss?uid=swg27006432.

Beispielanwendungen

- **ObjectGridSamplesSA:** Diese Beispielanwendung umfasst mehrere J2EE-Beispiele, die in der Datei `objectgridSamples.jar` gepackt sind und die ObjectGrid-Funktionen veranschaulichen. Diese J2SE-Beispiele können in einer J2SE-Umgebung ausgeführt werden.
- **ObjectGridSample:** Diese J2EE-Beispielanwendung veranschaulicht, wie Servlets und Enterprise-Beans in Sitzungen die ObjectGrid-Funktionen verwenden. Diese Beispielanwendung wird in einer EAR-Datei bereitgestellt.
- **ObjectGridPartitionCluster:** Diese J2EE-Beispielanwendung veranschaulicht, wie WPF und ObjectGrid zusammenarbeiten und das Interface ObjectGridEvent verwenden, um Objektänderungen weiterzugeben. Außerdem demonstriert sie, wie Sie die kontextbasierte Weiterleitung aktivieren können, um Integrität und Konsistenz des ObjectGrid-Framework zu gewährleisten. Diese Beispielanwendung wird in einer EAR-Datei bereitgestellt.
- **ObjectGridJMSSamples:** Diese Beispielanwendung enthält verschiedene J2EE-Beispiele, gepackt in der Datei ObjectGridJMSSamples.zip, die veranschaulichen, wie Sie in einer Umgebung mit einer JVM oder einer Cluster-Umgebung mit der JMS-Funktion Änderungen aus einer ObjectGrid-Instanz in eine andere ObjectGrid-Instanz übertragen können. Diese J2EE-Beispielanwendungen sind nur im Web unter der folgenden Adresse verfügbar: http://www-1.ibm.com/support/docview.wss?uid=swg27006432.

Funktionalität der Beispielanwendungen

Tabelle 1. Funktionalität der Beispielanwendungen

Funktionsbereich	Beispiel ObjectGrid SamplesSA	Beispiel ObjectGrid Sample	Beispiel ObjectGridPartition Cluster	Beispiel ObjectGrid JMSSamples
ObjectGrid EventListener			x	x
Transaction Callback	x	x	x	
Loader	x	x	x	
MapEvent Listener	x			
Object Transformer	x	x	x	x
Optimistic Callback	x	x	x	
Kopiermodus für BackingMap	x	x		
Verteilte Invalidierung			x	x
Verteilte Aktualisierung			x	x
LogSequence-Verarbeitung				x
Partitionierungs-Feature (WPF)			x	
Java Message Service (JMS)				x

Position

Die folgende Tabelle beschreibt, in welchen Verzeichnissen Sie die Dateien mit der Erweiterung .jar nach der Installation von WebSphere Extended Deployment finden:

Tabelle 2. Position der Beispielanwendungen

Beispielanwendung	Position
ObjectGridSamplesSA	*Installationsstammverz*\optionalLibraries\ObjectGrid
ObjectGridSample	*Installationsstammverz*\optionalLibraries\ObjectGrid
ObjectGridPartitionCluster	*Installationsstammverz*\installableApps

Die anderen Beispiele wie ObjectGridJMSSamples sind im Web unter der Adresse http://www-1.ibm.com/support/docview.wss?uid=swg27006432 verfügbar. Außerdem finden Sie unter der folgenden Webadresse Artikel von IBM DeveloperWorks, die interessante Themen behandeln: http://www.ibm.com/developerworks. Suchen Sie nach **ObjectGrid**.

Umgebungen für die Ausführung der Beispielanwendungen

Einige Beispielanwendungen können in einer J2SE-Umgebung, andere müssen in einer J2EE-Umgebung ausgeführt werden. Einige können in einer Einzelserverinstanz, andere müssen in einem Cluster ausgeführt werden. In der folgenden Tabelle sind die Ausführungsumgebungen für die Beispielanwendungen aufgelistet.

Einschränkung: Sie können ObjectGrid in einer Umgebung mit WebSphere Extended Deployment Version 6.0 einsetzen. ObjectGrid kann auch in Umgebungen mit Java 2 Platform, Standard Edition (J2SE) Version 1.4.2 und höher oder Umgebungen mit WebSphere Application Server Version 6.02 und höher verwendet werden, sofern bestimmte zusätzliche Lizenzvereinbarungen eingehalten werden. Nähere Einzelheiten hierzu können Sie bei Ihrem Vertriebsbeauftragten erfragen.

Tabelle 3. Ausführungsumgebungen für die Beispielanwendungen

		ObjectGrid SamplesSA	**ObjectGrid Sample**	**ObjectGrid Partition Cluster**	**ObjectGrid JMSSamples**
J2SE	Eclipse	x			
	Befehlszeile	x			
WebSphere Application Server Version 6.0.*x*	Einzelserver		x		x
	Cluster		x		x
	Rational Application Developer Unit Test Environment (UTE)		x		
WebSphere Extended Deployment Version 6.0	Einzelserver		x		x
	Cluster		x	x	x

Kapitel 5. Übersicht über die Anwendungsprogrammierschnittstelle von ObjectGrid

Dieses Kapitel beschreibt, wie Sie ObjectGrid mit XML oder mit Programmierschnittstellen konfigurieren. Außerdem finden Sie in diesem Kapitel Informationen zur Implementierung der externen Interfaces, die ObjectGrid bereitstellt. Zu allen Fällen finden Sie eine Übersicht, eine Beschreibung der API-Interfaces und Beispiele.

API-Dokumentation

Die JavaDoc zu ObjectGrid ist die maßgebliche Informationsquelle zu den APIs. Sie finden die JavaDoc im folgenden Verzeichnis Ihrer Installation von WebSphere Extended Deployment: *Installationsstammverzeichnis*\web\xd\apidocs.

Interface ObjectGridManager

Die Klasse ObjectGridManagerFactory und das Interface ObjectGridManager sind ein Mechanismus, mit dem Sie ObjectGrid-Instanzen erstellen, aufrufen und zwischenspeichern können. Die Klasse ObjectGridManagerFactory ist eine statische Helper-Klasse für den Zugriff auf das Interface ObjectGridManager, ein Singleton. Das Interface ObjectGridManager enthält verschiedene komfortable Methoden, mit dem Sie Instanzen eines ObjectGrid-Objekts erstellen können. Außerdem vereinfacht das Interface ObjectGridManager das Erstellen und Zwischenspeichern von ObjectGrid-Instanzen, auf die mehrere Benutzer zugreifen können.

createObjectGrid-Methoden

Dieser Abschnitt enthält Informationen zu den sieben createObjectGrid-Methoden des Interface ObjectGridManager.

createObjectGrid-Methoden

Das Interface ObjectGridManager hat sieben createObjectGrid-Methoden. Im Folgenden ist ein einfaches Szenario beschrieben:

Einfacher Fall mit Standardkonfiguration

Das folgende einfache Szenario zeigt, wie Sie eine ObjectGrid-Instanz erstellen können, die von mehreren Benutzern gemeinsam genutzt werden kann.

```
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.
/*Fortsetzung des Beispiels folgt..*/
```

Mit diesem Java-Code-Snippet wird die ObjectGrid-Instanz Employees erstellt und zwischengespeichert. Die ObjectGrid-Instanz Employees wird mit der Standardkonfiguration initialisiert und ist danach zur Verwendung bereit. Der zweite Para-

meter in der Methode createObjectGrid ist auf `true` gesetzt. Damit wird das Interface ObjectGridManager angewiesen, die erstellte ObjectGrid-Instanz zwischenzuspeichern. Wenn dieser Parameter auf `false` gesetzt ist, wird die Instanz nicht zwischengespeichert. Jede ObjectGrid-Instanz hat einen `Namen`. Basierend auf diesem Namen kann die Instanz von mehreren Clients oder Benutzern gemeinsam genutzt werden.

XML-Konfiguration

ObjectGrid ist in hohem Maße konfigurierbar. Das vorherige Beispiel hat veranschaulicht, wie eine einfache ObjectGrid-Instanz ohne Konfiguration erstellt werden kann. Anhand dieses Beispiels können Sie eine vorkonfigurierte ObjectGrid-Instanz erstellen, die auf einer XML-Konfigurationsdatei basiert. Es sind zwei Methoden zum Konfigurieren einer ObjectGrid-Instanz verfügbar. Sie können die Instanz über das Programm konfigurieren oder eine XML-basierte Konfigurationsdatei verwenden. Eine Kombination beider Ansätze ist ebenfalls möglich.

Mit dem Interface ObjectGridManager können Sie eine ObjectGrid-Instanz basierend auf einer XML-Konfiguration erstellen. Das Interface ObjectGridManager besitzt mehrere Methoden, an die ein URL als Argument übergeben werden kann. Jede XML-Datei, die an das Interface ObjectGridManager übergeben wird, muss anhand des Schemas validiert werden. Die XML-Validierung kann nur inaktiviert werden, wenn die Datei bereits validiert worden ist und seit der letzten Validierung keine Änderungen mehr vorgenommen wurden. Das Inaktivieren der Validierung verringert zwar den Systemaufwand geringfügig, birgt aber das Risiko, dass eine ungültige XML-Datei verwendet wird. IBM Java Developer Kit (JDK) 1.4.2 unterstützt die Validierung von XML-Dateien. Wenn Sie ein JDK verwenden, dass diese Unterstützung nicht bietet, müssen Sie die XML-Datei unter Umständen mit Apache Xerces validieren.

Das folgende Java-Code-Snippet veranschaulicht, wie eine XML-Konfigurationsdatei zum Erstellen einer ObjectGrid-Instanz übergeben wird.

```
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; // XML-Validierung aktivieren
boolean cacheInstance = true; // Instanz zwischenspeichern
String objectGridName="Employees"; // Name des ObjectGrid-URL
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
 ObjectGridManagerFactory.getObjectGridManager();
 ObjectGrid employees = oGridManager.createObjectGrid(objectGridName,
                        allObjectGrids,
                        validateXML,
                        cacheInstance);
```

Die XML-Datei kann Konfigurationsdaten für mehrere ObjectGrid-Instanzen enthalten. Das vorherige Code-Snippet gibt die ObjectGrid-Instanz "Employees" zurück, vorausgesetzt, dass die Konfiguration für "Employees" in der Datei definiert ist.

Es sind sieben createObjectGrid-Methoden verfügbar. Die Methoden sind im folgenden Codeblock dokumentiert.

```
/**
* Eine einfache Factory-Methode, die eine ObjectGrid-Instanz zurückgibt.
* Es wird ein eindeutiger Name zugeordnet.
* Die ObjectGrid-Instanz wird nicht zwischengespeichert.
* Die Benutzer können anschließend mit {@link ObjectGrid#setName(String)} den
* ObjectGrid-Namen ändern.
*
* @return ObjectGrid Eine ObjectGrid-Instanz mit einem eindeutigen Namen.
* @throws ObjectGridException Jeder Fehler, der beim Erstellen der
*                             ObjektGrid-Instanz auftritt.
* @ibm-api
*/
public ObjectGrid createObjectGrid() throws ObjectGridException;
/**
* Eine einfache Factory-Methode, die eine ObjectGrid-Instanz mit dem
* angegebenen Namen zurückgibt. Die ObjectGrid-Instanzen können
* zwischengespeichert werden. Wenn bereits eine ObjectGrid-Instanz
* mit diesem Namen zwischengespeichert wurde, wird eine Ausnahme
* des Typs ObjectGridException ausgelöst.
*
* @param objectGridName Der Name der zu erstellenden ObjectGrid-Instanz.
* @param cacheInstance true, wenn die ObjectGrid-Instanz zwischengespeichert
*                             werden soll.
* @return Eine ObjectGrid-Instanz.
* @Der Name wurde bereits zwischengespeichert, oder jeder andere
* Fehler, der beim Erstellen der ObjectGrid-Instanz auftritt.
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, boolean cacheInstance)
 throws ObjectGridException;
/**
* Eine ObjectGrid-Instanz mit dem angegebenen ObjectGrid-Namen erstellen.
* Die erstellte ObjectGrid-Instanz wird zwischengespeichert.
* @param objectGridName Der Name der zu erstellenden ObjectGrid-Instanz.
* @return Eine ObjectGrid-Instanz.
* @ObjectGridException auslösen, wenn bereits eine ObjectGrid-Instanz mit
* diesem Namen zwischengespeichert wurde, oder jeder Fehler, der beim Erstellen
* der ObjectGrid-Instanz auftritt.
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName)
 throws ObjectGridException;
/**
* Eine ObjectGrid-Instanz basierend auf dem angegebenen ObjectGrid-Namen und der
* XML-Datei erstellen. Die in der XML-Datei definierte ObjectGrid-Instanz wird mit
* dem angegebenen ObjectGrid-Namen erstellt und zurückgegeben. Falls die angegebene
* ObjectGrid-Instanz nicht in der XML-Datei vorhanden ist, wird eine Ausnahme
* ausgelöst.
*
* Die ObjecGrid-Instanz kann zwischengespeichert werden.
*
* Wenn der URL null ist, wird er einfach ignoriert. In diesem Fall verhält
* sich die Methode wie {@link #createObjectGrid(String, boolean)}.
*
* @param objectGridName Der Name der zurückzugebenden ObjectGrid-Instanz. Er
* darf nicht null sein.
* @param xmlFile Der URL einer korrekt formatierten XML-Datei, die auf dem
*                ObjectGrid-Schema basiert.
* @param enableXmlValidation Sofern true, wird die XML-Datei validiert.
* @param cacheInstance Ein Boolescher Wert, der angibt, ob die in der
* XML-Datei definierten ObjectGrid-Instanzen zwischengespeichert
* werden oder nicht. Mit dem Wert true werden die Instanzen
* zwischengespeichert.
*
* @throws ObjectGridException, wenn bereits eine ObjectGrid-Instanz dieses
* Namens zwischengespeichert wurde, der ObjectGrid-Name nicht in der XML-Datei
* enthalten ist oder ein anderer Fehler beim Erstellen der ObjectGrid-Instanz
```

```
* auftritt.
* @return Eine ObjectGrid-Instanz.
* @see ObjectGrid
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, final URL xmlFile,
final boolean enableXmlValidation, boolean cacheInstance) throws
 ObjectGridException;
/**
* Eine XML-Datei verarbeiten und basierend auf der Datei eine Liste der
* ObjectGrid-Objekte erstellen.
* Diese ObjectGrid-Instanzen können zwischengespeichert werden.
* Bei dem Versuch, eine neu erstellte ObjectGrid-Instanz zwischenzuspeichern,
* die denselben Namen hat wie eine bereits zwischengespeicherte Instanz,
* wird eine Ausnahme des Typs ObjectGridException ausgelöst.
*
* @param xmlFile Die Datei, die eine oder mehrere ObjectGrid-Instanzen
*                definiert.
* @param enableXmlValidation Mit dem Wert true wird die XML-Datei
*                            auf der Basis des Schemas validiert.
* @param cacheInstances Mit dem Wert true werden alle auf der Basis
*                       der Datei erstellten ObjectGrid-Instanzen
*                       zwischengespeichert.
* @return Eine ObjectGrid-Instanz.
* @throws ObjectGridException beim Versuch, eine ObjectGrid-Instanz
*                             zwischenzuspeichern, die denselben Namen hat
*                             wie eine bereits zwischengespeicherte Instanz,
*                             oder jeder andere Fehler, der beim Erstellen
*                             der ObjectGrid-Instanz auftritt.
* @ibm-api
*/
public List createObjectGrids(final URL xmlFile,
final boolean enableXmlValidation,
boolean cacheInstances)
throws ObjectGridException;
/*** Alle in der XML-Datei enthaltenen ObjectGrid-Instanzen erstellen.
* Die XML-Datei wird basierend auf dem Schema validiert. Jede erstellte
* ObjectGrid-Instanz wird zwischengespeichert. Es wird eine Ausnahme
* des Typs ObjectGridException ausgelöst, wenn versucht wird, eine neu
* erstellte ObjectGrid-Instanz zwischenzuspeichern, die denselben Namen
* hat wie eine bereits zwischengespeicherte Instanz.
* @param xmlFile Die zu verarbeitende XML-Datei. Die ObjectGrid-Instanzen
* werden basierend auf dem Inhalt der Datei erstellt.
* @return Eine Liste der erstellten ObjectGrid-Instanzen.
* @throws ObjectGridException, wenn bereits eine ObjectGrid-Instanz mit
* einem der in der XML-Datei enthaltenen Namen zwischengespeichert wurde,
* oder jeder andere Fehler, der während der Erstellung der ObjectGrid-Instanzen
* auftritt.
* @ibm-api
*/
public List createObjectGrids(final URL xmlFile) throws ObjectGridException;
/**
* Die XML-Datei verarbeiten und nur dann eine einzelne ObjectGrid-Instanz
* mit dem angegebenen ObjectGrid-Namen erstellen, wenn eine ObjectGrid-Instanz
* mit diesem Namen in der Datei enthalten ist.  Falls keine ObjectGrid-Instanz
* mit diesem Namen in der XML-Datei definiert ist, wird eine Ausnahme des
* Typs ObjectGridException ausgelöst. Die erstellte ObjectGrid-Instanz wird
* zwischengespeichert.
* @param objectGridName Der Name der zu erstellenden ObjectGrid-Instanz. Diese
* ObjectGrid-Instanz muss in der XML-Datei definiert werden.
* @param xmlFile Die zu verarbeitende XML-Datei.
* @return Eine neu erstellte ObjectGrid-Instanz.
* @throws ObjectGridException, wenn bereits eine ObjectGrid-Instanz dieses
* Namens zwischengespeichert wurde, der ObjectGrid-Name nicht in der XML-Datei
* enthalten ist oder ein anderer Fehler beim Erstellen der ObjectGrid-Instanz
* auftritt.
```

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

getObjectGrid-Methoden

Verwenden Sie die getObjectGrid-Methoden zum Abrufen zwischengespeicherter Instanzen.

Eine zwischengespeicherte Instanz abrufen

Da die ObjectGrid-Instanz Employees mit dem Interface ObjectGridManager zwischengespeichert wurde, können andere Benutzer mit dem folgenden Code-Snippet auf diese Instanz zugreifen:

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

Im Folgenden werden zwei getObjectGrid-Methoden beschrieben, die zwischengespeicherte ObjectGrid-Instanzen zurückgeben.

```
/**
* Eine Liste der ObjectGrid-Instanzen zurückgeben, die zuvor
* zwischengespeichert wurden.
* Null zurückgeben, wenn keine ObjectGrid-Instanzen zwischengespeichert wurden.
* @return Eine Liste der zuvor zwischengespeicherten ObjectGrid-Instanzen.
* @ibm-api
*/
public List getObjectGrids();
/**
* Diese Methode verwenden, wenn bereits eine ObjectGrid-Instanz
* vorhanden ist. Sie gibt zwischengespeicherte ObjectGrid-Instanzen
* nach Namen zurück. Die Methode gibt null zurück, wenn keine
* ObjectGrid-Instanz mit diesem ObjectGrid-Namen zwischengespeichert
* wurde.
*
* @param objectGridName Der Name der zwischengespeicherten
*                       ObjectGrid-Instanz.
* @return Eine bereits vorhandene zwischengespeicherte ObjectGrid-Instanz.
*
* @since WAS XD 6.0
* @ibm-api
*
*/
public ObjectGrid getObjectGrid(String objectGridName);
```

removeObjectGrid-Methoden

Dieser Abschnitt beschreibt, wie die beiden removeObjectGrid-Methoden verwendet werden.

Eine ObjectGrid-Instanz entfernen

Sie können für das Entfernen von ObjectGrid-Instanzen eine der folgenden removeObjectGrid-Methoden verwenden. ObjectGridManager verwaltet anschließend keine Referenzen auf die entfernten Instanzen. Es sind zwei Methoden verfügbar. Die eine Methode akzeptiert einen Booleschen Parameter. Wenn der Boolesche Parameter auf **true** gesetzt ist, wird die Methode destroy für die ObjectGrid-Instanz aufgerufen. Der Aufruf der Methode destroy für die ObjectGrid-Instanz bewirkt, dass die ObjectGrid-Instanz beendet wird, und gibt alle von der Instanz verwendeten Ressourcen frei. Im Folgenden wird beschrieben, wie die beiden removeObjectGrid-Methoden verwendet werden.

```
/**
* Eine ObjectGrid-Instanz aus dem Cache für ObjectGrid-Instanzen entfernen
* @param objectGridName Der Name der aus dem Cache zu entfernenden
* ObjectGrid-Instanz.
* @throws ObjectGridException, falls keine ObjectGrid-Instanz mit dem
* angegebenen Namen im Cache gefunden wird.
* @ibm-api
*/
public void removeObjectGrid(String objectGridName) throws ObjectGridException;
/**
* Eine ObjectGrid-Instanz aus dem Cache für ObjectGrid-Instanzen entfernen
* und die der Instanz zugeordneten Ressourcen freigeben.
* @param objectGridName Der Name der aus dem Cache zu entfernenden
* ObjectGrid-Instanz.
* @param destroy Die ObjectGrid-Instanz löschen und die der Instanz
* zugeordneten Ressourcen freigeben.
* @throws ObjectGridException, falls keine ObjectGrid-Instanz mit dem
* angegebenen Namen im Cache gefunden wird.
* @ibm-api
*/
public void removeObjectGrid(String objectGridName, boolean destroy)
throws ObjectGridException;
```

Mit dem Interface ObjectGridManager die Gültigkeitsdauer einer ObjectGrid-Instanz steuern

Dieser Abschnitt beschreibt, wie Sie mit dem Interface ObjectGridManager die Gültigkeitsdauer einer ObjectGrid-Instanz unter Verwendung von Startup-Beans und eines Servlet steuern können.

Gültigkeitsdauer einer ObjectGrid-Instanz in einer Startup-Bean verwalten

Mit einer Startup-Bean können Sie die Gültigkeitsdauer einer ObjectGrid-Instanz steuern. Eine Startup-Bean wird beim Anwendungsstart geladen. Mit einer Startup-Bean kann Code ausgeführt werden, wenn eine Anwendung wie erwartet gestartet oder gestoppt wird. Verwenden Sie zum Erstellen einer Startup-Bean das Home-Interface `com.ibm.websphere.startupservice.AppStartUpHome` und das Remote-Interface `com.ibm.websphere.startupservice.AppStartUp`. Implementieren Sie die Methoden `start` und `stop` in der Bean. Die Methode `start` wird aufgerufen, wenn die Anwendung gestartet wird. Die Methode `stop` wird aufgerufen, wenn die Anwendung beendet wird. Sie können die Methode `start` verwenden, um ObjectGrid-Instanzen zu erstellen. Mit der Methode `stop` können ObjectGrid-Instanzen gelöscht werden. Das folgende Code-Snippet veranschaulicht, wie die Gültigkeitsdauer einer ObjectGrid-Instanz in einer Startup-Bean verwaltet wird.

```
public class MeineStartupBean implements javax.ejb.SessionBean {
private ObjectGridManager objectGridManager;
/*
* Die Methoden im Interface SessionBean wurden im Hinblick auf
* die Übersichtlichkeit in diesem Beispiel weggelassen.
*/
public boolean start(){
 // Startup-Bean starten
 // Diese Methode wird beim Anwendungsstart aufgerufen.
 objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
 try {
  // Zwei ObjectGrid-Instanzen erstellen und diese Instanzen
  // zwischenspeichern.
  ObjectGrid bookstoreGrid =
   objectGridManager.createObjectGrid("bookstore", true);
  bookstoreGrid.defineMap("book");
  ObjectGrid videostoreGrid =
```

```
   objectGridManager.createObjectGrid("videostore", true);
  // In der JVM können diese ObjectGrid-Instanzen jetzt
  // mit der Methode getObjectGrid(String)
  // vom Interface ObjectGridManager abgerufen werden
 } catch (ObjectGridException e) {
  e.printStackTrace();
  return false;
 }
 return true;
}
public void stop(){
 // Startup-Bean stoppen
 // Diese Methode wird aufgerufen, wenn die Anwendung gestoppt wird.
 try {
  // Die zwischengespeicherten ObjectGrid-Instanzen entfernen
  // und löschen.
  objectGridManager.removeObjectGrid("bookstore", true);
  objectGridManager.removeObjectGrid("videostore", true);
  } catch (ObjectGridException e) {
   e.printStackTrace();
  }
 }
}
```

Nach dem Aufruf der Methode `start` können die neu erstellten ObjectGrid-Instanzen vom Interface ObjectGridManager abgerufen werden. Wenn die Anwendung beispielsweise ein Servlet enthält, kann das Servlet mit dem folgenden Code-Snippet auf diese ObjectGrid-Instanzen zugreifen:

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

Gültigkeitsdauer einer ObjectGrid-Instanz in einem Servlet verwalten

Eine Möglichkeit für die Verwaltung der Gültigkeitsdauer einer ObjectGrid-Instanz in einem Servlet ist das Erstellen der ObjectGrid-Instanz mit der Methode `init` und das Löschen der ObjectGrid-Instanz mit der Methode `destroy`. Wenn die ObjectGrid-Instanz zwischengespeichert wird, kann Sie mit dem Servlet-Code abgerufen und bearbeitet werden. Der folgende Beispielcode veranschaulicht, wie eine ObjectGrid-Instanz in einem Servlet erstellt, bearbeitet und gelöscht wird.

```
public class MeinObjectGridServlet extends HttpServlet implements Servlet {
 private ObjectGridManager objectGridManager;
 public MeinObjectGridServlet() {
 super();
}
public void init(ServletConfig arg0) throws ServletException {
 super.init();
 objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
 try {
  // Eine ObjectGrid-Instanz mit dem Namen bookstore erstellen
  // und zwischenspeichern.
  ObjectGrid bookstoreGrid =
   objectGridManager.createObjectGrid("bookstore", true);
  bookstoreGrid.defineMap("book");
 } catch (ObjectGridException e) {
  e.printStackTrace();
 }
}
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
 ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
```

```
 BackingMap bookMap = bookstoreGrid.getMap("book");
 // Führt Operationen für die zwischengespeicherte ObjectGrid-Instanz aus.
 // ...
}
public void destroy() {
 super.destroy();
 try {
  // Die zwischengespeicherte ObjectGrid-Instanz bookstore entfernen
  // und löschen.
  objectGridManager.removeObjectGrid("bookstore", true);
  } catch (ObjectGridException e) {
  e.printStackTrace();
  }
 }
}
```

Trace für ObjectGrid konfigurieren

Dieser Abschnitt beschreibt, wie die Trace-Verarbeitung für ObjectGrid konfiguriert wird.

J2SE-Umgebung (Java 2 Platform, Standard Edition)

Wenn Sie Debug-Informationen an IBM senden müssen, verwenden Sie den Trace-Mechanismus, um den Debug-Trace abzurufen. Das folgende Beispiel zeigt, wie Sie den Debug-Trace in einer J2SE-Umgebung abrufen:

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

Das Beispiel enthält keine Trace-Verarbeitung für die integrierten Evictor-Plug-ins für ObjectGrid. Wenn Sie die von ObjectGrid bereitgestellten Evictor-Plug-ins verwenden und Probleme auftreten, die mit der Bereinigung in Zusammenhang stehen könnten, aktivieren Sie die Trace-Verarbeitung für ObjectGrid und die Evictor-Plug-ins von ObjectGrid wie im folgenden Beispiel gezeigt:

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

Umgebung mit WebSphere Application Server

In einer Umgebung mit WebSphere Application Server ist es nicht erforderlich, den Trace mit dem Interface ObjectGridManager zu konfigurieren. Sie können die Trace-Spezifikation in der Administrationskonsole festlegen.

Interface ObjectGrid

Dieser Abschnitt beschreibt die Methoden, die zum Modifizieren einer ObjectGrid-Instanz erforderlich sind.

Einführung

ObjectGrid ist ein erweiterbares, transaktionsorientiertes Framework für das Zwischenspeichern (Caching) von Objekten, das auf dem Java-Interface Map basiert. Die Operationen der ObjectGrid-API sind in einer transaktionsorientierten Arbeitseinheit gruppiert und können durch benutzerdefinierte Plug-ins erweitert werden. ObjectGrid ist ein benannter logischer Container, der BackingMaps enthält. Nähere Informationen zum Interface BackingMap finden Sie im Abschnitt „Interface BackingMap” auf Seite 47.

Erstellen und initialisieren

Die erforderlichen Schritte zum Erstellen einer ObjectGrid-Instanz sind im Abschnitt über das Interface ObjectGridManager beschrieben. Eine ObjectGrid-Instanz kann über das Programm oder mit XML-Konfigurationsdateien erstellt werden. Nähere Informationen finden Sie im Abschnitt „Interface ObjectGridManager" auf Seite 37.

Get-, Set- und Factory-Methoden

Das Interface ObjectGrid enthält die folgenden Methoden:

Tabelle 4. Methoden des Interface ObjectGrid

Methode	Beschreibung
`BackingMap defineMap(String name);`	*defineMap* ist eine Factory-Methode, mit der Sie eine eindeutig benannte BackingMap-Instanz definieren können. Nähere Informationen zum Interface BackingMap finden Sie im Abschnitt „Interface BackingMap" auf Seite 47.
`BackingMap getMap(String name);`	*getMap* gibt eine zuvor mit der Methode *defineMap* definierte BackingMap-Instanz zurück. Mit dieser Methode können Sie die BackingMap-Instanz konfigurieren, falls diese noch nicht mit XML-Konfigurationsdateien konfiguriert wurde.
`public Session getSession() throws ObjectGridException, TransactionCallbackException;`	*getSession* gibt ein Session-Objekt zurück, das die Funktionen begin, commit und rollback für eine Arbeitseinheit enthält. Nähere Informationen zu Session-Objekten finden Sie im Abschnitt „Interface Session" auf Seite 51.
`void initialize() throws ObjectGridException;`	*initialize* initialisiert die ObjectGrid-Instanz und macht sie allgemein verfügbar. Diese Methode wird beim Aufruf der Methode `getSession` implizit aufgerufen, falls die ObjectGrid-Instanz noch nicht initialisiert ist.
`void destroy();`	*destroy*: Das Framework wird gelöscht und kann nach dem Aufruf dieser Methode nicht mehr verwendet werden.
`//Schlüsselwörter.`	
`void associateKeyword(Serializable parent, Serializable child);`	*associateKeyword*: ObjectGrid stellt einen flexiblen Invalidierungsmechanismus bereit, der auf Schlüsselwörtern basiert. Nähere Informationen zu Schlüsselwörtern finden Sie im Abschnitt „Schlüsselwörter" auf Seite 58. Diese Methode verbindet zwei Schlüsselwörter in einer gerichteten Beziehung. Wenn ein übergeordnetes Objekt ungültig gemacht wird, wird auch das untergeordnete Objekt ungültig gemacht. Wird das untergeordnete Objekt ungültig gemacht, hat dies keine Auswirkung auf das übergeordnete Objekt.
`//Sicherheit`	
`void setSecurityEnabled()`	*setSecurityEnabled* aktiviert die Sicherheit. Die Sicherheit ist standardmäßig inaktiviert.

Tabelle 4. Methoden des Interface ObjectGrid (Forts.)

Methode	Beschreibung
`void setPermissionCheckPeriod(long period);`	*setPermissionCheckPeriod*: Diese Methode akzeptiert einen einzigen Parameter, der angibt, wie oft die Berechtigung für einen Clientzugriff geprüft wird. Wenn der Parameter den Wert `0` hat, weisen alle Methoden den Autorisierungsmechanismus (JAAS-Autorisierung oder benutzerdefinierte Autorisierung) an zu prüfen, ob das aktuelle Subjekt die erforderliche Berechtigung besitzt. Diese Strategie kann sich je nach Autorisierungsimplementierung auf die Leistung auswirken. Dieser Autorisierungstyp von Berechtigung ist jedoch verfügbar, falls er benötigt wird. Wenn der Parameterwert kleiner als `0` ist, gibt dieser den Zeitraum (in Millisekunden) an, für den die Berechtigungen zwischengespeichert werden, bevor sie vom Autorisierungsmechanismus aktualisiert werden. Dieser Parameter liefert eine erheblich bessere Leistung. Sollten sich die Backend-Berechtigungen jedoch in diesem Zeitraum ändern, ist es möglich, dass die ObjectGrid-Instanz den Zugriff zulässt oder verhindert, obwohl der Sicherheits-Provider des Backend modifiziert wurde.
`void setAuthorizationMechanism(int authMechanism);`	*setAuthorizationMechanism* legt den Autorisierungsmechanismus fest. Die Standardeinstellung ist `SecurityConstants.JAAS_AUTHORIZATION`.

Plug-ins für das Interface ObjectGrid

Das Interface ObjectGrid hat mehrere optionale Plug-in-Punkte für weitergehende Interaktionen.

```
void addEventListener(ObjectGridEventListener cb);
void removeEventListener(ObjectGridEventListener cb);
void setTransactionCallback(TransactionCallback callback);
// Sicherheitsbezogene Plug-ins
void setSubjectValidation(SubjectValidation subjectValidation);
void setSubjectSource(SubjectSource source);
void setMapAuthorization(MapAuthorization mapAuthorization);
```

- *ObjectGridEventListener*: Das Interface ObjectGridEventListener wird verwendet, um Benachrichtigungen zu empfangen, wenn bedeutende Ereignisse in der ObjectGrid-Instanz eintreten. Zu diesen Ereignissen gehören die ObjectGrid-Initialisierung, der Beginn einer Transaktion, das Ende einer Transaktion und das Löschen einer ObjectGrid-Instanz. Zur Überwachung solcher Ereignisse erstellen Sie eine Klasse, die das Interface ObjectGridEventListener implementiert. Fügen Sie diese Klasse anschließend der ObjectGrid-Instanz hinzu. Diese Listener werden dem Session-Objekt hinzugefügt und aus diesem entfernt. Nähere Informationen finden Sie in den Abschnitten „Listener" auf Seite 88 und „Interface Session" auf Seite 51.
- *TransactionCallback*: Das Listener-Interface TransactionCallback ermöglicht, dass transaktionsorientierte Ereignisse wie begin-, commit- und rollback-Signale an dieses Interface gesendet werden. In der Regel wird das Listener-Interface TransactionCallback zusammen mit einem Ladeprogramm (Loader) verwendet.

Nähere Informationen finden Sie in den Abschnitten „Plug-in TransactionCallback" auf Seite 119 und „Loader" auf Seite 102. Diese Ereignisse können anschließend verwendet werden, um Transaktionen mit einer externen Ressource oder innerhalb mehrerer Ladeprogramme zu koordinieren.

- *SubjectValidation*: Wenn die Sicherheit aktiviert ist, kann dieses Plug-in verwendet werden, um eine Klasse `javax.security.auth.Subject` zu validieren, die an die ObjectGrid-Instanz übergeben wird.
- *MapAuthorization*: Wenn die Sicherheit aktiviert ist, kann dieses Plug-in verwendet werden, um ObjectMap-Zugriff auf die Principals zu autorisieren, die vom Objekt 'Subject' dargestellt werden.
- *SubjectSource*: Wenn die Sicherheit aktiviert ist, kann dieses Plug-in verwendet werden, um ein Objekt 'Subject' abzurufen, das den ObjectGrid-Client darstellt. Dieses Objekt 'Subject" wird anschließend für die ObjectGrid-Autorisierung verwendet.

Interface BackingMap

Jede ObjectGrid-Instanz enthält eine Sammlung von BackingMap-Objekten.

Jedes BackingMap-Objekt wird benannt und mit der Methode `defineMap` des Interface ObjectGrid einer ObjectGrid-Instanz hinzugefügt. Die Methode `defineMap` gibt eine BackingMap-Instanz zurück, die anschließend verwendet wird, um das Verhalten einer bestimmten Map-Instanz zu definieren.

Das Interface Session wird verwendet, um eine Transaktion zu starten und die ObjectMap- bzw. JavaMap-Instanz abzurufen, die für die Durchführung transaktionsorientierter Interaktionen zwischen einer Anwendung und einem BackingMap-Objekt erforderlich ist. Die Transaktionsänderungen werden jedoch erst auf das BackingMap-Objekt angewendet, wenn die Transaktion festgeschrieben wird. Ein BackingMap-Objekt ist ein Speicher-Cache mit festgeschriebenen Daten für eine bestimmte Map-Instanz. Nähere Informationen zum Interface Session finden Sie im Abschnitt „Interface Session" auf Seite 51.

Das Interface com.ibm.websphere.objectgrid.BackingMap stellt Methoden für das Festlegen (set) von BackingMap-Attributen bereit. Manche der set-Methoden unterstützen die Erweiterung eines BackingMap-Objekts durch mehrere benutzerdefinierte Plug-ins. Die folgende Liste enthält die `set`-Methoden, mit denen Sie Attribute festlegen können und die die Implementierung benutzerdefinierter Plug-ins unterstützen:

```
 // Methoden für die Festlegung von BackingMap-Attributen
public void setReadOnly(boolean readOnlyEnabled);
public void setNullValuesSupported(boolean nullValuesSupported);
public void setLockStrategy( LockStrategy lockStrategy );
public void setCopyMode(CopyMode mode, Class valueInterface);
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);

// Methoden für die Festlegung optionaler, benutzerdefinierter Plug-ins,
// die von der Anwendung bereitgestellt werden
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 );
```

Für jede der aufgelisteten set-Methoden gibt es eine entsprechende get-Methode.

BackingMap-Attribute

Jede BackingMap-Instanz hat die folgenden Attribute, mit denen das BackingMap-Verhalten modifiziert und gesteuert werden kann:

- Attribut *ReadOnly*. Dieses Attribut gibt an, ob die Map-Instanz schreibgeschützt ist oder Lese- und Schreibzugriffe zulässt. Wenn Sie dieses Attribut für die Map-Instanz nicht setzen, lässt die Map-Instanz Lese- und Schreibzugriffe standardmäßig zu. Wenn Sie den Schreibschutz für eine BackingMap-Instanz setzen, optimiert ObjectGrid das System, sofern möglich, für Lesezugriffe.
- Attribut *NullValuesSupported*. Dieses Attribut gibt an, ob ein Nullwert in die Map-Instanz gestellt werden kann. Wenn Sie dieses Attribut nicht setzen, unterstützt die Map-Instanz keine Nullwerte. Falls Nullwerte von der Map-Instanz unterstützt werden und eine `get`-Operation einen Nullwert zurückgibt, kann dies bedeuten, dass der Wert null ist oder dass die Map-Instanz den von der Operation `get` angegebenen Schlüssel nicht enthält.
- Attribut *LockStrategy*. Dieses Attribut bestimmt, ob das BackingMap-Objekt einen Sperrenmanager verwendet. Wenn ein Sperrenmanager verwendet wird, gibt das Attribut LockStrategy an, ob eine optimistische oder pessimistische Strategie für das Sperren der Map-Einträge verwendet wird. Es wird eine optimistische Sperrstrategie verwendet, wenn das Attribut nicht gesetzt ist. Nähere Einzelheiten zu den unterstützten Sperrstrategien finden Sie im Abschnitt „Sperren" auf Seite 64.
- Attribut *CopyMode*. Dieses Attribut bestimmt, ob die BackingMap-Instanz eine Kopie eines Wertobjekts erstellt, wenn in der Festschreibungsphase einer Transaktion ein Wert aus der Map gelesen oder in die BackingMap-Instanz gestellt wird. Es werden verschiedene Kopiermodi unterstützt, damit die Anwendung Leistung und Datenintegrität gegeneinander abwägen kann. Wenn dieses Attribut nicht gesetzt ist, wird der Kopiermodus `COPY_ON_READ_AND_COMMIT` verwendet. Dieser Kopiermodus bietet zwar nicht die beste Leistung, aber den größten Schutz vor Datenintegritätsproblemen. Nähere Informationen zu den Kopiermodi finden Sie im Abschnitt „Best Practices für die Methode copyMode" auf Seite 188.
- Attribut *NumberOfBuckets*. Dieses Attribut gibt die Anzahl der von der BackingMap-Instanz zu verwendenden Hash-Buckets an. Die BackingMap-Implementierung verwendet eine Hash-Tabelle. Wenn die BackingMap-Instanz eine hohe Anzahl von Einträgen enthält, können Sie mit einer höheren Anzahl von Buckets eine bessere Leistung erzielen. Die Anzahl der Schlüssel, die denselben Bucket haben, geht mit zunehmender Anzahl von Buckets zurück. Außerdem sind bei einer höheren Anzahl von Buckets mehr gemeinsame Zugriffe möglich. Dieses Attribut ist hilfreich für die Leistungsoptimierung. Wenn das Attribut NumberOfBuckets nicht in der Anwendung gesetzt ist, wird der Standardwert `503` verwendet.
- Attribut *NumberOfLockBuckets*. Dieses Attribut gibt die Anzahl der Sperren-Buckets an, die der Sperrenmanager für diese BackingMap-Instanz verwenden kann. Wenn das Attribut LockStrategy auf OPTIMISTIC oder PESSIMISTIC gesetzt ist, wird ein Sperrenmanager für die BackingMap-Instanz erstellt. Der Sperrenmanager verwendet eine Hash-Tabelle, um die Einträge zu verfolgen, die von einer oder mehreren Transaktionen gesperrt werden. Wenn die Hash-Tabelle viele Einträge enthält, können Sie mit einer höheren Anzahl von Buckets eine bessere Leistung erzielen, weil die Anzahl kollidierender Schlüssel für denselben

Bucket mit zunehmender Anzahl von Buckets abnimmt. Außerdem sind bei einer höheren Anzahl von Buckets mehr gemeinsame Zugriffe möglich. Wenn Sie das Attribut LockStrategy auf NONE setzen, verwendet die BackingMap-Instanz keinen Sperrenmanager. In diesem Fall hat das Attribut numberOfLockBuckets keine Wirkung. Wenn Sie dieses Attribut nicht setzen, wird der Standardwert 383 verwendet.

- Attribut *LockTimeout*. Dieses Attribut wird verwendet, wenn die BackingMap-Instanz einen Sperrenmanager verwendet. Die BackingMap-Instanz verwendet einen Sperrenmanager, wenn das Attribut LockStrategy auf OPTIMISTIC oder PESSIMISTIC gesetzt ist. Der Attributwert wird in Sekunden angegeben und bestimmt, wie lange der Sperrenmanager auf die Erteilung einer Sperre wartet. Wenn dieses Attribut nicht gesetzt ist, wird für LockTimeout ein Wert von 15 Sekunden verwendet. Nähere Einzelheiten zu den Ausnahmen, die beim Warten auf eine Sperre auftreten können, finden Sie im Abschnitt „Pessimistisches Sperren" auf Seite 65.
- Attribut *TtlEvictorType*. Jede BackingMap-Instanz hat einen eigenen Evictor, der einen zeitbasierten Algorithmus verwendet, um die Einträge zu ermitteln, die aus der Map gelöscht werden können. Standardmäßig ist dieser integrierte Evictor nicht aktiv. Sie können den Evictor mit der Methode setTtlEvictorType aktivieren. Die gültigen Werte für diese Methode sind `CREATION_TIME`, `LAST_ACCESS_TIME` und `NONE`. Der Wert `CREATION_TIME` gibt an, dass der Evictor den Wert des Attributs TimeToLive und die Zeit, zu der der Map-Eintrag in der BackingMap-Instanz erstellt wurde, addiert, um den Zeitpunkt zu bestimmen, zu dem der Map-Eintrag aus der BackingMap-Instanz gelöscht wird. Der Wert `LAST_TIME` gibt an, dass der Evictor den Wert des Attributs TimeToLive und die Zeit, zu der der letzte Zugriff auf den Map-Eintrag durch eine in der Anwendung ausgeführte Transaktion erfolgt ist, addiert, um den Zeitpunkt zu bestimmen, zu dem der Map-Eintrag gelöscht wird. Der Map-Eintrag wird nur gelöscht, wenn der letzte Zugriff einer Transaktion länger zurückliegt, als der mit dem Attribut TimeToLive angegebene Wert. Der Wert NONE gibt an, dass der Evictor inaktiv bleiben und keine Map-Einträge löschen soll. Wenn dieses Attribut nicht gesetzt wird, ist NONE der Standardwert, und der Evictor bleibt inaktiv. Nähere Einzelheiten zum integrierten TTL-Evictor finden Sie im Abschnitt „Evictor" auf Seite 93.
- Attribut *TimeToLive*. Mit diesem Attribut wird die Anzahl der Sekunden angegeben, die der integrierte TTL-Evictor zur Erstellungs- oder letzten Zugriffszeit jedes Eintrags addieren muss (siehe Beschreibung des Attributs TtlEvictorType). Wenn dieses Attribut nicht gesetzt ist, wird der Sonderwert null verwendet, um eine unbegrenzte Lebensdauer festzulegen. Ist eine unbegrenzte Lebensdauer definiert, löscht der Evictor keine Map-Einträge.

Das folgende Beispiel zeigt, wie eine BackingMap-Instanz someMap in einer ObjectGrid-Instanz someGrid und verschiedene Attribute der BackingMap-Instanz mit den set-Methoden des Interface BackingMap definiert werden:

```
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 ); // Lese-/Schreibzugriff (Standardeinstellung) überschreiben
bm.setNullValuesSupported(false); // Nullwerte zulassen (Standard) überschreiben
bm.setLockStrategy( LockStrategy.PESSIMISTIC ); // OPTIMISTIC (Standard) überschreiben
```

```
bm.setLockTimeout( 60 ); // Standardwert von 15 Sekunden überschreiben
bm.setNumberOfBuckets(251); // Standard überschreiben (Primzahlen am besten)
bm.setNumberOfLockBuckets(251); // Standard überschreiben (Primzahlen am besten)

...
```

Plug-ins für das Interface BackingMap

Das Interface BackingMap hat mehrere optionale Plug-in-Punkte für weitergehende Interaktionen mit dem ObjectGrid-Framework:

- **Plug-in ObjectTransformer**: Für einige Map-Operationen muss eine BackingMap-Instanz möglicherweise den Schlüssel oder Wert eines Eintrags in der BackingMap-Instanz serialisieren, entserialisieren oder kopieren. Die BackingMap-Instanz kann diese Aktionen ausführen, indem sie eine Standardimplementierung des Interface ObjectTransformer bereitstellt. Eine Anwendung kann ihre Leistung mit einem eigenen ObjectTransformer-Plug-in verbessern, das die BackingMap zum Serialisieren, Entserialisieren oder Kopieren eines Eintragsschlüssels bzw. -wertes in der BackingMap-Instanz verwendet.
- **Plug-in Evictor**: Der integrierte TTL-Evictor verwendet einen zeitbasierten Algorithmus, um zu entscheiden, wann ein Eintrag aus der BackingMap-Instanz gelöscht wird. Manche Anwendungen müssen möglicherweise einen anderen Algorithmus verwenden. Das Plug-in Evictor stellt der BackingMap einen angepassten Evictor zur Verfügung. Das Plug-in Evictor wird zusätzlich zum integrierten TTL-Evictor bereitgestellt und ist kein Ersatz für diesen. ObjectGrid stellt ein angepasstes Evictor-Plug-in bereit, das bekannte Algorithmen wie LRU (Least Recently Used, zuletzt verwendet) oder LFU (Least Frequently Used, am wenigsten verwendet) implementiert. Anwendungen können eines der bereitgestellten Evictor-Plug-ins oder ein eigenes Evictor-Plug-in verwenden.
- **Plug-in MapEventListener**: Eine Anwendung möchte möglicherweise über BackingMap-Ereignisse wie das Löschen eines Map-Eintrags oder das Vorabladen einer BackingMap-Instanz informiert werden. Eine BackingMap-Instanz ruft Methoden im Plug-in MapEventListener auf, um eine Anwendung über BackingMap-Ereignisse zu benachrichtigen. Eine Anwendung kann Benachrichtigungen über verschiedene BackingMap-Ereignisse empfangen. Dazu müssen Sie mit der Methode setMapEventListener angepasste MapEventListener-Plug-ins für die BackingMap-Instanz bereitstellen. Die Anwendung kann die aufgelisteten MapEventListener-Objekte mit der Methode addMapEventListener oder der Methode removeMapEventListener modifizieren.
- **Plug-in Loader**: Ein BackingMap-Objekt ist der Speicher-Cache für eine Map-Instanz. Ein Loader-Plug-in ist eine Option, die die BackingMap-Instanz verwendet, um Daten in den und aus dem Hauptspeicher zu verschieben, und ein persistenter Speicher für die BackingMap-Instanz. Beispielsweise kann ein JDBC-Loader verwendet werden, um Daten in und aus BackingMap-Instanzen und relationalen Tabellen einer relationalen Datenbank zu verschieben. Als persistenter Speicher für eine BackingMap-Instanz muss muss keine relationale Datenbank verwendet werden. Mit dem Loader können auch Daten zwischen einer BackingMap-Instanz und einer Datei, einem Hibernate-Map, einer J2EE-Entity-Bean, einem anderen Anwendungsserver usw. verschoben werden. Die Anwendung muss für jede verwendete Technologie ein angepasstes Loader-Plug-in bereitstellen, um Daten zwischen der BackingMap-Instanz und dem persistenten Speicher zu verschieben. Falls die Anwendung keinen Loader bereitstellt, wird aus der BackingMap-Instanz ein einfacher Speicher-Cache.
- **Plug-in OptimisticCallback**: Wenn das Attribut LockStrategy für eine BackingMap-Instanz auf OPTIMISTIC gesetzt ist, muss die BackingMap-Instanz oder ein Loader-Plug-in Vergleichsoperationen für die Map-Werte durchführen.

Das Plug-in OptimisticCallback wird von der BackingMap-Instanz und dem Loader verwendet, um die Versionsvergleichsoperationen nach einer optimistischen Strategie durchzuführen.

Interface Session

Dieser Abschnitt beschreibt, wie Anwendungen Transaktionen mit dem Interface Session starten und beenden. Das Interface Session unterstützt außerdem den Zugriff auf die anwendungsbasierten Interfaces ObjectMap und JavaMap.

Einführung

Jede ObjectMap- und JavaMap-Instanz ist direkt an ein bestimmtes Session-Objekt gebunden. Jeder Thread, der auf ein ObjectGrid-Objekt zugreifen möchte, muss zuerst eine Session-Instanz vom ObjectGrid-Objekt anfordern. Eine Session-Instanz kann nicht von mehreren Threads gemeinsam genutzt werden. ObjectGrid verwendet keinen lokalen Thread-Speicher für die einzelnen Threads. Es können jedoch plattformspezifische Bedingungen gelten, die die Übergabe einer Session-Instanz von einem Thread an einen anderen einschränken.

Methoden

Das Interface Session stellt die folgenden Methoden bereit. Nähere Informationen zu den folgenden Methoden finden Sie in der API-Dokumentation:

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

 void begin()
 throws TransactionAlreadyActiveException, TransactionException;

 void beginNoWriteThrough()
 throws TransactionAlreadyActiveException, TransactionException;

 public void commit()
 throws NoActiveTransactionException, TransactionException;

 public void rollback()
 throws NoActiveTransactionException, TransactionException;

 public void flush()
 throws TransactionException; ObjectGrid getObjectGrid();

 TxID getTxID()
 throws NoActiveTransactionException;

 boolean isWriteThroughEnabled();

 void setTransactionType(String tranType);

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

Eine Sitzung anfordern

Eine Anwendung fordert mit der Methode `ObjectGrid.getSession` eine Session-Instanz von einem ObjectGrid-Objekt an. Das folgende Code-Snippet zeigt, wie eine Session-Instanz angefordert wird:

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

Nach dem Erhalt der Session-Instanz verwaltet der Thread zur eigenen Verwendung eine Referenz auf diese Sitzung. Bei jedem Aufruf der Methode getSession wird ein neues Session-Objekt zurückgegeben.

Transaktionen und Sitzungen

Mit einer Session-Instanz können Transaktionen gestartet, festgeschrieben und rückgängig gemacht werden. Operationen für BackingMap-Objekte, die mit ObjectMap- und JavaMap-Instanzen ausgeführt werden, sind am effizientesten in einer Session-Transaktion. Nach dem Start einer Transaktion werden alle Änderungen, die im Rahmen dieser Transaktion an BackingMap-Objekten vorgenommen werden, so lange in einem besonderen Transaktions-Cache gespeichert, bis die Transaktion festgeschrieben wird. Wenn eine Transaktion festgeschrieben wird, werden die anstehenden Änderungen auf die BackingMap-Objekte und Loader angewendet und damit den anderen Clients dieser ObjectGrid-Instanz bereitgestellt.

ObjectGrid unterstützt auch die Möglichkeit, Transaktionen automatisch festzuschreiben. Wenn ObjectMap-Operationen außerhalb des Kontextes einer aktiven Transaktion durchgeführt werden, wird eine implizite Transaktion gestartet, bevor die Operation und die Transaktion automatisch festgeschrieben werden und die Steuerung an die Anwendung zurückgegeben wird.

```
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"); // Automatisches Festschreiben
```

Methode Session.flush

Die Methode `Session.flush` ist nur sinnvoll, wenn einer BackingMap-Instanz ein Loader zugeordnet ist. Die Methode ruft den Loader mit den derzeit im Transaktions-Cache gespeicherten Änderungen auf. Der Loader wendet die Änderungen auf das Backend an. Diese Änderungen werden nicht festgeschrieben, wenn die Methode Session.flush aufgerufen wird. Wenn eine Session-Transaktion nach dem Aufruf der Methode Session.flush festgeschrieben wird, werden nur die Aktualisierungen in den Loader geschrieben, die nach dem Aufruf der Methode Session.flush vorgenommen wurden. Wenn eine Session-Transaktion nach dem Aufruf der Methode Session.flush rückgängig gemacht wird, werden die in den Loader geschriebenen Änderungen zusammen mit allen in der Transaktion anstehenden Änderungen verworfen. Verwenden Sie die Methode Session.flush sparsam, da sie die Ausführung von Stapeloperationen für einen Loader einschränkt. Das folgende Beispiel zeigt die Verwendung der Methode `Session.flush`:

```
Session session = objectGrid.getSession();
session.begin();
// einige Änderungen vornehmen
...
session.flush(); // die Änderungen in den Loader schreiben, aber noch nicht festschreiben
// weitere Änderungen vornehmen...
...
session.commit();
```

Methode ohne Durchschreiben

Einige ObjectGrid-Maps werden von einem Loader gestützt, der einen persistenten Speicher für die Daten in der Map bietet. Manchmal ist es sinnvoll, die Daten nur in der ObjectGrid-Map festzuschreiben und nicht in den Loader zu schreiben. Das Interface Session stellt für diesen Zweck die Methode `beginNoWriteThough` bereit. Die Methode `beginNoWriteThrough` startet wie die Methode `begin` eine Transaktion. Wenn Sie die Methode `beginWriteThrough` verwenden, werden die Daten beim Festschreiben der Transaktion nur im ObjectGrid-Speicher-Cache und nicht in dem persistenten Speicher des Loader festgeschrieben. Diese Methode ist sehr hilfreich, wenn Daten vorher in die Map geladen werden.

Das Interface Session enthält auch die Methode `isWriteThroughEnabled`, mit der der Typ der derzeit aktiven Transaktion bestimmt werden kann.

```
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
// Änderungen vornehmen...
session.commit(); // Diese Änderungen werden nicht in den Loader geschrieben
```

Objekt TxID anfordern

Das Objekt TxID ist ein nicht transparentes Objekt, das die aktive Transaktion angibt. Das Objekt TxID kann für Folgendes verwendet werden:

- Für einen Vergleich, wenn Sie eine bestimmte Transaktion suchen.
- Zum Speichern gemeinsam genutzter Daten von TransactionCallback- und Loader-Objekten.

Nähere Informationen zum Slot-Feature für Objekte finden Sie in den Abschnitten „Plug-in TransactionCallback” auf Seite 119 und „Loader” auf Seite 102.

Transaktionstyp für die Leistungsüberwachung festlegen

Wenn Sie ObjectGrid in einem Anwendungsserver von WebSphere Application Server verwenden, müssen Sie möglicherweise den Transaktionstyp für die Leistungsüberwachung zurücksetzen. Sie können den Transaktionstyp mit der Methode setTransactionType festlegen. Nähere Informationen zur Methode setTransactionType finden Sie im Abschnitt „ObjectGrid-Leistung mit WebSphere Application Server Performance Monitoring Infrastructure (PMI) überwachen” auf Seite 149.

Ein vollständiges LogSequence-Objekt verarbeiten

ObjectGrid kann Sätze von Map-Änderungen an andere ObjectGrid-Listener weitergeben, um Maps von einer JVM (Java Virtual Machine) an eine andere zu verteilen. Nähere Informationen zu dieser Funktionalität finden Sie im Abschnitt „Weitergabe von verteilten Transaktionen in ObjectGrid” auf Seite 156. Damit der Listener die empfangenen LogSequence-Objekte einfacher verarbeiten kann, stellt das Interface Session die Methode `processLogSequence` bereit. Diese Methode untersucht jedes LogElement-Objekt im LogSequence-Objekt und führt die entsprechende Operation (z. B. insert, update, invalidate usw.) für die BackingMap-Instanz durch, die im Element MapName des LogSequence-Objekts angegeben ist. Damit die Methode `processLogSequence` aufgerufen werden kann, muss eine ObjectGrid-Sitzung aktiv sein. Außerdem muss die Anwendung die entsprechenden COMMIT- und ROLLBACK-Aufrufe absetzen, um die Sitzung abzuschließen. Für diesen Methodenaufruf wird keine automatische Festschreibung unterstützt.

Normalerweise startet der empfangende ObjectGridEventListener in der fernen JVM eine Sitzung mit der Methode `beginNoWriteThrough`, die verhindert, dass Änderungen endlos weitergegeben werden. Danach wird die Methode `processLogSequence` aufgerufen und abschließend wird die Transaktion festgeschrieben bzw. rückgängig gemacht.

```
// Das bei der Initialisierung des ObjectGridEventListener
// übergebene Session-Objekt verwenden
session.beginNoWriteThrough();
// Das empfangene LogSequence-Objekt verarbeiten
try {
 session.processLogSequence(receivedLogSequence);
} catch (Exception e) {
 session.rollback(); throw e;
}
// Änderungen festschreiben
session.commit();
```

Interfaces ObjectMap und JavaMap

Dieser Abschnitt beschreibt, wie Anwendungen unter Verwendung der Interfaces ObjectMap und JavaMap mit ObjectGrid interagieren. Diese beiden Interfaces werden für transaktionsorientierte Interaktionen zwischen Anwendungen und BackingMap-Objekten verwendet.

Interface ObjectMap

Eine ObjectMap-Instanz wird von einem Session-Objekt abgerufen, das dem aktuellen Thread entspricht. Das Interface ObjectMap ist das wichtigste Mittel, das Anwendungen zur Verfügung steht, um Änderungen an Einträgen in einer BackingMap-Instanz vorzunehmen.

Eine ObjectMap-Instanz anfordern

Eine Anwendung ruft mit der Methode `Session.getMap(String)` eine ObjectMap-Instanz von einem Session-Objekt ab. Das folgende Code-Snippet veranschaulicht, wie eine ObjectMap-Instanz angefordert wird:

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

Eine ObjectMap-Instanz entspricht einem bestimmten Session-Objekt. Wenn Sie die Methode getMap mehrfach für ein bestimmtes Session-Objekt aufrufen und jeweils denselben BackingMap-Namen verwenden, wird immer dieselbe ObjectMap-Instanz zurückgegeben.

Transaktionen automatisch festschreiben

Wie bereits erwähnt, sind Operationen für BackingMap-Objekte, die mit ObjectMap- und JavaMap-Instanzen ausgeführt werden, am effizientesten in einer Session-Transaktion. ObjectGrid unterstützt die automatische Festschreibung, wenn Methoden in den Interfaces ObjectMap und JavaMap außerhalb einer Session-Transaktion aufgerufen werden. Die Methoden starten eine implizite Transaktion, führen die angeforderte Operation durch und schreiben dann die implizite Transaktion fest.

Methodensemantik

Im Folgenden ist die Semantik der einzelnen Methoden in den Interfaces ObjectMap und JavaMap beschrieben. Die Methode setDefaultKeyword, die Methode invalidateUsingKeyword und die Methoden, die ein Argument Serializable haben, sind im Abschnitt „Schlüsselwörter" auf Seite 58 beschrieben. Eine Beschreibung der Methode setTimeToLive finden Sie im Abschnitt „Evictor" auf Seite 93. Nähere Informationen zu diesen Methoden finden Sie in der API-Dokumentation.

Methode containsKey
Bestimmt, ob ein Schlüssel einen Wert im BackingMap- oder Loader-Objekt hat. Wenn eine Anwendung Nullwerte unterstützt, kann mit dieser Methode bestimmt werden, ob eine Nullreferenz, die von einer get-Operation zurückgegeben wird, auf einen Nullwert verweist oder anzeigt, dass das BackingMap- bzw. Loader-Objekt den Schlüssel nicht enthält.

Methode flush
Die Semantik dieser Methode gleicht der Semantik der Methode flush im Interface Session. Die beiden Methoden unterscheiden sich insofern, dass die Methode flush des Interface Session die aktuell anstehenden Änderungen für alle modifizierten Maps in der aktuellen Sitzung anwendet. Mit dieser Methode hingegen werden nur die Änderungen in dieser ObjectMap-Instanz in den Loader geschrieben.

Methode get
Ruft den Eintrag aus der BackingMap-Instanz ab. Wenn der Eintrag nicht in der BackingMap-Instanz gefunden wird, der Backing-Map aber ein Loader zugeordnet ist, versucht die Methode, den Eintrag vom Loader abzurufen. Die Methode `getAll` wird für die Unterstützung der Stapelverarbeitung von get-Methoden bereitgestellt.

Methode getForUpdate
Entspricht im Wesentlichen der Methode `get`. Mit der Methode `getForUpdate` werden der BackingMap-Instanz und dem Loader jedoch mitgeteilt, dass beabsichtigt wird, den Eintrag zu aktualisieren. Ein Loader kann diesen Hinweis verwenden, um eine Abfrage des Typs `SELECT for UPDATE` an ein Datenbank-Backend abzusetzen. Wenn eine pessimistische Sperrstrategie für die BackingMap-Instanz definiert ist, sperrt der Sperrenmanager den Eintrag. Die Methode `getAllForUpdate` wird für die Unterstützung von Stapelabrufen bereitgestellt.

Methode insert
Fügt einen Eintrag in die BackingMap-Instanz und den Loader ein. Mit dieser Methode teilen Sie der BackingMap-Instanz und dem Loader mit, dass Sie einen noch nicht vorhandenen Eintrag einfügen möchten. Wenn Sie diese Methode für einen vorhandenen Eintrag aufrufen, wird beim Aufruf der Methode bzw. beim Festschreiben der aktuellen Transaktion eine Ausnahme ausgelöst.

Methode invalidate
Die Semantik dieser Methode richtet sich nach dem Wert des Parameters **isGlobal**, der an die Methode übergeben wird. Die Methode `invalidateAll` wird für die Unterstützung der Stapelverarbeitung von invalidate-Methoden bereitgestellt.

Es wird eine lokale Invalidierung durchgeführt, wenn der Parameter **isGlobal** mit dem Wert *false* an die Methode `invalidate` übergeben wird. Bei der lokalen Invalidierung werden alle Änderungen verworfen, die am Eintrag im Transaktions-Cache vorgenommen wurden. Wenn die Anwendung eine Methode `get` absetzt, wird der Eintrag dem zuletzt festgeschrie-

benen Wert in der BackingMap-Instanz entnommen. Sollte in der BackingMap-Instanz kein Eintrag vorhanden sein, wird der Eintrag dem Wert entnommen, der zuletzt in den Loader geschrieben wurde bzw. dort festgeschrieben wurde. Wenn eine Transaktion festgeschrieben wird, haben die Einträge, die als lokal invalidiert gekennzeichnet sind, keine Auswirkung auf die BackingMap-Instanz. Alle Änderungen, die in den Loader geschrieben wurden, werden auch dann festgeschrieben, wenn der Eintrag invalidiert wurde.

Es wird eine globale Invalidierung durchgeführt, wenn der Parameter **isGlobal** mit dem Wert *true* an die Methode `invalidate` übergeben wird. Bei der globalen Invalidierung werden alle anstehenden Änderungen für den Eintrag im Transaktions-Cache verworfen. Außerdem wird der BackingMap-Wert bei nachfolgenden Operationen, die für den Eintrag durchgeführt werden, umgangen. Wenn eine Transaktion festgeschrieben wird, werden alle Einträge die als global invalidiert gekennzeichnet sind, aus der BackingMap-Instanz gelöscht.

Stellen Sie sich beispielsweise den folgenden Anwendungsfall vor: Die BackingMap-Instanz wird von einer Datenbanktabelle gestützt, die eine Spalte für automatische Erhöhung enthält. Inkrementspalten sind hilfreich, um Datensätzen eindeutige Nummern zuzuordnen. Die Anwendung fügt einen Eintrag ein. Nach dem Einfügen des Eintrags muss die Anwendung die Folgenummer für die eingefügte Zeile wissen. Die Anwendung weiß, dass ihre Kopie des Objekts alt ist, und nutzt deshalb die globale Invalidierung, um den Wert vom Loader abzurufen. Der folgende Code veranschaulicht diesen Anwendungsfall:

```
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();
```

Dieses Codebeispiel fügt einen Eintrag für *Billy* hinzu. Das Versionsattribut von Person wird über eine Spalte für automatische Erhöhung in der Datenbank gesetzt. Die Anwendung setzt zuerst einen Befehl insert ab. Anschließend setzt sie einen Befehl flush ab, der bewirkt, dass der Befehl insert an den Loader und an die Datenbank gesendet wird. Die Datenbank setzt den Wert in der Versionsspalte auf die nächst höhere Zahl und kennzeichnet damit das Objekt Person in der Transaktion als veraltet. Zum Aktualisieren des Objekts führt die Anwendung eine globale Invalidierung durch. Die nächste Methode `get`, die abgesetzt wird, ruft den Eintrag vom Loader ab und ignoriert den Wert der Transaktion. Der Eintrag wird mit dem aktualisierten Versionswert aus der Datenbank abgerufen.

Methode put

Die Semantik der Methode put richtet sich danach, ob zuvor in der Transaktion eine Methode `get` für den Schlüssel aufgerufen wurde. Wenn die Anwendung eine Operation `get` ausführt, die einen vorhandenen Eintrag aus der BackingMap-Instanz oder dem Loader zurückgibt, wird der Aufruf der Methode `put` als Aktualisierung interpretiert, und es wird der vorherige Wert aus der Transaktion zurückgegeben. Eine Methode `put` ohne vorherige Methode `get` oder eine vorherige `get`, die keinen Eintrag gefunden hat, wird als Einfügeoperation (insert) eingestuft. Die Semantik der Methoden insert und update gilt, wenn die Operation put festgeschrieben wird.

Die Methode `putAll` wird für die Unterstützung der Stapelverarbeitung von insert- und update-Methoden bereitgestellt.

Methode remove
Entfernt den Eintrag aus BackingMap-Instanz und Loader, sofern ein solcher als Plug-in verfügbar ist. Falls der Eintrag zuvor in der Transaktion abgerufen wurde, gibt diese Methode den Wert zurück. Die Methode `removeAll` wird zur Unterstützung der Stapelverarbeitung von remove-Methoden ohne Rückgabewerte bereitgestellt.

Methode setCopyMode
Legt einen Kopiermodus für diese ObjectMap-Instanz fest. Mit dieser Methode kann eine Anwendung den Kopiermodus überschreiben, der in der BackingMap-Instanz angegeben ist. Der angegebene Kopiermodus bleibt so lang wirksam, bis die Methode `clearCopyMode` aufgerufen wird. Beide Methoden werden außerhalb von Transaktionen aufgerufen. Während der Ausführung einer Transaktion kann der Kopiermodus nicht geändert werden.

Methode touch
Aktualisiert die letzte Zugriffszeit für einen Eintrag. Diese Methode ruft keinen Wert von der BackingMap-Instanz ab. Verwenden Sie diese Methode in einer eigenen Transaktion. Wenn der angegebene Schlüssel nicht in der BackingMap-Instanz vorhanden ist, weil er invalidiert oder entfernt wurde, tritt während der COMMIT-Verarbeitung eine Ausnahme ein.

Methode update
Aktualisiert einen Eintrag in der BackingMap-Instanz und im Loader explizit. Mit dieser Methode teilen Sie der BackingMap-Instanz und dem Loader mit, dass ein vorhandener Eintrag aktualisiert werden soll. Es wird eine Ausnahme ausgelöst, wenn Sie diese Methode für einen Eintrag aufrufen, der beim Aufruf der Methode bzw. während der COMMIT-Verarbeitung nicht vorhanden ist.

Interface JavaMap

Eine JavaMap-Instanz wird von einem ObjectMap-Objekt abgerufen. Das Interface JavaMap hat dieselben Methodensignaturen wie ObjectMap, aber eine andere Ausnahmebehandlung. JavaMap erweitert das Interface `java.util.Map`, so dass alle Ausnahmen Instanzen der Klasse `java.lang.RuntimeException` sind. Da JavaMap das Interface `java.util.Map` erweitert, können Sie ObjectGrid problemlos für eine vorhandene Anwendung einsetzen, die das Interface `java.util.Map` für die Zwischenspeicherung von Objekten verwendet.

JavaMap anfordern

Eine Anwendung ruft mit der Methode `ObjectMap.getJavaMap` eine JavaMap-Instanz von einem ObjectMap-Objekt ab. Das folgende Code-Snippet veranschaulicht, wie eine JavaMap-Instanz angefordert wird.

```
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;
```

Eine JavaMap-Instanz wird von der ObjectMap-Instanz gestützt, bei dem sie angefordert wurde. Wenn Sie getJavaMap mehrfach für eine bestimmte ObjectMap-Instanz aufrufen, wird immer dieselbe JavaMap-Instanz zurückgegeben.

Unterstützte Methoden

Das Interface JavaMap unterstützt nur einen Teil der Methoden des Interface java.util.Map. Das Interface `java.util.Map` unterstützt die folgenden Methoden:

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

Alle anderen Methoden, die vom Interface `java.util.Map` übernommen werden, lösen eine Ausnahme des Typs `java.lang.UnsupportedOperationException` aus.

Schlüsselwörter

Das Interface ObjectGrid stellt einen flexiblen Invalidierungsmechanismus bereit, der auf Schlüsselwörtern basiert. Ein *Schlüsselwort* ist eine Instanz eines serialisierbaren Objekts, die nicht null ist. Sie können BackingMap-Einträgen beliebig Schlüsselwörter zuordnen.

Einträgen Schlüsselwörter zuordnen

Sie können Einträgen Schlüsselwörter zuordnen, müssen dies aber nicht. Die Methoden in den Interfaces ObjectMap und JavaMap, die Einträge bearbeiten, (z. B. `get`, `update`, `put`, `insert` und `touch`) haben alle Versionen, mit denen allen Einträgen, die von der Methode geändert werden, ein Schlüsselwort zugeordnet werden kann. Neue Schlüsselwortzuordnungen sind in der aktuellen Transaktion nur so lange sichtbar, bis die Transaktion festgeschrieben wird. Nach dem Festschreiben wird die neue Zuordnung auf die BackingMap-Instanz angewendet und ist danach für alle anderen Transaktionen sichtbar. Falls während der COMMIT-Verarbeitung ein Fehler auftritt, der dazu führt, dass die Transaktion rückgängig gemacht wird, oder falls ein Benutzer eine aktive Transaktion rückgängig macht, werden auch die neuen Schlüsselwortzuordnungen rückgängig gemacht. Der folgende Code zeigt, wie einem neuen Eintrag ein Schlüsselwort zugeordnet wird:

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

Dieser Beispielcode fügt der BackingMap-Instanz einen neuen Eintrag hinzu und ordnet diesem das Schlüsselwort `"New York"` zu. Eine Anwendung, die Einträge einfügt, muss auch beim Abrufen von Einträgen Schlüsselwörter zuordnen. Die Anwendung muss jedem abgerufenen Eintrag ein Schlüsselwort zuordnen. Dies gilt auch, wenn ein Eintrag mehrfach abgerufen wird. Schauen Sie sich das folgende Codebeispiel an:

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

Dieser Beispielcode stellt sicher, dass dem abgerufenen Eintrag das Schlüsselwort "New York" zugeordnet wird. Eine Anwendung kann einem Eintrag mehrere Schlüsselwörter zuordnen, aber es ist nur ein Schlüsselwort pro Methodenaufruf zulässig. Wenn Sie einem Eintrag mehrere Schlüsselwörter zuordnen möchten, müssen Sie einen anderen Methodenaufruf absetzen. Beispiel:

```
sess.begin();
Person p = (Person)map.get("Billy", "New York");
map.touch("Billy", "Anderes Schlüsselwort");
map.get("Billy", "Noch ein anderes Schlüsselwort");
sess.commit();
```

Standardschlüsselwörter

Die Methode `setDefaultKeyword` in den Interfaces ObjectMap und JavaMap ist eine Möglichkeit, Einträgen ein bestimmtes Schlüsselwort zuzuordnen, ohne die Schlüsselwortversion der Methoden `get`, `insert`, `put`, `update` oder `touch` zu verwenden. Bei der Verwendung einer Schlüsselwortversion dieser Methoden wird das Standardschlüsselwort ignoriert und das angegebene Schlüsselwortobjekt verwendet.

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

In diesem Beispiel wird `Billy` das Standardschlüsselwort "New York" zugeordnet. `Bob` wird kein Standardschlüsselwort zugeordnet, weil zum Abrufen des Eintrags Bob ein explizites Schlüsselwort an den get-Aufruf übergeben wurde. Dem Eintrag "Jimmy" werden keine Schlüsselwörter zugeordnet, weil das Standardschlüsselwort zurückgesetzt und kein explizites Schlüsselwortargument an den Aufruf der Methode `get` übergeben wurde.

Einträge mit Schlüsselwörtern ungültig machen

Mit der Methode `invalidateUsingKeyword` der Interfaces ObjectMap und JavaMap werden alle Einträge, denen ein Schlüsselwort in der zugehörigen BackingMap-Instanz zugeordnet ist, ungültig gemacht. Auf diese Weise können zusammengehörige Einträge in einer einzigen Operation ungültig gemacht werden.

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

In diesem Beispiel wird der Eintrag für "Billy" ungültig gemacht und nicht in die BackingMap-Instanz eingefügt. Der Eintrag für "Bob" wird nicht ungültig gemacht, da er nach dem Aufruf der Methode `invalidateUsingKeyword` eingefügt wurde. Die Methode `invalidateUsingKeyword` invalidiert die Einträge basierend auf den Schlüsselwortzuordnungen, die beim Aufruf der Methode gültig sind.

Gruppierung von Schlüsselwörtern

Schlüsselwörter können auch in einer hierarchischen Beziehung gruppiert werden. Ein übergeordnetes Schlüsselwort kann mehrere untergeordnete Schlüsselwörter haben, und ein untergeordnetes Schlüsselwort mehrere übergeordnete Schlüssel-

wörter. Wenn eine Anwendung beispielsweise die Schlüsselwörter ″Dublin″, ″Paris″, ″New York″ und ″Los Angeles″ verwendet, kann sie die folgenden Schlüsselwortgruppierungen hinzufügen:

- ″USA″ umfasst ″New York″ und ″Los Angeles″
- ″Europe″ umfasst ″Dublin″ und ″Paris″
- ″World″ umfasst ″USA″ und ″Europe″

Wenn das Schlüsselwort ″USA″ ungültig gemacht wird, werden alle Einträge ungültig gemacht, denen das Schlüsselwort ″New York″ oder ″Los Angeles″ zugeordnet sind. Wenn das Schlüsselwort ″World″ ungültig gemacht wird, werden alle Einträge ungültig gemacht, denen die Gruppierung ″USA″ oder ″Europe″ zugeordnet ist. Schlüsselwortzuordnungen werden mit der Methode `associateKeyword` des Interface ObjectGrid definiert. Werden einem übergeordneten Schlüsselwort nach dem Aufruf einer Methode `invalidateUsingKeyword` untergeordnete Schlüsselwörter hinzugefügt, werden die dem untergeordneten Schlüsselwort zugeordneten Einträge nicht ungültig gemacht. Der folgende Beispielcode definiert die zuvor beschriebenen Schlüsselwortzuordnungen:

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

LogElement- und LogSequence-Objekte

Wenn eine Anwendung während einer Transaktion Änderungen an einer Map-Instanz vornimmt, werden diese Änderungen in einem LogSequence-Objekt verfolgt. Ändert die Anwendung einen Map-Eintrag, gibt es dazu ein entsprechendes LogElement-Objekt, das die Einzelheiten zur vorgenommenen Änderung enthält. Die Loader erhalten ein LogSequence-Objekt für eine bestimmte Map, wenn eine Anwendung eine Flush- oder COMMIT-Methode für die Transaktion aufruft. Der Loader iteriert durch die LogElement-Objekte im LogSequence-Objekt und wendet jedes LogElement-Objekt auf das Backend an.

Auch die in einer ObjectGrid-Instanz registrierten ObjectGridEventListener verwenden LogSequence-Objekte. Diese Listener erhalten für jede Map in einer festgeschriebenen Transaktion ein LogSequence-Objekt. Anwendungen können diese Listener wie Auslöser in einer konventionellen Datenbank verwenden, um festzustellen, ob sich bestimmte Einträge ändern.

Dieser Abschnitt beschreibt vier protokollbezogene Interfaces bzw. Klassen, die vom ObjectGrid-Framework bereitgestellt werden:

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

LogElement

Ein LogElement-Objekt stellt eine Operation dar, die während einer Transaktion für einen Eintrag ausgeführt wird. Ein LogElement-Objekt hat die folgenden Attribute. Die am häufigsten verwendeten Attribute sind die beiden zuerst in der Liste aufgeführten: *type* und *current value*:

type Das Attribut *type* eines Protokollelements gibt den Typ der Operation an, die dieses Protokollelement darstellt. Gültige Werte für *type* sind die folgenden Konstanten, die im Interface `LogElement` definiert sind: `INSERT`, `UPDATE`, `DELETE`, `EVICT`, `FETCH` und `TOUCH`.

current value
current value gibt den neuen Wert für die Operation `INSERT`, `UPDATE` oder `FETCH` an. Bei den Operationen `TOUCH`, `DELETE` und `EVICT` ist "current value" null. Dieser Wert kann in ValueProxyInfo umgesetzt werden, wenn ein ValueInterface verwendet wird.

CacheEntry
Sie können eine Referenz auf das CacheEntry-Objekt aus dem LogElement-Objekt abrufen und die im CacheEntry-Objekt definierten Methoden verwenden, um die erforderlichen Informationen abzurufen.

pending state
Wenn *pending state* gleich `true` ist, wurde die von diesem Protokollelement dargestellte Änderung noch nicht auf den Loader angewendet. Hat das Attribut den Wert `false`, wurde die Änderung (wahrscheinlich mit der Operation flush) auf den Loader angewendet.

versioned value
versioned value gibt einen Wert an, der für die Versionssteuerung verwendet werden kann.

new keywords
Das Attribut *new keywords* enthält alle neuen Schlüsselwörter, die diesem Eintrag zugeordnet wurden.

last access time
Gibt die letzte Zugriffszeit für den Eintrag an.

LogSequence

In den meisten Transaktionen werden Operationen für mehrere Einträge in einer Map ausgeführt, so dass mehrere LogElement-Objekte erstellt werden. Vor diesem Hintergrund ist es sinnvoll, ein zusammengesetztes Objekt zu haben, das mehrere LogElement-Objekte umfasst. Das Interface LogSequence dient diesem Zweck und enthält eine Liste von LogElement-Objekten. Es stellt die folgenden Methoden bereit:

Methode `size`
Gibt die Anzahl der LogElement-Objekte im angegebenen LogSequence-Objekt zurück.

Methode `getAllChanges`
Gibt einen Iterator für alle Änderungen im angegebenen LogSequence-Objekt zurück.

Methode `getPendingChanges`
Gibt einen Iterator für alle anstehenden Änderungen zurück. Dieser Iterator wird in der Regel von einem Loader verwendet, um nur die anstehenden Änderungen im persistenten Speicher zu speichern.

Methode `getChangesByKeys`
Gibt basierend auf dem Eingabeparameter einen Iterator für die LogElement-Objekte zurück, die den Zielschlüssel enthalten.

Methode `getChangesByTypes`
Gibt einen Iterator für die LogElement-Objekte zurück, die den angegebenen LogElement-Typ haben.

Methode `getMapName`
Gibt den Namen der BackingMap-Instanz zurück, für die die Änderungen gelten. Der Aufrufende kann diesen Namen als Eingabe für die Methode `Session.getMap(string)` verwenden.

Methode `isDirty`
Gibt zurück, ob dieses LogSequence-Objekt LogElement-Objekte enthält, die eine Map-Instanz *verunreinigen* (dirty) würden. Wenn das LogSequence-Objekt LogElement-Objekte enthält, die einen anderen Typ als Fetch oder Get haben, wird das LogSequence-Objekt als "dirty" eingestuft.

LogElement- und LogSequence-Objekte werden in ObjectGrid und benutzerdefinierten ObjectGrid-Plug-ins häufig verwendet, wenn Operationen von einer Komponente bzw. einem Server an eine andere Komponente bzw. einen anderen Server weitergegeben werden. Beispielsweise kann ein LogSequence-Objekt von der ObjectGrid-Funktion für verteilte Transaktionsweitergabe verwendet werden, um die Änderungen an andere Server weiterzugeben, oder vom Loader auf den persistenten Speicher angewendet werden. LogSequence-Objekte werden hauptsächlich von den folgenden Interfaces verwendet:

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

Nähere Einzelheiten zu diesen Interfaces finden Sie in der API-Dokumentation.

Loader-Beispiel

Dieser Abschnitt beschreibt, wie die LogSequence- und LogElement-Objekte in einem Loader verwendet werden. Ein *Loader* wird verwendet, um Daten aus einem persistenten Speicher zu laden und in einem solchen zu speichern. Die Methode `batchUpdate` des Interface Loader verwendet LogSequence-Objekte. Beispiel:

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

Die Methode `batchUpdate` wird aufgerufen, wenn eine ObjectGrid-Instanz alle aktuellen Änderungen auf den Loader anwenden muss. Der Loader erhält, gekapselt in einem LogSequence-Objekt, eine Liste der LogElement-Objekte für die Map. Die Implementierung der Methode `batchUpdate` muss durch die Änderungen iterieren und sie auf das Backend anwenden. Das folgende Code-Snippet zeigt, wie ein Loader ein LogSequence-Objekt verwendet. Das Snippet iteriert durch die Änderungen und erstellt drei JDBC-batch-Anweisungen (Java Database Connectivity): eine mit Einfügeoperationen (insert), eine mit Aktualisierungsoperationen (update) und eine dritte mit Löschoperationen (delete):

```
public void batchUpdate(TxID tx, LogSequence sequence)
throws LoaderException
{
 // Zu verwendende SQL-Verbindung anfordern
 Connection conn = getConnection(tx);
 try
 {
  // Die Liste der Änderungen verarbeiten und vorbereitete Anweisungen für
  // die Ausführung von SQL-Aktualisierungs-, Einfüge- oder Löschoperationen
```

```
  // im Stapelbetrieb erstellen. Die Anweisungen werden in stmtCache
  // zwischengespeichert.
  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;
  }
 }
 // Die Stapelanweisungen ausführen, die mit der Schleife erstellt wurden
 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;
}
}
```

Das obige Beispiel veranschaulicht die allgemeine Verarbeitungslogik des LogSequence-Arguments. Die Erstellung einer SQL-Anweisung `insert`, `update` oder `delete` wird nicht im Detail gezeigt. Das Beispiel zeigt, dass die Methode `getPendingChanges` für ein LogSequence-Argument aufgerufen wird, um einen Iterator für die LogElement-Objekte anzufordern, die ein Loader verarbeiten muss. Mit der Methode `LogElement.getType().getCode()` kann festgestellt werden, ob ein LogElement-Objekt für eine SQL-Anweisung `insert`, `update` oder `delete` bestimmt ist.

Evictor-Beispiel

Dieses Beispiel veranschaulicht, wie LogSequence- und LogElement-Elemente in einem Evictor verwendet werden. Ein Evictor wird verwendet, um Map-Einträge auf der Basis bestimmter Kriterien aus der BackingMap zu löschen. Die Methode apply des Interface Evictor verwendet LogSequence-Objekte. Beispiel:

```
/**
* Diese Methode wird während der Cache-Festschreibung verwendet, damit das
* Bereinigungsprogramm (Evictor) die Verwendung des Objekts in einer BackingMap-Instanz
* verfolgen kann. Diese Methode meldet außerdem alle Einträge, die gelöscht
* bereinigt.
*
* @param sequence LogSequence-Objekt mit den an der Map vorgenommenen Änderungen
*/
void apply(LogSequence sequence);
```

Wie die Methode apply LogSequence-Objekte verwendet, können Sie dem Codebeispiel im Abschnitt „Evictor" auf Seite 93 entnehmen.

LogSequenceFilter und LogSequenceTransformer

Manchmal müssen die LogElement-Objekte gefiltert werden, damit nur LogElement-Objekte mit bestimmten Kriterien akzeptiert und andere Objekte zurückgewiesen werden. Wenn Sie ein LogElement-Objekt basierend auf einem bestimmten Kriterium serialisieren möchten, können Sie die Funktion LogSequenceFilter verwenden, die dieses Problem mit der folgenden Methode löst:

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

Diese Methode gibt true zurück, wenn das angegebene LogElement-Objekt in der Operation verwendet werden soll. Andernfalls gibt die Methode den Wert false zurück.

LogSequenceTransformer ist eine Klasse, die die oben beschriebene Funktion LogSequenceFilter verwendet. Diese Klasse verwendet LogSequenceFilter, um LogElement-Objekte zu filtern und die akzeptierten LogElement-Objekte anschließend zu serialisieren. Diese Klasse hat zwei Methoden. Dies ist die erste Methode:

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

Diese Methode erlaubt dem Aufrufenden, einen Filter zu definieren, um die LogElement-Objekte zu ermitteln, die in den Serialisierungsprozess aufgenommen werden. Mit dem Parameter **DistributionMode** kann der Aufrufende den Serialisierungsprozess steuern. Der Wert muss nicht serialisiert werden, wenn der Verteilungsmodus auf Invalidierung eingestellt ist. Die zweite Methode dieser Klasse ist folgendermaßen:

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

Diese Methode liest die serialisierte Protokollsequenz (LogSequence), die von der Methode `serialize` aus dem bereitgestellten Objekteingabestrom erstellt wurde.

Sperren

Dieser Abschnitt beschreibt die Sperrstrategie, die von einer ObjectGrid-BackingMap-Instanz unterstützt wird.

Für jede BackingMap-Instanz kann eine der folgenden Sperrstrategien konfiguriert werden:

- Pessimistisches Sperren
- Optimistisches Sperren
- Keine Sperrstrategie

Das folgende Beispiel zeigt, wie die Sperrstrategie für BackingMap-Instanzen definiert wird. In dem Beispiel wird für die BackingMap-Instanzen `map1`, `map2` und `map3` eine jeweils andere Sperrstrategie definiert:

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

```
 ObjectGridManagerFactory.getObjectGridManager().createObjectGrid("test");
BackingMap bm = og.defineMap("map1");
bm.setLockStrategy( LockStrategy.PESSIMISTIC );
bm = og.defineMap("map2");
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
bm = og.defineMap("map3");
bm.setLockStrategy( LockStrategy.NONE );
```

Damit keine Ausnahme des Typs `java.lang.IllegalStateException` ausgelöst wird, muss die Methode `setLockStrategy` vor dem Aufruf der Methode `initialize` oder `getSession` für die ObjectGrid-Instanz aufgerufen werden.

Bei der Verwendung der Sperrstrategien `PESSIMISTIC` und `OPTIMISTIC` wird ein Sperrenmanager für die BackingMap-Instanz erstellt. Der Sperrenmanager verwendet eine Hash-Tabelle, um die Einträge zu verfolgen, die von einer oder mehreren Transaktionen gesperrt werden. Wenn die Hash-Tabelle viele Einträge enthält, können Sie mit einer höheren Anzahl von Sperren-Buckets eine bessere Leistung erzielen. Das Risiko von Kollisionen bei der Java-Synchronisierung nimmt mit zunehmender Anzahl von Buckets ab. Außerdem sind bei einer höheren Anzahl von Sperren-Buckets mehr gemeinsame Zugriffe möglich. Das folgende Beispiel zeigt, wie eine Anwendung die Anzahl der Sperren-Buckets für eine bestimmte BackingMap-Instanz definieren kann:

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

Damit keine Ausnahme des Typs `java.lang.IllegalStateException` ausgelöst wird, muss die Methode `setNumberOfLockBuckets` vor dem Aufruf der Methode `initialize` oder `getSession` für die ObjectGrid-Instanz aufgerufen werden. Der Parameter für die Methode `setNumberOfLockBuckets` ist ein primitiver Java-Integer, der die Anzahl der zu verwendenden Sperren-Buckets angibt. Die Verwendung einer Primzahl gewährleistet eine gleichmäßige Verteilung der Map-Einträge auf die Sperren-Buckets. Im Hinblick auf die Leistung empfiehlt es sich, für die Anzahl der Sperren-Buckets zunächst ungefähr 10 Prozent der erwarteten Anzahl von BackingMap-Einträgen festzulegen.

Pessimistisches Sperren

Verwenden Sie die pessimistische Sperrstrategie für das Lesen und Schreiben von Maps, wenn die anderen Sperrstrategien nicht geeignet sind.

Wenn für eine ObjectGrid-Map-Instanz die Sperrstrategie PESSIMISTIC konfiguriert ist, wird eine pessimistische Transaktionssperre für einen BackingMap-Eintrag angefordert, wenn eine Transaktion den Eintrag aus der BackingMap-Instanz abruft. Die pessimistische Sperre bleibt so lange bestehen, bis die Anwendung die Transaktion abschließt. Die pessimistische Sperrstrategie wird in der Regeln in den folgenden Situationen verwendet:

- Die BackingMap-Instanz ist mit oder ohne Loader konfiguriert, und es sind keine Versionsinformationen verfügbar.
- Die BackingMap-Instanz wird direkt von einer Anwendung verwendet, die die Unterstützung von ObjectGrid für die Steuerung des gemeinsamen Zugriffs benötigt.
- Es sind Versionsinformationen verfügbar, aber bei den Aktualisierungstransaktionen für die BackingMap-Einträge kommt es häufig zu Kollisionen, die bei einer optimistischen Aktualisierung zu Fehlern führen.

Da die pessimistische Sperrstrategie die größten Auswirkungen auf Leistung und Skalierbarkeit hat, sollte diese Strategie nur dann für das Lesen und Schreiben von Maps verwendet werden, wenn die anderen Sperrstrategien nicht geeignet sind, d.

h. bei einer optimistischen Aktualisierung häufig Fehler auftreten und in solchen Fällen eine Wiederherstellung von der Anwendung nur schwer durchführbar ist.

ObjectMap-Methoden und Sperrmodi

Wenn eine Anwendung die Methoden des Interface ObjectMap verwendet, versucht ObjectGrid automatisch, eine pessimistische Sperre für den aufgerufenen BackingMap-Eintrag anzufordern. ObjectGrid verwendet basierend auf der Methode, die die Anwendung im Interface ObjectMap aufruft, einen der folgenden Sperrmodi:

- Die Methoden `get` und `getAll` fordern eine *S*-Sperre an, d. h. den gemeinsamen Sperrmodus für den Schlüssel eines BackingMap-Eintrags. Die S-Sperre bleibt so lange bestehen, bis die Transaktion abgeschlossen wird. Dieser Sperrmodus lässt gemeinsame Zugriffe von Transaktionen zu, die versuchen, eine S- oder U-Sperre (Upgrade-fähige Sperre) für den denselben Schlüssel anzufordern, blockieren aber andere Transaktionen, die versuchen, eine exklusive Sperre (X-Sperre) für denselben Schlüssel zu erhalten.
- Die Methoden `getForUpdate` und `getAllForUpdate` fordern eine *U-Sperre* für den Schlüssel eines BackingMap-Eintrags an. Die U-Sperre bleibt so lange bestehen, bis die Transaktion abgeschlossen wird. Der Sperrmodus U lässt gemeinsame Zugriffe von Transaktionen zu, die eine S-Sperre für denselben Schlüssel anfordern, blockiert aber andere Transaktionen, die versuchen, eine U- oder X-Sperre für denselben Schlüssel zu erhalten.
- Die Methoden `put`, `putAll`, `remove`, `removeAll`, `insert`, `update` und `touch` fordern eine *X-Sperre* für den Schlüssel eines BackingMap-Eintrags an. Die X-Sperre bleibt so lange bestehen, bis die Transaktion abgeschlossen wird. Der X-Sperrmodus stellt sicher, dass jeweils nur eine Transaktion einen BackingMap-Eintrag mit einem bestimmten Schlüsselwert einfügt, aktualisiert oder entfernt. Eine X-Sperre blockiert alle anderen Transaktionen, die versuchen, eine S-, U- oder X-Sperre für denselben Schlüssel zu erhalten.
- Die Methoden `global invalidate` und `global invalidateAll` fordern eine X-Sperre für jeden ungültig gemachten BackingMap-Eintrag an. Die X-Sperre bleibt so lange bestehen, bis die Transaktion abgeschlossen wird. Es werden keine Sperren für die Methoden `local invalidate` und `local invalidateAll` angefordert, da bei Aufrufen dieser Methoden keine BackingMap-Einträge ungültig gemacht werden.

Aus den beschriebenen Definitionen geht hervor, dass der Sperrmodus S schwächer ist als der Sperremodus U, da er beim Zugriff auf einen BackingMap-Eintrag eine höhere Anzahl gleichzeitig ausgeführter Transaktionen unterstützt. Der Sperrmodus U ist geringfügig stärker als der Sperrmodus S, da er andere Transaktionen blockiert, die eine U- oder X-Sperre anfordern. Der Sperrmodus S blockiert nur solche Transaktionen, die eine X-Sperre anfordern. Dieser kleine Unterschied ist wichtig, weil er das Auftreten von gegenseitigem Sperren (Deadlock) verhindern kann. Der Sperrmodus X ist der stärkste, da er alle anderen Transaktionen blockiert, die versuchen, eine S-, U- oder X-Sperre für denselben BackingMap-Eintrag zu erhalten. Dieser Modus stellt sicher, dass jeweils nur eine Transaktion einen BackingMap-Eintrag einfügt, aktualisiert oder entfernt, und verhindert, dass Aktualisierungen verloren gehen, wenn mehrere Transaktionen versuchen, denselben BackingMap-Eintrag zu aktualisieren.

Die folgende Tabelle ist eine Kompatibilitätsmatrix für die Sperrmodi, die die beschriebenen Modi noch einmal zusammenfasst und zeigt, welche Sperrmodi miteinander kompatibel sind. Die Zeilen der Matrix geben einen bereits erteilten Sperrmodus an. Die Spalten geben den Sperrmodus an, der von einer anderen

Transaktion angefordert wurde. Enthält eine Spalte den Wert **Ja**, wird der von der anderen Transaktion angeforderte Sperrmodus erteilt, weil er mit dem bereits erteilten Sperrmodus kompatibel ist. Ein **Nein** zeigt an, dass der Sperrmodus nicht kompatibel ist und die andere Transaktion warten muss, bis die erste Transaktion ihre Sperre wieder freigibt.

Tabelle 5. Kompatibilität und Stärke des Sperrmodus

Sperre	**Kompatible Sperren**			**Stärke**
	S (gemeinsamer Zugriff)	U (Upgrade-fähig)	X (exklusiv)	
S (gemeinsamer Zugriff)	J	J	N	schwach
U (Upgrade-fähig)	J	N	N	normal
X (exklusiv)	N	N	N	stark

Wartezeitlimit für Sperren

Jede ObjectGrid-BackingMap-Instanz hat ein Standardwartezeitlimit für Sperren. Das Zeitlimit stellt sicher, dass eine Anwendung nicht endlos auf die Erteilung einer Sperre wartet, weil aufgrund eines Anwendungsfehlers eine Situation mit gegenseitigem Sperren eingetreten ist. Die Anwendung kann mit dem Interface BackingMap das Standardwartezeitlimit für Sperren überschreiben. Das folgende Beispiel veranschaulicht, wie das Wartezeitlimit für Sperren für ″map1″ auf 60 gesetzt wird:

```
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 );
```

Damit keine Ausnahme des Typs `java.lang.IllegalStateException` ausgelöst wird, müssen die Methoden `setLockStrategy` und `setLockTimeout` vor dem Aufruf der Methode `initialize` oder `getSession` für die ObjectGrid-Instanz aufgerufen werden. Der Parameter für die Methode `setLockTimeout` ist ein primitiver Java-Integer, der angibt, wie lange (in Sekunden) ObjectGrid auf die Erteilung einer Sperre warten soll. Wenn das für die BackingMap-Instanz konfigurierte Wartezeitlimit bei einer Transaktion abläuft, löst ObjectGrid eine Ausnahme des Typs `com.ibm.websphere.objectgrid.LockTimeoutException` aus.

Wenn eine Ausnahme des Typs LockTimeoutException ausgelöst wird, muss die Anwendung feststellen, ob die Zeitlimitüberschreitung darauf zurückzuführen ist, dass die Anwendung langsamer als erwartet läuft oder ob eine Situation mit gegenseitigem Sperren vorliegt. Wenn eine Situation mit gegenseitigem Sperren eingetreten ist, kann die Ausnahme nicht durch Erhöhen des Wartezeitlimits für Sperren behoben werden. Das Erhöhen des Zeitlimits führt lediglich dazu, dass die Ausnahme später eintritt. Sollte sich das Problem jedoch durch das Erhöhen des Zeitlimits beheben lassen, ist das Problem darauf zurückzuführen, dass die Anwendung langsamer als erwartet läuft. In diesem Fall muss die Anwendung die Ursache für das Leistungsproblem feststellen.

Gegenseitiges Sperren

Schauen Sie sich die folgende Sequenz von Sperrmodusanforderungen an:

```
Transaktion 1 wird eine X-Sperre für Schlüssel1 erteilt.
Transaktion 2 wird eine X-Sperre für Schlüssel2 erteilt.
Transaktion 1 hat eine X-Sperre für Schlüssel2 angefordert.
(Transaktion 1 blockiert und muss auf die Sperre warten, die Transaktion 2 erteilt wurde.)
Transaktion 2 hat eine X-Sperre für Schlüssel1 angefordert.
(Transaktion 2 blockiert und muss auf die Sperre warten, die Transaktion 1 erteilt wurde.)
```

Diese Anforderungssequenz ist eine klassische Beispielsituation, in der sich zwei Transaktionen gegenseitig sperren, die mehrere Sperren in unterschiedlicher Reihenfolge anfordern. Um eine solche Situation zu verhindern, müssen die Transaktionen die Sperren in derselben Reihenfolge anfordern. Wenn eine optimistische Sperrstrategie verwendet wird und die Methode flush im Interface ObjectMap von der Anwendung nicht verwendet wird, fordert die Transaktion nur Sperrmodi nur im COMMIT-Zyklus an. Während des COMMIT-Zyklus bestimmt ObjectGrid die Schlüssel für die BackingMap-Einträge, die gesperrt werden müssen und fordert die Sperrmodi in der Reihenfolge der Schlüssel an. Mit diesem Verfahren verhindert ObjectGrid einen Großteil klassischer Situationen mit gegenseitigem Sperren. ObjectGrid kann jedoch nicht alle möglichen Situationen mit gegenseitigem Sperren verhindern. Es gibt diverse Szenarios, die die Anwendung berücksichtigen muss. Im Folgenden sind diese Szenarios und die entsprechenden Maßnahmen beschrieben.

Szenario 1: ObjectGrid kann eine Situation mit gegenseitigem Sperren erkennen, ohne auf eine Überschreitung des Wartezeitlimits für Sperren warten zu müssen. Wenn dieses Szenario eintritt, löst ObjectGrid eine Ausnahme des Typs `com.ibm.websphere.objectgrid.LockDeadlockException` aus. Schauen Sie sich das folgende Code-Snippet an:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Billy");
// Billy hatte Geburtstag, also machen wir ihn ein Jahr älter.
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
sess.commit();
```

Es ist unwahrscheinlich, dass mehrere Transaktionen versuchen, Billys Alter heraufzusetzen. Also stellen Sie vor, dass seine Frau ihn ein Jahr älter machen möchte und damit Billy selbst und seine Frau die Transaktion gleichzeitig ausführen. In dieser Situation erhalten beide Transaktionen beim Aufruf der Methode `person.get("Billy")` eine S-Sperre für den Eintrag **Billy** der BackingMap-Instanz PERSON. Mit dem Aufruf der Methode `person.put ("Billy", p)` versuchen beide Transaktionen, die S-Sperre in eine X-Sperre umzustufen. Jetzt werden beide Transaktionen blockiert und warten darauf, dass die jeweils andere ihre S-Sperre freigibt. Damit sperren sich die beiden Transaktionen gegenseitig. Ein solches gegenseitiges Sperren tritt auf, wenn mehrere Transaktionen versuchen, eine schwächere Sperre für denselben BackingMap-Eintrag in eine stärkere Sperre umzustufen. In diesem Szenario löst die ObjectGrid-Instanz eine Ausnahme des Typs `LockDeadlockException` anstelle einer Ausnahme des Typs `LockTimeoutException` aus.

Die Anwendung kann die Ausnahme des Typs `LockDeadlockException` für das vorherige Beispiel verhindern, indem Sie anstelle der pessimistischen Sperrstrategie eine optimistische Sperrstrategie verwendet. Die Verwendung einer optimistischen Sperrstrategie wird empfohlen, wenn die Map im Wesentlichen nur gelesen und

nur selten aktualisiert wird. Nähere Einzelheiten zur optimistischen Strategie finden Sie im Abschnitt „Optimistisches Sperren" auf Seite 71. Wenn Sie die pessimistische Sperrstrategie verwenden müssen, können Sie anstelle der get-Methode im vorherigen Beispiel die Methode `getForUpdate` verwenden. Damit fordert die erste Transaktion, die die Methode `getForUpdate` aufruft, eine U-Sperre anstelle einer S-Sperre an. Dieser Sperrmodus bewirkt, dass die zweite Transaktion blockiert wird, wenn sie die Methode `getForUpdate` aufruft, weil jeweils nur einer Transaktion eine U-Sperre erteilt wird. Da die zweite Transaktion blockiert wird, erhält sie keine Sperre für den BackingMap-Eintrag "Billy". Die erste Transaktion wird nicht blockiert, wenn sie versucht, mit dem Aufruf der put-Methode die U-Sperre in eine X-Sperre umzustufen. Diese Variante veranschaulicht, warum der U-Sperrmodus den Namen Upgrade-fähig hat. Nach Abschluss der ersten Transaktion wird die Blockierung der zweiten Transaktion aufgehoben und dieser Transaktion eine U-Sperre erteilt. Eine Anwendung kann Szenarios mit gegenseitigem Sperren durch Sperrenumstufung verhindern, indem Sie anstelle der Methode `get` die Methode `getForUpdate` aufruft, wenn eine pessimistische Sperrstrategie verwendet wird.

Wichtig: Diese Lösung verhindert jedoch nicht, dass Transaktionen mit Lesezugriff einen BackingMap-Eintrag lesen können. Transaktionen mit Lesezugriff rufen die Methode `get`, aber nicht die Methoden `put`, `insert`, `update` und `remove` auf. Die Anzahl gemeinsamer Zugriffe ist ebenso hoch wie bei der Verwendung der Methode `get`. Sie nimmt lediglich ab, wenn die Methode `getForUpdate` von mehreren Transaktionen für denselben BackingMap-Eintrag aufgerufen wird.

Wenn eine Transaktion die Methode `getForUpdate` für mehrere BackingMap-Einträge aufruft, müssen Sie darauf achten, dass die U-Sperren von jeder Transaktion in derselben Reihenfolge angefordert werden. Angenommen, die erste Transaktion ruft die Methode `getForUpdate` für den Schlüssel 1 und die Methode `getForUpdate` für den Schlüssel 2 auf. Eine andere gleichzeitig ausgeführte Transaktion ruft die Methode `getForUpdate` für dieselben Schlüssel auf, aber in umgekehrter Reihenfolge. Dies führt zu einer klassischen Situation mit gegenseitigem Sperren, weil unterschiedliche Transaktionen dieselben Sperren in unterschiedlicher Reihenfolge anfordern. Die Anwendung muss sicherstellen, dass jede Transaktion, die auf mehrere BackingMap-Einträge zugreift, die Schlüsselreihenfolge einhält, damit sich die Transaktionen nicht gegenseitig sperren. Da die U-Sperre beim Aufruf der Methode `getForUpdate` und nicht bei der Festschreibung (COMMIT) angefordert wird, kann ObjectGrid die Sperrenanforderungen nicht sortieren, wie es im COMMIT-Zyklus möglich ist. In diesem Fall muss die Anwendung die Sperrenreihenfolge steuern.

Wenn Sie die Methode `flush` des Interface ObjectMap vor einer COMMIT-Anforderung aufrufen, sind unter Umständen weitere Aspekte bei der Sperrenreihenfolge zu beachten. Die Methode flush wird in der Regel verwendet, um die Änderungen, die am BackingMap-Eintrag vorgenommen wurden, über das Loader-Plug-in auf das Backend zu schreiben. In diesem Fall verwendet das Backend einen eigenen Sperrenmanager für die Steuerung des gemeinsamen Zugriffs, so dass Wartezeiten für Sperren und Situationen mit gegenseitigem Sperren eher im Backend als im Sperrenmanager von ObjectGrid zu erwarten sind. Schauen Sie sich die folgende Transaktion an:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
 sess.begin();
 activeTran = true;
 Person p = (IPerson)person.get("Billy");
```

```
 p.setAge( p.getAge() + 1 );
 person.put( "Billy", p );
 person.flush();
 ...
 p = (IPerson)person.get("Tom");
 p.setAge( p.getAge() + 1 );
 sess.commit();
 activeTran = false;
}
finally
{
 if ( activeTran ) sess.rollback();
}
```

Angenommen, eine andere Transaktion aktualisiert die Person Tom, ruft die Methode `flush` auf und aktualisiert anschließend die Person Billy. In diesem Fall führt die Verzahnung der beiden Transaktionen zu einem gegenseitigen Sperren in der Datenbank:

```
Der Transaktion 1 wird beim Ausführen der Methode flush eine X-Sperre für "Billy" erteilt.
Der Transaktion wird beim Ausführen der Methode flush eine X-Sperre für "Tom" erteilt.
Transaktion 1 hat während der COMMIT-Verarbeitung eine X-Sperre für "Tom" angefordert.
(Transaktion 1 wird blockiert und muss auf die Sperre warten, die Transaktion 2 erteilt wurde.)
Transaktion 2 hat während der COMMIT-Verarbeitung eine X-Sperre für "Billy" angefordert.
(Transaktion 2 wird blockiert und muss auf die Sperre warten, die Transaktion 1 erteilt wurde.)
```

Dieses Beispiel veranschaulicht, dass die Verwendung der Methode `flush` eher zu einem gegenseitigen Sperren in der Datenbank als in ObjectGrid führen kann. Dieses Beispiel kann bei jeder Sperrstrategie auftreten. Die Anwendung ist für die Verhinderung solcher Situationen mit gegenseitigem Sperren verantwortlich, wenn die Methode `flush` verwendet wird und die BackingMap-Instanz ein Loader-Plug-in hat. Das vorherige Beispiel zeigt noch einen weiteren Grund, warum ObjectGrid ein Wartezeitlimit für Sperren unterstützt. Eine auf eine Datenbanksperre wartende Transaktion wartet möglicherweise, obwohl sie eine Sperre für einen ObjectGrid-BackingMap-Eintrag hat. Probleme auf Datenbankebene können zu übermäßigen Wartezeiten auf eine ObjectGrid-Sperre führen, woraufhin ObjectGrid eine Ausnahme des Typs `LockTimeoutException` auslöst.

Ausnahmebehandlung

Die Beispiele in diesem Abschnitt enthalten keine Ausnahmebehandlung. Um zu verhindern, dass Sperren übermäßig lange gehalten werden, wenn eine Ausnahme des Typs `LockTimeoutException` oder `LockDeadlockException` ausgelöst wird, muss eine Anwendung unerwartete Ausnahmen abfangen und die Methode `rollback` aufrufen, wenn unerwartete Ereignisse eintreten. Ändern Sie das vorherige Code-Snippet wie im folgenden Beispiel gezeigt:

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
 sess.begin();
 activeTran = true;
 Person p = (IPerson)person.get("Billy");
// Billy hatte Geburtstag, also machen wir ihn ein Jahr älter.
 p.setAge( p.getAge() + 1 );
 person.put( "Billy", p );
 sess.commit();
 activeTran = false;
}
```

```
finally
{
 if ( activeTran ) sess.rollback();
}
```

Der Block `finally` in diesem Code-Snippet stellt sicher, dass eine Transaktion rückgängig gemacht wird, wenn eine unerwartete Ausnahme eintritt. Er behandelt nicht nur die Ausnahme `LockDeadlockException`, sondern auch alle anderen unerwarteten Ausnahmen, die eintreten können. Der Block `finally` behandelt den Fall, bei dem während des Aufrufs der Methode COMMIT eine Ausnahme eintritt. Dieses Beispiel ist jedoch nicht die einzige Möglichkeit für die Behandlung unerwarteter Ausnahmen. Es kann Fälle geben, in denen eine Anwendung unerwartete Ausnahmen abfangen und eine ihrer eigenen Ausnahmen auslösen möchte. Sie können die gewünschten catch-Blöcke hinzufügen, aber die Anwendung muss sicherstellen, dass das Code-Snippet nicht beendet wird, ohne die Transaktion abzuschließen.

Optimistisches Sperren

Die optimistische Sperrstrategie geht davon aus, dass niemals zwei Transaktionen versuchen, denselben BackingMap-Eintrag zu aktualisieren, während sie gleichzeitig ausgeführt werden. Aufgrund dieser Annahme ist es nicht erforderlich, eine Sperre für die gesamte Dauer der Transaktion aufrecht zu erhalten, da es unwahrscheinlich ist, dass mehrere Transaktionen den BackingMap-Eintrag gleichzeitig aktualisieren.

Die optimistische Sperrstrategie wird normalerweise in den folgenden Fällen verwendet:

- Eine BackingMap-Instanz ist mit oder ohne Loader konfiguriert, und es sind Versionsinformationen verfügbar.
- Eine BackingMap-Instanz wird im Wesentlichen nur gelesen, d. h. die BackingMap-Einträge werden von Transaktionen hauptsächlich gelesen und nur selten eingefügt, aktualisiert oder entfernt.
- Es wird häufiger ein BackingMap-Eintrag eingefügt, aktualisiert oder entfernt als gelesen, aber es sind nur wenige Transaktionskollisionen für denselben BackingMap-Eintrag zu verzeichnen.

Wie bei der pessimistischen Sperrstrategie bestimmen die Methoden im Interface ObjectMap, wie ObjectGrid versucht, eine Sperre für den gewünschten BackingMap-Eintrag anzufordern. Es bestehen jedoch einige wichtige Unterschiede zwischen der pessimistischen und der optimistischen Strategie:

- Wie bei der pessimistischen Sperrstrategie wird beim Aufruf der Methoden `get` und `getAll` eine *S*-Sperre angefordert. Beim optimistischen Sperren bleibt die S-Sperre jedoch nicht bis zum Abschluss der Transaktion bestehen. Die S-Sperre wird hier freigegeben, bevor die Methode zur Anwendung zurückkehrt. Die Sperre wird angefordert, damit ObjectGrid sicherstellen kann, dass nur festgeschriebene Daten von anderen Transaktionen für die aktuelle Transaktion sichtbar sind. Nachdem ObjectGrid sichergestellt hat, dass die Daten festgeschrieben wurden, wird die S-Sperre freigegeben. In der COMMIT-Phase wird eine optimistische Versionsprüfung durchgeführt, um sicherzustellen, dass keine andere Transaktion den BackingMap-Eintrag geändert hat, nachdem die aktuelle Transaktion die S-Sperre freigegeben hat.
- Anders als bei der pessimistischen Sperrstrategie werden die Methoden `getForUpdate` und `getAllForUpdate` genauso wie die Methoden `get` und `getAll` behandelt, d. h. am Anfang der Methode wird eine S-Sperre angefordert, und die S-Sperre wird freigegeben, bevor die Transaktion zur Anwendung zurückkehrt.

- Alle anderen ObjectMap-Methoden werden exakt wie bei der pessimistischen Sperrstrategie behandelt. Beim Aufruf der COMMIT-Methode wird eine X-Sperre für jeden BackingMap-Eintrag angefordert, der eingefügt, aktualisiert, entfernt, berührt oder invalidiert wird. Die X-Sperre bleibt so lange bestehen, bis die COMMIT-Verarbeitung für die Transaktion abgeschlossen ist.

Diese Sperrstrategie wird als optimistisch bezeichnet, weil sie auf optimistischen Annahmen beruht. Die optimistische Sperrstrategie geht davon aus, dass niemals zwei Transaktionen versuchen, denselben BackingMap-Eintrag zu aktualisieren, während sie gleichzeitig ausgeführt werden. Aufgrund dieser Annahme ist es nicht erforderlich, eine Sperre für die gesamte Dauer der Transaktion aufrecht zu erhalten, da es unwahrscheinlich ist, dass mehrere Transaktionen den BackingMap-Eintrag gleichzeitig aktualisieren. Weil jedoch keine Sperre gehalten wird, könnte eine andere, gleichzeitig ausgeführte Transaktion den BackingMap-Eintrag aktualisieren, nachdem die aktuelle Transaktion ihre S-Sperre freigibt. Für die Behandlung dieses Falls ruft ObjectGrid zum Zeitpunkt der COMMIT-Operation eine X-Sperre ab und führt eine optimistische Versionsprüfung durch, um sicherzustellen, dass der BackingMap-Eintrag nicht von einer anderen geändert wurde, nachdem die aktuelle Transaktion den Eintrag aus der BackingMap-Instanz gelesen hat. Wenn eine andere Transaktion den Eintrag geändert hat, schlägt die Versionsprüfung fehl, und es wird eine Ausnahme des Typs `OptimisticCollistionException` ausgelöst. Diese Ausnahme bewirkt, dass die aktuelle Transaktion rückgängig gemacht wird und die vollständige Transaktion von der Anwendung wiederholt werden muss. Die optimistische Sperrstrategie ist sehr hilfreich, wenn eine BackingMap-Instanz in der Hauptsache nur gelesen wird und es relativ unwahrscheinlich ist, dass gleichzeitige Aktualisierungen für denselben BackingMap-Eintrag vorgenommen werden.

Strategie ohne Sperren

Wenn für eine BackingMap-Instanz das Attribut LockStrategy mit NONE konfiguriert ist, werden keine Transaktionssperren für einen BackingMap-Eintrag angefordert. Sinnvoll ist diese Strategie in einem Szenario, in dem eine Anwendung ein Persistenzmanager ist, z. B. ein J2EE-EJB-Container, oder persistente Daten mit der Hibernate-Funktion abruft. In diesem Szenario ist die BackingMap-Instanz ohne Loader konfiguriert und wird vom Persistenzmanager als Daten-Cache verwendet. Der Persistenzmanager übernimmt in diesem Szenario die Steuerung des gemeinsamen Zugriffs zwischen Transaktionen, die auf dieselben ObjectGrid-BackingMap-Einträge zugreifen. Die ObjectGrid-Instanz muss für die Steuerung des gemeinsamen Zugriffs keine Transaktionssperren anfordern. Dies setzt voraus, dass der Persistenzmanager seine Transaktionssperren freigibt, bevor er die ObjectGrid-Map mit den festgeschriebenen Änderungen aktualisiert. Sollte dies nicht der Fall sein, muss eine pessimistische oder optimistische Sperrstrategie verwendet werden. Angenommen, der Persistenzmanager eines EJB-Containers aktualisiert die ObjectGrid-Map mit Daten, die in der vom EJB-Container verwalteten Transaktion festgeschrieben wurden. Wenn die Aktualisierung der ObjectGrid-Map vor der Freigabe der Transaktionssperren durch den Persistenzmanager erfolgt, kann die Strategie ohne Sperren (NONE) verwendet werden. Falls die Aktualisierung der ObjectGrid-Map nach der Freigabe der Transaktionssperren durch den Persistenzmanager erfolgt, muss die optimistische (OPTIMISTIC) oder pessimistische (PESSIMISTIC) Sperrstrategie verwendet werden.

Die Strategie ohne Sperren kann auch verwendet werden, wenn die Anwendung eine BackingMap-Instanz direkt verwendet und für die BackingMap-Instanz ein Loader konfiguriert ist. In diesem Szenario verwendet der Loader die Unterstützung für die Steuerung des gemeinsamen Zugriffs, die ein Verwaltungssystem für relationale Datenbanken (RDBMS, Relational Database Management System) bereit-

stellt. Für den Zugriff auf die Daten in einer relationalen Datenbank verwendet der Loader JDBC (Java Database Connectivity) oder die Hibernate-Funktion. Die Loader-Implementierung kann einen optimistischen oder pessimistischen Ansatz verwenden.

Mit einem Loader, der eine optimistische Sperrstrategie oder Versionssteuerungsmethode verwendet, kann ein höheres Maß an gemeinsamen Zugriffen und Leistung erzielt werden. Ein Loader verwendet eines der folgenden Verfahren, um ein optimistisches Sperrkonzept zu implementieren:

- Der Loader verwendet Versionsinformationen, die von dem Backend bereitgestellt werden, mit dem der Loader verbunden ist. Diese Versionsinformationen können in einem verdeckten Feld enthalten sein, das vom Backend beispielsweise über eine spezielle SQL-Anweisungssyntax verfügbar gemacht wird.
- Der Loader verwendet JDBC, um eine SQL-Operation durchzuführen, und der erste Ansatz ist nicht praktikabel. In diesem Fall wird mit der Methode `LogElement.Type.getCode` angegeben, ob die SQL-Operation eine Aktualisierungs- (update), Einfüge- (insert) oder Löschoperation (delete) ist. Wenn die Methode `getCode` den Wert `LogElement.CODE_UPDATE` zurückgibt, verwendet der Loader die Methode `getVersionedValue` im Interface LogElement, um das zu verwendende Versionsobjekt in der where-Klausel einer überqualifizierten SQL-Anweisung update abzurufen. Die überqualifizierte `where`-Klausel wird verwendet, um sicherzustellen, dass die Zeile nicht von einer anderen Transaktion geändert wurde. Der Loader verwendet die Methode `updateVersionedObjectForValue` im Interface `OptimisticCallback`, um ein neues Versionsobjekt zu generieren. Mit der neuen Version wird die Versionsspalte der aktualisierten Zeile aktualisiert. Anschließend verwendet der Loader die Methode `setVersionedValue` im Interface `LogElement`, um das LogElement-Objekt mit demselben Versionswert zu aktualisieren, mit dem auch die Zeile aktualisiert wurde. Der Abschnitt „Loader" auf Seite 102 beschreibt anhand eines Beispiels, wie ein Loader die Interfaces LogElement und OptimisticCallback zum Implementieren dieses optimistischen Konzepts verwendet.
- Wenn für eine BackingMap-Instanz ein Value-Interface konfiguriert ist, kann der Loader mit der Methode `getCurrentValue` des Interface LogElement und des Interface ValueProxyInfo die Attribute abrufen, die von der Transaktion geändert wurden. Dieses Konzept erlaubt dem Loader die Verwendung einer überqualifizierten `where`-Klausel, die jedes der geänderten Attribute enthält. Der Abschnitt „Loader" auf Seite 102 beschreibt anhand eines Beispiels, wie ein Loader die Interfaces LogElement und ValueProxyInfo zum Implementieren eines optimistischen Konzepts verwendet.

Ein Loader, der die Unterstützung für pessimistische Sperrstrategien des zugrunde liegenden Backend verwendet, kann den Parameter **forUpdate** verwenden, der in der Methode get des Interface Loader übergeben wird. Dieser Parameter hat den Wert *true*, wenn die Anwendung zum Abrufen der Daten die Methode `getForUpdate` des Interface ObjectMap verwendet hat. Der Loader kann mit diesem Parameter feststellen, ob eine Upgrade-fähige Sperre für die zu lesende Zeile angefordert werden soll. DB2 ruft beispielsweise eine Upgrade-fähige Sperre ab, wenn eine SQL-Anweisung select eine Klausel `for update` enthält. Dieses Konzept bietet denselben Schutz vor Situationen mit gegenseitigem Sperren, wie im Abschnitt ″Pessimistisches Sperren″ beschrieben wird.

ObjectGrid-Sicherheit

Die Sicherheitsmechanismen von ObjectGrid unterstützen den Schutz von Map-Daten über Programmierung und über Konfiguration.

Übersicht

ObjectGrid stellt Sicherheitsmechanismen bereit, mit denen Sie Map-Daten schützen können. Die ObjectGrid-Sicherheit ist in den JAAS-Mechanismus (Java Authentication and Authorization Services) integriert, der ein integraler Bestandteil der Java-2-Sicherheit ist. Dieser Abschnitt beschreibt die Mechanismen, die ObjectGrid verwendet, um den Zugriff auf Map-Daten zu autorisieren. Außerdem wird die Integration mit der Sicherheit von WebSphere Application Server beschrieben und veranschaulicht. Die Beispiel-Enterprise-Anwendung ObjectGridSample in der Datei `ObjectGridSample.ear` veranschaulicht, wie die Sicherheit konfiguriert und verwendet werden kann. Sie können die ObjectGrid-Sicherheit mit den folgenden beiden Methoden aktivieren:

- **Konfiguration:** Mit einer XML-Datei können Sie eine ObjectGrid-Instanz definieren und die Sicherheit für diese Instanz aktivieren. Im Folgenden sehen Sie die Datei `secure-objectgrid-definition.xml`, die in der Beispiel-Enterprise-Anwendung ObjectGridSample verwendet wird. In dieser XML-Datei wird die Sicherheit mit der folgenden Einstellung aktiviert: `securityEnabled="true"`.

```
<objectGrids>
 <objectGrid  name="secureClusterObjectGrid"
        securityEnabled="true"
       authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
  <bean  id="TransactionCallback"
     classname="com.ibm.websphere.samples.objecgrid.HeapTransactionCallback"/>
....
</objectGrids>
```

- **Programmierung:** Wenn Sie eine ObjectGrid-Instanz mit der API erstellen möchten, rufen Sie die folgende Methode im Interface ObjectGrid auf, um die Sicherheit zu aktivieren:

```
/**
* ObjectGrid-Sicherheit aktivieren
*/
void setSecurityEnabled();
```

Nähere Informationen zur ObjectGrid-Sicherheit finden Sie in den folgenden Abschnitten:

- Authentifizierung
- ObjectGrid-Autorisierung
- Sicherheit mit WebSphere Extended Deployment

Authentifizierung

ObjectGrid stützt sich bei der Authentifizierung auf die Umgebung, d. h. die Anwendungsserver oder die Anwendungen.

ObjectGrid selbst stellt kein Authentifizierungsverfahren bereit. Wenn Sie eine ObjectGrid-Instanz in einer Umgebung mit WebSphere Application Server oder WebSphere Extended Deployment verwenden, können die Anwendungen die Sicherheitsauthentifizierungsmechanismen von WebSphere Application Server verwenden. Wird eine ObjectGrid-Instanz in einer J2SE-Umgebung (Java 2 Platform, Standard Edition) ausgeführt, muss die Anwendung die Authentifizierungen mit der JAAS-Authentifizierung oder anderen Authentifizierungsverfahren selbst verwalten. Nähere Informationen zur Verwendung der JAAS-Authentifizierung finden Sie im Referenzhandbuch zu JAAS.

Der Vertrag zwischen einer Anwendung und einer ObjectGrid-Instanz wird mit dem Objekt `javax.security.auth.Subject` getroffen. Nachdem der Client vom Anwendungsserver oder der Anwendung authentifiziert wurde, kann die Anwen-

dung das authentifizierte Objekt `javax.security.auth.Subject` abrufen und dieses Subject-Objekt verwenden, um mit dem Aufruf der Methode ObjectGrid.getSession(Subject) eine Sitzung von der ObjectGrid-Instanz anzurufen. Mit diesem Subject-Objekt wird der Zugriff auf die Map-Daten autorisiert. Dieser Vertrag wird als *Mechanismus mit Übergabe des Subjekts* bezeichnet. Schauen Sie sich die API `ObjectGrid.getSession(Subject)` an:

```
/**
* Ermöglicht dem Cache, ein spezielles Subjekt anstelle des in der
* ObjectGrid-Instanz konfigurierten Subjekts für den Abruf einer
* Sitzung zu verwenden.
* @param Subjekt
* @return Eine Session-Instanz
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws InvalidSubjectException, wenn das übergebene Subjekt gemäß
* SubjectValidation-Mechanismus ungültig ist.
*/
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;
```

Die Methode getSession im Interface ObjectGrid kann ebenfalls zum Abruf eines Session-Objekts verwendet werden:

```
/**
* Gibt ein Session-Objekt zurück, das jeweils von einem Thread verwendet werden kann.
* Dieses Session-Objekt kann nicht von mehreren Threads gemeinsam genutzt werden,
* ohne es in einem kritischen Programmabschnitt einzubetten. Obwohl das zentrale
* Framework das Verschieben des Objekts zwischen Threads unterstützt, können
* TransactionCallback und Loader diese Nutzung, insbesondere in J2EE-Umgebungen,
* verhindern. Wenn die Sicherheit aktiviert ist, wird SubjectSource zum
* Abrufen eines Subject-Objekts verwendet.
*
* Wenn die Methode initialize() vor dem ersten getSession-Aufruf nicht
* aufgerufen wurde, findet eine implizite Initialisierung statt. Auf
* diese Weise wird sichergestellt, dass die gesamte Konfiguration
* vor der Verwendung in der Laufzeitumgebung abgeschlossen ist.
*
* @see #initialize()
* @return Eine Session-Instanz
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws IllegalStateException, falls diese Methode nach dem
* Aufruf der Methode destroy() aufgerufen wird.
*/
public Session getSession()
throws ObjectGridException, TransactionCallbackException;
```

Wie in der API-Dokumentation spezifiziert verwendet diese Methode bei aktivierter Sicherheit das Plug-in SubjectSource, um ein Subject-Objekt abzurufen. Das Plug-in SubjectSource ist eines der Sicherheits-Plug-ins, die in ObjectGrid für die Unterstützung der Authentifizierung definiert sind.

Sicherheitsbezogene Plug-ins

ObjectGrid stellt zwei Sicherheits-Plug-ins bereit, die mit dem Mechanismus mit Übergabe des Subjekts in Zusammenhang stehen: SubjectSource und SubjectValidation.

Plug-in SubjectSource

Das Plug-in SubjectSource wird vom Interface com.ibm.websphere.objectgrid.security.plugins.SubjectSource dargestellt und für den Abruf eines Subject-Objekts von einer aktiven ObjectGrid-Umgebung verwendet. Diese

ObjectGrid-Umgebung kann eine Anwendung sein, die ObjectGrid verwendet, oder ein Anwendungsserver, in dem die Anwendung implementiert ist.

Das Plug-in SubjectSource ist eine Alternative zum Mechanismus mit Übergabe des Subjekts. Unter Verwendung des Mechanismus mit Übergabe des Subjekts ruft die Anwendung das Subject-Objekt ab und verwendet dieses für den Abruf des ObjectGrid-Session-Objekts. Die ObjectGrid-Laufzeitumgebung ruft mit dem Plug-in SubjectSource das Subject-Objekt ab und verwendet dieses für den Abruf des Session-Objekts. Der Mechanismus mit Übergabe des Objekts übergibt die Steuerung der Subject-Objekte den Anwendungen, wohingegen der Mechanismus des Plug-in SubjectSource den Anwendungen den Abruf des Subject-Objekts abnimmt.

Das Plug-in SubjectSource kann zum Abrufen eines Subject-Objekts für einen ObjectGrid-Client und anschließend für die ObjectGrid-Autorisierung verwendet werden. Wenn die Methode ObjectGrid.getSession() aufgerufen wird, wird die Methode Subject getSubject() throws ObjectGridSecurityException () von der ObjectGrid-Laufzeitumgebung aufgerufen, wenn die Sicherheit aktiviert ist.

ObjectGrid stellt eine Standardimplementierung des Plug-in SubjectSource bereit: `com.ibm.websphere.objectgrid.security.plugins.builtins. WSSubjectSourceImpl`. Mit dieser Implementierung kann ein aufrufendes Subjekt (Caller-Subjekt) oder ein RunAs-Subjekt aus dem Thread abgerufen werden, wenn eine Anwendung in WebSphere Application Server ausgeführt wird. Wenn Sie ObjectGrid in WebSphere Application Server einsetzen, können Sie diese Klasse als SubjectSource-Implementierungsklasse konfigurieren. Das folgende Code-Snippet zeigt den Hauptverarbeitungsablauf der Methode WSSubjectSourceImpl.getSubject():

```
Subject s = null;
try {
 if (finalType == RUN_AS_SUBJECT) {
  // RunAs-Subjekt abrufen
  s = com.ibm.websphere.security.auth.WSSubject.getRunAsSubject();
 }
 else if (finalType == CALLER_SUBJECT) {
  // Caller-Subjekt abrufen
  s = com.ibm.websphere.security.auth.WSSubject.getCallerSubject();
 }
}
catch (WSSecurityException wse) {
 throw new ObjectGridSecurityException(wse);
}
return s;
```

Nähere Einzelheiten finden Sie in der API-Dokumentation zum Plug-in SubjectSource und zu WSSubjectSourceImpl.

Plug-in SubjectValidation

Das Plug-in SubjectValidation, dargestellt vom Interface `com.ibm.websphere.objectgrid.security.plugins.SubjectValidation`, ist ein weiteres Sicherheits-Plug-in. Mit dem Plug-in SubjectValidation kann geprüft werden, ob ein Objekt `javax.security.auth.Subject`, das an die ObjectGrid-Instanz übergeben oder vom Plug-in SubjectSource abgerufen wird, ein gültiges Subjekt ist, das nicht modifiziert wurde.

Die Methode Subject validateSubject(Subject subject) throws InvalidSubjectException; im Interface SubjectValidation akzeptiert ein Subject-Objekt und gibt ein Subject-Objekt zurück. Ob ein Subject-Objekt gültig ist und ein whatSubject-Objekt zurückgegeben wird, richtet sich nach Ihren Implemen-

tierungen. Falls das Subject-Objekt nicht gültig ist, sollte eine Ausnahme des Typs `InvalidSubjectException` ausgelöst werden.

Sie können dieses Plug-in verwenden, wenn Sie dem an die Methode übergebenen Subject-Objekt nicht vertrauen. In der Regel werden die Anwendungsentwickler, die den Code zum Abrufen des Subject-Objekts implementieren, jedoch als vertrauenswürdig eingestuft.

Eine Implementierung dieses Plug-in benötigt Unterstützung vom Ersteller des Subject-Objekts, da nur der Ersteller weiß, ob das Subject-Objekt manipuliert wurde. Sollte der Subject-Ersteller nicht wissen, ob das Subject-Objekt manipuliert wurde, ist dieses Plug-in nicht hilfreich.

ObjectGrid stellt eine Standardimplementierung des Plug-in SubjectValidation bereit: die Klasse `com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl`. Mit dieser Klasse kann die Gültigkeit des authentifizierten Subjekts von WebSphere Application Server geprüft werden. Sie können diese Klasse als SubjectValidation-Implementierungsklasse konfigurieren, wenn Sie ObjectGrid in WebSphere Application Server einsetzen. Die WSSubjectValidationImpl-Implementierung stuft ein Subject-Objekt als gültig ein, wenn das Token mit dem Berechtigungsnachweis, das diesem Subject-Objekt zugeordnet ist, nicht modifiziert wurde. Anders ausgedrückt, Benutzer könnten andere Teile des Subject-Objekts ändern. Die WSSubjectValidationImpl-Implementierung fordert von WebSphere Application Server das ursprüngliche Subject-Objekt an, das dem Token mit dem Berechtigungsnachweis entspricht, und gibt das ursprüngliche Subject-Objekt als geprüftes Subject-Objekt zurück. Deshalb haben Änderungen, die nicht am Token mit dem Berechtigungsnachweis, sondern an anderen Teilen des Subject-Inhalts vorgenommen werden, keine Auswirkung. Das folgende Code-Snippet veranschaulicht den Hauptverarbeitungsablauf der Methode `WSSubjectValidationImpl.validateSubject(Subject)`:

```
// Anmeldekontext mit Schema WSLogin erstellen
// und einen Callback-Handler übergeben.
LoginContext lc = new LoginContext("WSLogin",
new WSCredTokenCallbackHandlerImpl(subject));
// Beim Aufruf dieser Methode werden die Methoden des Callback-Handler
// aufgerufen, um den Benutzer anzumelden.
lc.login();
// Subjekt aus dem Anmeldekontext abrufen
return lc.getSubject();
```

In dem Code-Snippet wird das Callback-Handler-Objekt für das Token mit dem Berechtigungsnachweis, das Objekt WSCredTOkenCallbackHandlerImpl, mit dem zu prüfenden Subject-Objekt erstellt. Anschließend wird ein LoginContext-Objekt mit dem Anmeldeschema "WSLogin" erstellt. Beim Aufruf der Methode lc.login() ruft die Sicherheit von WebSphere Application Server das Token mit dem Berechtigungsnachweis vom Subject-Objekt ab und gibt dann das entsprechende Subject-Objekt als geprüftes Subject-Objekt zurück. Nähere Einzelheiten finden Sie in der API-Dokumentation zu SubjectValidation und WSSubjectValidationImpl.

Plug-in-Konfiguration

Sie können die Plug-ins SubjectValidation und SubjectSource auf zwei Arten konfigurieren:

- **XML-Konfiguration:** Mit einer XML-Datei können Sie eine ObjectGrid-Instanz definieren und diese beiden Plug-ins konfigurieren. Das folgende

Beispiel zeigt, wie die Klasse WSSubjectSourceImpl als SubjectSource-Plug-in und die Klasse WSSubjectValidation als SubjectValidation-Plug-in konfiguriert wird:

```
<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>
```

- **Über das Programm:** Wenn Sie eine ObjectGrid-Instanz mit den APIs erstellen möchten, können Sie die folgenden Methoden verwenden, um die Plug-ins SubjectSource und SubjectValidation zu konfigurieren:

```
/**
* SubjectValidation-Plug-in für diese ObjectGrid-Instanz konfigurieren. Mit
* dem SubjectValidation-Plug-in kann geprüft werden, ob das übergebene
* Subject-Objekt ein gültiges Subject-Objekt ist. Nähere Einzelheiten
* finden Sie unter {@link SubjectValidation}.
* @param subjectValidation SubjectValidation-Plug-in
*/
void setSubjectValidation(SubjectValidation subjectValidation);
/**
* SubjectSource-Plug-in konfigurieren. Mit einem SubjectSource-Plug-in
* kann ein Subject-Objekt für die Darstellung des ObjectGrid-Clients
* von der Umgebung abgerufen werden.
*
* @param source SubjectSource-Plug-in
*/
void setSubjectSource(SubjectSource source);
```

Eigenen JAAS-Authentifizierungscode schreiben

Für die Authentifizierung können Sie Ihren eigenen JAAS-Authentifizierungscode schreiben. Sie müssen eigene Anmeldemodule schreiben und diese Anmeldemodule anschließend für Ihr Authentifizierungsmodul konfigurieren.

Das Anmeldemodul empfängt Informationen zu einem Benutzer und authentifiziert den Benutzer. Bei diesen Informationen kann es sich um alles handeln, was den Benutzer identifiziert. Dies können beispielsweise eine Benutzer-ID und ein Kennwort, ein Clientzertifikat oder Ähnliches sein. Nach dem Empfang der Informationen prüft das Anmeldemodul, ob diese ein gültiges Subjekt darstellen, und erstellt anschließend ein Subject-Objekt. Derzeit sind mehrere Implementierungen von Anmeldemodulen verfügbar.

Nachdem Sie ein Anmeldemodul geschrieben haben, müssen Sie dieses Anmeldemodul konfigurieren, damit es von der Laufzeitumgebung verwendet werden kann. Es muss eine Konfigurationsdatei für JAAS-Anmeldemodule konfiguriert werden, die das Anmeldemodul und das zugehörige Authentifizierungsschema enthält. Beispiel:

```
FileLogin
{
com.acme.auth.FileLoginModule required debug=true
};
```

Das Authentifizierungsschema ist FileLogin und das Anmeldemodul `com.acme.auth.FileLoginModule`. Das erforderliche Token zeigt an, dass das Modul FileLoginModule diese Anmeldung validieren muss, oder das Schema schlägt fehl. Sie können die Konfigurationsdatei für JAAS-Anmeldemodule auf die folgenden Arten definieren:

- Den Dateinamen mit der Einstellung `login.config.url` in der Datei java.security festlegen. Beispiel:

  ```
  login.config.url.1=file:${java.home}/lib/security/file.login
  ```

- Den Dateinamen in der Befehlszeile mit den JVM-Argumenten `Djava.security.auth.login.config` festlegen. Beispiel:

  ```
  Djava.security.auth.login.config ==$JAVA_HOME/jre/lib/security/file.login
  ```

Nähere Einzelheiten zum Schreiben und Konfigurieren von Anmeldemodulen finden Sie im Lernprogramm zur JAAS-Authentifizierung.

ObjectGrid-Autorisierung

Die ObjectGrid-Autorisierung basiert auf dem Objekt Subject. ObjectGrid unterstützt zwei Arten von Autorisierungsmechanismen: die Autorisierung mit Java Authentication and Authorization Service (JAAS) und die benutzerdefinierte Autorisierung.

Berechtigungsklasse MapPermission

In ObjectGrid wird eine allgemein zugängliche (public) Klasse `com.ibm.websphere.objectgrid.security.MapPermission` bereitgestellt, mit der Berechtigungen für die ObjectGrid-Ressourcen, insbesondere die Methoden der Interfaces ObjectMap und JavaMap dargestellt werden. ObjectGrid definiert die folgenden Berechtigungszeichenfolgen für den Zugriff auf die Methoden der Interfaces ObjectMap und JavaMap:

- **read**: Erteilt die Berechtigung zum Lesen der Map-Daten. Die ganzzahlige Konstante wird mit `MapPermission.READ` definiert.
- **write**: Erteilt die Berechtigung zum Aktualisieren der Map-Daten. Die ganzzahlige Konstante wird mit `MapPermission.WRITE` definiert.
- **insert**: Erteilt die Berechtigung zum Einfügen der Daten in die Map. Die ganzzahlige Konstante wird mit `MapPermission.INSERT` definiert.
- **remove**: Erteilt die Berechtigung zum Entfernen der Daten aus der Map. Die ganzzahlige Konstante wird mit `MapPermission.REMOVE` definiert.
- **invalidate**: Erteilt die Berechtigung zum Invalidieren der Map-Daten. Die ganzzahlige Konstante wird mit `MapPermission.INVALIDATE` definiert.
- **all**: Erteilt alle Berechtigungen (read, write, insert, remove und invalidate). Die ganzzahlige Konstante wird mit `MapPermission.ALL` definiert.

Sie können ein Objekt MapPermission erstellen, indem Sie den vollständig qualifizierten Namen der ObjectGrid-Map im Format [*ObjectGrid-Name*].[*ObjectMap-Name*] sowie die Berechtigungszeichenfolge oder den entsprechenden ganzzahligen Wert übergeben. Eine Berechtigungszeichenfolge kann eine durch Kommata begrenzte Zeichenfolge mit Berechtigungen sein, wie z. B. `"read, insert"`. Die Berechtigungszeichenfolge `"all"` bedeutet, dass alle Berechtigungen erteilt werden. Ein ganzzahliger Berechtigungswert kann jede der zuvor genannten ganzzahligen Berechtigungskonstanten oder ein mathematischer "oder"-Wert mit mehreren ganzzahligen Berechtigungskonstanten wie `MapPermission.GET|MapPermission.PUT` sein. Die Autorisierung findet statt, wenn ein Client eine Methode des Interface ObjectMap oder JavaMap aufruft. Die ObjectGrid-Laufzeitumgebung überprüft die Berechti-

gungen für verschiedene Methoden. Falls der Client nicht die erforderlichen Berechtigungen besitzt, wird eine Ausnahme des Typs `AccessControlException` ausgelöst.

In der folgenden Tabelle sind die Berechtigungen und die Methoden aufgelistet, die die jeweilige Berechtigung erfordern.

Tabelle 6. Methodenberechtigungen

Berechtigung	**Methode com.ibm.websphere.objectgrid.ObjectMap oder com.ibm.websphere.objectgrid.JavaMap**
read	`boolean containsKey(Object)`
	`boolean equals(Object)`
	`Object get(Object)`
	`Object get(Object, Serializable)`
	`List getAll(List)`
	`List getAll(List keyList, Serializable)`
	`List getAllForUpdate(List)`
	`List getAllForUpdate(List, Serializable)`
	`Object getForUpdate(Object)`
	`Object getForUpdate (Object, Serializable)`
write	`Object put(Object key, Object value)`
	`void put (Object, Object, Serializable)`
	`void putAll(Map)`
	`void putAll(Map, Serializable)`
	`void update (Object, Object)`
	`void update (Object, Object, Serializable)`
insert	`public void insert (Object, Object)`
	`void insert (Object, Object, Serializable)`
	`remove Object remove (Object)`
	`void removeAll(Collection)`
invalidate	`public void invalidate (Object, boolean)`
	`void invalidateAll(Collection, boolean)`
	`void invalidateUsingKeyword(Serializable, boolean)`
	`int setTimeToLive(int)`

Autorisierungsmechanismen

ObjectGrid unterstützt zwei Arten von Autorisierungsmechanismen: die Autorisierung mit Java Authentication and Authorization Service (JAAS) und die benutzerdefinierte Autorisierung.

Die JAAS-Autorisierung erweitert die Java-Sicherheitsrichtlinien um benutzerdefinierte Zugriffssteuerungen. Berechtigungen können basierend auf dem ausgeführten Code und dem Ausführenden des Codes erteilt werden. Die JAAS-Autorisierung ist Teil des IBM Software Development Kit (SDK) Version 1.4.

ObjectGrid unterstützt mit dem Plug-in `com.ibm.websphere.objectgrid.security.plugins.MapAuthorization` auch die benutzerdefinierte Autorisierung. Sie können Ihren eigenen Autorisierungsmechanismus implementieren, wenn Sie die JAAS-Autorisierung nicht verwenden möchten. Wenn Sie einen eigenen Autorisierungsmechanismus verwenden, können Sie Policy-Datenbanken, Policy-Server oder Tivoli Access Manager für die Verwaltung der ObjectGrid-Autorisierungen verwenden.

Sie können den ObjectGrid-Autorisierungsmechanismus auf zwei Arten konfigurieren:

- **XML-Konfiguration:** Mit einer XML-Datei können Sie eine ObjectGrid-Instanz definieren und den Autorisierungsmechanismus auf `AUTHORIZATION_MECHANISM_JAAS` oder `AUTHORIZATION_MECHANISM_CUSTOM` setzen. Im Folgenden sehen Sie die Datei `secure-objectgrid-definition.xml`, die in der Beispiel-Enterprise-Anwendung ObjectGridSample verwendet wird:

  ```
  <objectGrids>
   <objectGrid name="secureClusterObjectGrid securityEnabled="true"
         authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS" >
    <bean id="TransactionCallback" className=
   "com.ibm.websphere.samples.objectgrid.HeapTransactionCallback"/>
  ....
  </objectGrids>
  ```

- **Über das Programm:** Wenn Sie eine ObjectGrid-Instanz mit den APIs erstellen möchten, können Sie die folgende Methode aufrufen, um den Autorisierungsmechanismus festzulegen:

  ```
  /**
  * Autorisierungsmechanismus festlegen. Der Standardwert ist
  * com.ibm.websphere.objectgrid.security.SecurityConstants.
  * AUTHORIZATION_MECHANISM_JAAS.
  * @param authMechanism Der Autorisierungsmechanismus für die Map
  */
  void setAuthorizationMechanism(int authMechanism);
  ```

JAAS-Autorisierung

Das Objekt `javax.security.auth.Subject` stellt einen authentifizierten Benutzer dar. Ein Subject-Objekt setzt sich aus einer Reihe von Principals zusammen, und jeder Principal repräsentiert eine Identität für diesen Benutzer. Beispielsweise könnte ein Subject-Objekt einen Namens-Principal (`"Joe Smith"`) und einen Gruppen-Principal (`"manager"`) haben.

Mit der JAAS-Autorisierungs-Policy können speziellen Principals Berechtigungen erteilt werden. ObjectGrid ordnet das Subject-Objekt dem aktuellen Zugriffssteuerungskontext zu. Bei jedem Aufruf einer Methode des Interface ObjectMap oder JavaMap stellt die Java-Laufzeitumgebung fest, ob die Policy die erforderliche Berechtigung nur einem bestimmten Principal erteilt. Diese Operation ist nur zulässig, wenn das Subject-Objekt, das dem Zugriffssteuerungskontext zugeordnet ist, diesen Principal enthält.

ObjectGrid hat eine spezielle Codebasis, die zur Überprüfung der JAAS-Autorisierung für die ObjectMap- und JavaMap-Methodenaufrufe verwendet wird. Diese spezielle Codebasis ist auf der folgenden Website verfügbar:
`http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction`.
Verwenden Sie diese Codebasis, um Principals ObjectMap- oder JavaMap-Berechtigungen zuzuordnen.

Die Policy-Schablone für die Erteilung der MapPermission ist wie folgt:

```
grant codeBase
 "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
  <Principal field(s)> {
  permission com.ibm.websphere.objectgrid.security.MapPermission
  "map_name", "action";
....
  permission com.ibm.websphere.objectgrid.security.MapPermission
  "map_name", "action";
 };
```

Das folgende Code-Snippet enthält ein Beispiel für ein Principal-Feld:

```
Principal Principal_class "principal_name"
```

Dem Wort `Principal` folgen der vollständig qualifizierte Name einer Principal-Klasse und ein Principal-Name. ″map_name″ steht für den vollständig qualifizierten Map-Namen im Format [*ObjectGrid-Name*].[*Map-Name*], z. B. `secureClusterObjectGrid.employees`. Die Aktion ist eine durch Kommata begrenzte Zeichenfolge mit den Berechtigungen, die in der Klasse MapPermission definiert sind, wie z. B. `read, insert` oder `all`.

Die Verwendung von Platzhalterzeichen wird in eingeschränktem Umfang unterstützt. Sie können den ObjectGrid-Namen und den Map-Namen durch einen Stern (`*`) ersetzen. Die Verwendung des Sterns zum Ersetzen eines Teil des ObjectGrid-Namens oder Map-Namens wird jedoch von ObjectGrid nicht unterstützt. Deshalb sind die Zeichenfolgen `*.employees`, `clusterObjectGrid.*` und `*.*` gültige Namen, aber die Zeichenfolge `cluster*.employees` nicht.

In der Beispielanwendung in der Datei `ObjectGridSample.ear` sind beispielsweise zwei Policy-Dateien für die Autorisierung definiert: `fullAccessAuth.policy` und `readInsertAccessAuth.policy`. Der Inhalt der Policy-Datei `readInsertAccessAuth.policy` ist wie folgt:

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/
  security/PrivilegedAction" /*Die Zeilen sind zur besseren Lesbarkeit umgebrochen*/
 Principal com.ibm.ws.security.common.auth.WSPrincipalImpl
  "principal_name" {
 permission com.ibm.websphere.objectgrid.security.MapPermission
  "secureClusterObjectGrid.employees", "read,insert";
 permission com.ibm.websphere.objectgrid.security.MapPermission
  "secureClusterObjectGrid.offices", "read,insert";
 permission com.ibm.websphere.objectgrid.security.MapPermission
  "secureClusterObjectGrid.sites", "read,insert";
 permission com.ibm.websphere.objectgrid.security.MapPermission
  "secureClusterObjectGrid.counters", "read,insert";
};
```

In dieser Policy-Datei werden einem bestimmten Principal nur die Berechtigungen insert und read für die vier Maps erteilt. Die Datei `fullAccessAuth.policy` erteilt einem Principal alle Berechtigungen für diese Maps. Bevor Sie die Anwendung ausführen, müssen Sie *principal_name* durch einen entsprechenden Wert ersetzen und die Principal-Klasse angeben. Der Wert, den Sie für *principal_name* angeben müssen, richtet sich nach der Benutzer-Registry. Wenn Sie beispielsweise ein lokales Betriebssystem (localOS) als Benutzer-Registry verwenden, ist der Maschinenname **MACH1**, die Benutzer-ID **user1** und `MACH1/user1` der **principal_name**.

Eine JAAS-Autorisierungs-Policy kann direkt in die Java-Policy-Datei oder in eine separate JAAS-Autorisierungsdatei gestellt und anschließend mit dem JVM-Argument `-Djava.security.auth.policy=file:[Datei_mit_JAAS-`

Autorisierungs-Policy] oder mit der Einstellung auth.poliyc.url.x=file:[Datei_mit_JAAS-Autorisierungs-Policy] in der Datei java.security definiert werden.

Benutzerdefinierte Autorisierung mit dem Plug-in MapAuthorization

ObjectGrid unterstützt mit dem Plug-in MapAuthorization auch die benutzerdefinierte Autorisierung. Sie können dieses Plug-in verwenden, um ObjectMap- und JavaMap-Zugriffe auf die im Subject-Objekt enthaltenen Principals zu autorisieren.

Die ObjectGrid-Laufzeitumgebung ruft die Methode boolean checkPermission(Subject subject, MapPermission permission) des Interface MapAuthorization auf, um zu prüfen, ob das übergebene Subject-Objekt die übergebenen Berechtigungen besitzt. Die Implementierung des Interface MapAuthorization muss true zurückgeben, wenn das Subject-Objekt die Berechtigungen besitzt, und false, wenn es diese nicht besitzt.

Ein typischer Implementierungszweck dieses Plug-in ist, die Principals aus dem Subject-Objekt abzurufen und unter Verwendung bestimmter Policys zu prüfen, ob den Principals die angegebenen Berechtigungen erteilt wurden. Diese Policys werden vom Benutzer definiert. Sie können beispielsweise in einer Datenbank, einer unverschlüsselten Datei oder einem TAM-Policy-Server (Tivoli Access Manager) definiert werden.

ObjectGrid stellt zwei Standardimplementierungen für dieses Plug-in bereit. Die Klasse com.ibm.websphere.objectgrid.security.plugins.builtins.JAASMapAuthorizationImpl ist eine Implementierung des Plug-in MapAuthorization, die für die Autorisierung einen JAAS-Mechanismus verwendet. Die Implementierungsklasse com.ibm.websphere.objectgrid.security.plugins.builtins.TAMMapAuthorizationImpl zeigt, wie Sie mit Tivoli Access Manager die ObjectGrid-Autorisierungen verwalten können. Das folgende Code-Snippet veranschaulicht den Hauptverarbeitungsablauf der Methode JAASMapAuthorizationImpl.checkPermission(Subject, MapPermission):

```
// Erstellt eine PrivilegedExceptionAction-Aktion für die Überprüfung
// der Berechtigungen.
PrivilegedExceptionAction action =
  MapPermissionCheckAction.getInstance(permission);
Subject.doAsPrivileged(subject, action, null);
```

Verwenden Sie dieses TAMMapAuthorizationImpl-Plug-in nicht sofort, da verschiedene restriktive Bedingungen erfüllt sein müssen:

- Das Subject-Objekt enthält einen Principal com.tivoli.mts.PDPrincipal.
- Im TAM-Policy-Server sind die folgenden Berechtigungen für das ObjectMap- bzw. JavaMap-Namensobjekt definiert. Der Name des im Policy-Server definierten Objekts muss mit dem ObjectMap- bzw. JavaMap-Namen übereinstimmen und das Format [*OBJECTGRID-NAME*].[*MAP-NAME*] haben. Die Berechtigung ist das erste Zeichen der Berechtigungszeichenfolgen, die in MapPermission definiert sind. Beispielsweise steht im Policy-Server definierte Berechtigung "r" für die Leseberechtigung für die ObjectMap-Instanz.

Intervall für Berechtigungsprüfung

ObjectGrid unterstützt aus Leistungsaspekten die Zwischenspeicherung der Ergebnisse der Berechtigungsprüfung. Wenn kein Intervall für die Berechtigungsprüfung definiert wird, ruft die ObjectGrid-Laufzeitumgebung beim Aufruf einer der in diesem Abschnitt aufgelisteten Metho-

den den konfigurierten Autorisierungsmechanismus auf, um den Zugriff zu autorisieren. Ist ein Intervall für die Berechtigungsprüfung definiert, wird der Autorisierungsmechanismus im definierten Intervall aufgerufen.

Das Intervall für die Berechtigungsprüfung kann auf zwei Arten konfiguriert werden:

- **XML-Konfiguration:** Mit einer XML-Datei können Sie eine ObjectGrid-Instanz definieren und anschließend das Intervall für die Berechtigungsprüfung festlegen. Im folgenden Beispiel wird ein Intervall von 45 Sekunden für die Berechtigungsprüfung festgelegt:

```
<objectGrids>
  <objectGrid name="SecureClusterObjectGrid" securityEnabled="true"
        authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS"
        permissionCheckPeriod="45">
   <bean id="TransactionCallback"
    className="com.ibm.websphere.samples.objectgrid.
    HeapTransactionCallback"/>
...
</objectGrids>
```

- **Über das Programm:** Wenn Sie eine ObjectGrid-Instanz mit den APIs erstellen möchten, können Sie die folgende Methode aufrufen, um das Intervall für die Berechtigungsprüfung festzulegen:

```
/**
* Diese Methode akzeptiert einen einzigen Parameter, der angibt,
* wie oft die Berechtigung für einen Clientzugriff geprüft werden soll.
* Wenn der Parameter den Wert 0 hat, weisen alle Methoden (get/put/
* update/remove/evict) den Autorisierungsmechanismus (JAAS-Autorisierung
* oder benutzerdefinierte Autorisierung) an zu prüfen, ob das aktuelle
* Subjekt die erforderliche Berechtigung besitzt. Dies kann aus
* Leistungssicht je nach Autorisierungsimplementierung sehr
* kostenintensiv sein, aber sollte es erforderlich sein, können
* Sie diesen Ansatz wählen.
* Wenn der Parameterwert kleiner als 0 ist, gibt dieser den
* Zeitraum (in Sekunden) an, für den die Berechtigungen
* zwischengespeichert werden, bevor sie vom Autorisierungsmechanismus
* aktualisiert werden.
* Dieser Parameter liefert eine erheblich bessere Leistung. Sollten sich
* die Backend-Berechtigungen jedoch in diesem Zeitraum ändern, ist es
* möglich, dass die ObjectGrid-Instanz den Zugriff zulässt oder verhindert,
* obwohl der Sicherheits-Provider des Backend modifiziert wurde.
*
* @param period Das Intervall für die Berechtigungsprüfung in Sekunden.
*/
void setPermissionCheckPeriod(int period);
```

Integration der Sicherheit mit WebSphere Extended Deployment

Eine der unterstützten Umgebungen für ObjectGrid ist WebSphere Extended Deployment. In diesem Abschnitt wird anhand der Beispielanwendung in der Datei `ObjectGridSample.ear` beschrieben, wie die Sicherheit von WebSphere Application Server für die Unterstützung des Mechanismus mit Übergabe des Subjekts mit der ObjectGrid-Sicherheit integriert werden kann.

Die Beispielanwendung `ObjectGridSample.ear` veranschaulicht die Verwendung von ObjectGrid in einer J2EE-Programmierumgebung. Die Beispielobjekte Employee, Office, Sites und Counters sind dieselben ObjectGrid-Maps wie im Eclipse-basierten Beispiel. Die Datei `ObjectGridSample.ear` demonstriert jedoch den sicheren Zugriff auf ObjectGrid. Dieser Abschnitt zeigt, wie in einem Server mit WebSphere Extended Deployment sicher auf ObjectGrid zugegriffen werden kann.

Sicherheitskonfiguration für die Anwendung

Die Sicherheitskonfiguration für diese Anwendung können Sie der folgenden Abbildung entnehmen:

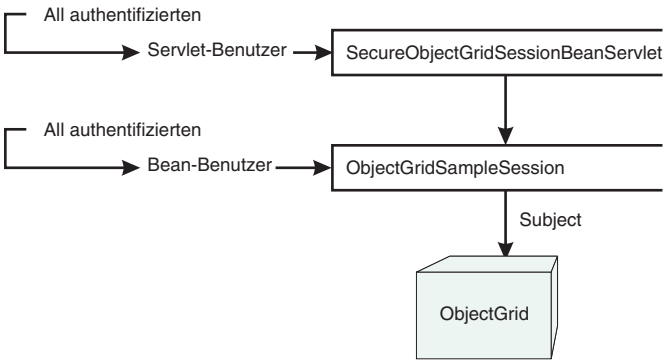


Abbildung 2. Sicherheitskonfiguration für die Beispieldatei ObjectGridSample.ear

Das Servlet SecureObjectGridSessionBeanServlet wird in einem Browser gestartet. Das Servlet ruft die Bean ObjectGridSampleSession auf, die wiederum die Objekt-Grid-Instanz aufruft.

Der Zugriff auf das Servlet securedObjectGridSessoinBeanServlet ist nur mit dem Sicherheitsaufgabenbereich servletUser möglich. ObjectGridSampleSession.executeActionWithSecurity kann nur mit dem Sicherheitsaufgabenbereich beanUser aufgerufen werden. servletUser und beanUser sind dem Sicherheitsaufgabenbereich **Alle authentifizierten Benutzer** zugeordnet, d. h. jeder authentifizierte Benutzer kann auf die ObjectGrid-Instanz zugreifen.

Subject-Objekt von WebSphere Extended Deployment abrufen

ObjectGrid stützt sich bei der Authentifizierung auf die Anwendungsserver oder Anwendungen. Sehen Sie sich das folgende Code-Snippet in der Methode `internalExecuteAction` der Klasse com.ibm.websphere.samples.objectgrid.basic.ejb.ObjectGridSampleSessionBean an:

```
private String internalExecuteAction(String action, boolean isSecurityEnabled,
String methodName) {
 ... ...
 Subject subject = null;
 if (isSecurityEnabled) {
  try {
   subject = WSSubject.getCallerSubject();
  }
  catch (Exception e) {
  throw new RuntimeException(e);
 }
}
```

In diesem Snippet wird ein Objekt Subject mit der Methode `WSSubject.getCallerSubject` vom Thread abgerufen. Dieses Subject-Objekt stellt das Caller-Subjekt dar.

Sitzung durch Übergabe des Subject-Objekts abrufen

Sehen Sie sich die Methode `init` in der Klasse com.ibm.websphere.samples.ObjectGridSampleWrapper an:

```
public void init(boolean isSecurityEnabled, Subject subject) {
 ... ...
 if(ivObjectGrid!=null){
  if (!isSecurityEnabled) {
   ivSession = ivObjectGrid.getSession();
  }
 else {
  ivSession = ivObjectGrid.getSession(subject);
}
```

Dieses Snippet zeigt, wie mit dem Subject-Objekt, das von WebSphere Application Server abgerufen wurde, das Sitzungsobjekt ivSession abgerufen wird.

Globale Sicherheit in WebSphere Application Server aktivieren

Wenn WebSphere Application Server die Authentifizierung handhaben soll, muss die globale Sicherheit in WebSphere Application Server aktiviert sein. Führen Sie die folgenden Schritte aus, um die globale Sicherheit in WebSphere Application Server zu aktivieren:

1. Klicken Sie in der Administrationskonsole auf **Sicherheit > Globale Sicherheit**.
2. Vergewissern Sie sich, dass das Markierungsfeld **Globale Sicherheit** ausgewählt ist. Das Markierungsfeld **Java-2-Sicherheit erzwingen** wird automatisch ausgewählt.
3. Wählen Sie für **Aktives Protokoll** den Eintrag `CSI und SAS` aus.
4. Wählen Sie für **Aktives Authentifizierungsverfahren** den Eintrag `Lightweight Third Party Authentication (LTPA)` aus.
5. Wählen Sie für **Aktive Benutzer-Registry** den Eintrag `LocalOS` aus. Sie können auch andere Benutzer-Registrys auswählen.
6. Klicken Sie auf **OK**.
7. Falls Sie die Benutzer-Registry noch nicht konfiguriert haben, werden Sie aufgefordert, die Benutzer-Registry zu konfigurieren. Wenn Sie `LocalOS` auswählen, werden Sie aufgefordert, Benutzer-ID und Kennwort des Servers anzugeben.
8. Speichern Sie alle Änderungen.
9. Starten Sie den Application Server erneut, um die Änderungen zu aktivieren, die an der globalen Sicherheit vorgenommen wurden.

Nach dem Neustart des Servers können Sie die in der Benutzer-Registry definierte Benutzer-ID/Kennwort-Kombination verwenden, um auf die Administrationskonsole zuzugreifen.

Datei mit JAAS-Autorisierungs-Policy zum JVM-Argument hinzufügen

In der Datei `ObjectGridSample.ear` sind zwei Autorisierungs-Policy-Dateien definiert: `fullAccessAuth.policy` und `readInsertAccessAuth.policy`. Im Folgenden sehen Sie den Inhalt der Datei `fullAccessAuth.policy`:

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
 Principal com.ibm.ws.security.common.auth.WSPrincipalImpl "principal_name" {
 permission com.ibm.websphere.objectgrid.security.MapPermission
  "secureClusterObjectGrid.employees", "all";
 permission com.ibm.websphere.objectgrid.security.MapPermission
  "secureClusterObjectGrid.offices", "all";
 permission com.ibm.websphere.objectgrid.security.MapPermission
  "secureClusterObjectGrid.sites", "all";
 permission com.ibm.websphere.objectgrid.security.MapPermission
  "secureClusterObjectGrid.counters", "all";
};
```

Sie können diese beiden Policy-Dateien extrahieren und in einem Verzeichnis speichern, z. B. im Verzeichnis
Installationsstammverzeichnis/profiles/*Profilname*/properties.

WebSphere Application Server verwendet eine spezielle Principal-Klasse mit dem Namen com.ibm.ws.security.common.auth.WSPrincipalImpl. Verwenden Sie diese Principal-Klasse.

Sie müssen "principal_name" durch einen gültigen Wert ersetzen. Der Wert richtet sich nach der Benutzer-Registry. Wählen Sie eine in der Registry Ihres lokalen Betriebssystems definierte Benutzer-ID aus. Wenn Sie beispielsweise "LocalOS" als Benutzer-Registry verwenden, ist der Maschinenname MACH1, die Benutzer-ID user1 und der Prinicpal-Name MACH1/user1.

WebSphere Application Server bietet keine Unterstützung für die JAAS-Autorisierungs-Policy in der Java-Policy-Datei. Deshalb können die Autorisierungs-Policys nicht in die Java-Policy-Datei gestellt werden. Sie müssen die JAAS-Autorisierungs-Policy mit dem JVM-Argument -Djava.security.auth.policy definieren. Gehen Sie zum Definieren der JVM-Argumente in der Administrationskonsole von WebSphere Extended Deployment wie folgt vor:

1. Navigieren Sie zum Serverziel.
2. Klicken Sie auf **Java- und Prozessverwaltung > Prozessdefinition > Java Virtual Machine**.
3. Fügen Sie dem Feld **Generische JVM-Argumente** den folgenden Text hinzu:

   ```
   -Djava.security.auth.policy=file:[Datei_mit_JAAS-Autorisierungs-Policy]
   ```

   Ersetzen Sie [*Datei_mit_JAAS-Autorisierungs-Policy*] durch den Namen der Datei mit der JAAS-Autorisierungs-Policy, z. B.:

   ```
   -Djava.security.auth.policy=file:c:/WASXD/AppServer/profiles/AppSrv01/
   properties/fullAccessAuth.policy
   ```

Anwendung installieren

Installieren Sie die Anwendung im Anwendungsserver. Während der Installation können Sie alle Standardwerte übernehmen.

Gesichertes Servlet ausführen

1. Starten Sie nach dem Anwendungsstart den Server mit dem URL http://*Host*:*Port*/ObjectGridSample. *Host* steht für den Namen der Maschine, auf der der Anwendungsserver ausgeführt wird, und *Port* für die Nummer des HTTP-Port, z. B. 9080.
2. Klicken Sie auf **SecureObjectGridSessionBeanServlet**. Geben Sie die Benutzer-ID, die Sie in der JAAS-Policy-Datei definiert haben, und das zugehörige Kennwort ein.
3. Auf der Seite erscheint eine Textzeile wie die folgende:

   ```
   The caller of the session is user_ID.
   ```

   Die Variable *Benutzer-ID* steht für die Benutzer-ID, die für die Authentifizierung verwendet wird.
4. Sie können auf alle Aktionsschaltflächen klicken, da die Policy-Datei dem Benutzer alle Berechtigungen für die Map erteilt.
5. Sie können den Wert des JVM-Arguments -Djava.security.auth.policy in readInsertAccessAuth.policy ändern und anschließend den Server erneut starten. Wenn Sie diese Policy verwenden, können die fünf Aktionen nicht ausge-

führt werden. Beispielsweise wird auf der Seite die folgende Ausnahme angezeigt, wenn Sie auf die Aktion **2. Update maps** klicken:

```
Caught Throwable: java.security.AccessControlException:
access denied (com.ibm.websphere.objectgrid.
```

Diese Ausnahme tritt ein, weil der Map in der Policy readInsertAccessAuth.policy nicht die MapPermission für Schreibzugriff erteilt wird.

Listener

ObjectGrid stellt zwei Listener-Interfaces bereit, die Sie erweitern können. Wenn Sie die Listener-Interfaces erweitern, können Sie Benachrichtigungen über Operationen empfangen, die für eine ObjectGrid-Instanz oder eine Map-Instanz ausgeführt werden.

Interface ObjectGridEventListener

Das Interface ObjectGridEventListener wird verwendet, um Benachrichtigungen zu empfangen, wenn wichtige Ereignisse in einer ObjectGrid-Instanz eintreten. Zu diesen Ereignissen gehören die ObjectGrid-Initialisierung, der Beginn einer Transaktion, das Ende einer Transaktion und das Löschen einer ObjectGrid-Instanz. Zur Überwachung solcher Ereignisse erstellen Sie eine Klasse, die das Interface ObjectGridEventListener implementiert. Fügen Sie diese Klasse anschließend der ObjectGrid-Instanz hinzu.

Änderungen in einer Map-Instanz überwachen

Die Methode ObjectGridEventListener#transactionEnd ist hilfreich für Anwendungen, die Einträge in den lokalen Map-Instanzen überwachen möchten. Eine Anwendung kann einen dieser Listener hinzufügen und anschließend mit der Methode transactionEnd überwachen, wann die Einträge geändert werden. Wenn die ObjectGrid-Instanz beispielsweise im verteilten Modus arbeitet, kann die Anwendung eingehende Änderungen überwachen. Angenommen, die replizierten Einträge sind für die neuesten Aktienkurse bestimmt. Der Listener kann den Eingang dieser Änderungen überwachen und eine zweite Map-Instanz aktualisieren, die den Wert einer Position in einem Portfolio verwaltet. Der Listener muss alle Änderungen mit dem Session-Objekt vornehmen, das dem Listener mit der Methode ObjectGridEventListener#initialize bereitgestellt wurde. Der Listener kann normalerweise zwischen lokalen Änderungen und eingehenden fernen Änderungen unterscheiden, indem er prüft, ob die Transaktion auf Durchschreiben eingestellt ist. Die von Peer-ObjektGrid-Instanzen eingehenden Änderungen werden immer durchgeschrieben.

Interface ObjectGridEventListener

Das Interface ObjectGridEventListener enthält die folgenden Methoden. Diese Methoden werden aufgerufen, wenn bestimmte wichtige Ereignisse in der ObjectGrid-Instanz eintreten.

```
/**
* Diese Methode wird aufgerufen, wenn die ObjectGrid-Instanz selbst initialisiert
* wird. Es wird eine verwendbare Session-Instanz an den Listener übergeben, damit
* eine in einer Map-Instanz empfangene LogSequence optional wiederholt werden kann.
*
* @param session Die Session-Instanz, der dieser Listener zugeordnet ist.
*/
void initialize(Session session);
/**
* Dieses Ereignis kennzeichnet den Anfang einer Transaktion (Session).
* Es wird eine Zeichenfolgeversion von TxID bereitgestellt, die bei
* Bedarf mit dem Ende der Transaktion (Session) korreliert wird.
```

```
* Der Typ der Transaktion (Session) wird mit dem Booleschen Parameter
* isWriteThroughEnabled übermittelt.
*
* @param txid Zeichenfolgeversion von TxID
* @param isWriteThroughEnabled Boolesches Flag, das anzeigt,
* ob das Session-Objekt mit beginNoWriteThrough gestartet wurde
*/
void transactionBegin(String txid, boolean isWriteThroughEnabled);
/**
* Signalisiert das Ende einer Transaktion (Session). Es wird eine
* Zeichenfolgeversion von TxID bereitgestellt, die bei Bedarf
* mit dem Anfang der Transaktion (Session) korreliert wird.
* Änderungen werden ebenfalls berichtet. Dieses Ereignis wird
* typischerweise für Benutzer verwendet, die eigene
* Peer-Invalidierungen oder Push-Methoden mit COMMIT verwenden.
* Die Änderungen werden vom Ereignis-Listener übermittelt.
* Aufrufe dieser Methode werden nach der Festschreibung
* und nacheinander, nicht parallel ausgeführt. Die Reihenfolge
* der Ereignisse entspricht der Festschreibungsreihenfolge.
*
* Wenn die Transaktion nicht ordnungsgemäß festgeschrieben wird
* (committed == false), ist das Collection-Objekt für die Änderungen
* leer.
*
* @param txid Zeichenfolgeversion von TxID
* @param isWriteThroughEnabled Ein Boolesches Flag, das anzeigt, ob die
* Session-Instanz mit beginNoWriteThrough gestartet wurde.
* @param committed Ein Boolesches Flag, das anzeigt, ob die Session-Instanz
* festgeschrieben (true) oder rückgängig gemacht wurde (false).
* @param changes Ein Collection-Objekt mit LogSequences, die für die
* aktuelle Session-Instanz verarbeitet wurden. Falls die Session-Instanz
* zurückgesetzt wurde, ist dieser Parameter null.
*/
void transactionEnd(String txid, boolean isWriteThroughEnabled,
boolean committed, Collection /**/ changes);
/**
* Diese Methode wird aufgerufen, wenn die ObjectGrid-Instanz
* gelöscht wird. Sie ist das Gegenstück zur Methode
* initialize. Wenn diese Methode aufgerufen wird, kann der
* ObjectGridEventListener alle von ihm verwendeten Ressourcen
* freigeben.
*/
void destroy();
```

ObjectGridEventListener-Objekte hinzufügen und entfernen

Eine ObjectGrid-Instanz kann mehrere ObjectGridEventListener-Objekte haben. Es sind zwei Methoden verfügbar, mit denen Sie der ObjectGrid-Instanz ObjectGridEventListener-Objekte hinzufügen können. Hinzugefügte ObjectGridEventListener-Objekte können auch einer ObjectGrid-Instanz auch wieder entfernt werden.

Mit der Methode addEventListener können Sie einer ObjectGrid-Instanz ein ObjectGridEventListener-Objekt hinzufügen.

```
/**
* Der Session-Instanz einen EventListener hinzufügen. Wichtige
* Ereignisse werden den interessierten Listenern über diesen
* Callback mitgeteilt. Es können mehrere Ereignis-Listener ohne
* implizierte Reihenfolge der Ereignisbenachrichtigungen registriert
* werden.
*
* Diese Methode kann vor und nach der Methode
* {@link ObjectGrid#initialize()} aufgerufen werden.
*
* @param cb Eine Instanz von ObjectGridEventListener
*/
void addEventListener(ObjectGridEventListener cb);
```

Wenn Sie eine Liste von ObjectGridEventListener-Objekten hinzufügen möchten, verwenden Sie die Methode setEventListeners:

```
/**
* Diese Methode überschreibt die aktuelle Liste der Callbacks und ersetzt sie
* durch die angegebene Liste von Callbacks.
*
* Diese Methode kann vor und nach der Methode
* {@link ObjectGrid#initialize()} aufgerufen werden.
* @param callbacks
*/
void setEventListeners(List callbacks);
```

Wenn Sie ein ObjectGridEventListener-Objekt aus einer ObjectGrid-Instanz entfernen möchten, verwenden Sie die Methode removeEventListener:

```
/**
* Entfernt ein EventListener-Objekt aus der Sitzung. Falls das gewünschte
* EventListener-Objekt nicht in der Session-Instanz vorhanden ist, wird ein
* Fehler zurückgegeben.
*
* Diese Methode kann vor und nach der Methode
* {@link ObjectGrid#initialize()} aufgerufen werden.
* @param cb Eine Instanz von ObjectGridEventListener
*/
void removeEventListener(ObjectGridEventListener cb);
```

Benutzerdefinierten ObjectGrid-Ereignis-Listener erstellen

Wenn Sie einen eigenen ObjectGrid-Ereignis-Listener verwenden möchten, müssen Sie zuerst eine Klasse erstellen, die das Interface com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener implementiert. Fügen Sie einer ObjectGrid-Instanz einen eigenen Listener hinzu, wenn Sie Benachrichtigungen über wichtige Ereignisse empfangen möchten. Ein ObjectGridEventListener-Objekt kann über das Programm oder mit einer XML-Datei konfiguriert werden:

- **Über das Programm:** Angenommen, der Klassenname des ObjectGrid-Ereignis-Listener ist com.company.org.MyObjectGridEventListener. Diese Klasse implementiert das Interface ObjectGridEventListener. Das folgende Code-Snippet erstellt das benutzerdefinierte ObjectGridEventListener-Objekt und fügt es einer ObjectGrid-Instanz hinzu:

  ```
  ObjectGridManager objectGridManager =
   ObjectGridManagerFactory.getObjectGridManager();
  ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
  MyObjectGridEventListener myListener = new MyObjectGridEventListener();
  myGrid.addEventListener(myListener);
  ```

- **XML-Konfiguration:** Ein ObjectGridEventListener-Objekt kann auch mit einer XML-Datei konfiguriert werden. Die folgende XML-Datei erstellt eine Konfiguration, die dem über das Programm erstellten ObjectGrid-Ereignis-Listener entspricht. Der folgende Text muss in der Datei myGrid.xml enthalten sein:

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

Stellen Sie diese Datei dem Interface ObjectGridManager zur Verfügung, um die Erstellung der Konfiguration zu vereinfachen. Das folgende Code-Snippet veranschaulicht, wie mit dieser XML-Datei eine ObjectGrid-Instanz erstellt wird. Die erstellte ObjectGrid-Instanz definiert ein ObjectGridEventListener-Objekt für die ObjectGrid-Instanz myGrid.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
 objectGridManager.createObjectGrid("myGrid", new URL(
  "file:etc/test/myGrid.xml"), true, false);
```

Interface MapEventListener

Verwenden Sie das Interface MapEventListener, wenn Sie wichtige Ereignisse zu einer Map empfangen möchten. Ereignisse werden an den MapEventListener gesendet, wenn ein Eintrag aus der Map gelöscht wird und wenn das vorherige Laden der Daten in eine Map abgeschlossen ist.

Interface MapEventListener

Das Interface MapEventListener enthält die folgenden Methoden. Implementieren Sie das Interface com.ibm.websphere.objectgrid.plugins.MapEventListener, wenn Sie einen eigenen MapEventListener erstellen möchten.

```
/**
* Diese Methode wird aufgerufen, wenn der angegebene Eintrag aus der
* Map gelöscht wird. Die Bereinigung kann von einem Evictor
* oder durch Aufruf einer der Invalidierungsmethoden im Interface
* ObjectMap durchgeführt werden.
*
* @param key Der Schlüssel für den gelöschten Map-Eintrag.
* @param value Der Wert, der im gelöschten Map-Eintrag enthalten
* war. Das Wertobjekt darf nicht geändert werden.
*
*/
void entryEvicted(Object key, Object value);
/**
* Diese Methode wird aufgerufen, wenn das vorherige Laden der Daten
* in die Map abgeschlossen ist.
*
* @param t Ein Throwable-Objekt, das anzeigt, ob das vorherige Laden ohne
* Auslösen eines Throwable abgeschlossen wurde. Eine Nullreferenz zeigt
* an, dass beim vorherigen Laden von Daten in die Map keine Throwable-Objekte
* erzeugt wurden.
*/
void preloadCompleted( Throwable t );
```

MapEventListener hinzufügen und entfernen

Mit den folgenden BackingMap-Methoden können MapEventListener für eine Map hinzugefügt und entfernt werden:

```
/**
* Fügt der BackingMap-Instanz einen MapEventListener hinzu.
*
* Diese Methode kann vor und nach der Methode
* ObjectGrid.initialize() aufgerufen werden.
* @param eventListener Eine nicht leere Referenz auf einen MapEventListener,
* die der Liste hinzugefügt werden soll.
*
* @throws IllegalArgumentException, falls eventListener null ist.
*
* @see MapEventListener
*/
public void addMapEventListener(MapEventListener
 eventListener );
```

```
/**
* Definiert die Liste der MapEventListener-Objekte.
*
* Wenn die BackingMap-Instanz bereits eine Liste mit
* MapEventListener-Objekten besitzt, wird diese Liste durch
* die Liste ersetzt, die als Argument an den aktuellen Aufruf
* dieser Methode übergeben wurde. Diese Methode kann vor und
* nach der Methode ObjectGrid.initialize() aufgerufen werden.
*
* @param eventListenerList Eine nicht leere Referenz auf eine
* Liste mit MapEventListener-Objekten.
*
* @throws IllegalArgumentException Wird ausgelöst, wenn
* eventListenerList gleich null ist oder das
* eventListenerList-Objekt eine leere Referenz oder
* ein Objekt enthält, das keine Instanz von MapEventListener ist.
*
* @see MapEventListener
*/
public void setMapEventListeners( List /*MapEventListener*/
 eventListenerList );
/**
* Entfernt einen MapEventListener für diese BackingMap-Instanz.
*
* Diese Methode kann vor und nach der Methode
* ObjectGrid.initialize() aufgerufen werden.
*
* @param eventListener Eine nicht leere Referenz auf einen
* Ereignis-Listener, der zuvor durch den Aufruf der Methode
* addMapEventListener(MapEventListener) oder der Methode
* setMapEventListeners(List) dieses Interface hinzugefügt wurde.
*
* @throws IllegalArgumentException, falls eventListener gleich null ist.
*
* @see MapEventListener
*/
public void removeMapEventListener(MapEventListener eventListener );
```

MapEventListener erstellen

Zum Erstellen eines eigenen MapEventListener müssen Sie das Interface com.ibm.websphere.objectgrid.plugins.MapEventListener implementieren. Wenn Sie den MapEventListener verwenden möchten, müssen Sie ihn einer BackingMap-Instanz hinzufügen. Ein MapEventListener kann über das Programm oder mit einer XML-Datei erstellt und konfiguriert werden:

- **Über das Programm:** Der Klassenname für den benutzerdefinierten MapEventListener ist com.company.org.MyMapEventListener. Diese Klasse implementiert das Interface MapEventListener. Das folgende Code-Snippet erstellt den benutzerdefinierten MapEventListener und fügt ihn einer BackingMap-Instanz hinzu:

  ```
  ObjectGridManager objectGridManager =
   ObjectGridManagerFactory.getObjectGridManager();
  ObjectGrid myGrid = objectGridManager.createObjectGrid("myGrid", false);
  BackingMap myMap = myGrid.defineMap("myMap");
  MyMapEventListener myListener = new MyMapEventListener();
  myMap.addMapEventListener(myListener);
  ```

- **XML-Konfiguration:** Ein MapEventListener kann auch mit einer XML-Datei konfiguriert werden. Mit der folgenden XML-Datei erstellen Sie eine Konfiguration, die dem zuvor über das Programm erstellten MapEventListener entspricht. Die folgende XML muss in der Datei `myGrid.xml` enthalten sein:

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

```
  ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
   <objectGrids>
    <objectGrid name="myGrid">
     <backingMap name="myMap" pluginCollectionRef="myPlugins" />
    </objectGrid>
   </objectGrids>
   <backingMapPluginCollections>
    <backingMapPluginCollection id="myPlugins">
     <bean id="MapEventListener"
        classname="com.company.org.MyMapEventListener" />
    </backingMapPluginCollection>
   </backingMapPluginCollection>
</objectGridConfig>
```

Stellen Sie diese Datei dem Interface ObjectGridManager zur Verfügung, um die Erstellung der Konfiguration zu vereinfachen. Das folgende Code-Snippet veranschaulicht, wie mit dieser XML-Datei eine ObjectGrid-Instanz erstellt wird. Die neu erstellte ObjectGrid-Instanz definiert einen MapEventListener für die BackingMap-Instanz myMap.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid myGrid =
 objectGridManager.createObjectGrid("myGrid", new URL(
 "file:etc/test/myGrid.xml"), true, false);
```

Evictor

ObjectGrid stellt einen Standardbereinigungsmechanismus bereit. Sie können aber auch einen eigenen Plug-in-Evictor verwenden.

Ein *Evictor* (oder Bereinigungsprogramm) steuert die Zugehörigkeit von Einträgen in einer BackingMap-Instanz. Der Standard-Evictor verwendet für jede BackingMap-Instanz eine Bereinigungs-Policy, die auf der *Lebensdauer* (TTL, Time to Live) der Einträge basiert. Wenn Sie einen eigenen Plug-in-Evictor nutzen, verwendet dieser in der Regel eine Bereinigungs-Policy, die auf der Anzahl der Einträge und nicht auf deren Lebensdauer basiert. In diesem Abschnitt werden beide Evictor-Typen beschrieben.

Standard-TTL-Evictor

ObjectGrid stellt für jede BackingMap-Instanz einen Evictor bereit, der auf der Basis der Lebensdauer (TTL, Time to Live) von Einträgen arbeitet. Der TTL-Evictor verwaltet für jeden erstellten Eintrag eine Verfallszeit. Wenn die Verfallszeit für einen Eintrag erreicht ist, löscht der Evictor den Eintrag aus der BackingMap-Instanz. Zur Minimierung der Leistungseinbußen kann der TTL-Evictor die Bereinigung eines Eintrags über die Verfallszeit hinweg hinauszögern. Ein Eintrag wird jedoch nie vor seiner Verfallszeit bereinigt.

Die BackingMap-Instanz hat Attribute, mit denen gesteuert werden kann, wie der TTL-Evictor die Verfallszeit für die einzelnen Einträge berechnet. Anwendungen definieren mit dem Attribut ttlType, wie der TTL-Evictor die Verfallszeit berechnet. Die gültigen Werte für das Attribut ttlType sind im Folgenden beschrieben:

- **None** zeigt an, dass ein Eintrag in der BackingMap-Instanz nie verfällt. Der TTL-Evictor löscht solche Einträge nicht.
- **Creation time** zeigt an, dass die Erstellungszeit eines Eintrags als Basis für die Berechnung der Verfallszeit verwendet wird.

- **Last access time** zeigt an, dass der Zeitpunkt des letzten Zugriffs auf einen Eintrag für die Berechnung der Verfallszeit herangezogen wird.

Wenn das Attribut ttlType in einer BackingMap-Instanz nicht definiert ist, wird der Standardwert **None** verwendet, so dass der TTL-Evictor keine Einträge löscht. Ist das Attribut ttlType auf **creation time** oder **last access time** gesetzt, wird der Wert des Attributs für die Lebensdauer in der BackingMap-Instanz der Erstellungszeit bzw. der Zeit des letzten Zugriffs hinzugefügt, um die Verfallszeit zu berechnen. Der Wert des Attributs für die Lebensdauer eines Eintrags in der BackingMap-Instanz wird in Sekunden angegeben. `0` ist ein spezieller Wert für dieses Attribut und wird verwendet, um anzuzeigen, dass der BackingMap-Eintrag eine unbegrenzte Lebensdauer hat, d. h. der Eintrag bleibt so lange in der Map-Instanz, bis die Anwendung ihn explizit entfernt oder invalidiert.

Attribute für TTL-Evictor angeben

TTL-Evictor werden BackingMap-Instanzen zugeordnet. Das folgende Code-Snippet veranschaulicht, wie Sie mit dem Interface BackingMap die erforderlichen Attribute so definieren können, dass beim Erstellen eines Eintrags die Verfallszeit auf 10 Minuten nach der Erstellungszeit gesetzt wird.

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

Das Methodenargument `setTimeToLive` hat den Wert `600`, weil die Lebensdauer in Sekunden angegeben wird. Der vorherige Code muss ausgeführt werden, bevor die Methode `initialize` für die ObjectGrid-Instanz aufgerufen wird. Diese BackingMap-Attribute können nach der Initialisierung der ObjectGrid-Instanz nicht mehr geändert werden. Nach der Ausführung des Codes erhält jeder Eintrag, der in die BackingMap-Instanz myMap eingefügt wird, eine Verfallszeit. Beim Erreichen der Verfallszeit löscht der TTL-Evictor den Eintrag.

Wenn eine Anwendung erfordert, dass die Verfallszeit auf die Zeit des letzten Zugriffs plus zehn Minuten gesetzt wird, muss eine Zeile des vorherigen Codes geändert werden. Das Argument, das an die Methode `setTtlEvictorType` übergeben wird, muss von `TTLType.CREATION_TIME` in `TTLType.LAST_ACCESS_TIME` geändert werden. Dann wird die Verfallszeit aus der Zeit des letzten Zugriffs zuzüglich 10 Minuten berechnet. Wenn ein Eintrag erstellt wird, entspricht die Zeit des letzten Zugriffs der Erstellungszeit.

Wenn Sie `TTLType.LAST_ACCESS_TIME` verwenden, können Sie mit den Interfaces ObjectMap und JavaMap die Lebensdauer in der BackingMap-Instanz überschreiben. Dieser Mechanismus ermöglicht einer Anwendung, für jeden erstellten Eintrag eine andere Lebensdauer zu verwenden. Angenommen, das vorherige Code-Snippet wurde verwendet, um das Attribut ttlType auf `LAST_ACCESS_TIME` zu setzen, und die Lebensdauer in der BackingMap-Instanz ist auf zehn Minuten eingestellt. Danach kann eine Anwendung die Lebensdauer für jeden Eintrag überschreiben, indem sie vor dem Erstellen oder Ändern eines Eintrags den folgenden Code ausführt:

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

In diesem Code-Snippet hat der Eintrag mit dem Schlüssel `key1` eine Verfallszeit, die der Erstellungszeit plus 30 Minuten entspricht, weil der Methodenaufruf `setTimeToLive( 1800 )` für die ObjectMap-Instanz deklariert wurde. Die Variable oldTimeToLive1 ist auf 600 eingestellt, weil der Wert für die Lebensdauer aus der BackingMap-Instanz als Standardwert verwendet wird, wenn die Methode `setTimeToLive` für die ObjectMap-Instanz noch nicht aufgerufen wurde.

Der Eintrag mit dem Schlüssel `key2` hat eine Verfallszeit, die sich aus der Erstellungszeit plus 20 Minuten berechnet, weil der Methodenaufruf `setTimeToLive( 1200 )` für die ObjectMap-Instanz deklariert ist. Die Variable oldTimeToLive2 ist auf 1800 eingestellt, weil der Wert für die Lebensdauer aus dem vorherigen Aufruf der Methode `ObjectMap.setTimeToLive` die Lebensdauer auf 1800 gesetzt hat.

Informationen zur Methode `setTimeToLive` in den Interfaces ObjectMap und JavaMap finden Sie in der API-Dokumentation. In der Dokumentation werden Sie darauf hingewiesen, dass eine Ausnahme des Typs `IllegalStateException` ausgelöst wird, wenn die Methode `BackingMap.getTtlEvictorType()` einen anderen Wert als den Wert von `TTLType.LAST_ACCESS_TIME` zurückgibt. Die Interfaces ObjectMap und JavaMap können nur dann zum Überschreiben der Lebensdauer verwendet werden, wenn Sie den TTL-Evictor-Typ `LAST_ACCESS_TIME` verwenden. Wenn Sie den TTL-Evictor-Typ `CREATION_TIME` oder `NONE` verwenden, können Sie mit dieser Methoden die Lebendauer nicht überschreiben.

Attribute für den TTL-Evictor in einer XML-Datei angeben

Anstatt das Interface BackingMap zu verwenden, um die BackingMap-Attribute für den TTL-Evictor über das Programm zu definieren, können Sie die BackingMap-Instanzen mit einer XML-Datei konfigurieren. Der folgende Beispielcode definiert die Attribute für drei unterschiedliche BackingMap-Instanzen:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
        xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
 <objectGrid name="grid1">
  <backingMap name="map1" ttlEvictorType="NONE" />
  <backingMap name="map2" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="1800" />
  <backingMap name="map3" ttlEvictorType="CREATION_TIME" timeToLive="1200" />
 </objectgrid>
</objectGrids>
```

Das vorherige Beispiel zeigt, dass die BackingMap-Instanz map1 den TTL-Evictor-Typ `NONE` verwendet. Die BackingMap-Instanz map2 verwendet den Typ `LAST_ACCESS_TIME` und hat eine Lebensdauer von 1800 Sekunden (oder 30 Minuten) definiert. Die BackingMap-Instanz map3 verwendet den TTL-Evictor-Typ `CREATION_TIME` und eine Lebensdauer von 1200 Sekunden (oder 20 Minuten).

Optionale Plug-in-Evictor

Der Standard-TTL-Evictor verwendet eine Bereinigungs-Policy, die auf Zeit basiert. Die Anzahl der Einträge in der BackingMap-Instanz hat keine Auswirkung auf die Verfallszeit eines Eintrags. Sie können einen optionalen Plug-in-Evictor verwenden, der die Bereinigung basierend auf der Anzahl der vorhandenen Einträge anstelle der Zeit durchführt. Die folgenden optionalen Plug-in-Evictor verwenden verschiedene bekannte Algorithmen, um zu entscheiden, ob Einträge gelöscht werden müssen, wenn eine BackingMap-Instanz eine bestimmte Größenbegrenzung überschreitet.

- **LRUEvictor** ist ein Evictor, der einen LRU-Algorithmus (*Least Recently Used*, zuletzt verwendet) einsetzt, um zu entscheiden, welche Einträge gelöscht werden müssen, wenn die BackingMap-Instanz die maximale Anzahl von Einträgen überschreitet.
- **LFUEvictor** ist ein Evictor, der einen LFU-Algorithmus (*Least Frequently Used*, am seltensten verwendet) einsetzt, um zu entscheiden, welche Einträge gelöscht werden müssen, wenn die BackingMap-Instanz die maximale Anzahl von Einträgen überschreitet.

Die BackingMap-Instanz informiert einen Evictor, wenn Einträge in einer Transaktion erstellt, geändert oder entfernt werden. Die BackingMap-Instanz überwacht diese Einträge und entscheidet, wann Einträge gelöscht werden müssen.

Eine BackingMap-Instanz hat keine Konfigurationsdaten für eine maximale Größe. Stattdessen werden Merkmale für den Evictor definiert, die dessen Verhalten steuern. Sowohl LRUEvictor als auch LFUEvictor haben ein Merkmal für maximale Größe, das die Evictor dazu veranlasst, mit dem Löschen von Einträgen zu beginnen, sobald die maximal Größe überschritten wird. Wie der TTL-Evictor löschen die LRU- und LFU-Evictor einen Eintrag nicht sofort, wenn die maximale Anzahl von Einträgen erreicht ist, um die Auswirkungen auf die Leistung so gering wie möglich zu halten.

Sollten die LRU- und LFU-Bereinigungsalgorithmen für eine bestimmte Anwendung nicht geeignet sein, können Sie einen eigenen Evictor für die gewünschte Bereinigungsstrategie schreiben.

Plug-in-Evictor angeben

Da Evictor BackingMap-Instanzen zugeordnet werden, können Sie mit dem Interface BackingMap den zu verwendenden Plug-in-Evictor angeben. Das folgende Code-Snippet zeigt, wie ein LRUEvictor für die BackingMap-Instanz map1 und ein LFUEvictor für die BackingMap-Instanz map2 angegeben wird:

```
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);
```

Das vorherige Snippet zeigt, dass ein LRUEvictor für die BackingMap-Instanz map1 mit einer maximalen Anzahl von `1000` Einträgen verwendet wird. Für die BackingMap-Instanz map2 wird der LFUEvictor mit einer maximalen Anzahl von `2000` Einträgen verwendet. Beide Evictor haben ein Merkmal für Ruhezeit, das anzeigt, wie lange der Evictor ruht, bevor er aktiviert wird und prüft, ob Einträge gelöscht werden müssen. Die Ruhezeit wird in Sekunden angegeben. Ein Wert von 15 Sekunden ist ein angemessener Kompromiss. Er gewährleistet, dass die Auswirkungen auf die Leistung relativ gering bleiben, und verhindert, dass die BackingMap-Instanz zu groß wird. Das Ziel ist, die Ruhezeit so lange wie möglich auszudehnen, ohne dadurch zu bewirken, dass die BackingMap-Instanz zu groß wird.

Die Methode `setNumberOfLRUQueues` setzt das LRUEvictor-Merkmal, das angibt, wie viele LRU-Warteschlangen der Evictor für die Verwaltung der LRU-Informationen verwendet. Es wird eine Gruppe von Warteschlangen verwendet, damit die LRU-Informationen für die einzelnen Einträge nicht alle in derselben Warteschlange gespeichert werden. Dieses Konzept kann die Leistung verbessern, da die Anzahl der BackingMap-Einträge, die in einem Warteschlangenobjekt synchronisiert werden müssen, minimiert wird. Die Anzahl der Warteschlangen zu erhöhen, ist eine gute Methode, um die Auswirkungen, die der LRU-Evictor auf die Leistung haben kann, so gering wie möglich zu halten. Es wird empfohlen, zunächst zehn Prozent der maximalen Eintragsanzahl für die Anzahl der Warteschlangen zu veranschlagen. In der Regel eignen sich Primzahlen am besten.

Die Methode `setNumberOfHeaps` setzt das LFUEvictor-Merkmal, um die Anzahl der binären Heap-Speicherobjekte festzulegen, die der LFUEvictor für die Verwaltung der LFU-Informationen verwendet. Auch hier wird eine Gruppe verwendet, um die Leistung zu verbessern. Es wird empfohlen, zehn Prozent der maximalen Eintragsanzahl zu verwenden. In der Regel eignen sich Primzahlen am besten.

Plug-in-Evictor in einer XML-Datei angeben

Anstatt verschiedene APIs zu verwenden, um einen Evictor über das Programm zu implementieren, können Sie, wie im folgenden Beispiel gezeigt, jede BackingMap-Instanz in einer XML-Datei konfigurieren:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
 <objectGrids>
  <objectGrid name="grid">
   <backingMap name="map1" ttlEvictorType="NONE" pluginCollectionRef="LRU" />
   <backingMap name="map2" ttlEvictorType="NONE" pluginCollectionRef="LFU" />
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPlugincollection id="LRU">
   <bean id="Evictor"
      className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
    <property name="maxSize" type="int" value="1000"
         description="set max size for LRU evictor">
    <property name="sleepTime" type="int" value="15"
         description="evictor thread sleep time" />
    <property name="numberOfLRUQueues" type="int" value="53"
         description="set number of LRU queues" />
```

```
   </bean>
  </backingMapPluginCollection>
  <backingMapPluginCollection id="LFU">
   <bean id="Evictor"
      className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor">
    <property name="maxSize" type="int" value="2000"
         description="set max size for LFU evictor">
    <property name="sleepTime" type="int" value="15"
         description="evictor thread sleep time" />
    <property name="numberOfHeaps" type="int" value="211"
         description="set number of LFU heaps" />
   </bean>
  </backingMapPluginCollection>
 </backingMapPluginCollections>
</objectGridConfig>
```

Benutzerdefinierten Evictor schreiben

ObjectGrid kann mit weiteren Bereingigungsalgorithmen erweitert werden. Sie müssen einen eigenen Evictor schreiben, der das Interface `com.ibm.websphere.objectgrid.plugins.Evictor` implementiert. Das Interface hat den folgenden Inhalt:

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

- Die Methode `initialize` wird während der Initialisierung des BackingMap-Objekts aufgerufen. Diese Methode initialisiert ein Evictor-Plug-in mit einer Referenz auf die BackingMap-Instanz und einer Referenz auf ein Objekt, das das Interface `com.ibm.websphere.objectgrid.plugins.EvictionEventCallback` implementiert.
- Die Methode `apply` wird aufgerufen, wenn Transaktionen, die auf einen oder mehrere Einträge der BackingMap-Instanz zugreifen, festgeschrieben werden. An diese Methode wird eine Referenz auf ein Objekt übergeben, das das Interface com.ibm.websphere.objectgrid.plugins.LogSequence implementiert. Mit dem Interface LogSequence kann ein Evictor-Plug-in bestimmen, welche BackingMap-Einträge von der Transaktion erstellt, geändert oder entfernt wurden. Ein Evictor-Plug-in verwendet diese Informationen, um zu entscheiden, wann und welche Einträge gelöscht werden müssen.
- Die Methode `destroy` wird aufgerufen, wenn die BackingMap-Instanz gelöscht wird. Mit dieser Methode kann ein Evictor-Plug-in alle Threads beenden, die erstellt wurden.

Das Interface EvictionEventCallback enthält die folgenden Methoden:

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

Die `EvictionEventCallback`-Methoden werden folgendermaßen von einem Evictor-Plug-in verwendet, um Rückrufe (Callbacks) an das ObjectGrid-Framework abzusetzen:

- Mit der Methode `setEvictorData` fordert ein Evictor das verwendete Framework auf, ein erstelltes Evictor-Objekt zu speichern und dieses Objekt dem mit dem Schlüsselargument angegebenen Eintrag zuzuordnen. Die Daten sind Evictor-

spezifisch und werden durch die Informationen bestimmt, die der Evictor benötigt, um den verwendeten Algorithmus zu implementieren. Beispielsweise verwaltet der Evictor in einem LFU-Algorithmus einen Zähler im Evictor-Datenobjekt, um festzuhalten, wie oft die Methode `apply` für ein LogElement aufgerufen wird, das auf einen Eintrag für einen bestimmten Schlüssel verweist.

- Mit der Methode `getEvictorData` ruft der Evictor die Daten ab, die er während eines vorherigen Aufrufs der Methode `apply` an die Methode `setEvictorData` übergeben hat. Falls die Evictor-Daten für das angegebene Schlüsselargument nicht gefunden werden, wird ein spezielles Objekt `KEY_NOT_FOUND`, das im Interface EvictorCallback definiert ist, zurückgegeben.
- Mit der Methode `evictEntries` fordert ein Evictor die Bereinigung eines oder mehrerer BackingMap-Einträge an.

Hinweis: Gehen Sie sorgfältig vor, wenn Sie die Sperrstrategie für die BackingMap-Instanz auf OPTIMISTIC oder PESSIMISTIC setzen. Die Transaktion, die die Methode `apply` im Interface Evictor aufruft, erhält eine X-Sperre für jeden BackingMap-Eintrag, der eingefügt, aktualisiert, entfernt, berührt oder invalidiert wird. Die Implementierung der Methode `evictEntries` im Interface EvictionEventCallback verwendet ein eigenes ObjectGrid-Session-Objekt zum Starten einer weiteren Transaktion. Jeder Schlüssel in der Liste der Schlüssel, die an die Methode `evictEntries` übergeben werden, bewirkt, dass die Transaktion evictEntries eine X-Sperre für den zu bereinigenden BackingMap-Eintrag anfordert. Da die Transaktion `apply` bereits eine X-Sperre für den BackingMap-Eintrag besitzt, wird die Transaktion evictEntries so lange blockiert, bis die Transaktion apply abgeschlossen ist. Um ein gegenseitiges Sperren der Transaktion apply und der Transaktion evictEntries zu verhindern, dürfen Sie zum Aufrufen der Methode `evictEntries` nicht denselben Thread verwenden, der für den Aufruf der Methode `apply` verwendet wurde. Wenn Sie denselben Thread verwenden, wird eine Ausnahme des Typs `LockTimeoutException` an die Transaktion evictEntries gesendet, um die Situation, in der sich die beiden Transaktionen gegenseitig sperren, aufzulösen. Um sicherzustellen, dass die Methode `evictEntries` in einem anderen Thread als die die Methode `apply` aufgerufen wird, implementieren Sie die Methode `initialize` im Interface Evictor so, dass sie, wie in den Beispielen in diesem Abschnitt gezeigt, einen neuen Thread erzeugt. Der erzeugte Thread ist der einzige Thread, der die Methode `evictEntries` aufruft. Die Methode apply wird von diesem Thread nur mit einer LogSequence aufgerufen, die ein LogElement des Typs EVICT enthält. Das LogElement des Typs EVICT bewirkt, dass der Evictor keine Maßnahmen ergreift, weil dies nicht erforderlich ist.

Die Interfaces Evictor und EvictorEventCallback ermöglichen einer Anwendung, einen Evictor einzusetzen, der einen benutzerdefinierten Bereinigungsalgorithmus implementiert. Das folgende Code-Snippet zeigt, wie Sie die Methode `initialize` des Interface Evictor implementieren:

```
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;
// Instanzvariablen
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();
}
```

Der vorherige Code speichert die Referenzen auf die BackingMap- und Callback-Objekte in Instanzvariablen, um sie den Methoden `apply` und `destroy` zur Verfügung zu stellen. In diesem Beispiel wird eine verlinkte Liste erstellt, die als FIFO-Warteschlange (*First in, First out*) für die Implementierung eines LRU-Algorithmus verwendet wird. Es wird ein Thread erzeugt, und eine Referenz auf den Thread wird als Instanzvariable verwaltet. Weil diese Referenz verwaltet wird, kann die Methode `destroy` den erzeugten Thread unterbrechen und beenden.

Das folgende Code-Snippet zeigt, wie die Methode `apply` des Interface Evictor implementiert werden kann. Die Synchronisationsanforderungen, um den Code Thread-sicher zu machen, werden hier ignoriert:

```
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;
public void apply(LogSequence sequence)
{
 Iterator iter = sequence.getAllChanges();
 while ( iter.hasNext() )
 {
  LogElement elem = (LogElement)iter.next();
  Object key = elem.getCacheEntry().getKey();
  LogElement.Type type = elem.getType();
  if ( type == LogElement.INSERT )
  {
   // Einfügevearbeitung durch Hinzufügen am Anfang der LRU-Warteschlange hier vornehmen
   EvictorData data = new EvictorData(key);
   evictorCallback.setEvictorData(key, data);
   queue.addFirst( data );
  }
  else if ( type == LogElement.UPDATE || type == LogElement.FETCH ||
   type == LogElement.TOUCH )
  {
   // Aktualisierungsverarbeitung durch Verschieben des EvictorData-Objekts
   // an den Anfang der Warteschlange hier vornehmen
   EvictorData data = evictorCallback.getEvictorData(key);
   queue.remove(data);
   queue.addFirst(data);
  }
  else if ( type == LogElement.DELETE || type == LogElement.EVICT )
  {
   // Entfernungsverarbeitung durch Entfernen des EvictorData-Objekts aus der
   // Warteschlange hier vornehmen
   EvictorData data = evictorCallback.getEvictorData(key);
   if ( data == EvictionEventCallback.KEY_NOT_FOUND )
   {
    // Hier wird angenommen, dass der asynchrone Evictor-Thread
    // den BackingMap-Eintrag gelöscht hat, bevor der Thread
    // die Chance hatte, die LogElement-Anforderung zu verarbeiten.
    // Wahrscheinlich ist in diesem Fall nichts zu tun.
   }
```

```
   else
   {
    // Schlüssel wurde gefunden. Die Evictor-Daten können verarbeitet werden.
    if ( data != null )
    {
     // Von der Methode remove zurückgegebenen Nullwert ignorieren,
     // da der erzeugte Evictor-Thread den Eintrag möglicherweise schon
     // aus der Warteschlange entfernt hat.
     // Der Code wird jedoch für den Fall benötigt, dass nicht
     // der Evictor-Thread für dieses LogElement verantwortlich ist.
     queue.remove( data );
    }
    else
    {
     // Je nachdem, wie Sie den Evictor schreiben, ist diese Möglichkeit
     // unter Umständen nicht vorhanden, oder sie kann auf einen
     // Fehler im Evictor hinweisen, der auf eine ungültige
     // Synchronisationslogik für den Thread zurückzuführen ist.
    }
   }
  }
 }
}
```

Die Einfügeverarbeitung in der Methode `apply` ist in der Regel für die Erstellung eines Evictor-Datenobjekts zuständig, das an die Methode `setEvictorData` des Interface EvictionEventCallback übergeben wird. Da dieser Evictor eine LRU-Implementierung veranschaulicht, werden die Evictor-Daten ebenfalls am Anfang der Warteschlange hinzugefügt, die von der Methode `initialize` erstellt wurde. Die Aktualisierungsverarbeitung in der Methode `apply` aktualisiert normalerweise das Evictor-Datenobjekt, das von einem früheren Aufruf der Methode `apply` erstellt wurde (z. B. während der Einfügeverarbeitung der Methode `apply`). Da dieser Evictor eine LRU-Implementierung ist, muss er das EvictorData-Objekt von der aktuellen Warteschlangenposition an den Anfang der Warteschlange verschieben. Der erzeugte Evictor-Thread entfernt das letzte EvictorData-Objekt in der Warteschlange, weil das letzte Warteschlangenelement den zuletzt verwendeten Eintrag darstellt. Die Annahme ist die, dass das EvictorData-Objekt eine Methode `getKey` enthält, so dass der Evictor-Thread die Schlüssel der Einträge kennt, die bereinigt werden müssen. Beachten Sie, dass dieses Beispiel die Synchronisationsanforderungen bezüglich der Thread-Sicherheit des Codes ignoriert. Ein echter benutzerdefinierter Evictor ist komplizierter, weil er die Synchronisations- und Leistungsengpässe berücksichtigt, die in Folge der Synchronisationspunkte auftreten.

Die folgenden Code-Snippets zeigen die Methode `destroy` und die Methode `run` des ausführbaren Thread, den die Methode `initialize` erzeugt hat:

```
// Die Methode destroy unterbricht einfach den Thread, der
// von der Methode initialize erzeugt wurde.
public void destroy()
{
 evictorThread.interrupt();
}

// Hier ist die methode run des Thread, der von der Methode initialize erzeugt wurde.
public void run()
{
 // Schleife, bis die Methode destroy den Thread unterbricht.
 boolean continueToRun = true;
 while ( continueToRun )
 {
  try
  {
```

```
   // Ruhezeit, bis die Warteschlange untersucht wird.
   // Das Evictor-Merkmal sleepTime (Ruhezeit) ist ein
   // gutes Beispiel.
   Thread.sleep( sleepTime );
   int queueSize = queue.size();
   // Einträge bereinigen, wenn die Warteschlangengröße die
   // maximal zulässige Größe überschreitet. Die maximale
   // ist demnach ein weiteres Evictor-Merkmal.
   int numToEvict = queueSize - maxSize;
   if ( numToEvict > 0 )
   {
    // Eintrag am Ende der Warteschlange entfernen, da am
    // Ende der zuletzt verwendete Eintrag steht.
    List evictList = new ArrayList( numToEvict );
    while( queueSize > ivMaxSize )
    {
     EvictorData data = null;
     try
     {
      EvictorData data = (EvictorData) queue.removeLast();
      evictList.add( data.getKey() );
      queueSize = queue.size();
     }
     catch ( NoSuchElementException nse )
     {
      // Die Warteschlange ist leer.
      queueSize = 0;
     }
    }
    // Bereinigung anfordern, wenn die Schlüsselliste nicht leer ist.
    if ( ! evictList.isEmpty() )
    {
     evictorCallback.evictEntries( evictList );
    }
   }
  }
  catch ( InterruptedException e )
  {
   continueToRun = false;
  }
 } // while-Schleife beenden
} // Methode run beenden.
```

Loader

Ein ObjectGrid-Loader (oder Ladeprogramm) ist eine Plug-in-Komponente, die es einer ObjectGrid-Map-Instanz ermöglicht, als Speicher-Cache für Daten aufzutreten, die in der Regel in einem persistenten Speicher auf demselben oder einem anderen System verwaltet werden.

Normalerweise wird eine Datenbank oder ein Dateisystem als persistenter Speicher verwendet. Es kann auch eine ferne Java Virtual Machine (JVM) als Datenquelle verwendet werden, was die Erstellung Hub-basierter Caches mit ObjectGrid ermöglicht. Ein Loader besitzt die Logik zum Lesen und Schreiben von Daten in einem persistenten Speicher.

Ein Loader ist ein Plug-in für eine ObjectGrid-BackingMap-Instanz. Einer BackingMap-Instanz kann jeweils nur ein Loader zugeordnet werden. Jede BackingMap-Instanz hat eine eigene Loader-Instanz. Die BackingMap-Instanz fordert alle Daten, die sie selbst nicht enthält, von ihrem Loader an. Alle Änderungen, die an der Map-Instanz vorgenommen werden, werden in den Loader geschrieben. Das Loader-Plug-in ermöglicht der BackingMap-Instanz, Daten zwischen der Map und ihrem persistenten Speicher zu verschieben.

Loader als Plug-in implementieren

Das folgende Code-Snippet veranschaulicht, wie ein von der Anwendung bereitgestellter Loader mit der ObjectGrid-API als Plug-in für die BackingMap-Instanz map1 implementiert wird:

```
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 );
```

Es wird vorausgesetzt, dass `MyLoader` die von der Anwendung bereitgestellte Klasse ist, die das Interface com.ibm.websphere.objectgrid.plugins.Loader implementiert. Da die Zuordnung eines Loader zu einer BackingMap-Instanz nach der Initialisierung von ObjectGrid nicht mehr geändert werden kann, muss der Code vor dem Aufruf der Methode `initialize` des aufgerufenen Interface ObjectGrid ausgeführt werden. Es wird eine Ausnahme des Typs `IllegalStateException` in einer Methode `setLoader` ausgelöst, wenn diese Methode nach der Initialisierung aufgerufen wird.

Der von der Anwendung bereitgestellte Loader kann definierte Merkmale haben. In dem Beispiel wird der Loader MyLoader verwendet, um Daten in einer relationalen Datenbank zu lesen und zu schreiben. Der Loader muss den Namen der Datenbank haben und die zu verwendende SQL-Isolationsstufe kennen. Der Loader MyLoader hat die Methoden `setDataBaseName` und `setIsolationLevel`, mit denen die Anwendung diese beiden Loader-Merkmale definieren kann.

Ein von einer Anwendung bereitgestellter Loader kann auch mit einer XML-Datei als Plug-in implementiert werden. Das folgende Beispiel zeigt, wie der Loader MyLoader mit demselben Datenbanknamen und derselben Isolationsstufe in der BackingMap map1 implementiert wird:

```
<?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>
```

Interface Loader implementieren

Ein von einer Anwendung bereitgestellter Loader muss das Interface *com.ibm.websphere.objectgrid.plugins.Loader* implementieren. Das Interface Loader hat die folgende Definition:

```
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;
}
```

Die folgenden Abschnitte enthalten Beschreibungen und Hinweise zur Implementierung jeder einzelnen Methode des Interface Loader.

Methode get

Die BackingMap-Instanz ruft die Loader-Methode `get` auf, um die Werte für eine mit dem Argument **keyList** übergebenen Schlüsselliste abzurufen. Die Methode get muss eine Liste des Typs java.lang.util.List zurückgeben, die für jeden Schlüssel in der Schlüsselliste jeweils einen Wert enthält. Der erste in der Werteliste zurückgegebene Wert entspricht dem ersten Schlüssel in der Schlüsselliste, der zweite zurückgegebene Wert dem zweiten Schlüssel in der Schlüsselliste usw. Wenn der Loader den Wert für einen in der Schlüsselliste enthaltenen Schlüssel nicht findet, muss der Loader das Wertobjekt `KEY_NOT_FOUND` zurückgeben, das im Interface Loader definiert ist. Da eine BackingMap-Instanz so konfiguriert werden kann, dass sie `null` als gültigen Wert akzeptiert, ist es sehr wichtig, dass der Loader das spezielle Objekt `KEY_NOT_FOUND` zurückgibt, wenn der Schlüssel nicht gefunden wird. Anhand dieses Wertes kann die BackingMap-Instanz zwischen einem `Nullwert` und einem Wert unterscheiden, der nicht vorhanden ist, weil der Schlüssel nicht gefunden wurde. Wenn eine BackingMap-Instanz `Nullwerte` nicht unterstützt und ein Loader null anstelle des Objekts `KEY_NOT_FOUND` für einen nicht vorhandenen Schlüssel zurückgibt, wird eine Ausnahme ausgelöst.

Das Argument **forUpdate** teilt dem Loader mit, ob die Anwendung eine Methode `get` oder eine Methode `getForUpdate` für die BackingMap-Instanz aufgerufen hat. Nähere Informationen finden Sie im Abschnitt zum Interface com.ibm.websphere.objectgrid.ObjectMap. Der Loader ist für die Implementierung einer Policy für die Steuerung des gemeinsamen Zugriffs zuständig, die den gemeinsamen Zugriff auf den persistenten Speicher regelt. Zahlreiche Verwaltungssysteme für relationale Datenbanken unterstützen beispielsweise die Syntax von `for update` in der SQL-Anweisung "select", die zum Lesen von Daten aus einer relationalen Tabelle verwendet wird. Der Loader kann die Syntax von `for update` in der SQL-Anweisung `select` verwenden, wenn der Boolesche Wert `true` als Argumentwert für den Parameter **forUpdate** in dieser Methode übergeben wird. In der Regel verwendet der Loader die Syntax von "for update" nur, wenn eine pessimistische Policy für die Steuerung des gemeinsamen Zugriffs verwendet wird. Bei einer optimistischen Steuerung des gemeinsamen Zugriffs verwendet der Loader die Syntax von "for update" in der SQL-Anweisung "select" nicht. Der Loader muss basierend auf der verwendeten Policy für die Steuerung des gemeinsamen Zugriffs entscheiden, ob das Argument **forUpdate** verwendet wird.

Eine Erläuterung des Parameters **txid** finden Sie im Abschnitt „Plug-in TransactionCallback" auf Seite 119.

Methode `batchUpdate`

Die Methode `batchUpdate` im Interface Loader ist sehr wichtig. Diese Methode wird aufgerufen, wenn eine ObjectGrid-Instanz alle aktuellen Änderungen auf den Loader anwenden muss. Der Loader erhält eine Liste mit den Änderungen für diese Map-Instanz. Die Änderungen werden iteriert und auf das Backend angewendet. Die Methode empfängt den aktuellen TxID-Wert und die anzuwendenden Änderungen. Das folgende Beispiel iteriert durch die Änderungen und fasst drei JDBC-Anweisungen, `insert`, `update` und `delete`, zu einem Ausführungsstapel zusammen.

```
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
{
 // SQL-Verbindung abrufen
 Connection conn = getConnection(tx);
 try
 {
  // Die Liste der Änderungen verarbeiten und vorbereitete Anweisungen für
  // die Ausführung von SQL-Aktualisierungs-, Einfüge- oder Löschoperationen
  // im Stapelbetrieb erstellen.
  Iterator iter = sequence.getPendingChanges();
  while ( iter.hasNext() )
  {
   LogElement logElement = (LogElement)iter.next();
   Object key = logElement.getCacheEntry().getKey();
   Object value = logElement.getCurrentValue();
   switch ( logElement.getType().getCode() )
   {
    case LogElement.CODE_INSERT:
    buildBatchSQLInsert( tx, key, value, conn );
    break;
    case LogElement.CODE_UPDATE:
    buildBatchSQLUpdate( tx, key, value, conn );
    break;
    case LogElement.CODE_DELETE:
    buildBatchSQLDelete( tx, key, conn );
    break;
   }
  }
  // Die Stapelanweisungen ausführen, die mit der Schleife erstellt wurden
  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;
 }
}
```

Das obige Beispiel veranschaulicht die allgemeine Verarbeitungslogik des LogSequence-Arguments. Die Erstellung einer SQL-Anweisung `insert`, `update` oder `delete` wird jedoch nicht im Detail gezeigt. Einige der wichtigen Punkte, die veranschaulicht werden, sind:

- Die Methode `getPendingChanges` wird für das LogSequence-Argument aufgerufen, um einen Iterator für die Liste der LogElements abzurufen, die der Loader verarbeiten muss.
- Mit der Methode `LogElement.getType().getCode()` wird festgestellt, ob das LogElement für eine SQL-Operation `insert`, `update` oder `delete` bestimmt ist.
- Es wird eine Ausnahme des Typs `SQLException` abgefangen und mit einer Ausnahme des Typs `LoaderException` verkettet, die berichtet, dass während der Stapelaktualisierung eine Ausnahme eingetreten ist.
- Die JDBC-Unterstützung für Stapelaktualisierungen wird verwendet, um die Anzahl der Abfragen zu minimieren, die das Backend absetzen muss.

Methode `preloadMap`

Während der Initialisierung von ObjectGrid wird jede definierte BackingMap-Instanz initialisiert. Wenn einer BackingMap-Instanz ein Loader-Plug-in besitzt, ruft die BackingMap-Instanz die Methode `preloadMap` im Interface Loader auf, um dem Loader zu ermöglichen, die Daten vorab aus dem Backend abzurufen und in die BackingMap-Instanz zu laden. Das folgende Beispiel setzt voraus, dass die ersten 100 Zeilen einer Tabelle mit dem Namen Employee aus der Datenbank gelesen und in die Map-Instanz geladen werden. Die Klasse EmployeeRecord ist eine von der Anwendung bereitgestellte Klasse, die die aus der Tabelle Employee gelesenen Mitarbeiterdaten enthält.

```
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
 boolean tranActive = false;
 ResultSet results = null;
 Statement stmt = null;
 Connection conn = null;
 try
 {
  session.beginNoWriteThrough();
  tranActive = true;
  ObjectMap map = session.getMap( backingMap.getName() );
  TxID tx = session.getTxID();
  // Verbindung mit automatischer Festschreibung abrufen, die
  // auf die Isolationsstufe 'read committed' (Lesen
  // mit COMMIT) gesetzt wird.
  conn = getAutoCommitConnection(tx);
  // Map-Instanz Employee vorher mit EmployeeRecord-Objekten
  // laden. Alle Employee-Einträge aus der Tabelle lesen,
  // aber das vorherige Laden auf die ersten 100 Zeilen
  // beschränken.
  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;
  }
  // Transaktion festschreiben
  session.commit();
  tranActive = false;
 }
 catch (Throwable t)
 {
  throw new LoaderException("preload failure: " + t, t);
 }
 finally
 {
  if ( tranActive )
  {
   try
   {
    session.rollback();
   }
   catch ( Throwable t2 )
   {
    // Rollback-Fehler tolerieren und Auslösen des
    // ursprünglichen Throwable zulassen
   }
  }
  // Hier auch andere Datenbankressourcen bereinigen, z. B.
  // Anweisungen, Dateien usw. schließen.
 }
}
```

Dieses Beispiel veranschaulicht Folgendes:

- Die BackingMap-Instanz preloadMap verwendet das Session-Objekt, das ihr als Sitzungsargument übergeben wurde.
- Die Transaktion `Session.beginNoWriteThrough()` wird anstelle der Methode `begin` zum Starten der Transaktion verwendet. Zum Laden der Map kann der Loader nicht für jede Operation `put` aufgerufen werden, die in dieser Methode enthalten ist.
- Der Loader kann Spalten der Tabelle Employee einem Feld im Java-Objekt EmployeeRecord zuordnen.
- Der Loader fängt alle Throwable-Ausnahmen ab und löst eine Ausnahme des Typs `LoaderException` mit der verketteten abgefangenen Throwable-Ausnahme aus.
- Der Block `finally` stellt sicher, dass alle Throwable-Ausnahmen, die zwischen dem Aufruf der Methode `beginNoWriteThrough` und dem Aufruf der Methode `commit` eintreten, bewirken, dass der Block `finally` die aktive Transaktion rückgängig macht. Diese ist wichtig, damit gewährleistet ist, dass jede Transaktion, die von der Methode `preloadMap` gestartet wurde, abgeschlossen ist, bevor sie zum Aufrufenden zurückkehrt. Der Block `finally` eignet sich auch für die Durchführung weiterer Bereinigungsaktionen, die möglicherweise erforderlich sind, wie z. B. das Schließend der JDBC-Verbindung und anderer JDBC-Objekte.

Das Beispiel `preloadMap` verwendet eine SQL-Anweisung "select", die alle Zeilen der Tabelle auswählt. In dem von Ihrer Anwendung bereitgestellten

Loader müssen Sie möglicherweise Loader-Merkmale definieren, um zu steuern, welcher Anteil der Tabelle vorher in die Map-Instanz geladen werden muss.

Da die Methode `preloadMap` nur einmal während der Initialisierung der BackingMap-Instanz aufgerufen wird, eignet sie sich auch für die Ausführung des Initialisierungscodes für den Loader. Selbst wenn ein Loader sich dagegen entscheidet, Daten vorab aus dem Backend abzurufen und in die Map-Instanz zu laden, muss er wahrscheinlich andere einmalige Initialisierungsoperationen durchführen, um die Effizienz anderer Loader-Methoden zu gewährleisten. Das folgende Beispiel zeigt, wie das TransactionCallback-Objekt und das OptimisticCallback-Objekt als Instanzvariablen des Loader zwischengespeichert werden, damit die anderen Methoden des Loader keine Methodenaufrufe absetzen müssen, um Zugriff auf diese Objekte zu erhalten. Die Werte für die ObjectGrid-Plug-ins können zwischengespeichert werden, da die TransactionCallback- und OptimisticCallback-Objekte nach der Initialisierung der BackingMap-Instanz nicht mehr geändert oder ersetzt werden können. Deshalb ist es möglich, diese Objektreferenzen als Instanzvariablen des Loader zwischenzuspeichern.

```
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;

// Instanzvariablen für Loader
MyTransactionCallback ivTcb; // MyTransactionCallback

// Erweitert TransactionCallback
MyOptimisticCallback ivOcb; // MyOptimisticCallback
// Implementiert OptimisticCallback
...
public void preloadMap(Session session, BackingMap backingMap)
 throws LoaderException
{
 // TransactionCallback- und OptimisticCallback-Objekte
 // in Instanzvariablen dieses Loader zwischenspeichern
 ivTcb = (MyTransactionCallback)
 session.getObjectGrid().getTransactionCallback();
 ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
 // Der Rest des preloadMap-Codes (wie im vorherigen Beispiel gezeigt).
}
```

Zugehörige Verweise

Hinweise zum Loader
Beachten Sie beim Implementieren eines Loader die folgenden Hinweise.

Hinweise zum Loader

Beachten Sie beim Implementieren eines Loader die folgenden Hinweise.

Hinweise zum vorherigen Laden

Jede BackingMap hat ein Boolesches Attribut preloadMode, mit dem angezeigt werden kann, ob das vorherige Laden von Daten in eine Map asynchron durchgeführt wird. Standardmäßig ist das Attribut preloadMode auf `false` gesetzt. Dieser Wert gibt an, dass die Initialisierung der BackingMap erst dann abgeschlossen wird, wenn das vorherige Laden der Daten in die Map abgeschlossen ist. Die Initialisierung der BackingMap wird beispielsweise nicht vor der Rückkehr der Methode `preloadMap` abgeschlossen. Wenn die Methode `preloadMap` ein großes Datenvolumen aus dem Backend lesen und in die Map laden soll, kann es relativ

lang dauern, bis die Initialisierung abgeschlossen wird. In diesem Fall können Sie eine BackingMap so konfigurieren, dass die Daten asynchron in die Map geladen werden. Hierfür müssen Sie das Attribut preloadMode auf true setzen. Diese Einstellung bewirkt, dass der Initialisierungscode für die BackingMap einen Thread erzeugt, der die Methode `preloadMap` aufruft, und lässt damit zu, dass die Initialisierung einer BackingMap abgeschlossen werden kann, während das vorherige Laden der Map noch aktiv ist.

Das folgende Code-Snippet veranschaulicht, wie das Attribut preloadMode so gesetzt wird, dass ein asynchrones vorheriges Laden ermöglicht:

```
BackingMap bm = og.defineMap( "map1" );
bm.setPreloadMode( true );
```

Das Attribut preloadMode kann, wie im folgenden Beispiel gezeigt, auch mit einer XML-Datei gesetzt werden:

```
<backingMap name="map1" preloadMode="true"
 pluginCollectionRef="map1" lockStrategy="OPTIMISTIC" />
```

Wenn der asynchrone Modus für das vorherige Laden verwendet wird, müssen die folgenden Punkte in der Implementierung der Methode `preloadMap` berücksichtigt werden:

- Eine Anwendung kann einen Map-Eintrag, der mit der Transaktion preloadMap noch geladen werden muss, abrufen, in die Map stellen und festschreiben. In einer solchen Situation kann die Transaktion preloadMap fehlschlagen und eine Ausnahme anzeigen, wenn die Transaktion versucht, die COMMIT-Operation durchzuführen. Wenn Sie beispielsweise eine optimistische Sperrstrategie für die Map verwenden, kann die Transaktion preloadMap wegen der Ausnahme OptimisticCollisionException fehlschlagen, die ausgelöst wird, wenn die COMMIT-Operation feststellt, dass die BackingMap bereits einen Map-Eintrag für den Schlüssel enthält, und das Versionsobjekt für den Map-Eintrag anzeigt, dass dass die Transaktion preloadMap mit einer veralteten Kopie der Daten arbeitet.

  Die Anwendung und der Loader müssen auf diese Eventualität vorbereitet sein. Zur Behebung dieses Problems kann die Anwendung ein MapEventListener- und ein Loader-Plug-in bereitstellen. Die Anwendung kann das MapEventListener-Plug-in verwenden, um den Abschluss des vorherigen Ladens abzuwarten, bevor Transaktionen gestartet werden, die Map-Einträge aktualisieren.

  Transaktionen mit Lesezugriff kann die Anwendung fortsetzen, ohne auf den Abschluss der vorherigen Ladens zu warten. Da die Anwendung die Plug-in-Interfaces MapEventListener und Loader implementiert, kann sie zwischen der Komplexität der Lösung und der Leistung abwägen. Mit zusätzlicher Komplexität kann ein Loader, der Daten vorher in eine partitionierte Map lädt, den Status der Vorabladevorgänge für die Partitionen überwachen. Auf dieser Basis dieser zusätzlichen Informationen können Sie MapEventListener- und Loader-Plug-ins entwerfen, die Transaktionen nur für solche Partitionen blockieren, für die das vorherige Laden noch nicht abgeschlossen ist. Der Abschnitt Kapitel 4, „ObjectGrid-Beispielanwendungen”, auf Seite 33 zeigt anhand eines Beispiels, wie eine Anwendung ein Plug-in MapEventListener bereitstellen kann, das das Interface MapEventListener so erweitert, dass Thread bis zum Abschluss des vorherigen Ladens blockiert werden.
- Sie können zum Laden großer Datenvolumen mehrere Threads verwenden. Verwenden Sie die Methode `session.getObjectGrid()`, um eine Referenz auf die ObjectGrid-Instanz abzurufen. Anschließend können Sie mit der Methode `getSession` in der ObjectGrid-Instanz weitere Sitzungen abrufen. Jeder Thread muss möglicherweise ein eigenes Session-Objekt haben, da einem Session-Objekt jeweils nur eine Transaktion zugeordnet werden kann. Jeder Thread liest *n* Zei-

len aus einer relationalen Tabelle, sofern dieses Backend eine relationale Datenbank ist. Der Loader muss eine Möglichkeit haben, die Anforderungen auf die Threads zu verteilen, um die erforderliche Zeit für das vorherige Laden einer Map zu reduzieren.

TxID und Verwendung des Interface TransactionCallback

Die Methoden `get` und `batchUpdate` im Interface Loader werden an ein TxID-Objekt übergeben, das die Session-Transaktion darstellt, die die Durchführung der Operation `get` bzw. `batchUpdate` erfordert. Es ist möglich, dass die Methoden `get` und `batchUpdate` in einer Transaktion mehrfach aufgerufen werden. Deshalb werden transaktionsbezogene Objekte, die der Loader benötigt, normalerweise in einem Slot des TxID-Objekts verwaltet. Es wird ein JDBC-Loader (Java Database Connectivity) verwendet, um zu zeigen, wie ein Loader die Interfaces TxID und TransactionCallback nutzt.

Möglich ist auch, mehrere ObjectGrid-Maps in derselben Datenbank zu speichern. Jede Map hat einen eigenen Loader, und jeder Loader muss möglicherweise eine Verbindung zu derselben Datenbank herstellen. Wenn die Loader dieselbe Datenbank verwenden, möchte jeder Loader dieselbe JDBC-Verbindung verwenden, damit die Änderungen, die an den Tabellen vorgenommen werden, im Rahmen derselben Datenbanktransaktion festgeschrieben werden. Normalerweise schreibt die Person, die die Loader-Implementierung erstellt, auch die TransactionCallback-Implementierung. Bei der Erweiterung des Interface TransactionCallback empfiehlt es sich, Methoden hinzuzufügen, die der Loader benötigt, um eine Datenbankverbindung anzufordern und um ausführbare Anweisungen zwischenzuspeichern. Der Grund für diese Vorgehensweise wird deutlich, wenn Sie sich anschauen, wie die Interfaces TransactionCallback und TxID vom Loader verwendet werden.

Der Loader könnte beispielsweise die folgende Erweiterung des Interface TransactionCallback erfordern:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
public interface MyTransactionCallback extends TransactionCallback
{
 Connection getAutoCommitConnection(TxID tx, String databaseName)
 throws SQLException;
 Connection getConnection(TxID tx, String databaseName,
 int isolationLevel ) throws SQLException;
 PreparedStatement getPreparedStatement(TxID tx, Connection conn,
 String tableName, String sql) throws SQLException;
 Collection getPreparedStatementCollection( TxID tx, Connection conn,
 String tableName );
}
```

Mit diesen neuen Methoden können die Loader-Methoden `get` und `batchUpdate` wie folgt eine Verbindung abrufen:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getConnection(TxID tx, int isolationLevel)
{
 Connection conn = ivTcb.getConnection(tx, databaseName, isolationLevel );
 return conn;
}
```

Im vorherigen Beispiel und den folgenden Beispielen sind *ivTcb* und *ivOcb* Instanzvariablen für den Loader, die so initialisiert wurde, wie es der Abschnitt „Hinweise zum vorherigen Laden" auf Seite 108 beschreibt. Die Variable *ivTcb* ist eine Referenz auf die MyTransactionCallback-Instanz und *ivOcb* eine Referenz auf die MyOptimisticCallback-Instanz. Die Variable *databaseName* ist eine Instanzvariable des Loader, die während der Initialisierung der BackingMap als Loader-Merkmal gesetzt wurde. Das Argument `isolationLevel` ist eine der JDBC-Verbindungskonstanten, die für die verschiedenen, von JDBC unterstützten Isolationsstufen definiert werden. Wenn der Loader eine optimistische Implementierung verwendet, ruft die Methode `get` die Daten aus der Datenbank normalerweise in einer JDBC-Verbindung mit automatischer Festschreibung ab. In diesem Fall kann der Loader einer Methode `getAutoCommitConnection` haben, die wie folgt implementiert ist:

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
private Connection getAutoCommitConnection(TxID tx)
{
 Connection conn = ivTcb.getAutoCommitConnection(tx, databaseName);
 return conn;
}
```

Sie erinnern sich, dass die Methode `batchUpdate` die folgende Anweisung switch enthält:

```
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;
}
```

Jede der `buildBatchSQL`-Methoden verwendet das Interface MyTransactionCallback, um eine ausführbare Anweisung abzurufen. Das folgende Code-Snippet zeigt, dass die Methode `buildBatchSQLUpdate` eine SQL-Anweisung "update" erstellt, die einen EmployeeRecord-Eintrag aktualisiert und diesen für die Stapelaktualisierung hinzufügt:

```
private void buildBatchSQLUpdate( TxID tx, Object key, Object value, Connection
 conn )
throws SQLException, LoaderException
{
 String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?, DEPTNO = ?,
 SEQNO = ?, MGRNO = ? where EMPNO = ?";
 PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn, "employee",
 sql );
 EmployeeRecord emp = (EmployeeRecord) value;
 sqlUpdate.setString(1, emp.getLastName());
 sqlUpdate.setString(2, emp.getFirstName());
 sqlUpdate.setString(3, emp.getDepartmentName());
 sqlUpdate.setLong(4, emp.getSequenceNumber());
 sqlUpdate.setInt(5, emp.getManagerNumber());
 sqlUpdate.setInt(6, key);
 sqlUpdate.addBatch();
}
```

Nachdem die batchUpdate-Schleife alle ausführbaren Anweisungen erstellt hat, ruft sie die Methode getPreparedStatementCollection auf. Diese Methode kann wie folgt implementiert werden:

```
private Collection getPreparedStatementCollection( TxID tx, Connection conn )
{
 return ( ivTcb.getPreparedStatementCollection( tx, conn, "employee" ) );
}
```

Wenn die Anwendung die Methode commit für das Session-Objekt aufruft, ruft der Session-Code die Methode commit in der Methode TransactionCallback auf, nachdem sie alle von der Transaktion vorgenommenen Änderungen in den Loader jeder Map, die von der Transaktion geändert wurde, geschrieben hat. Da alle Loader die Methode MyTransactionCallback verwenden, um eine Verbindung und die erforderlichen ausführbaren Anweisungen abzurufen, weiß die Methode TransactionCallback, welche Verbindung sie verwenden muss, um das Festschreiben der Änderungen beim Backend anzufordern. Das Erweitern des Interface TransactionCallback um die von den einzelnen Loadern benötigten Methoden hat somit die folgenden Vorteile:

- Das TransactionCallback-Objekt kapselt die Nutzung der TxID-Slots für transaktionsbezogene Daten, und der Loader benötigt keine Informationen zu den TxID-Slots. Der Loader muss lediglich die Methoden kennen, die mit dem Interface MyTransactionCallback für die vom Loader benötigten Funktionen dem TransactionCallback-Objekt hinzugefügt werden.
- Das TransactionCallback-Objekt kann sicherstellen, dass alle Loader, die eine Verbindung zu demselben Backend herstellen, dieselbe Verbindung nutzen, damit ein zweiphasiges Festschreibprotokoll vermieden werden kann.
- Das TransactionCallback-Objekt kann sicherstellen, dass die Verbindung zum Backend abgeschlossen wird, indem bei Bedarf eine COMMIT- oder ROLLBACK-Operationen in der Verbindung aufgerufen wird.
- TransactionCallback kann sicherstellen, dass beim Abschluss einer Transaktion die Bereinigung der Datenbankressourcen durchgeführt wird.
- Das TransactionCallback-Objekt kann verbergen, ob es eine verwaltete Verbindung von einer verwalteten Umgebung wie WebSphere Application Server oder von einem anderen J2EE-kompatiblen Anwendungsserver abruft. Dies hat den Vorteil, dass derselbe Loader für verwaltete und nicht verwaltete Umgebungen verwendet werden kann. Es muss nur das TransactionCallback-Plug-in geändert werden.

Der Abschnitt „Plug-in TransactionCallback" auf Seite 119 beschreibt ausführlich, wie die TransactionCallback-Implementierung die TxID-Slots für transaktionsbezogene Daten verwendet.

OptimisticCallback

Wie bereits erwähnt, kann der Loader entscheiden, eine optimistische Strategie für die Steuerung des gemeinsamen Zugriffs zu verwenden. In diesem Fall muss das Beispiel für die Methode buildBatchSQLUpdate geringfügig geändert werden, um die optimistische Strategie zu implementieren. Eine optimistische Strategie kann auf verschiedene Arten verwendet werden. Eine typische Methode ist, eine Zeitmarkenspalte oder eine Zählerspalte für die Folgenummer zu verwenden, um die einzelnen Aktualisierungen der Spalte als Versionen zu verwalten. Angenommen, die Tabelle "employee" hat eine Spalte für die Folgenummer, die sich jedesmal um eins erhöht, wenn die Zeile aktualisiert wird.

In diesem Fall ändern Sie die Deklaration der Methode buildBatchSQLUpdate so, dass ihr anstelle des Schlüssel/Wert-Paars das Objekt LogElement übergeben wird. Außerdem muss die Methode das Objekt OptimisticCallback verwenden, das als Plug-in in der BackingMap implementiert ist, um das Ausgangsversionsobjekt abzurufen und das Versionsobjekt zu aktualisieren. Das folgende Beispiel zeigt eine geänderte Methode buildBatchSQLUpdate, die die Instanzvariable *ivOcb* verwendet, die wie im Abschnitt zu preloadMap initialisiert wurde:

```
private void buildBatchSQLUpdate( TxID tx, LogElement le,
 Connection conn )throws SQLException, LoaderException
{
 // Ausgangsversionsobjekt abrufen, wenn dieser Map-Eintrag als letzter in der Datenbank
 // gelesen oder aktualisiert wurde.
 Employee emp = (Employee) le.getCurrentValue();
 long initialVersion = ((Long) le.getVersionedValue()).longValue();
 // Von OptimisticCallback das Employee-Objekt mit dem neuen Versionswert
 // aktualisieren lassen.
 ivOcb.updateVersionedObjectForValue( emp );
 // Versionsobjekt für die SQL-Operaton update aus dem aktualisierten
 // employee-Objekt abrufen
 Long currentVersion = (Long)ivOcb.getVersionedObjectForValue( emp );
 long nextVersion = currentVersion.longValue();
 // LogElement mit dem aktualisierten Versionsobjekt aktualisieren
 le.setVersionedValue( currentVersion );
 // Jetzt die SQL-Anweisung update erstellen, die das Versionsobjekt
 // in die where-Klausel für die optimistische Prüfung einfügt
 String sql = "update EMPLOYEE set LASTNAME = ?, FIRSTNAME = ?,
 DEPTNO = ?,SEQNO = ?, MGRNO = ? where EMPNO = ? and SEQNO = ?";
 PreparedStatement sqlUpdate = ivTcb.getPreparedStatement( tx, conn,
 "employee", sql );
 sqlUpdate.setString(1, emp.getLastName());
 sqlUpdate.setString(2, emp.getFirstName());
 sqlUpdate.setString(3, emp.getDepartmentName());
 sqlUpdate.setLong(4, nextVersion );
 sqlUpdate.setInt(5, emp.getManagerNumber());
 sqlUpdate.setInt(6, key);
 sqlUpdate.setLong(7, initialVersion);
 sqlUpdate.addBatch();
}
```

Das Beispiel zeigt, dass das LogElement verwendet wird, um den Ausgangsversionswert abzurufen. Wenn die Transaktion zum ersten Mal auf den Map-Eintrag zugreift, wird ein LogElement mit dem employee-Ausgangsobjekt erstellt, das aus der Map abgerufen wurde. Das employee-Objekt wird außerdem an die Methode getVersionedObjectForValue im Interface OptimisticCallback übergeben, und das Ergebnis wird im LogElement gespeichert. Diese Verarbeitung findet statt, bevor eine Anwendung eine Referenz auf das Employee-Objekt empfängt und die Gelegenheit erhält, eine Methode für das Objekt aufzurufen, das den Status des Employee-Objekts ändert.

Wenn die Methode batchUpdate im Interface Loader aufgerufen wird und der LogElement-Typ anzeigt, dass der Map-Eintrag von der Anwendung aktualisiert wurde, ruft der Code die Methode updateVersionedObjectForValue im Interface OptimisticCallback auf, damit das neue Versionsobjekt generiert wird. Das LogElement wird mit diesem neuen Versionsobjekt aktualisiert, indem der Loader die Methode setVersionedObject für das LogElement aufruft. Dieser Schritt ist erforderlich, weil die Anwendung die Methode flush anstelle der Methode commit im Interface Session aufrufen könnte. Der Loader kann in einer Transaktion mehrfach für denselben Schlüssel aufgerufen werden. Das LogElement muss jedesmal, wenn die Zeile in der Tabelle employee aktualisiert wird, mit dem neuen Versionsobjekt aktualisiert werden.

Sobald der Loader das Ausgangsversionsobjekt und das Folgeversionsobjekt hat, kann er eine SQL-Anweisung "update" ausführen, die die Spalte SEQNO auf den Wert des Folgeversionsobjekts setzt und den Wert des Ausgangsversionsobjekts in der `where`-Klausel verwendet. Diese Methode wird manchmal auch als überqualifizierte `update`-Anweisung bezeichnet. Die Verwendung der überqualifizierten `update`-Anweisung ermöglicht der relationalen Datenbank zu prüfen, ob die Zeile zwischen dem Lesen der Daten aus der Datenbank und der Aktualisierung der Datenbank durch die Transaktion von einer anderen Transaktion geändert wurde. Wenn die Zeile von einer anderen Transaktion geändert wurde, zeigt das Zählerfeld, das von der Stapelaktualisierung (batch update) zurückgegeben wird, an, dass keine Zeilen für diesen Schlüssel aktualisiert wurden. Der Loader muss sicherstellen, dass die Zeile nicht von der SQL-Operation `update` aktualisiert wurde. Falls die Zeile geändert wurde, zeigt der Loader eine Ausnahme des Typs `com.ibm.websphere.objectgrid.plugins.OptimisticCollisionException` an, um das Session-Objekt darüber zu informieren, dass die Methode `batchUpdate` fehlgeschlagen ist, weil mehrere Transaktionen gleichzeitig versucht haben, dieselbe Zeile in der Datenbanktabelle zu aktualisieren. Diese Ausnahme leitet eine ROLLBACK-Operation für das Session-Objekt ein, und die Anwendung muss die gesamte Transaktion wiederholen. Die Annahme ist, dass eine Wiederholung erfolgreich verläuft. Deshalb wird die Strategie als optimistisch bezeichnet. Die optimistische Strategie bietet eine bessere Leistung, wenn die Daten nur selten geändert und gleichzeitig ausgeführte Transaktionen nur selten versuchen, dieselbe Zeile zu aktualisieren.

Im Folgenden sind weitere Möglichkeiten beschrieben, wie ein Loader eine optimistische Strategie implementieren kann:

- Es ist keine Zeitmarken- oder Folgenummernspalte vorhanden. In diesem Fall gibt die Methode `getVersionObjectForValue` im Interface OptimisticCallback einfach das Wertobjekt selbst als Version zurück. Bei dieser Strategie muss der Loader eine `where`-Klausel erstellen, die jedes der Felder im Ausgangsversionsobjekt enthält. Diese Strategie ist nicht sehr effizient, und nicht alle Spaltentypen eignen sich zur Verwendung in der `where`-Klausel einer überqualifizierten SQL-Anweisung "update". Deshalb wird diese Strategie in der Regel nicht verwendet.
- Es ist keine Zeitmarken- oder Folgenummernspalte vorhanden. Aber anders als bei der zuvor beschriebenen Strategie enthält die `where`-Klausel nur die Wertfelder, die von der Transaktion geändert wurden. Eine Möglichkeit festzustellen, welche Felder geändert werden, besteht darin, den Kopiermodus für die BackingMap auf `CopyMode.COPY_ON_WRITE` einzustellen. Dieser Kopiermodus erfordert, dass ein Value-Interface an die Methode `setCopyMode` im Interface BackingMap übergeben wird. Die BackingMap erstellt dynamische Proxy-Objekte, die das angegebene value-Interface implementieren. In diesem Kopiermodus kann der Loader jeden Wert in ein com.ibm.websphere.objectgrid.plugins.ValueProxyInfo-Objekt umsetzen. Das Interface ValueProxyInfo hat eine Methode, mit der der Loader die Liste der Attribute abrufen kann, die von der Transaktion geändert wurden. Diese Methode ermöglicht dem Loader, die Methode `get` im Interface value für die Attribute aufzurufen, um die geänderten Daten abzurufen und eine SQL-Anweisung "update" zu erstellen, die nur die geänderten Attribute setzt. Jetzt kann die `where`-Klausel mit der Primärschlüsselspalte und den Spalten der geänderten Attribute erstellt werden. Diese Strategie ist wesentlich effizienter als die zuvor beschriebene Strategie. Allerdings muss für diese Strategie mehr Code im Loader geschrieben und für die Verarbeitung der unterschiedlichen Permutationen unter Umständen ein größerer Cache für ausführbare Anweisungen verwendet werden. Wenn die Transaktionen jedoch in der Regel nur wenige Attribute ändern, sollte diese Einschränkung kein Problem darstellen.

- Einige relationale Datenbanken besitzen möglicherweise eine API, die Sie bei der automatischen Verwaltung der Spaltendaten unterstützt, was hilfreich für eine optimistische Versionssteuerung ist. Lesen Sie in Ihrer Datenbankdokumentation nach, ob diese Möglichkeit gegeben ist.

Plug-in ObjectTransformer

Verwenden Sie das Plug-in ObjectTransformer, wenn Sie eine hohe Leistung benötigen. Wenn Sie Leistungsprobleme bei der CPU-Belastung sehen, fügen Sie jeder Map ein ObjectTransformer-Plug-in hinzu. Falls Sie kein ObjectTransformer-Plug-in hinzufügen, werden 60 - 70 % der Gesamt-CPU-Zeit mit dem Serialisieren und Kopieren von Einträgen verbracht.

Zweck

Das Plug-in ObjectTransformer ermöglicht Anwendungen, benutzerdefinierte Methoden für die folgenden Operationen bereitzustellen:

- Schlüssel für einen Eintrag serialisieren oder entserialisieren
- Wert für einen Eintrag serialisieren und entserialisieren
- Schlüssel oder Wert für einen Eintrag kopieren

Wenn Sie kein ObjectTransformer-Plug-in bereitstellen, müssen Sie in der Lage sein, die Schlüssel und Werte zu serialisieren und zu entserialisieren, da ObjectGrid für das Kopieren der Objekte eine Serialisierungs-/Entserialisierungssequenz verwendet. Diese Methode ist kostenintensiv. Verwenden Sie deshalb ein ObjectTransformer-Plug-in, wenn die Leistung ein kritischer Faktor ist. Der Kopiervorgang findet statt, wenn eine Anwendung ein Objekt zum ersten Mal in einer Transaktion sucht. Sie können diesen Kopiervorgang verhindern, indem Sie den Kopiermodus der Map auf `NO_COPY` einstellen. Durch Einstellen des Kopiermodus `COPY_ON_READ` können Sie den Kopieraufwand reduzieren. Optimieren Sie die Kopieroperationen, wenn die Anwendung dies erfordert, indem Sie eine eigene Kopiermethode in diesem Plug-in bereitstellen. Ein solches Plug-in kann die Kopierkosten von 65 – 70 % auf 2/3 % der Gesamt-CPU-Zeit verringern.

Die Objektserialisierung wird auch direkt verwendet, wenn die ObjectGrid-Instanz im verteilten Modus ausgeführt wird. Die LogSequence verwendet das Plug-in ObjectTransformer als Unterstützung bei der Serialisierung von Schlüsseln und Werten, bevor die Änderungen an die Peers in der ObjectGrid-Instanz übertragen werden. Sie müssen vorsichtig vorgehen, wenn Sie eine eigene Serialisierungsmethode anstelle der integrierten JDK-Serialisierung verwenden. Die Versionssteuerung von Objekten ist ein komplexer Vorgang, und es können Probleme bei der Versionskompatibilität auftreten, wenn Sie nicht sicherstellen, dass Ihre eigenen Methoden für die Versionssteuerung geeignet sind.

Die folgende Liste beschreibt ausführlich, wie die ObjectGrid-Instanz versucht, Schlüssel und Werte zu serialisieren:

- Wenn Sie ein eigenes ObjectTransformer-Plug-in schreiben und implementieren, ruft ObjectGrid Methoden in den ObjectTransformer-Methoden auf, um Schlüssel und Werte zu serialisieren und Kopien von Objektschlüsseln und -werten abzurufen.
- Wenn Sie kein eigenes ObjectTransformer-Plug-in verwenden, serialisiert und entserialisiert ObjectGrid die Schlüssel und Werte mit dem Standardverfahren. Beim Standardverfahren wird jedes Objekt als Externalizable- oder Serializable-Objekt implementiert.

- Wenn das Objekt das Interface Externalizable unterstützt, wird die Methode `writeExternal` aufgerufen. Mit Objekten, die als Externalizable-Objekt implementiert werden, können Sie eine bessere Leistung erzielen.
- Wenn das Objekt das Interface Externalizable nicht unterstützt und das Interface Serializable implementiert, wird das Objekt mit der Methode `ObjectOutputStream` gespeichert.

Interface ObjectTransformer

Nähere Informationen zum Interface ObjectTransformer finden Sie in der API-Dokumentation. Das Interface ObjectTransformer enthält die folgenden Methoden, die Schlüssel oder Werte serialisieren und entserialisieren oder Schlüssel und Werte kopieren:

```
public interface ObjectTransformer
{
 void serializeKey(Object key, ObjectOutputStream stream)
  throws IOException;
 void serializeValue(Object value, ObjectOutputStream stream)
  throws IOException;
 Object inflateKey(ObjectInputStream stream)
  throws IOException, ClassNotFoundException;
 Object inflateValue(ObjectInputStream stream)
  throws IOException, ClassNotFoundException;
 Object copyKey(Object value);
 Object copyValue(Object value);
}
```

Verwendung des Interface ObjectTransformer

Sie können das Interface ObjectTransformer in den folgenden Situationen verwenden:

- nicht serialisierbares Objekt
- serialisierbares Objekt, aber Leistung der Serialisierung verbessern
- Schlüssel oder Wert kopieren

Im folgenden Beispiel wird ObjectGrid zum Speichern der Klasse Stock verwendet:

```
/**
* Stock-Objekt für ObjectGrid-Demo
*
*
*/
public class Stock implements Cloneable {
 String ticket;
 double price;
 String company;
 String description;
 int serialNumber;
 long lastTransactionTime;
 /**
 * @return Gibt die Beschreibung zurück.
 */
 public String getDescription() {
  return description;
 }
 /**
 * @param description Die festzulegende Beschreibung
 */
 public void setDescription(String description) {
  this.description = description;
 }
 /**
```

```
 * @return Gibt den lastTransactionTime-Wert zurück
 */
 public long getLastTransactionTime() {
  return lastTransactionTime;
 }
 /**
 * @param lastTransactionTime Der festzulegende lastTransactionTime-Wert
 */
 public void setLastTransactionTime(long lastTransactionTime) {
  this.lastTransactionTime = lastTransactionTime;
 }
 /**
 * @return Gibt den Preis zurück
 */
 public double getPrice() {
  return price;
 }
 /**
 * @param price Der festzulegende Preis
 */
 public void setPrice(double price) {
  this.price = price;
 }
 /**
 * @return Gibt die Seriennummer zurück
 */
 public int getSerialNumber() {
  return serialNumber;
 }
 /**
 * @param serialNumber Die festzulegende Seriennnummer
 */
 public void setSerialNumber(int serialNumber) {
  this.serialNumber = serialNumber;
 }
 /**
 * @return Gibt das Ticket zurück
 */
 public String getTicket() {
  return ticket;
 }
 /**
 * @param ticket Das festzulegende Ticket
 */
 public void setTicket(String ticket) {
  this.ticket = ticket;
 }
 /**
 * @return Gibt das Unternehmen zurück
 */
 public String getCompany() {
  return company;
 }
 /**
 * @param company Das festzulegende Unternehmen
 */
 public void setCompany(String company) {
  this.company = company;
 }
 // Klonen
 public Object clone() throws CloneNotSupportedException
 {
  return super.clone();
 }
}
```

Sie können eine eigene Objektumsetzerklasse für die Klasse Stock schreiben:

```
/**
* Benutzerdefinierte Implementierung des ObjectGrid-Plug-in ObjectTransformer
* für Stock-Objekt
*
*/
public class MyStockObjectTransformer implements ObjectTransformer {
 /* (keine 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);
 }

 /* (keine 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());
 }

/* (keine 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;
 }

/* (keine 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;
 }

/* (keine 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)
  {
   // Datenstrom erzeugen
  }
 }

/* (keine 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;
 }
}
```

Implementieren Sie anschließend die angepasste Klasse MyStockObjectTransformer in der BackingMap-Instanz:

```
ObjectGridManager ogf=ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogf.getObjectGrid("NYSE");
BackingMap bm = og.defineMap("NYSEStocks");
MyStockObjectTransformer ot = new MyStockObjectTransformer();
bm.setObjectTransformer(ot);
```

Plug-in TransactionCallback

Eine Anwendung implementiert normalerweise ein TransactionCallback-Plug-in und einen Loader als Paar. Der Loader ist für den Abruf der Daten aus dem Backend sowie für das Anwenden von Änderungen auf das Backend zuständig. Das Abrufen und Schreiben der Daten findet normalerweise im Kontext einer ObjectGrid-Transaktion statt.

Das TransactionCallback-Plug-in hat die folgenden Zuständigkeiten:

- Es reserviert Slots für einen transaktionsspezifischen Zustand, der für die Transaktion und den Loader erforderlich ist.
- Es ordnet eine ObjectGrid-Transaktion einer Plattformtransaktion zu.
- Es definiert den transaktionsspezifischen Zustand, wenn die ObjectGrid-Instanz eine Transaktion startet.
- Es schreibt die Transaktion fest, wenn die ObjectGrid-Transaktion festgeschrieben wird.
- Er macht die Transaktion rückgängig, wenn die die ObjectGrid-Transaktion rückgängig gemacht wird.

Die ObjectGrid-Instanz ist kein XA-Transaktionskoordinator. Die ObjectGrid-Instanz stützt sich in Bezug auf diese Funktionalität auf die Plattform. Die ObjectGrid-Methoden `begin`, `commit` und `rollback`, die in einer Sitzung verwendet werden, sind Aufrufe, die sich auf den Lebenszyklus beziehen. Das Plug-in TransactionCallback muss diese Ereignisse empfangen und die Transaktionsfunktionen der Plattform für die von den Loadern verwendeten Ressourcen nutzen. Dieser Artikel untersucht verschiedene Szenarios und beschreibt, wie Sie das TransactionCallback-Plug-in für diese Szenarios schreiben.

Übersicht über das Plug-in TransactionCallback

Das Plug-in TransactionCallback ist ein POJO, das das Interface TransactionCallback implementiert. Das Interface TransactionCallback gleicht dem folgenden Beispiel:

```
public interface TransactionCallback
{
 void initialize(ObjectGrid objectGrid) throws TransactionCallbackException;
 void begin(TxID id) throws TransactionCallbackException;
 void commit(TxID id) throws TransactionCallbackException;
 void rollback(TxID id) throws TransactionCallbackException;
 boolean isExternalTransactionActive(Session session);
}
```

Methode `initialize`

Die Methode `initialize` wird bei der Initialisierung der ObjectGrid-Instanz aufgerufen. Der Callback reserviert die erforderlichen Slots für das TxID-Objekt. Normalerweise wird ein Slot für jeden Teil des Zustands bzw. Objekts reserviert, der bzw. das beim Transaktionsstart mit der Methode `begin` erstellt werden soll. Angenommen, Sie möchten einen Persistenzmanager für ObjectGrid als Loader verwenden. Wenn dieser Persistenzmanager Sitzungs- und Transaktionszustandobjekte hat, ruft das TransactionCallback-Plug-in eine Sitzung und eine Transaktion ab und verwaltet die Referenzen auf diese beiden Objekte in Slots des TxID-Objekts. In diesem Fall sieht die Methode `initialize` wie folgt aus:

```
/**
* Wird aufgerufen, wenn das Grid initialisiert wird. Die Slots
* werden nur im TxID-Objekt reserviert.
*/
public void initialize(ObjectGrid objectGrid) throws
 TransactionCallbackException
{
 // Einen Slot für die Transaktion des Persistenzmanagers reservieren
 TXslot = objectGrid.reserveSlot(TxID.SLOT_NAME);
 // Einen Slot für die Sitzung des Persistenzmanagers reservieren
 SessionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
}
```

Ein TxID-Objekt hat Slots. Die Slots sind Einträge in einem Array des Typs ArrayList. Plug-ins können einen Eintrag im Array ArrayList reservieren, indem sie die Methode `ObjectGrid.reserveSlot` aufrufen und anzeigen, dass sie einen Slot im TxID-Objekt reservieren möchten. Die Methode gibt daraufhin den nächsten Eintragsindex an die Anwendung zurück. Anschließend kann die Anewndung Informationen in diesem Slot speichern. Die folgenden Methoden veranschaulichen dieses Verfahren.

Methode `begin`

Die ObjectGrid-Instanz ruft diese Methode auf, wenn sie eine neue Transaktion startet. Das Plug-in ordnet dieses Ereignis einer echten Transaktion zu, die die Loader dann für die eingehenden Methodenaufrufe `get` und `update` verwenden können, bevor die Methode `commit` aufgerufen wird. Im Folgenden sehen Sie ein Beispiel für die Methode `begin`, die eine ObjectGrid-Methode begin der Persistenzmanager-Methode begin für Transaktionen zuordnet:

```
/**
* Diese Methode wird aufgerufen, wenn das Grid eine neue Transaktion startet.
* wird nur eine Persistenzmanagertransaktion erstellt und begin aufgerufen.
* Anschließend wird die Transaktion im TxID-Slot gespeichert, damit sie später
* ohne ThreadLocal usw. abgerufen werden kann.
*/
```

```
public void begin(TxID id) throws TransactionCallbackException
{
 Session PMsession = getPMcurrentSession();
 Transaction tx = PMsession.beginTransaction();
 id.putSlot(TXslot, tx);
 id.putSlot(SessionSlot, PMsession);
}
```

Dieses Beispiel stützt sich auf die Tatsache, dass die Methode `initialize` zwei Slots im TxID-Objekt reserviert hat. Ein Slot ist für die Sitzung und der andere Slot für die Transaktion des Persistenzmanagers bestimmt. Die Methode `begin` fordert den Persistenzmanager auf, eine Sitzung abrufen, speichert das Objekt im indexierten SessionSlot-Slot und erstellt anschließend eine Transaktion in der Sitzung. Dann speichert sie eine Referenz auf diese Transaktion mit dem indexierten TXSlot-Slot.

Methode `commit`

Die Methode `commit` wird aufgerufen, wenn eine ObjectGrid-Transaktion festgeschrieben wird. Die Daten aller Loader wurden bereits in das Backend geschrieben. Das Plug-in ist dafür zuständig, die Plattform über dieses COMMIT-Ereignis zu informieren.

```
/**
* Diese Methode wird aufgerufen, wenn das Grid eine Transaktion festschreiben
* möchte. Sie wird nur an den Persistenzmanager weitergegeben.
*/
public void commit(TxID id) throws TransactionCallbackException
{
 Transaction tx = (Transaction)id.getSlot(TXslot);
 tx.commit();
}
```

Die Methode sucht die Transaktion des Persistenzmanagers, die im Slot gespeichert ist, und ruft anschließend die Methode `commit` auf.

Methode `rollback`

Diese Methode wird aufgerufen, wenn eine ObjectGrid-Transaktion eine Transaktion rückgängig machen möchte. Das Plug-in leitet dies an den Transaktionsmanager der Plattform weiter. Das zugehörige Code-Snippet sehen Sie im Folgenden:

```
/**
* Diese Methode wird aufgerufen, wenn das Grid eine Transaktion rückgängig
* machen möchte. Dies wird nur an den Persistenzmanager weitergegeben.
*/
public void rollback(TxID id) throws TransactionCallbackException
{
 Transaction tx = (Transaction)id.getSlot(TXslot);
 tx.rollback();
}
```

Diese Methode ist der Methode `commit` sehr ähnlich. Sie ruft eine Referenz auf die Transaktion des Persistenzmanagers aus einem Slot ab und ruft anschließend die Methode `rollback` auf.

Methode `isExternalTransactionActive`

Eine ObjectGrid-Sitzung wird normalerweise im Modus mit automatischer Festschreibung oder im Transaktionsmodus ausgeführt. Modus mit automatischer Festschreibung bedeutet, dass bei jedem Methodenaufruf, der in der Sitzung an die ObjectMap-Instanzen abgesetzt wird, eine implizite Transaktion erstellt wird. Wenn keine Transaktion aktiv ist und eine Anwendung einen Aufruf an eine ObjectMap-Methode absetzt, ruft das Framework diese Methode im Plug-in TransactionCallback auf, um zu prü-

fen, ob eine entsprechende Transaktion aktiv ist. Wenn diese Methode den Wert true zurückgibt, leitet das Framework automatisch eine begin-Operation ein, andernfalls automatisch eine COMMIT-Operation. Diese Methode ermöglicht die Integration von ObjectGrid in Umgebungen, in denen die Anwendung die Methoden `begin`, `commit` und `rollback` in den Plattform-APIs und nicht in den ObjectGrid-APIs aufruft.

Szenario: Einfache JDBC-basierte J2SE-Umgebung

Dieses Beispiel verwendet eine J2SE-Umgebung, in der die Anwendung einen JDBC-basierten Loader hat. Es sind zwei Maps mit jeweils einem Loader vorhanden. Die Loader greifen auf unteschiedliche Tabellen in der Datenbank zu. Das Plug-in TransactionCallback ruft eine JDBC-Verbindung ab und ruft anschließend die Methoden `begin`, `commit` und `rollback` in der Verbindung auf. Im Folgenden sehen Sie die TransactionCallback-Implementierung für das Beispiel:

```
public class JDBCTCB implements TransactionCallback
{
 DataSource datasource;
 int connectionSlot;
 public JDBCTCB(DataSource ds)
 {
  datasource = ds;
 }
 public void initialize(ObjectGrid objectGrid)
  throws TransactionCallbackException
  {
   connectionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
  }
 public void begin(TxID id) throws TransactionCallbackException
 {
  try
  {
   Connection conn = datasource.getConnection();
   conn.setAutoCommit(false);
   id.putSlot(connectionSlot, conn);
  }
  catch(SQLException e)
  {
   throw new TransactionCallbackException("Cannot start transaction", e);
  }
 }
 public void commit(TxID id) throws TransactionCallbackException
 {
  Connection conn = null;
  try
 {
   conn = (Connection)id.getSlot(connectionSlot);
   conn.commit();
   conn.close();
  }
  catch(SQLException e)
  {
   throw new TransactionCallbackException("Cannot commit transaction", e);
  }
  finally {
   if (conn!=null) {
    try {
    conn.close();
    }
    catch (SQLException closeE) {
   }
  }
 }
 }
```

```
public void rollback(TxID id) throws TransactionCallbackException
{
 Connection conn = null;
 try
 {
  conn = (Connection)id.getSlot(connectionSlot);
  conn.rollback();
  conn.close();
 }
 catch(SQLException e)
 {
  throw new TransactionCallbackException("Cannot rollback transaction", e);
 }
 finally {
  if (conn!=null) {
   try {
    conn.close();
   }
   catch (SQLException closeE) {
   }
  }
 }
}
public boolean isExternalTransactionActive(Session session)
{
 return false;
}
public int getConnectionSlot()
{
 return connectionSlot;
}
}
```

Diees Beispiel zeigt ein TransactionCallback-Plug-in, das die Ereignisse der ObjectGrid-Transaktion in eine JDBC-Verbindung konvertiert. Bei der Initialisierung des Plug-in reserviert es einen Slot, in dem es eine Referenz auf die JDBC-Verbindung speichert. Anschließend ruft die Methode `begin` eine JDBC-Verbindung für die neue Transaktion ab, inaktiviert die automatische Festschreibung und speichert dann eine Referenz auf die Verbindung im TxID-Slot. Die Methoden `commit` und `rollback` rufen die Verbindung aus dem TxID-Slot ab und rufen die entsprechende Methode in der Verbindung auf. Die Methode `isExternalTransaction` gibt immer false zurück, d. h. die Anwendung muss für die Steuerung der Transaktionen explizit die Transaktions-APIs von ObjectGrid verwenden. Ein Loader, der paarweise mit diesem Plug-in verbunden ist, ruft die JDBC-Verbindung aus dem TxID-Objekt ab. Im Folgenden sehen Sie ein Beispiel für einen Loader:

```
public class JDBCLoader implements Loader
{
 JDBCTCB tcb;
 public void preloadMap(Session session, BackingMap backingMap)
 throws LoaderException
 {
 tcb = (JDBCTCB)session.getObjectGrid().getTransactionCallback();
 }
 public List get(TxID txid, List keyList, boolean forUpdate)
 throws LoaderException
 {
  Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
  // Hier get implementieren
 return null;
 }
 public void batchUpdate(TxID txid, LogSequence sequence)
 throws LoaderException, OptimisticCollisionException
 {
```

```
  Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
  // Hier Stapelaktualisierung implementieren
 }
}
```

Der Loader ruft beim Aufruf der Methode `initialize` eine Referenz auf die JDBCTCB-Instanz ab. Anschließend ruft er das mit JDBCTCB abgerufene Connection-Objekt ab, wenn dies in den Methoden `get` und `batchUpdate` erforderlich ist. TransactionCallback-Implementierungen und Loader werden normalerweise als Paare geschrieben, die miteinander kooperieren. Die TransactionCallback-Implementierung verarbeitet die Transaktion und speichert die Objekte, die der Loader benötigt, in Slots im TxID-Objekt. Anschließend implementieren die Loader `get`- und `batchUpdate`-Methoden im Kontext einer Transaktion, die vom TransactionCallback mit Ressourcen verwaltet wird, die normalerweise vom TCB abgerufen werden.

Szenario: Umgebung mit einer Servlet-Steuerkomponente

In diesem Szenario verwendet die ObjectGrid-Instanz zwar auch einen JDBC-basierten Loader, aber in einer verwalteten Servlet-Steuerkomponente. Der Container geht davon aus, dass die Methode `UserTransaction` zum Starten und Festschreiben von Transaktionen verwendet wird. Dieses Szenario unterscheidet ist geringfügig vom J2SE-Szenario, da keine Referenz auf die JDBC-Verbindung in einem TxID-Slot gespeichert werden muss. Die JDBC-Verbindung wird hier vom Container verwaltet. Wenn eine Containertransaktion aktiv ist und eine Datenquelle zum Suchen einer Verbindung genutzt wird, wird immer dieselbe Verbindung verwendet, da der Container sich merkt, welche Verbindungen von der Transaktion verwendet werden, und stets dieselbe Verbindung zurückgibt, wenn die Methode `DataSource.getConnection` aufgerufen wird. Angenommen, die Datenquellenreferenz ist im folgenden Beispiel als `gemeinsam nutzbar` konfiguriert:

```
public class ManagedJDBCTCB implements TransactionCallback {
 UserTransaction tx;
 public void initialize(ObjectGrid objectGrid)
 throws TransactionCallbackException
 {
  try
  {
   InitialContext ic = new InitialContext();
   tx = (UserTransaction)ic.lookup("java:comp/UserTransaction");
  }
  catch(NamingException e)
  {
   throw new TransactionCallbackException("Cannot find UserTransaction", e);
  }
 }
 public void begin(TxID id) throws TransactionCallbackException
 {
  try
  {
   tx.begin();
  }
  catch(SystemException e)
  {
   throw new TransactionCallbackException("Cannot begin tx", e);
  }
  catch(NotSupportedException e)
  {
   throw new TransactionCallbackException("Cannot begin tx", e);
  }
 }
 public void commit(TxID id) throws TransactionCallbackException
 {
```

```
 try
 {
  tx.commit();
 }
 catch(SystemException e)
 {
  throw new TransactionCallbackException("Cannot commit tx", e);
 }
 catch(HeuristicMixedException e)
 {
  throw new TransactionCallbackException("Cannot commit tx", e);
 }
 catch(RollbackException e)
 {
  throw new TransactionCallbackException("Cannot commit tx", e);
 }
 catch(HeuristicRollbackException e)
 {
  throw new TransactionCallbackException("Cannot commit tx", e);
 }
}

public void rollback(TxID id) throws TransactionCallbackException
{
 try
 {
  tx.rollback();
 }
 catch(SystemException e)
 {
  throw new TransactionCallbackException("Cannot commit tx", e);
 }
}
public boolean isExternalTransactionActive(Session session) {
 return false;
}
}
```

Dieses Beispiel ruft eine Referenz auf die Methode `UserTransaction` in der Methode `initialize` ab und ordnet anschließend die Methoden `begin`, `commit` und `rollback` den entsprechenden `UserTransaction`-Methoden zu. Slots sind nicht erforderlich, da der Container sicherstellt, dass die richtigen Verbindungsinformationen für diese Transaktion abgerufen werden. Im Folgenden sehen Sie den JDBC-Loader für diese TransactionCallback-Implementierung:

```
public class ManagedJDBCLoader implements Loader
{
 DataSource myDataSource;
 ManagedJDBCLoader(DataSource ds)
 {
  myDataSource = ds;
 }
 public void preloadMap(Session session, BackingMap backingMap)
 throws LoaderException
 {
 }
 public List get(TxID txid, List keyList, boolean forUpdate)
 throws LoaderException
 {
  try
  {
   Connection conn = myDataSource.getConnection();
   // Hier get für die Verbindung implementieren
   return null;
  }
  catch(SQLException e)
```

```
  {
   throw new LoaderException("Cannot get objects", e);
  }
 }
 public void batchUpdate(TxID txid, LogSequence sequence)
 throws LoaderException, OptimisticCollisionException
 {
  try
  {
   Connection conn = myDataSource.getConnection();
   // Hier update für die Verbindung implementieren
  }
  catch(SQLException e)
  {
   throw new LoaderException("Cannot update objects", e);
  }
 }
}
```

Dieses Beispiel kann einfacher sein als die JDBC-Basisversion, weil der Container die Verbindungen verwaltet und sicherstellt, dass die Methode `DataSource.getConnection` stets dieselbe Verbindung zurückgibt, wenn sie in derselben aktiven Transaktion aufgerufen wird. Versuchen Sie deshalb nicht, die Verbindung in einem Slot zwischenzuspeichern, auch wenn die Anwendung die Verbindung bei Bedarf zwischenspeichert.

Interface OptimisticCallback

Sie können ein Plug-in-fähiges Objekt für optimistischen Callback verwenden, das das Interface com.ibm.websphere.objectgrid.plugins.OptimisticCallback implementiert.

Zweck

Das Interface OptimisticCallback unterstützt optimistische Vergleichsoperationen für die Werte einer Map. Ein OptimisticCallback-Objekt ist erforderlich, wenn die optimistische Sperrstrategie, wie im Abschnitt „Optimistisches Sperren" auf Seite 71 beschrieben, verwendet wird. ObjectGrid stellt eine Standardimplementierung des Interface OptimisticCallback bereit. Normalerweise muss die Anwendung jedoch eine eigene Implementierung des Interface OptimisticCallback bereitstellen.

Von der Anwendung bereitgestelltes OptimisticCallback-Objekt implementieren

Das folgende Beispiel veranschaulicht, wie eine Anwendung ein OptimisticCallback-Objekt für die BackingMap ″employee″ in der ObjectGrid-Instanz grid1 implementieren kann:

```
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 );
```

Das Objekt EmployeeOptimisticCallbackImpl im vorherigen Beispiel muss das Interface OptimisticCallback implementieren. Die Anwendung kann, wie im folgenden Beispiel gezeigt, auch eine XML-Datei verwenden, um ihr OptimisticCallback-Objekt zu implementieren:

```
<?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>
```

Standardimplementierung

Das Framework ObjectGrid stellt eine Standardimplementierung des Interface OptimisticCallback bereit, die verwendet wird, wenn die Anwendung nicht, wie im vorherigen Abschnitt gezeigt, ein eigenes OptimisticCallback-Objekt bereitstellt. Die Standardimplementierung gibt immer den Sonderwert NULL_OPTIMISTIC_VERSION als Versionsobjekt für den Wert zurück und aktualisiert das Versionsobjekt nicht. Diese Aktion macht aus dem optimistischen Vergleich eine Funktion ohne Operation. In den meisten Fällen ist dies nicht erwünscht, wenn die optimistische Sperrstrategie verwendet wird. Ihre Anwendungen müssen das Interface OptimisticCallback implementieren und ihre eigenen OptimisticCallback-Implementierungen bereitstellen, damit die Standardimplementierung nicht verwendet wird. Es gibt jedoch ein Szenario, in dem die OptimisticCallback-Standardimplementierung hilfreich ist. Stellen Sie sich die folgende Situation vor:

- Es ist ein Loader für die BackingMap-Instanz implementiert.
- Der Loader weiß ohne Unterstützung eines OptimisticCallback-Plug-in, wie der optimistische Vergleich durchzuführen ist.

Woher weiß der Loader ohne Unterstützung eines OptimisticCallback-Objekts, wie bei der optimistischen Versionssteuerung vorzugehen ist? Der Loader kennt das value-Klassenobjekt und weiß, welches Feld des Wertobjekts als Wert für die optimistische Versionssteuerung verwendet wird. Stellen Sie sich beispielsweise vor, dass das folgende Interface als Wertobjekt für die Map ″employees″ verwendet wird.

```
public interface Employee
{
// Folgenummer für optimistische Versionssteuerung
public long getSequenceNumber();
public void setSequenceNumber(long newSequenceNumber);
// Weitere get/set-Methoden für andere Felder des Objekts Employee
}
```

In diesem Fall weiß der Loader, dass er mit der Methode getSequenceNumber die aktuellen Versionsinformationen für ein Employee-Wertobjekt abrufen kann. Er erhöht den zurückgegebenen Wert um eins, um eine neue Versionsnummer zu generieren, bevor er den persistenten Speicher mit dem neuen Employee-Wert

aktualisiert. Bei einem JDBC-Loader wird die aktuelle Folgenummer in der where-Klausel einer überqualifizierten SQL-Anweisung ″update″ verwendet, und die neu generierte Folgenummer wird in der Folgenummernspalte gesetzt. Eine andere Möglichkeit ist die, dass der Loader eine vom Backend bereitgestellte Funktion verwendet, die eine verdeckte Spalte, die für die optimistische Versionssteuerung geeignet ist, automatisch aktualisiert. Manchmal kann möglicherweise eine gespeicherte Prozedur oder ein Trigger verwendet werden, um eine Spalte mit Versionsinformationen zu verwalten. Wenn der Loader eines dieser Verfahren für die Verwaltung von Informationen für die optimistische Versionssteuerung verwendet, muss die Anwendung keine OptimisticCallback-Implementierung bereitstellen. In diesem Fall kann die OptimisticCallback-Standardimplementierung verwendet werden, weil der Loader die optimistische Versionssteuerung ohne Unterstützung eines OptimisticCallback-Objekts vornehmen kann.

Interface OptimisticCallback implementieren

Das Interface OptimisticCallback enthält die folgenden Methoden und Sonderwerte:

```
public interface OptimisticCallback
{
 final static Byte NULL_OPTIMISTIC_VERSION;
 Object getVersionedObjectForValue(Object value);
 void updateVersionedObjectForValue(Object value);
 void serializeVersionedValue(Object versionedValue,
 ObjectOutputStream stream) throws IOException;
 Object inflateVersionedValue(ObjectInputStream stream) throws
 IOException, ClassNotFoundException;
}
```

Die folgende Liste enthält eine Beschreibung oder einen spezielle Hinweise für jede der Methoden im Interface OptimisticCallback:

NULL_OPTIMISTIC_VERSION
Dieser Sonderwert wird von der Methode getVersionedObjectForValue zurückgegeben, wenn die OptimisticCallback-Standardimplementierung anstelle einer anwendungseigenen OptimisticCallback-Implementierung verwendet wird.

Methode getVersionedObjectForValue
Diese Methode kann eine Kopie des Wertes oder ein Attribut des Wertes zurückgeben, die bzw. das für die Versionssteuerung verwendet werden kann. Sie wird aufgerufen, wenn ein Objekt einer Transaktion zugeordnet wird. Wenn eine BackingMap keinen Loader hat, verwendet die BackingMap diesen Wert während der Festschreibung, um einen optimistischen Versionsvergleich durchzuführen. Der optimistische Versionsvergleich wird von der BackingMap verwendet, um sicherzustellen, dass sich die Version nicht geändert hat, nachdem die Transaktion zum ersten Mal auf den Map-Eintrag zugegriffen hat. Wenn eine andere Transaktion die Version für diesen Map-Eintrag bereits geändert hat, schlägt der Versionsvergleich fehl, und die BackingMap zeigt eine Ausnahme des Typs OptimisticCollisionException an, um eine ROLLBACK-Operation für die Transaktion zu erzwingen. Wenn der BackingMap ein Loader zugeordnet ist, verwendet die BackingMap die Informationen für die optimistische Versionssteuerung nicht. Vielmehr ist in diesem Fall der Loader für den optimistischen Versionsvergleich und die eventuelle Aktualisierung der Versionsinformationen zuständig. Normalerweise ruft der Loader das Ausgangsversionsobjekt von dem LogElement ab, das an die Methode batch-

Update des Loader übergeben wurde. Die Methode batchUpdate wird aufgerufen, wenn eine flush-Operation ausgeführt oder eine Transaktion festgeschrieben wird.

Der folgende Code zeigt die vom Objekt EmployeeOptimisticCallbackImpl verwendete Implementierung:

```
public Object getVersionedObjectForValue(Object value)
{
 if (value == null)
 {
  return null;
 }
 else
 {
  Employee emp = (Employee) value;
  return new Long( emp.getSequenceNumber() );
 }
}
```

Wie im vorherigen Beispiel gezeigt, wird das Attribut sequenceNumber in einem java.lang.Long-Objekt zurückgegeben, das der Loader erwartet. Dies impliziert, dass dieselbe Person, die den Loader geschrieben hat, auch die EmployeeOptimisticCallbackImpl-Implementierung geschrieben oder zumindest eng mit der Person, die EmployeeOptimisticCallbackImpl implementiert hat, zusammengearbeitet und sich auf den von der Methode getVersionedObjectForValue zurückgegebenen Wert geeinigt hat.

Die OptimisticCallback-Standardimplementierung gibt wie zuvor beschrieben den Sonderwert NULL_OPTIMISTIC_VERSION als Versionsobjekt zurück.

Methode updateVersionedObjectForValue

Diese Methode wird aufgerufen, wenn eine Transaktion einen Wert aktualisiert hat und ein neues Versionsobjekt erforderlich ist. Wenn die Methode getVersionedObjectForValue ein Attribut des Wertes zurückgibt, aktualisiert diese Methode normalerweise den Attributwert mit einem neuen Versionsobjekt. Gibt die Methode getVersionedObjectForValue eine Kopie des Wertes zurück, führt die Methode in der Regel keine Aktionen aus. Die OptimisticCallback-Standardimplementierung führt keine Aktionen aus, da die Standardimplementierung von getVersionedObjectForValue immer den Sonderwert NULL_OPTIMISTIC_VERSION als Versionsobjekt zurückgibt.

Der folgende Code zeigt die Implementierung, die von dem im Abschnitt ″OptimisticCallback″ verwendeten Objekt EmployeeOptimisticCallbackImpl verwendet wird:

```
public void updateVersionedObjectForValue(Object value)
{
 if ( value != null )
 {
  Employee emp = (Employee) value;
  long next = emp.getSequenceNumber() + 1;
  emp.updateSequenceNumber( next );
 }
}
```

Wie im vorherigen Beispiel gezeigt, wird das Attribut sequenceNumber um eins erhöht, damit der java.lang.Long-Wert, der beim nächsten Aufruf der Methode getVersionedObjectForValue zurückgegeben wird, ein Wert vom Typ ″long″ ist, der der ursprünglichen Folgenummer plus 1 entspricht und z. B. den nächsten Versionswert für diese employee-Instanz darstellt. Auch

hier impliziert das Beispiel, dass dieselbe Person, die den Loader geschrieben hat, auch die EmployeeOptimisticCallbackImpl-Implementierung oder zumindest eng mit der Person, von der EmployeeOptimisticCallbackImpl implementiert wurde, zusammengearbeitet hat.

Methode serializeVersionedValue
Diese Methode schreibt den Versionswert in den angegebenen Datenstrom. Je nach Implementierung kann der Versionswert verwendet werden, um Kollisionen bei einer optimistischen Aktualisierung festzustellen. In einigen Implementierungen ist der Versionswert eine Kopie des ursprünglichen Wertes. Andere Implementierungen können eine Folgenummer oder ein anderes Objekt haben, das die Version des Wertes anzeigt. Da die eigentliche Implementierung unbekannt ist, wird diese Methode für die ordnungsgemäße Durchführung der Serialisierung bereitgestellt. Die Standardimplementierung führt einen writeObject-Aufruf durch.

Methode inflateVersionedValue
Diese Methode verwendet die serialisierte Version des Versionswertes und gibt das Objekt mit dem echten Versionswert zurück. Je nach Implementierung kann der Versionswert verwendet werden, um Kollisionen bei einer optimistischen Aktualisierung festzustellen. In einigen Implementierungen ist der Versionswert eine Kopie des ursprünglichen Wertes. Andere Implementierungen können eine Folgenummer oder ein anderes Objekt haben, das die Version des Wertes anzeigt. Da die eigentliche Implementierung unbekannt ist, wird diese Methode für die ordnungsgemäße Durchführung der Entserialisierung bereitgestellt. Die Standardimplementierung führt einen readObject-Aufruf durch.

ObjectGrid-Konfiguration

Sie können ObjectGrid über das Programm oder mit einer XML-Datei konfigurieren. Das Interface ObjectGridManager ist der Ausgangspunkt für beide Konfigurationsmethoden.

Das Interface ObjectGridManager enthält mehrere Methoden, die Sie für das Erstellen einer ObjectGrid-Instanz verwenden können. Eine vollständig Beschreibung der einzelnen Methoden finden Sie im Abschnitt Interface ObjectGridManager. Die folgenden Abschnitte beschreiben die Methoden für das Konfigurieren der ObjectGrid-Instanz:

- Der Abschnitt ObjectGrid-Basiskonfiguration beschreibt, wie Sie eine einfache XML-Datei mit einer ObjectGrid- und einer BackingMap-Instanz erstellen.
- Der Abschnitt Vollständige ObjectGrid-Konfiguration enthält Definitionen der einzelnen Elemente und Attribute der XML-Datei und beschreibt, wie Sie dasselbe Ergebnis durch Programmierung erzielen.
- Der Abschnitt ObjectGrid-Konfiguration im gemischten Modus beschreibt, wie Sie die beiden zuvor genannten Konfigurationsmethoden in Kombination verwenden.

ObjectGrid-Basiskonfiguration

Dieser Abschnitt beschreibt, wie Sie eine einfache XML-Datei mit dem Namen `bookstore.xml` erstellen, die eine ObjectGrid- und eine BackingMap-Instanz definiert.

Die ersten Zeilen der Datei sind der erforderliche Header jeder ObjectGrid-XML-Datei. Die folgende XML-Datei definiert die ObjectGrid-Instanz *bookstore* mit der BackingMap-Instanz *book*:

bookstore.xml

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

Die XML-Datei wird an das Interface ObjectGridManager übergeben, das basierend auf der Datei eine ObjectGrid-Instanz erstellt. Das folgende Code-Snippet prüft die Gültigkeit der Datei `bookstore.xml` im Hinblick auf das XML-Schema und erstellt eine ObjectGrid-Instanz mit dem Namen *bookstore*. Die neu erstellte ObjectGrid-Instanz wird nicht zwischengespeichert.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore",
new URL("file:etc/test/bookstore.xml"), true, false);
```

Der folgende Code erzielt dasselbe Ergebnis ohne XML-Datei. Verwenden Sie diesen Code, um über das Programm eine BackingMap-Instanz in einer ObjectGrid-Instanz zu definieren. Dieser Code erstellt die BackingMap-Instanz ″book″ in der ObjectGrid-Instanz ″bookstoreGrid″:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
 ("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
```

Vollständige ObjectGrid-Konfiguration

Dieser Abschnitt enthält ausführliche Anleitungen zum Konfigurieren einer ObjectGrid-Instanz. Jedes Element und Attribut der XML-Datei wird definiert. Es werden XML-Beispieldateien gezeigt, denen der Code gegenüberstellt wird, mit dem dieselbe Task über das Programm realisiert werden kann.

Auf die folgende XML-Datei, `sample1.xml`, wird im gesamten Abschnitt verwiesen. Die Elemente und Attribute dieser Datei werden im Anschluss an dieses Beispiel ausführlich beschrieben.

Datei *sample1.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
 <objectGrids>
  <objectGrid name="bookstore">
   <bean id="ObjectGridEventListener"
     classname="com.company.organization.MyObjectGridEventListener" />
   <backingMap name="books" pluginCollectionRef="collection1" />
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="collection1">
   <bean id="Evictor"
    classname="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
```

```
    <property name="maxSize" type="int" value="321" />
   </bean>
  </backingMapPluginCollection>
 </objectGridConfig>
```

Element objectGridConfig

Anzahl der Vorkommen: 1

Untergeordnete Elemente: Element objectGrids und Element backingMapPlugin-Collections

Das Element objectGridConfig ist das Ausgangselement der XML-Datei. Es muss im XML-Dokument wie im Bespiel gezeigt angegeben werden. Dieses Element konfiguriert den Namespace für die Datei und die Schemaposition. Das Schema wird in der Datei `objectGrid.xsd` definiert. ObjectGrid sucht diese Datei im Stammverzeichnis der Datei `ObjectGrid.jar`.

Element objectGrids

Anzahl der Vorkommen: 1

Untergeordnetes Element: Element objectGrid

Das Element objectGrids ist ein Container für alle objectGrid-Elemente. In der Datei `sample1.xml` enthält das Element objectGrids ein objectGrid-Element mit dem Namen `bookstore`.

Element objectGrid

Anzahl der Vorkommen: 1 bis n

Untergeordnete Elemente: Element bean und Element backingMap

Verwenden Sie das Element objectGrid, um eine ObjectGrid-Instanz in einer XML-Datei zu definieren. Jedes der Attribute im Element objectGrid entspricht einer Methode im Interface ObjectGrid.

```
<objectGrid
(1) name="objectGridName"
(2) securityEnabled="true|false"
(3) authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS|
          AUTHORIZATION_MECHANISM_CUSTOM"
(4) permissionCheckPeriod="permission check period"
/>
```

Attribute:

1. Attribut **name** (erforderlich): Gibt den Namen für die ObjectGrid-Instanz an. Wenn dieses Attribut fehlt, schlägt die Validierung der XML-Datei fehl.
2. Attribut **securityEnabled** (optional, Standardeinstellung `false`): Wenn Sie dieses Attribut auf `true` setzen, wird die Sicherheit für ObjectGrid aktiviert. Standardmäßig ist die Sicherheit inaktiviert.
3. Attribut **authorizationMechanism** (optional, Standardeinstellung `AUTHORIZATION_MECAHNISM_JAAS`): Legt den Autorisierungsmechanismus für diese ObjectGrid-Instanz fest. Die gültigen Werte für dieses Attribut sind `AUTHORIZATION_MECHANISM_JAAS` und `AUTHORIZATION_MECHANISM_CUSTOM`. Setzen Sie das Attribut auf `AUTHORIZATION_MECHANISM_CUSTOM`, wenn Sie ein eigenes MapAu-

thorization-Plug-in verwenden. Diese Einstellung wird wirksam, wenn das Attribut securityEnabled auf `true` gesetzt ist.

4. Attribut **permissionCheckPeriod** (optional, Standardeinstellung `0`): Gibt einen ganzzahligen Wert (in Sekunden) an, der festlegt, wie oft die Berechtigung für einen Clientzugriff geprüft wird. Wenn dieses Attribut den Wert `0` hat, fordert jede Methode `get`, `put`, `update`, `remove` und `evict` den Autorisierungsmechanismus (JAAS- oder benutzerdefinierte Autorisierung) auf zu prüfen, ob das aktuelle Subjekt berechtigt ist. Ein Wert größer als `0` gibt den Zeitraum (in Sekunden) an, für den die Berechtigungen zwischengespeichert werden, bevor sie vom Autorisierungsmechanismus aktualisiert werden. Diese Einstellung wird wirksam, wenn das Attribut securityEnabled auf `true` gesetzt ist.

Die folgende XML-Beispieldatei, `objectGridAttr.xml`, zeigt eine Möglichkeit für die Konfiguration der Attribute einer objectGrid-Instanz. In diesem Beispiel ist die Sicherheit aktiviert, der Autorisierungsmechanismus auf JAAS gesetzt und das Intervall für die Berechtigungsprüfung auf 45 Sekunden eingestellt.

Datei *objectGridAttr.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
 <objectGrids>
  <objectGrid name="bookstore" securityEnabled="true"
   authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS"
   permissionCheckPeriod="45">
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

Der folgende Code veranschaulicht, wie Sie die Konfiguration, die Sie mit der Datei `objectGridAttr.xml` im vorherigen Beispiel erstellt haben, über das Programm realisieren.

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory
.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore", false);

bookstoreGrid.setSecurityEnabled();
bookstoreGrid.setAuthorizationMechanism(
SecurityConstants.AUTHORIZATION_MECHANISM_JAAS);
bookstoreGrid.setPermissionCheckPeriod(45);
```

Element backingMap

Anzahl der Vorkommen: 0 bis n

Untergeordnete Elemente: keine

Das Element backingMap wird verwendet, um eine BackingMap-Instanz in einer ObjectGrid-Instanz zu erstellen. Jedes der Attribute im Element backingMap entspricht einer Methode im Interface BackingMap.

```
<backingMap
(1) name="backingMapName"
(2) readOnly="true|false"
(3) pluginCollectionRef="reference to backingMapPluginCollection"
(4) numberOfBuckets="number of buckets"
(5) preloadMode="true|false"
(6) lockStrategy="OPTIMISTIC|PESSIMISTIC|NONE"
(7) numberOfLockBuckets="number of lock buckets"
```

```
(8) lockTimeout="lock timeout"
(9) copyMode="COPY_ON_READ_AND_COMMIT|COPY_ON_READ|
  COPY_ON_WRITE|NO_COPY"
(10) valueInterfaceClassName="value interface class name"
(11) copyKey="true|false"
(12) nullValuesSupported="true|false"
(13) ttlEvictorType="CREATION_TIME|LAST_ACCESS_TIME|NONE"
(14) timeToLive="time to live"
/>
```

Attribute:

1. Attribut **name** (erforderlich): Gibt den Namen für die BackingMap-Instanz an. Wenn dieses Attribut fehlt, schlägt die Validierung der XML-Datei fehl.
2. Attribut **readOnly** (optional, Standardeinstellung `false`): Wenn Sie dieses Attribut auf `true` setzen, ist die BackingMap-Instanz schreibgeschützt. Hat das Attribut den Wert `false`, sind Lese- und Schreibzugriff auf die BackingMap-Instanz zulässig. Wenn Sie keinen Wert für das Attribut festlegen, wird die Standardeinstellung verwendet, d. h. es sind Lese- und Schreibzugriffe zulässig.
3. Attribut **pluginCollectionRef** (optional): Gibt eine Referenz auf ein backingMapPluginCollection-Plug-in an. Der Wert dieses Attribut muss mit dem Wert des Attributs id eines backingMapCollection-Plug-in übereinstimmen. Die Validierung schlägt fehl, wenn keine übereinstimmende ID vorhanden ist. Diese Referenz hat den Zweck, dass BackingMap-Plug-ins einfach wiederverwendet werden können.
4. Attribut **numberOfBuckets** (optional, Standardeinstellung `503`): Die Anzahl der von der BackingMap-Instanz zu verwendenden Buckets. Die BackingMap-Implementierung verwendet eine Hash-Tabelle. Wenn in der BackingMap viele Einträge vorhanden sind, kann mit einer höheren Anzahl von Buckets eine bessere Leistung erzielt werden, weil das Risiko von Kollisionen mit zunehmender Anzahl von Buckets abnimmt. Außerdem sind bei einer höheren Anzahl von Buckets mehr gemeinsame Zugriffe möglich.
5. Attribut **preloadMode** (optional, Standardeinstellung `false`): Setzt den Modus für vorheriges Laden, wenn ein Loader-Plug-in für diese BackingMap-Instanz definiert ist. Wenn das Attribut auf `true` gesetzt ist, wird die Methode `Loader.preloadMap(Session, BackingMap)` asynchron aufgerufen. Andernfalls wird die Ausführung der Methode beim Laden der Daten blockiert, so dass der Cache bis zum Abschluss des vorherigen Ladens nicht verfügbar ist. Das vorherige Laden wird während der ObjectGrid-Initialisierung durchgeführt.
6. Attribut **lockStrategy** (optional, Standardeinstellung `OPTIMISTIC`): Setzt die Sperrstrategie für die BackingMap-Instanz. Die Sperrstrategie bestimmt, ob der interne ObjectGrid-Sperrenmanager verwendet wird, wenn eine Transaktion auf einen Map-Eintrag zugreift. Die gültigen Werte für dieses Attribut sind `OPTIMISTIC`, `PESSIMISTIC` und `NONE`.

   Der Wert `OPTIMISTIC` wird normalerweise für eine Map verwendet, die kein Loader-Plug-in hat, die zum größten Teil nur gelesen wird und für die der Persistenzmanager, der ObjectGrid als Neben-Cache verwendet, oder die Anwendung keine Sperrstrategie bereitstellt. Bei der optimistischen Sperrstrategie wird beim Festschreiben eine exklusive Sperre für einen Map-Eintrag angefordert, der eingefügt, aktualisiert oder entfernt wird. Die Sperre stellt sicher, dass die Versionsinformationen nicht von einer anderen Transaktion geändert werden, während die Transaktion, die festgeschrieben wird, eine optimistische Versionsprüfung durchführt.

   Der Wert `PESSIMISTIC` wird normalerweise für eine Map verwendet, die kein Loader-Plug-in hat und für die weder der Persistenzmanager mit objectGrid

als Neben-Cache, noch ein Loader-Plug-in oder eine Anwendung eine Sperrstrategie bereitstellt. Die pessimistische Sperrstrategie wird verwendet, wenn die optimistische Strategie zu häufig fehlschlägt, weil Aktualisierungstransaktionen für denselben Map-Eintrag zu häufig kollidieren. Die optimistische Strategie kann fehlschlagen, wenn die Map nicht nur gelesen wird oder wenn viele Clients auf eine kleine Map zugreifen.

Der Wert NONE gibt an, dass die Verwendung des internen Sperrenmanager nicht erforderlich ist, weil die Steuerung des gemeinsamen Zugriffs außerhalb der ObjectGrid-Instanz realisiert wird, z. B. vom Persistenzmanager, der ObjectGrid als Neben-Cache verwendet, von einer Anwendung oder von einem Loader-Plug-in, das die Datenbanksperren für die Steuerung des gemeinsamen Zugriffs verwendet.

7. Attribut **numberOfLockBuckets** (optional, Standardeinstellung 383): Legt die Anzahl der Sperren-Buckets fest, die vom Sperrenmanager für diese BackingMap-Instanz verwendet wird. Wenn das Attribut lockStrategy auf OPTIMISTIC oder PESSIMISTIC gesetzt ist, wird ein Sperrenmanager für die BackingMap-Instanz erstellt. Der Sperrenmanager verwendet eine Hash-Tabelle, um die Einträge zu verfolgen, die von einer oder mehreren Transaktionen gesperrt werden. Wenn in der Hash-Tabelle viele Einträge vorhanden sind, kann mit einer höheren Anzahl von Sperren-Buckets eine bessere Leistung erzielt werden, weil das Risiko von Kollisionen mit zunehmender Anzahl von Buckets abnimmt. Außerdem sind bei einer höheren Anzahl von Sperren-Buckets mehr gemeinsame Zugriffe möglich. Wenn das Attribut lockStrategy auf NONE gesetzt ist, wird von dieser BackingMap-Instanz kein Sperrenmanager verwendet. In diesem Fall ist die Definition des Attributs numberOfLockBuckets nicht erforderlich.
8. Attribut **lockTimeout** (optional, Standardeinstellung 15): Legt das Sperrzeitlimit fest, das der Sperrenmanager für diese BackingMap-Instanz verwendet. Wenn das Attribut lockStrategy auf OPTIMISTIC oder PESSIMISTIC gesetzt ist, wird ein Sperrenmanager für die BackingMap-Instanz erstellt. Um gegenseitiges Sperren zu verhindern, arbeitet der Sperrenmanager mit einem Standardzeitlimit, das die Wartezeit für das Erteilen einer Sperre vorgibt. Bei einer Überschreitung dieses Zeitlimits wird eine Ausnahme des Typs LockTimeoutException ausgelöst. Der Standardwert von 15 Sekunden ist für die meisten Anwendungen ausreichend. Auf Systemen mit hoher Arbeitslast können jedoch Zeitlimitüberschreitungen auftreten, selbst wenn kein gegenseitiges Sperren vorliegt. In diesem Fall können Sie mit dieser Methode das Zeitlimit für Sperren vom Standardwert auf den erforderlichen Wert erhöhen, erhöhen, um solche Zeitlimitüberschreitungen und die daraufhin ausgelösten Ausnahmen zu verhindern. Wenn die Sperrstrategie auf NONE eingestellt ist, wird von der BackingMap kein Sperrenmanager verwendet. In diesem Fall ist die Definition des Attributs lockTimeout nicht erforderlich.
9. Attribut **copyMode** (optional, Standardeinstellung COPY_ON_READ_AND_COMMIT): Das Attribut copyMode bestimmt, ob eine Operation get für einen Eintrag in der BackingMap-Instanz den echten Wert, eine Kopie des Wertes oder einen Proxy für den Wert zurückgibt. Die gültigen Werte für das Attribut copyMode sind COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE und NO_COPY.

   Der Modus COPY_ON_READ_AND_COMMIT stellt sicher, dass eine Anwendung keine Referenz auf das Wertobjekt in der BackingMap besitzt und stattdessen mit einer Kopie des Wertes arbeitet.

   Der Modus COPY_ON_READ bietet im Vergleich mit dem Modus COPY_ON_READ_AND_COMMIT eine bessere Leistung, weil der Kopiervorgang beim Festschreiben einer Transaktion wegfällt. Um die Integrität der BackingMap-Daten zu gewährleisten, sichert die Anwendung zu, jede Referenz auf einen

Eintrag zu löschen, nachdem die Transaktion festgeschrieben wurde. Dieser Modus bewirkt, dass eine Methode `ObjectMap.get` eine Kopie des Wertes anstelle einer Referenz auf den Wert zurückgibt, um sicherzustellen, dass die Änderungen, die von der Anwendung an dem Wert vorgenommen werden, so lange keine Auswirkungen auf den BackingMap-Wert haben, bis die Transaktion festgeschrieben wird.

Der Modus `COPY_ON_WRITE` bietet im Vergleich mit dem Modus `COPY_ON_READ_AND_COMMIT` eine bessere Leistung, weil der Kopiervorgang beim ersten Aufruf der Methode `ObjectMap.get` für einen bestimmten Schlüssel wegfällt. Stattdessen gibt die Methode `ObjectMap.get` einen Proxy anstelle einer direkten Referenz auf das Wertobjekt zurück. Der Proxy stellt sicher, dass nur dann eine Kopie des Wertes erstellt wird, wenn die Anwendung die Methode `set` im Interface value aufruft.

Im Modus `NO_COPY` kann eine Anwendung zusichern, dass sie ein Wertobjekt, das mit einer Methode `ObjectMap.get` angefordert wird, nicht ändert, um Leistungsvorteile zu erzielen. Wenn Sie diesen Modus verwenden, wird der Wert nicht kopiert.

10. Attribut **valueInterfaceClassName** (optional): Wenn das Attribut copyMode auf `COPY_ON_WRITE` gesetzt ist, ist ein Attribut valueInterfaceClassName erforderlich. In allen anderen Modi wird das Attribut ignoriert. Im Modus mit Kopieren beim Schreiben (Copy on write) wird ein Proxy verwendet, wenn die Methode `ObjectMap.get` aufgerufen wird. Der Proxy stellt sicher, dass nur dann eine Kopie des Wertes erstellt wird, wenn die Anwendung die Methode `set` in der Klasse aufruft, die im Attribut valueInterfaceClassName angegeben ist.
11. Attribut **copyKey** (optional, Standardeinstellung `false`): Dieses Attribut bestimmt, ob der Schlüssel kopiert werden muss, wenn ein Map-Eintrag erstellt wird. Das Kopieren des Schlüsselobjekts ermöglicht der Anwendung, für alle ObjectMap-Operationen dasselbe Schlüsselobjekt zu verwenden. Wenn Sie das Attribut auf `true` setzen, wird das Schlüsselobjekt kopiert, wenn ein Map-Eintrag erstellt wird.
12. Attribut **nullValuesSupported** (optional, Standardeinstellung `true`): Die Unterstützung von Nullwerten bedeutet, dass ein Nullwert in einer Map gespeichert werden kann. Wenn dieses Attribut auf `true` gesetzt ist, werden Nullwerte in der ObjectMap-Instanz unterstützt, andernfalls nicht. Falls Nullwerte unterstützt werden und eine `get`-Operation einen Nullwert zurückgibt, kann dies bedeuten, dass der Wert `null` ist oder dass die Map den übergebenen Schlüssel nicht enthält.
13. Attribut **ttlEvictorType** (optional, Standardeinstellung `NONE`): Das Attribut ttlEvictorType bestimmt, wie die Verfallszeit eines BackingMap-Eintrags berechnet wird. Die gültigen Werte für dieses Attribut sind `CREATION_TIME`, `LAST_ACCESS_TIME` und `NONE`.

    `NONE` zeigt an, dass ein Eintrag keine Verfallszeit hat und so lange in der BackingMap bleiben kann, bis die Anwendung den Eintrag explizit entfernt oder ungültig macht.

    Der Wert `CREATION_TIME` zeigt an, dass die Verfallszeit die Summe aus der Erstellungszeit des Eintrags und dem Wert des Attributs timeToLive ist.

    `LAST_ACCESS_TIME` zeigt an, dass die Verfallszeit die Summe aus der letzten Zugriffszeit des Eintrags und dem Wert des Attributs timeToLive ist.
14. Attribut **timeToLive** (optional, Standardeinstellung `0`): Die Lebensdauer jedes Map-Eintrags in Sekunden. Der Standardwert `0` bedeutet, dass die Lebensdauer des Map-Eintrags nicht beschränkt ist, sofern die Anwendung den Ein-

trag nicht explizit entfernt oder ungültig macht. Hat das Attribut einen anderen Wert als 0, wird der TTL-Evictor verwendet, um den Map-Eintrag basierend auf diesem Wert zu löschen.

Die folgende XML-Datei, backingMapAttr.xml, zeigt eine backingMap-Beispielkonfiguration. Dieses Beispiel verwendet alle optionalen Attribute mit Ausnahme des Attributs pluginCollectionRef. Der Abschnitt „Element backingMapPluginCollection " auf Seite 141 enthält ein Beispiel für die Verwendung des Attributs pluginCollectionRef.

Datei *backingMapAttr.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
 <objectGrids>
  <objectGrid name="bookstore">
   <backingMap name="book" readOnly="true" numberOfBuckets="641"
    preloadMode="false" lockStrategy="OPTIMISTIC"
    numberOfLockBuckets="409" lockTimeout="30" copyMode="COPY_ON_WRITE"
    valueInterfaceClassName=
     "com.ibm.websphere.samples.objectgrid.CounterValueInterface"
    copyKey="true" nullValuesSupported="false"
    ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3000" />
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

Der folgende Beispielcode veranschaulicht, wie Sie die Konfiguration, die Sie mit der Datei backingMapAttr.xml im vorherigen Beispiel erstellt haben, über das Programm realisieren:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
bookMap.setReadOnly(true);
bookMap.setNumberOfBuckets(641);
bookMap.setPreloadMode(false);
bookMap.setLockStrategy(LockStrategy.OPTIMISTIC);
bookMap.setNumberOfLockBuckets(409);
bookMap.setLockTimeout(30);
// when setting copy mode to COPY_ON_WRITE, a valueInterface class is required
bookMap.setCopyMode(CopyMode.COPY_ON_WRITE,
com.ibm.websphere.samples.objectgrid.CounterValueInterface.class);
bookMap.setCopyKey(true);
bookMap.setNullValuesSupported(false);
bookMap.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);
bookMap.setTimeToLive(3000); // Lebensdauer auf 50 Minuten setzen
```

Element bean

Anzahl der Vorkommen (im Element objectGrid): 0 bis n

Anzahl der Vorkommen (im Element backingMapPluginCollection): 0 bis n

Untergeordnetes Element: Element property

Verwenden Sie das Element bean, um Plug-ins zu definieren. Plug-ins können mit ObjectGrid- und BackingMap-Instanzen verbunden werden.

ObjectGrid-Plug-ins:

- Plug-in TransactionCallback
- Plug-in ObjectGridEventListener
- Plug-in SubjectSource
- Plug-in MapAuthorization
- Plug-in SubjectValidation

BackingMap-Plug-ins:
- Plug-in Loader
- Plug-in ObjectTransformer
- Plug-in OptimisticCallback
- Plug-in Evictor
- Plug-in MapEventListener

Attribute für das Element bean

```
<bean
(1) id="TransactionCallback|ObjectGridEventListener|SubjectSource|
  MapAuthorization|SubjectValidation|Loader|ObjectTransformer|
  OptimisticCallback|Evictor|MapEventListener"
(2) className="class name"
/>
```

1. Attribut **id** (erforderlich): Gibt den Typ des zu erstellenden Plug-in an. Für ein Element bean, das ein untergeordnetes Element des Elements objectGrid ist, sind die gültigen Werte `TransactionCallback`, `ObjectGridEventListener`, `SubjectSource`, `MapAuthorization` und `SubjectValidation`. Für ein Element bean, das ein untergeordnetes Element des Elements backingMapPluginCollection ist, sind die gültigen Werte `Loader`, `ObjectTransformer`, `OptimisticCallback`, `Evictor` und `MapEventListener`. Jeder der gültigen Werte für das Attribut id stellt ein Interface dar.
2. Attribut **className** (erforderlich): Gibt den Namen der Klasse an, die zum Erstellen des Plug-in instanziert werden soll. Die Klasse muss das Interface für den Plug-in-Typ implementieren.

Die folgende Beispieldatei `bean.xml` zeigt, wie Sie das Element bean zum Konfigurieren von Plug-ins verwenden. In dieser XML-Datei wird der ObjectGrid-Instanz "bookstore" ein ObjectGridEventListener-Plug-in hinzugefügt. Der Klassenname für diese Bean ist `com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener`. Diese Klasse implementiert das Interface `com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener`.

Außerdem wird in der Beispieldatei `bean.xml` ein BackingMap-Plug-in definiert. Der BackingMap-Instanz wird ein Evictor-Plug-in hinzugefügt. Da die Bean-ID Evictor ist, muss das Attribut className eine Klasse angeben, die das Interface `com.ibm.websphere.objectgrid.plugins.Evictor` implementiert. Die Klasse `com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor` implementiert dieses Interface. Die BackingMap-Instanz verweist mit dem Attribut pluginCollectionRef auf ihre Plug-ins. Nähere Informationen zum Hinzufügen von Plug-ins zu einer BackingMap-Instanz finden Sie im Abschnitt „Interface BackingMap" auf Seite 47.

Datei *bean.xml*

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

```
   <bean id="ObjectGridEventListener"
     className="com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener" />
    <backingMap name="book" pluginCollectionRef="bookPlugins" />
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="bookPlugins">
   <bean id="Evictor"
     classname="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
  </backingMapPluginCollection>
 </backingMapPluginCollections>
</objectGridConfig>
```

Der folgende Code veranschaulicht, wie Sie die Konfiguration, die Sie mit der Datei `bean.xml` erstellt haben, über das Programm realisieren.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
 ("bookstore", false);
TranPropListener tranPropListener = new TranPropListener();
bookstoreGrid.addEventListener(tranPropListener);

BackingMap bookMap = bookstoreGrid.defineMap("book");
Evictor lruEvictor = new
 com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
bookMap.setEvictor(lruEvictor);
```

Element property

Anzahl der Vorkommen: 0 bis n

Untergeordnete Elemente: keine

Das Element property wird verwendet, um Plug-ins Merkmale hinzuzufügen. Der Name des Merkmals entspricht einer Methode `set` in der Klasse, die mit dem Attribut className der Bean, die das Merkmal enthält, angegeben wird.

Attribute für das Element property

```
<property
(1) name="name"
(2) type="java.lang.String|boolean|java.lang.Boolean|int|
  java.lang.Integer|double|java.lang.Double|byte|
  java.lang.Byte|short|java.lang.Short|long|
  java.lang.Long|float|java.lang.Float|char|
  java.lang.Character"
(3) value="value"
(4) description="description"
/>
```

1. Attribut **name** (erforderlich): Gibt den Namen des Merkmals an. Der Wert, der diesem Attribut zugeordnet ist, muss einer Methode `set` in der Klasse entsprechen, die mit dem Attribut className im Element bean angegeben ist. Wenn das Attribut className der Bean beispielsweise auf `com.ibm.MyPlugin` gesetzt ist und der Name des Merkmals `size` ist, muss die Klasse com.ibm.MyPlugin eine Methode `setSize` enthalten.
2. Attribut **type** (erforderlich): Gibt den Typ des Merkmals an. Dies ist der Typ des Parameters, der an die mit dem Attribut ″name″ angegebene Methode `set` übergeben wird. Die gültigen Werte sind primitive Java-Typen, ihre `java.lang`-Gegenstücke und `java.lang.String`. Die Attribute ″name″ und ″type″ müssen einer Methodendeklaration im Attribut className der Bean entsprechen. Wenn

der Name beispielsweise `size` und der Typ `int` ist, muss eine Methode `setSize(int)` in der Klasse vorhanden sein, die mit dem Attribut className für die Bean angegeben ist.

3. Attribut **value** (erforderlich): Gibt den Wert des Merkmals an. Dieser Wert wird in den Typ konvertiert, der mit dem Attribut ″type″ angegeben ist, und anschließend im Aufruf an die Methode `set`, die mit den Attributen ″name″ und ″type″ angegeben ist, als Parameter verwendet. Der Wert dieses Attributs wird nicht validiert. Der Entwickler des Plug-in muss sicherstellen, dass der übergebene Wert gültig ist. Falls der Parameter nicht gültig ist, kann der Entwickler eine Ausnahme des Typs IllegalArgumentException in der Methode set anzeigen. Der Entwickler muss dafür sorgen, dass diese Ausnahme angezeigt wird. Von ObjectGrid wird diese Ausnahme nicht implementiert.
4. Attribut **description** (optional): Verwenden Sie dieses Attribut, um eine Beschreibung des Merkmals zu schreiben.

Die folgende Datei `property.xml` zeigt, wie einer Bean ein Element property hinzugefügt wird. In diesem Beispiel wird einem Evictor ein Merkmal mit dem Namen `maxSize` und dem Typ `int` hinzugefügt. Der Evictor `com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor` hat eine Methodendeklaration, die der Methode `setMaxSize(int)` entspricht. Der ganzzahlige Wert `499` wird an die Methode `setMaxSize(int)` in der Klasse `com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor` übergeben.

Datei property.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
 <objectGrids>
  <objectGrid name="bookstore">
   <backingMap name="book" pluginCollectionRef="bookPlugins" />
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="bookPlugins">
   <bean id="Evictor"
    className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
    <property name="MaxSize" type="int" value="449"
     description="The maximum size of the LRU Evictor" />
   </bean>
  </backingMapPluginCOllection>
 </backingMapPluginCollections>
</objectGridConfig>
```

Mit dem folgenden Code erzielen Sie dasselbe Ergebnis wie mit der Datei `property.xml`:

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
 ("bookstore", false);

BackingMap bookMap = bookstoreGrid.defineMap("book");

LRUEvictor lruEvictor =
new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
// Wenn Sie die XML-Datei verwenden, bewirkt das hinzugefügte Merkmal
// den folgenden Aufruf
lruEvictor.setMaxSize(449);
bookMap.setEvictor(lruEvictor);
```

Element backingMapPluginCollections

Anzahl der Vorkommen: 0 bis 1

Untergeordnete Elemente: Element backingMapPluginCollection

Das Element backingMapPluginCollections ist ein Container für alle backingMapPluginCollection-Elemente. In der Datei `sample1.xml` enthält das Element backingMapPluginCollections ein Element backingMapPluginCollection mit der ID `collection1`.

Element backingMapPluginCollection

Anzahl der Vorkommen: 0 bis n

Untergeordnetes Element: Element bean

Im Element backingMapPluginCollection werden BackingMap-Plug-ins definiert. Jedes Element backingMapPluginCollection wird anhand seines Attributs id identifiziert. Jedes Element backingMap muss mit dem Attribut pluginCollectionRef im Element backingMap auf seine Plug-ins verweisen. Wenn mehrere BackingMap-Instanzen vorhanden sind, für die ähnliche Plug-ins konfiguriert werden müssen, kann jede auf dasselbe Element backingMapPluginCollection verweisen.

Attribute für das Element backingMapPluginCollection

```
<backingMapPluginCollection
(1) id="id"
/>
```

1. Attribut **id** (erforderlich): Die Kennung (ID) für das Element backingMapPluginCollection. Jede ID muss eindeutig sein. Auf die ID wird mit dem Attribut pluginCollectionRef des Elements backingMap verwiesen. Wenn der Wert eines Attributs pluginCollectionRef mit keiner ID eines Elements backingMapPluginCollection übereinstimmt, schlägt die XML-Validierung fehl. Es können beliebig viele backingMap-Elemente auf dasselbe Element backingMapPluginCollection verweisen.

Die folgende Datei `collection.xml` zeigt, wie das Element backingMapPluginCollection verwendet wird. In dieser Datei werden drei backingMap-Elemente definiert. Die backingMap-Elemente `book` und `customer` verwenden das backingMapPluginCollection-Element mit dem Namen `collection1`. Jede dieser beiden backingMap-Elemente hat einen eigenen LRUEvictor. Das backingMap-Element `employee` verweist auf das backingMapPluginCollection-Element `collection2`. Für dieses backingMap-Element ist ein LFUEvictor als Evictor-Plug-in und die Klasse EmployeeOptimisticCallbackImpl als OptimisticCallback-Plug-in definiert.

Datei collection.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
 <objectGrids>
  <objectGrid name="bookstore">
   <backingMap name="book" pluginCollectionRef="collection1" />
   <backingMap name="customer" pluginCollectionRef="collection1" />
   <backingMap name="employee" pluginCollectionRef="collection2" />
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
```

```
  <backingMapPluginCollection id="collection1">
   <bean id="Evictor"
     className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
  </backingMapPluginCollection>
  <backingMapPluginCollection id="collection2">
   <bean id="Evictor"
     className="com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor" />
   <bean id="OptimisticCallback"
     className="com.ibm.websphere.samples.objectgrid.
      EmployeeOptimisticCallBackImpl" />
  </backingMapPluginCollection>
 </backingMapPluginCollections>
</objectGridConfig>
```

Der folgende Code veranschaulicht, wie Sie die Konfiguration, die Sie mit der Datei `collection.xml` erstellt haben, über das Programm realisieren.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();

ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
 ("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
LRUEvictor bookEvictor = new LRUEvictor();
bookMap.setEvictor(bookEvictor);

BackingMap customerMap = bookstoreGrid.defineMap("customer");
LRUEvictor customerEvictor = new LRUEvictor();
customerMap.setEvictor(customerEvictor);

BackingMap employeeMap = bookstoreGrid.defineMap("employee");
LFUEvictor employeeEvictor = new LFUEvictor();
employeeMap.setEvictor(employeeEvictor);
OptimisticCallback employeeOptCallback =
 new EmployeeOptimisticCallbackImpl();
employeeMap.setOptimisticCallback(employeeOptCallback);
```

ObjectGrid-Konfiguration im gemischten Modus

Für die Konfiguration von ObjectGrid können Sie XML-Konfigurationsdateien und Programmschnittstellen in Kombination verwenden.

Für eine solche gemischte Konfiguration erstellen Sie zuerst eine XML-Datei, die dann an das Interface ObjectGridManager übergeben wird. Nachdem eine ObjectGrid-Instanz basierend auf der XML-Datei erstellt worden ist, kann die Instanz über das Programm bearbeitet werden, sofern die Methode `ObjectGrid.initialize()` noch nicht aufgerufen wurde. Die Methode `ObjectGrid.getSession()` ruft implizit die Methode `ObjectGrid.initialize()` auf, falls diese von der Anwendung noch nicht aufgerufen wurde.

Beispiel

Im Folgenden wird gezeigt, wie Sie eine Konfiguration im gemischten Modus erstellen. Die folgende Datei `mixedBookstore.xml` wird verwendet.

Datei mixedbookstore.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/ ..objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">
 <objectGrids>
  <objectGrid name="bookstore">
   <backingMap name="book" readOnly="true" numberOfBuckets="641"
```

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

Das folgende Code-Snippet, das zeigt, wie die XML-Datei an ObjectGridManager übergeben wird, und die neu erstellte ObjectGrid-Instanz werden weiter bearbeitet.

```
ObjectGridManager objectGridManager =
 ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore",
new URL("file:etc/test/document/mixedBookstore.xml"), true, false);
// An dieser Stelle ist ObjectGrid-Instanz erstellt, die in der
// XML-Datei definiert wurde.

// JEtzt wird die erstellte und konfigurierte BackingMap-Instanz geändert
BackingMap bookMap = bookstoreGrid.getMap("book");
// Die XML-Datei hat readOnly auf true gesetzt
// Hier wird das Attribut in false geändert
bookMap.setReadOnly(false);

// Die XML-Datei hat nullValuesSupported nicht gesetzt,
// d. h. es wird standardmäßig der Wert true angenommen.
// Jetzt wird das Attribut auf false gesetzt.
bookMap.setNullValuesSupported(false);

// Den in der XML-Datei definierten Evictor abrufen
// und das Attribut maxSize setzen
LFUEvictor lfuEvictor = (LFUEvictor) bookMap.getEvictor();
lfuEvictor.setMaxSize(443);

bookstoreGrid.initialize();
// Eine weitere Konfiguration dieser ObjectGrid-Instanz
// ist nach diesem Aufruf von initialize nicht zulässig
```

Kapitel 6. ObjectGrid mit WebSphere Application Server integrieren

Verwenden Sie ObjectGrid mit den Features, die von WebSphere Application Server bereitgestellt werden, um Ihre Anwendungen mit der Funktionalität von ObjectGrid zu erweitern.

Installieren Sie WebSphere Application Server und WebSphere Extended Deployment. Nachdem Sie WebSphere Extended Deployment installiert haben, können Sie Ihren J2EE-Anwendungen ObjectGrid-Funktionen hinzufügen.

Die ObjectGrid-API kann in einer für WebSphere Application Server bestimmten J2EE-Anwendung verwendet werden. Die Datei `ObjectGrid.jar` befindet sich nach der Installation von WebSphere Extended Deployment im Verzeichnis `\base\lib`. Zusätzlich zur Integration der ObjectGrid-API in das Programmiermodell für J2EE-Anwendungen können Sie die Unterstützung für verteilte Transaktionsweitergabe nutzen. Mit dieser Unterstützung können Sie ObjectGrid-Instanzen konfigurieren, die die Ergebnisse von Transaktionsfestschreibungen in einem Cluster mit WebSphere Application Server koordinieren.

1. Führen Sie die grundlegenden Programmierschritte durch, um eine J2EE-Anwendung für ObjectGrid zu befähigen. Nähere Informationen finden Sie im Abschnitt „ObjectGrid in eine J2EE-Umgebung integrieren".
2. Überwachen Sie die Leistungsdaten für Ihre ObjectGrid-Anwendungen. Nähere Informationen finden Sie im Abschnitt „ObjectGrid-Leistung mit WebSphere Application Server Performance Monitoring Infrastructure (PMI) überwachen" auf Seite 149.
3. Sie können Aktualisierungen von Cache-Einträgen während der Transaktionsfestschreibung in einer ObjectGrid zwischen zwei oder mehr ObjectGrid-Instanzen koordinieren, die in einem oder mehreren Clustern denselben Instanznamen haben. Nähere Informationen finden Sie im Abschnitt „Weitergabe von verteilten Transaktionen in ObjectGrid" auf Seite 156.
4. Wenn ObjectGrid integriert ist, können die Transaktionen von einem externen Transaktionskoordinator gestartet und beendet werden. Nähere Informationen finden Sie im Abschnitt „ObjectGrid und Interaktionen mit externen Transaktionen" auf Seite 161.
5. Verwenden Sie das Partitionierungs-Feature (WPF) mit ObjectGrid. Das Feature ObjectGrid bietet die Möglichkeit, Schlüssel/Wert-Paare nach Transaktion zwischenzuspeichern, und das Partitionierungs-Feature unterstützt die kontextbasierte Weiterleitung anhand von Objektmerkmalen. Nähere Informationen finden Sie im Abschnitt „ObjectGrid und WPF integrieren" auf Seite 164.

Sie können ObjectGrid auch mit JMS verwenden, um Änderungen zwischen unterschiedlichen Schichten oder in Umgebungen mit unterschiedlichen Plattformen zu verteilen. Nähere Informationen finden Sie im Abschnitt „Java Message Service für die Verteilung von Transaktionsänderungen" auf Seite 202.

ObjectGrid in eine J2EE-Umgebung integrieren

ObjectGrid unterstützt die Programmiermodelle mit Servlets und Enterprise JavaBeans in der J2EE-Umgebung.

Dieser Abschnitt beschreibt die grundlegenden Programmierschritte für das Aktivieren einer J2EE-Anwendung mit ObjectGrid.

1. Definieren Sie eine ObjectGrid-Konfiguration. Sie können eine ObjectGrid-Konfiguration mit XML-Dateien, mit der Programmierschnittstelle oder mit einer Kombination von XML-Dateien und Programmierschnittstelle definieren. Nähere Informationen hierzu finden Sie im Abschnitt ObjectGrid-Konfiguration.
2. Erstellen Sie ein URL-Objekt. Wenn die ObjectGrid-Konfiguration in einer XML-Datei enthalten ist, erstellen Sie ein URL-Objekt, das auf diese XML-Datei zeigt. Sie können dieses URL-Objekt verwenden, um mit der API ObjectGridManager ObjectGrid-Instanzen zu erstellen. Wenn die XML-Datei mit der ObjectGrid-Konfiguration in einer WAR- oder JAR-Datei enthalten ist, kann sie als Ressourcen für den Klassenlader des Web- und EJB-Moduls aufgerufen werden. Wenn die XML-Datei mit der ObjectGrid-Konfiguration beispielsweise im Ordner WEB−INF der WAR-Datei des Webmoduls enthalten ist, können die in dieser WAR-Datei enthaltenen Servlets nach dem folgenden Muster ein URL-Objekt erstellen:

   ```
   URL url =className.class.getClassLoader().
   getResource("META-INF/objectgrid-definition.xml");
   URL objectgridUrl = ObjectGridCreationServlet.class.getClassLoader().
   getResource("WEB-INF/objectgrid-definition.xml");
   ```

3. ObjectGrid-Instanzen erstellen oder abrufen. Verwenden Sie die API ObjectGridManager, um ObjectGrid-Instanzen abzurufen und zu erstellen. Die API ObjectGridManager ermöglicht Ihnen, ObjectGrid-Instanzen in einer XML-Datei zu erstellen oder Dienstprogrammmethoden zu verwenden, um eine einfache ObjectGrid-Instanz zu erstellen. Anwendungen müssen die API ObjectGridManagerFactory verwenden, um eine Referenz auf die API ObjectGridManager abzurufen. Schauen Sie sich das folgende Codebeispiel an:

   ```
   import com.ibm.websphere.objectgrid.ObjectGridManager;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory ;
   ...
   ObjectGridManager objectGridManager = ObjectGridManagerFactory.
   getObjectGridManager();
   ObjectGrid ivObjectGrid = objectGridManager.
   createObjectGrid(objectGridName, objectgridUrl, true, true);
   ```

   Nähere Informationen zur API ObjectGridManager finden Sie im Abschnitt Interface ObjectGridManager.
4. ObjectGrid-Instanzen initialisieren. Verwenden Sie die Methode ″initialize″ im Interface ObjectGrid, um mit dem Starten der ObjectGrid-Instanz und der Session-Instanzen zu beginnen. Diese Methode ″initialize″ wird als optional eingestuft, weil der erste Aufruf der Methode getSession eine implizite Initialisierung durchführt. Nach dem Aufruf dieser Methode gilt die ObjectGrid-Konfiguration als abgeschlossen und für die Verwendung in der Laufzeitumgebung bereit. Alle weiteren Aufrufe von Konfigurationsmethoden, wie z. B. der Aufruf der Methode defineMap(String mapName), führt zu einer Ausnahme.
5. Session- und ObjectMap-Instanz abrufen. Eine Sitzung ist ein Container für die ObjectGrid-Maps, die von ObjectMap-Instanzen dargestellt werden. Ein Thread muss sein eigenes Session-Objekt abrufen, um mit der ObjectGrid-Basisinstanz zu interagieren. Sie können sich diese Technik so vorstellen, dass eine Sitzung jeweils nur von einem Thread verwendet werden kann. Die Sitzung kann von mehreren Threads verwendet werden, wenn die Threads nacheinander ausgeführt werden. Wenn jedoch eine J2EE-Verbindung oder Transaktionsinfrastruktur verwendet wird, kann das Session-Objekt nicht von mehreren Threads gemeinsam genutzt werden. Eine Analogie für dieses Objekt ist eine JDBC-Verbindung zu einer Datenbank.

Eine ObjectMap-Map ist eine interne Kennung (Handle) für eine benannte Map. Maps müssen homogene Schlüssel und Werte haben. Eine ObjectMap-Instanz kann nur von dem Thread verwendet werden, der derzeit der Sitzung zugeordnet ist, die für den Abruf dieser ObjectMap-Instanz verwendet wurde. Session- und ObjectMap-Objekte können nicht von mehreren Threads gleichzeitig genutzt werden. In einer Transaktion werden Schlüsselwörter angewendet. Bei der Zurücksetzung einer Transaktion werden alle Schlüsselwortzuordnungen, die während der Transaktion vorgenommen wurden, rückgängig gemacht-.Schauen Sie sich dazu das folgende Codebeispiel an:

```
Session ivSession = ivObjectGrid.getSession();
ObjectMap ivEmpMap = ivSession.getMap("employees");
ObjectMap ivOfficeMap = ivSession.getMap("offices");
ObjectMap ivSiteMap = ivSession.getMap("sites");
ObjectMap ivCounterMap = ivSession.getMap("counters");
```

6. Sitzung starten, Objekte lesen oder schreiben und die Sitzung festschreiben oder rückgängig machen. Map-Operationen müssen in einem Transaktionskontext ausgeführt werden. Die Methode begin des Session-Objekts wird verwendet, um einen expliziten Transaktionskontext zu starten. Nach dem Start der Sitzung können Anwendungen mit der Durchführung von Map-Operationen beginnen. Zu den am häufigsten verwendeten Operationen gehören Aufrufe der Methoden `update`, insert und remove für Objekte in Maps. Am Ende von Map-Operationen wird die Methode commit oder rollback des Session-Objekts aufgerufen, um den expliziten Transaktionskontext festzuschreiben bzw. rückgängig zu machen. Schauen Sie sich das folgende Programmierbeispiel an:

```
ivSession.begin();
Integer key = new Integer(1);
if (ivCounterMap.containsKey(key) == false) {
 ivCounterMap.insert(key, new Counter(10));
}
ivSession.commit();
```

Sie haben die grundlegenden Programmierschritte ausgeführt, um eine J2EE-Anwendung mit ObjectGrid zu aktivieren.

Nähere Informationen finden Sie in den Abschnitten ″J2EE-Anwendungen mit Unterstützung von ObjectGrid entwickeln″ und ″Hinweise zur Integration von J2EE-Anwendungen und ObjectGrid″.

J2EE-Anwendungen mit Unterstützung von ObjectGrid entwickeln

Mit dieser Task können Sie den Erstellungspfad oder Klassenpfad für J2EE-Anwendungen erstellen, die ObjectGrid unterstützen. Der Klassenpfad muss die Datei `objectgrid.jar` enthalten, die sich im Verzeichnis `$install_root/lib` befindet.

Entwickeln Sie eine J2EE-Anwendung, die ObjectGrid unterstützt. Nähere Informationen hierzu finden Sie im Abschnitt ObjectGrid in eine J2EE-Umgebung integrieren.

Diese Task veranschaulicht, wie Sie in IBM Rational Software Development Platform Version 6.0 die Datei `objectgrid.jar` in den Erstellungspfad (Build-Pfad) einfügen.

1. Klicken Sie in der Sicht ″Projekt-Explorer″ der J2EE-Perspektive mit der rechten Maustaste auf das **Web**- oder **EJB**-Projekt und wählen Sie **Eigenschaften** aus. Das Fenster ″Eigenschaften″ erscheint.

2. Wählen Sie in der linken Anzeige die Option **Java-Erstellungspfad** aus, klicken Sie in der rechten Anzeige auf das Register **Bibliotheken** und klicken Sie anschließend auf **Variable hinzufügen**. Das Fenster **Neuer Eintrag in Variable CLASSPATH** erscheint.
3. Klicken Sie auf **Variablen konfigurieren**, um das Fenster **Vorgabe** zu öffnen.
4. Fügen Sie einen neuen Eintrag für die Variable hinzu.
   a. Klicken Sie auf **Neu**.
   b. Geben Sie `OBJECTGRID_JAR` im Feld **Name** ein. Klicken Sie auf **Datei**, um das Fenster **JAR-Auswahl** zu öffnen.
   c. Navigieren Sie in das Verzeichnis `/lib` und klicken Sie auf die Datei **objectgrid.jar**. Klicken Sie anschließend auf **Öffnen**, um das Fenster **JAR-Auswahl** zu schließen.
   d. Klicken Sie auf **OK** , um das Fenster **Neuer Variableneintrag** zu schließen.

   Die Variable `OBJECTGRID_JAR` wird in der Liste der Klassenpfadvariablen angezeigt.
5. Klicken Sie auf **OK**, um das Fenster **Vorgabe** zu schließen.
6. Wählen Sie in der Variablenliste die Variable `OBJECTGRID_JAR` aus und klicken Sie auf **OK**, um das Fenster **Neuer Eintrag in Variable CLASSPATH** zu schließen. Die Variable `OBJECTGRID_JAR` wird in der Anzeige **Bibliotheken** angezeigt.
7. Klicken Sie auf **OK**, um das Fenster **Eigenschaften** zu schließen.

Fügen Sie in IBM Rational Software Development Platform Version 6.0 die Datei `objectgrid.jar` in den Erstellungspfad ein.

Hinweise für die Integration von J2EE-Anwendungen und ObjectGrid

Verwenden Sie die folgenden Hinweise, wenn Sie eine J2EE-Anwendung (Java 2 Platform, Enterprise Edition) mit ObjectGrid integrieren.

Startup-Beans und ObjectGrid

Sie können Startup-Beans für eine Anwendung verwenden, um eine ObjectGrid-Instanz zu starten, wenn eine Anwendung gestartet wird, und die ObjectGrid-Instanz zu löschen, wenn die Anwendung gestoppt wird. Eine Startup-Bean ist eine Stateless-Session-Bean mit einem Remote-Home-Interface com.ibm.websphere.startupservice.AppStartUpHome und einem Remote-Interface com.ibm.websphere.startupservice.AppStartUp. Wenn WebSphere Application Server eine Enterprise JavaBean (EJB) sieht, wird die Startup-Bean erkannt. Das Remote-Interface hat zwei Methoden, die Methode ″start″ und die Methode ″stop″. Mit der Methode ″start″ können Sie das Grid booten, und mit der Methode ″stop″ können Sie das Grid löschen. Die Anwendung kann eine Referenz auf das Grid verwalten. Hierfür wird bei Bedarf mit der Methode ObjectGridManager.getObjectGrid eine Referenz abgerufen. Nähere Informationen hierzu finden Sie im Abschnitt Interface ObjectGridManager.

Klassenlader und ObjectGrid-Instanzen

Sie müssen vorsichtig vorgehen, wenn Sie eine ObjectGrid-Instanz für mehrere Anwendungsmodule gemeinsam nutzen, die unterschiedliche Klassenlader verwenden. Anwendungsmodule, die unterschiedliche Klassenlader verwenden, funktionieren nicht und lösen in der Anwendung Ausnahmebedingungen bei der Klassenumsetzung aus. Eine ObjectGrid-Instanz darf nur von Anwendungs-

modulen gemeinsam genutzt werden, die denselben Klassenlader verwenden, oder wenn die Anwendungsobjekte, z. B. die Plug-ins, Schlüssel und Werte, in einem gemeinsamen Klassenlader enthalten sind.

Gültigkeitsdauer von ObjectGrid-Instanzen in einem Servlet verwalten

Sie können die Gültigkeitsdauer von ObjectGrid-Instanzen mit den Methoden init und destroy eines Servlet verwalten. Verwenden Sie die Methode init, um die erforderlichen ObjectGrid-Instanzen für die Anwendung zu erstellen und zu initialisieren. Nach dem Erstellen und Zwischenspeichern der ObjectGrid-Instanzen können Sie die Instanzen mit der API ObjectGridManager mit ihrem Namen abrufen. Verwenden Sie die Methode "destroy", um die ObjectGrid-Instanzen zu löschen und Systemressourcen freizugeben. Nähere Informationen hierzu finden Sie im Abschnitt Interface ObjectGridManager.

ObjectGrid-Leistung mit WebSphere Application Server Performance Monitoring Infrastructure (PMI) überwachen

ObjectGrid unterstützt Performance Monitoring Infrastructure (PMI) in einem Anwendungsserver von WebSphere Application Server oder WebSphere Extended Deployment. PMI erfasst Leistungsdaten für Laufzeitanwendungen und stellt Interfaces bereit, die externe Anwendungen für die Überwachung von Leistungsdaten unterstützen.

Nähere Informationen zu den von ObjectGrid bereitgestellten Statistiken finden Sie im Abschnitt ObjectGrid-Statistiken.

ObjectGrid verwendet das PMI-Feature von WebSphere Application Server, um seine eigene PMI-Instrumentierung hinzuzufügen. Mit dieser Methode können Sie ObjectGrid-PMI in der Administrationskonsole oder mit JMX-Interfaces (Java Management Extensions) aktivieren und inaktivieren. Außerdem können Sie mit den Standard-PMI- und JMX-Interfaces, die von Überwachungstools wie Tivoli Performance Viewer verwendet werden, auf ObjectGrid-Statistiken zugreifen.

1. Aktivieren Sie PMI für ObjectGrid. Sie müssen PMI aktivieren, um die PMI-Statistiken anzeigen zu können. Nähere Informationen finden Sie im Abschnitt „PMI für ObjectGrid aktivieren" auf Seite 152.
2. Rufen Sie die PMI-Statistiken für ObjectGrid ab. Überprüfen Sie die Leistung Ihrer ObjectGrid-Anwendungen in Tivoli Performance Viewer. Nähere Informationen finden Sie im Abschnitt „PMI-Statistiken für ObjectGrid abrufen" auf Seite 155.

ObjectGrid-Statistiken

ObjectGrid stellt zwei PMI-Module (Performance Monitoring Infrastructure) bereit: objectGridModule und mapModule.

Modul objectGridModule

Das Modul objectGridModule enthält eine Zeitstatistik, die Antwortzeit der einer Transaktion. Eine ObjectGrid-Transaktion ist definiert als die Dauer zwischen dem Aufruf der Methode Session.begin und dem Aufruf der Methode Session.commit. Diese Dauer wird als Antwortzeit der Transaktion überwacht.

Das Stammelement des Moduls objectGridModule ist das Element ObjectGrids, das als Eingangspunkt für die ObjectGrid-Statistiken dient. Dieses Stammelement enthält ObjectGrid-Instanzen als untergeordnete Elemente, die wiederum Transaktionstypen als untergeordnete Elemente besitzen. Die Antwortzeitstatistik wird jedem Transaktionstyp zugeordnet. Die Struktur des Moduls objectGrid-Module können Sie der folgenden Abbildung entnehmen:

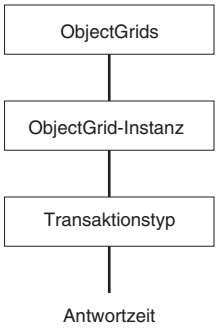


Abbildung 3. Struktur des Moduls ObjectGridModule

Die folgende Abbildung zeigt ein Beispiel für die Struktur des PMI-Moduls für ObjectGrid. In diesem Beispiel sind zwei ObjectGrid-Instanzen im System vorhanden: objectGrid1 und objectGrid2. Die Instanz objectGrid1 hat zwei Transaktionstypen, aktualisieren (update) und lesen (read), und die Instanz objectGrid2 nur einen Transaktionstyp, aktualisieren (update).

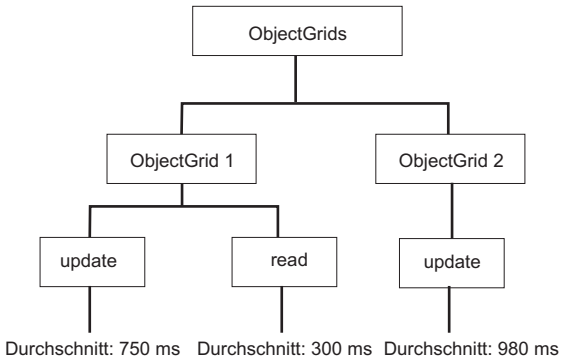


Abbildung 4. Struktur des PMI-Moduls für ObjectGrid

Transaktionstypen werden von Anwendungsentwicklern definiert, weil diese wissen, welche Typen von Transaktionen in den Anwendungen verwendet werden. Der Transaktionstyp wird mit der folgenden Methode Session.setTransactionType(String) definiert:

```
/**
* Definiert den Transaktionstyp für zukünftige Transaktionen
*
* Nach dem Aufruf dieser Methode haben alle zukünftigen Transaktionen
* denselben Typ, sofern kein anderer Transaktionstyp definiert wird.
* Wenn Sie keinen Transaktionstyp definieren, wird der Standardtyp
* TRANSACTION_TYPE_DEFAULT verwendet.
*
* Transaktionstypen werden hauptsächlich für die Überwachung statistischer
* Daten verwendet. Die Idee ist, Transaktionen, die
* dieselben Merkmale haben, einer Kategorie (Typ) zuzuordnen, damit jeweils
* eine Antwortzeitstatistik für die Überwachung jedes Transaktionstyps
* verwendet werden kann.
*
```

```
* Diese Art der Überwachung ist sinnvoll, wenn Ihre Anwendung unterschiedliche
* Transaktionstypen verwendet.
* Die Verarbeitungsdauer mancher Typen von Transaktionen, wie z. B. update,
* ist länger als die von anderen. Auf der Basis des Transaktionstyps können
* unterschiedliche Transaktionen mit unterschiedlichen Statistiken überwacht
* werden. Auf diese Weise sind die Statistiken effizienter.
*
* @param tranType Der Transaktionstyp für zukünftige Transaktionen
*/
void setTransactionType(String tranType);
```

Das folgende Beispiel setzt den Transaktionstyp auf `updatePrice`:

```
// Transaktionstyp auf updatePrice setzen
// Die Zeit zwischen session.begin() und session.commit() wird in der Zeitstatistik
// für "updatePrice" überwacht.
session.setTransactionType("updatePrice");
session.begin();
map.update(stockId, new Integer(100));
session.commit();
```

Die erste Zeile zeigt an, dass die nachfolgende Transaktion den Typ `updatePrice` hat. Die ObjectGrid-Instanz enthält eine Statistik `updatePrice` für die Sitzung im Beispiel. Mit JMX-Interfaces (Java Management Extensions) können Sie die Antwortzeit für updatePrice-Transaktionen abrufen. Außerdem können Sie die zusammengefasste Statistik für alle Transaktionstypen der angegebenen ObjectGrid-Instanz abrufen.

Modul mapModule

Das PMI-Modul mapModule enthält drei Statistiken für ObjectGrid-Maps:

- **Map hit rate:** Diese BoundedRangeStatistic-Statistik überwacht die Trefferrate für eine Map. Die Trefferrate ist ein variabler Wert zwischen `0` und `100` einschließlich, der den Prozentsatz der Map-Treffer im Vergleich mit den get-Operationen für die Map darstellt.
- **Number of entries**: Diese CountStatistic-Statistik überwacht die Anzahl der Einträge in der Map.
- **Loader batch update response time**: Diese TimeStatistic-Statistik überwacht die Antwortzeit für die Stapelaktualisierungsoperation (batch update) des Loader.

Das Stammelement des Moduls mapModule ist das ObjectGrid-Element Maps, das als Eingangspunkt für die ObjectGrid-Map-Statistiken dient. Dieses Stammelement hat ObjectGrid-Instanzen als untergeordnete Elemente, die wiederum Map-Instanzen als untergeordnete Elemente besitzen. Jede Map-Instanz hat die drei aufgelisteten Statistiken. Die Struktur des Moduls mapModule können Sie der folgenden Abbildung entnehmen:

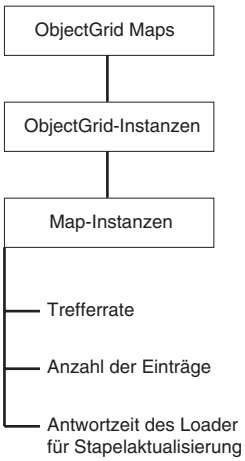


Abbildung 5. Struktur des Moduls mapModule

Die folgende Abbildung zeigt ein Beispiel für die Struktur des Moduls mapModule:

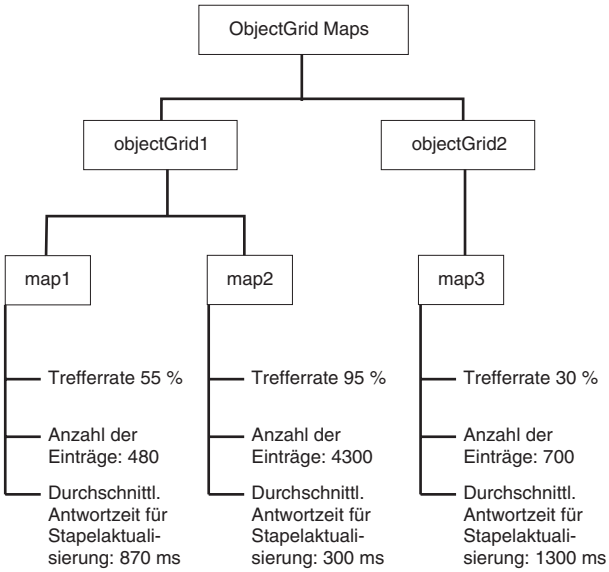


Abbildung 6. Beispiel für die Struktur des Moduls mapModule

PMI für ObjectGrid aktivieren

Sie können WebSphere Application Server Performance Monitoring Infrastructure (PMI) verwenden, um Statistiken auf jeder Ebene zu aktivieren und zu inaktivieren. Beispielsweise könnten Sie nur die Trefferratenstatistik für eine bestimmte Map aktivieren und die beiden anderen Statistiken zur Anzahl der Einträge in der Map und zur Antwortzeit des Loader für die Stapelaktualisierung inaktivieren. Dieser Abschnitt beschreibt, wie Sie mit der Administrationskonsole und den wsadmin-Scripts PMI für ObjectGrid aktivieren.

Mit WebSphere Application Server PMI haben Sie einen differenzierten Mechanismus, mit dem Sie Statistiken auf jeder Ebene aktivieren und inaktivieren können. Beispielsweise könnten Sie nur die Trefferratenstatistik für eine bestimmte Map aktivieren und die beiden anderen Statistiken zur Anzahl der Einträge in der Map

und zur Antwortzeit des Loader für die Stapelaktualisierung inaktivieren. Dieser Abschnitt beschreibt, wie Sie mit der Administrationskonsole und den wsadmin-Scripts PMI für ObjectGrid aktivieren.

1. Öffnen Sie die Administrationskonsole, z. B. mit http://localhost:9060/ibm/console.
2. Klicken Sie auf **Überwachung und Optimierung > Performance Monitoring Infrastructure >** *Servername*.
3. Vergewissern Sie sich, dass **Performance Monitoring Infrastructure (PMI) aktivieren** ausgewählt ist. Diese Einstellung ist standardmäßig aktiviert. Sollte die Einstellung nicht aktiviert sein, wählen Sie das Markierungsfeld aus und starten Sie anschließend den Server erneut.
4. Klicken Sie auf **Benutzerdefiniert**. Wählen Sie in der Konfigurationsstruktur die Module **ObjectGrid** und **ObjectGrid Maps** aus. Aktivieren Sie die Statistiken für jedes Modul.

Die Transaktionstypkategorie für ObjectGrid-Statistiken wird zur Laufzeit erstellt. In der Anzeige "Laufzeit" sehen Sie nur die Unterkategorien der ObjectGrid- und Map-Statistiken.

Sie können beispielsweise die folgenden Schritte ausführen, um PMI-Statistiken für die Beispielanwendung zu aktivieren:

1. Starten Sie die Anwendung mit der Webadresse `http://Host:Port/ObjectGridSample`. Host und Port stehen für den Hostnamen und die HTTP-Port-Nummer des Servers, in dem die Beispielanwendung installiert ist.
2. Klicken Sie in der Beispielanwendung auf **ObjectGridCreationServlet** und anschließend auf die Aktionsschaltflächen 1, 2, 3, 4 und 5, um verschiedene Aktionen für ObjectGrid und die Maps zu generieren. Schließen Sie diese Servlet-Seite noch nicht.
3. Kehren Sie in die Administrationskonsole zurück und klicken Sie auf **Überwachung und Optimierung > Performance Monitoring Infrastructure >** *Servename*. Klicken Sie auf das Register **Laufzeit**.
4. Klicken Sie auf den Radioknopf **Benutzerdefiniert**.
5. Erweitern Sie in der Laufzeitstruktur das Modul **ObjectGrid Maps** und klicken Sie anschließend auf **clusterObjectGrid**. In der Gruppe **ObjectGrid Maps** ist eine ObjectGrid-Instanz mit dem Namen clusterObjectGrid und in der Gruppe clusterObjectGrid sind vier Maps vorhanden: counters, employees, offices und sites. Die **ObjectGrids**-Instanz enthält eine clusterObjectGrid-Instanz. In dieser Instanz ist ein Transaktionstyp mit dem Namen `DEFAULT` vorhanden.
6. Sie können die gewünschten Statistiken aktivieren. Zu Demonstrationszwecken können Sie für die Map "employees" die Statistik für die Anzahl der Einträge (**number of map entries**) und für den Transaktionstyp DEFAULT die Statistik für die Antwortzeit (**transaction response time**) aktivieren.

Die Aktivierung von PMI können Sie mit Scripting automatisieren. Nähere Informationen hierzu finden Sie im Abschnitt PMI für ObjectGrid mit Scripting aktivieren.

PMI für ObjectGrid mit Scripting aktivieren

Sie können das Aktivieren von PMI für ObjectGrid mit dem Tool wsadmin automatisieren.

Ihr Anwendungsserver muss gestartet und eine ObjectGrid-fähige Anwendung muss installiert sein. Außerdem müssen Sie sich anmelden und das Tool wsadmin

verwenden können. Nähere Informationen zum Tool wsadmin finden Sie im Artikel Scripting (wsadmin) verwenden im WebSphere Extended Deployment Version 6.0.x Information Center.

Verwenden Sie diese Task, um das Aktivieren von PMI zu automatisieren. Informationen zum Aktivieren von PMI mit der Administrationskonsole finden Sie im Abschnitt PMI für ObjectGrid aktivieren.

1. Öffnen Sie eine Eingabeaufforderung. Navigieren Sie in das Verzeichnis *Installationsstammverzeichnis*/bin. Geben Sie wsadmin ein, um das Befehlszeilentool wsadmin zu starten.
2. Ändern Sie die PMI-Laufzeitkonfiguration für ObjectGrid. Prüfen Sie, ob PMI für den Server aktiviert ist. Setzen Sie dazu die folgenden Befehle ab:

   ```
   wsadmin>set s1 [$AdminConfig getid /Cell:ZELLENNAME/Node:KNOTENNAME/Server:
    NAME_DES_ANWENDUNGSSERVERS/]
   wsadmin>set pmi [$AdminConfig list PMIService $s1]
   wsadmin>$AdminConfig show $pmi.
   ```

   Wenn PMI nicht aktiviert ist, führen Sie die folgenden Befehle aus, um PMI zu aktivieren:

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

   Zum Aktivieren von PMI müssen Sie den Server erneut starten.
3. Setzen Sie die Variablen, um aus der Statistikgruppe eine eigene Gruppe zu machen. Führen Sie die folgenden Befehle aus:

   ```
   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. Definieren Sie eine benutzerdefinierte Statistikgruppe: Führen Sie den folgenden Befehl aus:

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
   ```
5. Setzen Sie die Variablen, um die PMI-Statistiken objectGridModule zu aktivieren. Führen Sie die folgenden Befehle aus:

   ```
   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. Setzen Sie die Zeichenfolge für die Statistiken. Führen Sie den folgenden Befehl aus:

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params $sigs
   ```
7. Setzen Sie die Variablen, um die PMI-Statistik mapModule zu aktivieren. Führen Sie die folgenden Befehle aus:

   ```
   wsadmin>set params2 [java::new {java.lang.Object[]} 2]
   wsadmin>$params2 set 0 [java::new java.lang.String mapModule=*]
   wsadmin>$params2 set 1 [java::new java.lang.Boolean false]
   wsadmin>set sigs2 [java::new {java.lang.String[]} 2]
   wsadmin>$sigs2 set 0 java.lang.String
   wsadmin>$sigs2 set 1 java.lang.Boolean
   ```
8. Setzen Sie die Zeichenfolge für die Statistiken. Führen Sie den folgenden Befehl aus:

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

Mit diesen Schritten haben Sie die PMI-Laufzeitumgebung für ObjectGrid aktiviert, die PMI-Konfiguration aber nicht geändert. Wenn Sie den Anwendungsserver erneut starten, gehen die PMI-Einstellungen bis auf die Aktivierung verloren.

Nachdem Sie PMI aktiviert haben, können Sie die PMI-Statistiken mit der Administrationskonsole oder mit Scripting anzeigen. Nähere Informationen finden Sie in den Abschnitten „PMI-Statistiken für ObjectGrid abrufen" und „PMI-Statistiken für ObjectGrid mit Scripts abrufen".

PMI-Statistiken für ObjectGrid abrufen

Rufen Sie die Leistungsstatistiken für Ihre ObjectGrid-Anwendungen ab.

Nachdem Sie die ObjectGrid-Statistiken aktiviert haben, können Sie sie abrufen. Informationen zum Aktivieren von PMI für ObjectGrid finden Sie im Abschnitt PMI für ObjectGrid aktivieren.

Verwenden Sie diese Task, um die Leistungsstatistiken für Ihre ObjectGrid-Anwendungen abzurufen.

1. Öffnen Sie die Administrationskonsole. Geben Sie beispielsweise http://localhost:9060/ibm/console ein.
2. Klicken Sie auf **Überwachung und Optimierung > Performance Viewer > Aktuelle Aktivität**.
3. Klicken Sie auf den Server, den Sie mit Tivoli Performance Viewer überwachen möchten, und aktivieren Sie die Überwachung.
4. Klicken Sie auf den Server, um die Seite "Performance Viewer" anzuzeigen.
5. Erweitern Sie die Konfigurationsstruktur. Klicken Sie auf **ObjectGrid-Caches > clusterObjectGrid** und wählen Sie **employees** aus. Klicken Sie auf **ObjectGrids > clusterObjectGrid** und wählen Sie **DEFAULT** aus.
6. In der ObjectGrid-Beispielanwendung können Sie zum Servlet ObjectGridCreationServlet zurückkehren und auf die Schaltfläche 1, **populate maps**, klicken. Die Statistiken werden im Viewer angezeigt.

Sie sehen die ObjectGrid-Statistiken in Tivoli Performance Viewer.

Der Abruf der Statistiken kann mit Java Management Extensions (JMX) oder mit dem Tool wsadmin automatisiert werden. Nähere Informationen hierzu finden Sie im Abschnitt „PMI-Statistiken für ObjectGrid mit Scripts abrufen".

PMI-Statistiken für ObjectGrid mit Scripts abrufen

Verwenden Sie diese Task, um Leistungsstatistiken für Ihre ObjectGrid-Anwendungen abzurufen.

Aktivieren Sie Performance Monitoring Infrastructure (PMI) in Ihrer Anwendungsserverumgebung. Nähere Informationen hierzu finden Sie in den Artikeln PMI für ObjectGrid aktivieren und PMI für ObjectGrid mit Scripts aktivieren. Außerdem müssen Sie sich anmelden und das Tool wsadmin verwenden können. Nähere Informationen zum Tool wsadmin finden Sie im Artikel "Scripting (wsadmin) verwenden" im WebSphere Extended Deployment Version 6.0.x Information Center.

Verwenden Sie diese Task, um Leistungsstatistiken für Ihre Anwendungsserverumgebung abzurufen. Die abrufbaren ObjectGrid-Statistiken sind im Abschnitt „ObjectGrid-Statistiken" auf Seite 149 beschrieben.

1. Öffnen Sie eine Eingabeaufforderung. Navigieren Sie in das Verzeichnis *Installationsstammverzeichnis*/bin. Geben Sie wsadmin ein, um das Befehlszeilentool wsadmin zu starten.
2. Setzen Sie die Variablen für die Umgebung. Führen Sie die folgenden Befehle aus:

```
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. Setzen Sie die Variablen, um mapModule-Statistiken abzurufen. Führen Sie die folgenden Befehle aus:

```
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. Rufen Sie die mapModule-Statistiken ab. Führen Sie den folgenden Befehl aus:

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

5. Setzen Sie die Variablen, um objectGridModule-Statistiken abzurufen. Führen Sie die folgenden Befehle aus:

```
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. Rufen Sie die objectGridModule-Statistiken ab. Führen Sie den folgenden Befehl aus:

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

Nähere Informationen zu den zurückgegebenen Statistiken finden Sie im Abschnitt „ObjectGrid-Statistiken" auf Seite 149.

Weitergabe von verteilten Transaktionen in ObjectGrid

Die Weitergabe von verteilten Transaktionen wird verwendet, um Aktualisierungen von Cache-Einträgen während der Transaktionsfestschreibung in einer ObjectGrid zwischen zwei oder mehr ObjectGrid-Instanzen zu koordinieren, die in einem oder mehreren Clustern denselben Instanznamen haben.

Bei der Weitergabe verteilter Transaktionen können Sie Invalidierungsanforderungen oder Schlüssel- und Wertaktualisierungen an Peers weitergeben, indem Sie dieselbe ObjectGrid-Instanz instanzieren.

Angenommen, Sie haben Cache-Instanzen, die hauptsächlich nur gelesen werden. Jetzt aktualisieren Sie ein bestimmtes Schlüssel/Wert-Paar. Um sicherzustellen, dass die anderen Cluster-Member keine veralteten Informationen für diesen Schlüssel bereitstellen, müssen Sie die Member der Application-Server-Cluster koordinieren, die Benutzern Daten bereitstellen. Sie können eine der folgenden Strategien verwenden:

- **Die Daten in anderen Servern ungültig machen.**

Sie haben beispielsweise Daten in einem Datenspeicher, z. B. einer Datenbank, gespeichert und einen ObjectGrid-Loader implementiert, der die Informationen von dieser Datenbankinstanz abruft. Wenn ein Benutzer auf Informationen auf den Servern zugreift, auf denen der Eintrag ungültig gemacht wurde, bewirkt die Abfrage, dass der ObjectGrid-Loader die neuen Daten aus einer Datenquelle abruft und dem Benutzer keine veraltete Version der Daten bereitstellt.

- **Die Daten auf den fernen Servern durch die aktuellen Informationen ersetzen.**

  Wenn für eine bestimmte ObjectGrid-Instanz kein Loader implementiert ist, können Sie den Cache mit den ObjectGrid-Map- oder JavaMap-APIs füllen. In diesem Fall kann es hilfreich sein, wenn der Wert für den bestimmten Eintrag, der in einem Cluster-Member aktualisiert wurde, an die fernen ObjectGrid-Instanzen weitergegeben und dort angewendet wird. Somit erhalten Benutzer, die Daten aus dem Cache abrufen, die aktuelle Version.

Die Aktualisierungs- und Invalidierungsaktionen sind asynchron, und es ist nicht sichergestellt, dass sie auf den fernen Cluster-Membern festgeschrieben werden. Dies ist ähnlich wie bei der Methode update, wenn mehrere Member unterschiedliche Versionen der Informationen besitzen. Wenn mehrere Member die Daten aktualisieren können, sind die Vorgaben für den Verteilungsmodus nicht konsistent.

Voraussetzungen

Zum Aktivieren der Weitergabe verteilter Transaktionen muss Ihre Konfiguration die folgenden Voraussetzungen erfüllen:

- Sie muss einen konfigurierten ObjectGridEventListener-Listener für die bestimmte ObjectGrid-Instanz haben. Sie können diesen Listener mit einer XML-Datei oder über das Programm konfigurieren. Nähere Informationen finden Sie im Abschnitt „Listener" auf Seite 88.
- Die ObjectGrid-Instanz muss in jeder Anwendungsserverinstanz dasselbe Attribut `name` haben, und diese Instanz muss initialisiert sein.
- Die ObjectGridEventListener-Klasse TranPropListener, die diese Funktion bereitstellt, darf nur in einer Anwendungsserverinstanz von WebSphere Application Server verwendet werden. Andernfalls kann die Klasse nicht initialisiert werden.
- Jeder Anwendungsserver, der eine ObjectGrid-Instanz enthält, muss bei derselben HA-Manager-Stammgruppe registriert sein.

Wenn Sie mit der Weitergabe verteilter Transaktionen außerhalb einer Application-Server-Umgebung arbeiten möchten, lesen Sie den Abschnitt Kapitel 8, „Änderungen an Peer-JVMs verteilen", auf Seite 199.

Übersicht

Mit dieser Listener-Klasse können Sie erfolgreiche Festschreibungen von ObjectGrid-Transaktionen an andere WebSphere Application Server weitergeben, die dieselbe ObjectGrid-Instanz enthalten. Die Weitergabe basiert auf dem ObjectGrid-Namen. Die ObjectGrid-Instanzen werden anhand des Attributs `name` unterschieden. Wenn zwei Instanzen denselben Namen haben und für beide Instanzen derselbe Listener, nur in unterschiedlichen Stammgruppen-Membern von WebSphere Application Server konfiguriert ist, werden Änderungen, die nicht schreibgeschützt sind, in Abhängigkeit von den vom Benutzer ausgewählten Optionen weitergegeben.

Es werden vier Optionen für die Verteilung bereitgestellt:

- push
- push mit Versionssteuerung (Standardeinstellung)

- invalidate
- invalidate mit Versionssteuerung

Nähere Informationen hierzu finden Sie in der Beschreibung der Klasse DistributionMode in der API-Dokumentation.

Zusätzlich zum Verteilungsmodus können Sie konfigurieren, ob der Listener Transaktionen an andere Cluster-Member weitergibt, nur Transaktionsfestschreibungen von anderen, ähnlich konfigurierten ObjectGrid-Instanzen empfängt oder Transaktionen für jede Anwendungsserverinstanz weitergibt und empfängt (Standardoption). Diese Unterstützung kann nur für die optimistische, asynchrone Festschreibung verteilter Transaktionen verwendet werden.

Sie können konfigurieren, ob die Komprimierung aktiviert werden soll. Im Allgemeinen erhöht die Komprimierungsunterstützung für die Datenströme typischer Transaktionsfestschreibungen den lokalen CPU- und Speicherbedarf einiger JVMs. Es wird ein komprimierter Datenstrom für die Instanzen der Map erstellt, die die Übertragung an Listener unterstützen und sich in anderen Stammgruppen-Membern befinden. Bei der Komprimierung ist jedoch eines zu berücksichtigen. In Szenarios mit sehr kurzen Nachrichten und wenig Inhalt für die Transaktionsfestschreibung kann der Systemaufwand höher sein, als durch die Komprimierung bei den IT-Ressourcen eingespart werden kann. Sie können die Komprimierung in dieser Klasse inaktivieren.

Diese Klasse wird nur in einer Laufzeitumgebung mit WebSphere Application Server ordnungsgemäß initialisiert und erfordert eine einwandfreie Ausführung des HA Manager.

Sie müssen jede ObjectGrid-Instanz, an die Sie verteilte Transaktionen senden und von der Sie verteilte Transaktionen empfangen möchten, in den gewünschten Cluster-Member-Instanzen von WebSphere Application Server mit diesem Listener konfigurieren. Diese Unterstützung unterstützt Worker-Threads, damit sichergestellt werden kann, dass der gesamte Transaktionsinhalt im Hintergrund verarbeitet wird. Diese Unterstützung wird auch als *asynchrone Unterstützung* bezeichnet.

Der Thread, der diese Verarbeitung durchführt, wird normalerweise während der ersten Initialisierung der ObjectGrid-Instanz gestartet. Die Implementierung des Interface ObjectGridManager stellt diesen Thread für eine JVM in den Cache. Verwenden Sie das Interface ObjectGridManager, um eine ObjectGrid-Instanz in einer JVM konsistent zu verwalten. Der Worker-Thread wird beendet, wenn der Server heruntergefahren wird.

Zur Steuerung der Unterstützung für Worker-Threads können Sie die bereitgestellten Methoden verwenden. Gehen Sie dabei jedoch sorgfältig vor. Diese Unterstützung sollte in der folgenden Reihenfolge verwendet werden:

1. Server starten.
2. Mit der Implementierung des Interface ObjectGridManager die erste Instanz erstellen.
3. Dafür sorgen, dass alle JVM-Benutzer die vorhandene ObjectGrid-Instanz in ObjectGridManager suchen.
4. Die Threads, die für die Unterstützung verteilter Transaktionen zuständig sind, beim Herunterfahren des Servers ordnungsgemäß und ohne Intervention eines Programmierers oder Administrators bereinigen.

Alle Anwendung, die ObjectGrid in einem WebSphere Application Server verwenden und Festschreibungsinformationen gemeinsam nutzen, müssen derselben HA-Manager-Stammgruppe angehören.

Zugehörige Verweise

„Modusoptionen für die Verteilte Transaktionsweitergabe" auf Seite 160
Sie können beim Verteilungsmodus zwischen vier Weitergabe-Policys wählen. Attribute steuern den Typ von Operation, die an die ObjectGrid-Instanz der anderen Cluster-Member übertragen wird, und geben vor, welche Aktion während einer fernen Transaktionsfestschreibung ausgeführt wird.

Verteilte Transaktionsweitergabe konfigurieren

Sie können die Unterstützung für die verteilte Transaktionsweitergabe mit XML oder mit den Programmschnittstellen konfigurieren. Mit der verteilten Transaktionsweitergabe können Sie Aktualisierungen der Cache-Einträge während der Festschreibung einer ObjectGrid-Sitzungstransaktion koordinieren.

XML-Konfiguration

Das folgende Beispiel veranschaulicht, wie Sie die Klasse `TranPropListener` für eine bestimmte ObjectGrid-Instanz konfigurieren:

```
<bean id="ObjectGridEventListener"
  className="com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener">
 <property name="propagateService" type="java.lang.String" value="all"
  description="Option all includes propagating and receiving transaction
   commits.(propagate) and (receive) other options. "/>
 <property name="propagateMode" type="java.lang.String" value="update"
  description="Propagate value (update), or just convert all updates to invalidates
  in remote cache (invalidate)." />
 <property name="propagateVersionOption" type="java.lang.String" value="enable"
  description="Enable the use of versioning (enable) or disable (disable)."/>
 <property name="compressionMode" type="java.lang.String" value="enable"
  description="Enable the use of stream compression for transmission
  enable (enable) or disable (disable)." />
</bean>
```

Konfiguration über das Programm

Sie können diese Unterstützung nicht nur mit XML, sondern auch wie folgt über das Programm konfigurieren:

```
...
String ogn, boolean enableDist, String pMode, String pVersion, String compressMode;
....

ObjectGridManager manager= ObjectGridManagerFactory.getObjectGridManager();
og=manager.createObjectGrid(ogName);

if (enableDist){
 TranPropListener tpl=new TranPropListener();
 if (pMode!=null){
  tpl.setPropagateMode(pMode);
 }
 if (pVersion!=null){
  tpl.setPropagateVersionOption(pVersion);
 }
 if (compressMode!=null) {
  tpl.setCompressionMode(compressMode);
 }
 og.addEventListener(tpl);
}
```

Modusoptionen für die Verteilte Transaktionsweitergabe

Sie können beim Verteilungsmodus zwischen vier Weitergabe-Policys wählen. Attribute steuern den Typ von Operation, die an die ObjectGrid-Instanz der anderen Cluster-Member übertragen wird, und geben vor, welche Aktion während einer fernen Transaktionsfestschreibung ausgeführt wird.

Zweck

Sie müssen die Attribute propagateMode und propagateVersionOption setzen, um eine bestimmte Policy zu implementieren. Die folgende Tabelle beschreibt die Policys und die Attributkombinationen, die Sie definieren müssen, wenn Sie die Weitergabe verteilter Transaktionen mit XML oder in einem Programm konfigurieren, um die gewünschte Strategie für die Transaktionsweitergabe einzusetzen.

Tabelle 7. Erforderliche Attributwerte für eine bestimmte Weitergabe-Policy

Weitergabe-Policy	**Wert des Attributs propagateMode**	**Wert des Attributs propagateVersionOption**
Invalidate	invalidate	disable
Invalidate conditional	invalidate	enable
Update	update	disable
Update conditional (Standardeinstellung)	update	enable

Die Policys für die Verteilungsmodi sind in der folgenden Liste beschrieben. Der Standardmodus ist ″Update conditional″.

Modus 'Invalidate'

Der Verteilungsmodus 'Invalidate' (oder INVALIDATE in der API-Dokumentation) wird verwendet, um invalidate-Operationen an die Empfänger der LogSequence zu übertragen. Alle Aktualisierungen, wie z. B. update-, delete- und evict-Operationen, in der aktuellen LogSequence werden auf der Empfängerseite als Invalidierungen übertragen. Die Empfänger des LogSequence-Objekts können Einträge, die wegen der soeben verarbeiteten LogSequence nicht mehr auf dem neuesten Stand sind, ungültig machen. Im Modus INVALIDATE werden nur die Schlüssel gesendet, die in der verteilten LogSequence enthalten sind. Mehr ist für eine unqualifizierte invalidate-Operation nicht erforderlich.

Modus 'Invalidate conditional'

Der Verteilungsmodus 'Invalidate conditional' (oder INVALIDATE_CONDITIONAL in der API-Dokumentation) ist dem Verteilungsmodus INVALIDATE sehr ähnlich. Allerdings werden im Verteilungsmodus 'Invalidate conditional' auch die VersionedValue-Daten berücksichtigt, die jedem Eintrag zugeordnet sind. Vor der Ausführung der invalidate-Operationen auf der Empfangsseite werden der VersionedValue-Wert und der aktuelle Wert in der ObjectMap auf der Empfängerseite verglichen. Wenn der VersionedValue-Wert älter ist als der aktuelle Wert, wird die invalidate-Operation nicht ausgeführt.

Update

Der Verteilungsmodus 'Update' (oder PUSH in der API-Dokumentation) wird verwendet, um Änderungen an die Empfänger der LogSequence zu übertragen. Alle

LogElements-Elemente im aktuellen LogSequence-Objekt werden als entsprechende LogElements auf der Empfängerseite übertragen. Deshalb können die Empfänger der LogSequence ihre Instanz der ObjectMap-Map ordnungsgemäß aktualisieren. Im Modus PUSH wird die LogSequence unbedingt verarbeitet, ohne Rücksicht auf optimistische Kollisionen. Eine im PUSH-Modus ausgeführte update-Aktion kann den Wert in der ObjectMap auf der Empfängerseite überschreiben, der mit einer anderen Transaktion erstellt wurde.

Update conditional

Der Verteilungsmodus 'Update conditional' (oder PUSH_CONDITIONAL in der API-Dokumentation) ist dem Verteilungsmodus PUSH sehr ähnlich. Allerdings werden im Verteilungsmodus 'Update Conditional' auch die VersionedValue-Daten berücksichtigt, die jedem Eintrag zugeordnet sind. Vor der Ausführung der update-Operationen auf der Empfangsseite werden der VersionedValue-Wert und der aktuelle Wert in der ObjectMap auf der Empfängerseite verglichen. Wenn der VersionedValue-Wert älter ist als der aktuelle Wert, wird die Operation nicht ausgeführt.

ObjectGrid und Interaktionen mit externen Transaktionen

In der Regel werden ObjectGrid-Transaktionen mit der Methode session.begin gestartet und mit der Methode session.commit beendet. Wenn ObjectGrid jedoch integriert ist, können die Transaktionen von einem externen Transaktionskoordinator gestartet und beendet werden. In diesem Fall müssen Sie nicht die Methode session.begin aufrufen und zum Beenden die Methode session.commit verwenden.

Koordination externer Transaktionen

Das ObjectGrid-Plug-in TransactionCallback enthält isExternalTransactionActive-(Session Sitzung) als zusätzliche Methode. Diese Methode ordnet die ObjectGrid-Sitzung einer externen Transaktion zu. Der Methoden-Header sieht wie folgt aus:

```
public synchronized boolean isExternalTransactionActive(Session Sitzung)
```

ObjectGrid kann beispielsweise in WebSphere Application Server und WebSphere Extended Deployment integriert werden. Der Schlüssel für diese nahtlose Integration ist die Nutzung der API ExtendedJTATransaction in WebSphere Application Server Version 5.x und Version 6.x. Wenn Sie jedoch WebSphere Application Server Version 6.0.2 verwenden, müssen Sie APAR PK07848 für die Unterstützung dieser Methode anwenden. Verwenden Sie den folgenden Beispielcode, um einer ObjectGrid-Sitzung eine Transaktions-ID von WebSphere Application Server zuzuordnen:

```
/**
* Diese Methode ist erforderlich, um einer objectGrid-Sitzung eine
* WebSphere-Transaktions-ID zuzuordnen.
*/
Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
 // Denken Sie daran, dass diese lokale ID bedeutet, dass die Sitzung
 // für spätere Nutzung gespeichert wird.
 localIdToSession.put(new Integer(jta.getLocalId()), session);
 return true;
}
```

Externe Transaktion abrufen

Es kann vorkommen, dass Sie ein externes Transaktionsserviceobjekt für das ObjectGrid-Plug-in TransactionCallback abrufen müssen. Im Server von WebSphere Application Server suchen Sie, wie im folgenden Beispiel gezeigt, das Objekt ExtendedJTATransaction im zugehörigen Namespace:

```
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");
 }
}
```

Für andere Produkte können Sie eine ähnliche Methode verwenden, um das Transaktionsserviceobjekte abzurufen.

Festschreibung mit externen Callbacks steuern

Das Plug-in TransactionCallback muss ein externes Signal empfangen, um die ObjectGrid-Sitzung festzuschreiben oder rückgängig zu machen. Zum Empfangen dieses externen Signals verwenden Sie den Callback des externen Transaktionsservice. Sie müssen das externe Callback-Interface implementieren und beim externen Transaktionsservice registrieren. Für WebSphere Application Server müssen Sie beispielsweise das Interface SynchronizationCallback implementieren. Beispiel:

```
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);
```

```
 // Session für localId suchen
 Session session = (Session)localIdToSession.get(lid);
 if(session != null)
 {
  try
  {
   // Wenn WebSphere Application Server beim Festschreiben
   // der Transaktion in der BackingMap festgeschrieben wird.
   // flush wurde bereits beforeCompletion ausgeführt.
   if(didCommit)
   {
    session.commit();
   }
   else
   {
    // Andernfalls Rollback
    session.rollback();
   }
  }
  catch(NoActiveTransactionException e)
  {
   // theoretisch unmöglich
  }
  catch(TransactionException e)
  {
   // falls flush bereits ausgeführt wurde, sollte
   // dieser Schritt nicht fehlschlagen
  }
  finally
  {
   // Sitzung immer aus der Map entfernen
   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 definiert formal keine
   // Methode, mit der gemeldet wird, dass die Transaktion
   // fehlgeschlagen ist. Definieren Sie deshalb Folgendes:
   throw new RuntimeException("Cache flush failed", e);
  }
 }
}
}
```

ObjectGrid-APIs mit dem Plug-in TransactionCallback verwenden

Wenn dieses Plug-in als TransactionCallback-Plug-in für eine ObjectGrid-Instanz verwendet wird, inaktiviert es die automatische Festschreibung. Im Folgenden sehen Sie das normale Verwendungsmuster für eine ObjectGrid-Instanz:

```
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
ogSession.begin();
```

```
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
ogSession.commit();
```

Wenn dieses TransactionCallback-Plug-in verwendet wird, nimmt ObjectGrid an, dass die Anwendung die ObjectGrid-Instanz verwendet, wenn eine containergesteuerte Transaktion vorhanden ist. Das vorherige Code-Snippet ändert sich für diese Umgebung wie folgt:

```
public void myMethod()
{
 UserTransaction tx = ...;
 tx.begin();
 Session ogSession = ...;
 ObjectMap myMap = ogSession.getMap("MyMap");
 MyObject v = myMap.get("key");
 v.setAttribute("newValue");
 myMap.update("key", v);
 tx.commit();
}
```

Die Methode myMethod gleicht einem einem Webanwendungsszenario. Die Anwendung verwendet das normale Interface UserTransaction, um Transaktionen zu starten, festzuschreiben und rückgängig zu machen. Die ObjectGrid-Instanz führt die Methoden begin und commit vor bzw. nach der Containertransaktion aus. Wenn die Methode eine EJB-Methode ist, die das Attribut TX_REQUIRES verwendet, müssen Sie die Referenz auf UserTransaction und die Aufrufe zum Starten und Festschreiben von Transaktionen entfernen. Anschließend funktioniert die Methode auf dieselbe Weise. In diesem Fall ist der Container für das Starten und Beenden der Transaktion zuständig.

ObjectGrid und WPF integrieren

Verwenden Sie die Beispielanwendung ObjectGridPartitionCluster, um sich mit den kombinierten Funktionen von ObjectGrid und des Partitionierungs-Feature (WPF) vertraut zu machen.

Der Abschnitt „ObjectGrid und WPF" auf Seite 165 beschreibt in einer Zusammenfassung, wie ObjectGrid und das Partitionierungs-Feature miteinander arbeiten.

Wenn Sie ObjectGrid zusammen mit dem Partitionierungs-Feature verwenden möchten, muss WebSphere Extended Deployment in Ihrer Umgebung installiert sein.

Die Beispielanwendung ObjectGridPartitionCluster veranschaulicht die kombinierten Funktionen von ObjectGrid und WPF. Das Feature ObjectGrid bietet die Möglichkeit, Schlüssel/Wert-Paare nach Transaktion zwischenzuspeichern, und das Partitionierungs-Feature unterstützt die kontextbasierte Weiterleitung anhand von Objektmerkmalen.

- Installieren und führen Sie die Beispielanwendung ObjectGridPartitionCluster aus. Nähere Informationen finden Sie im Abschnitt „Beispielanwendung ObjectGridPartitionCluster installieren und ausführen" auf Seite 166.
- Wenn Sie den Quellcode der Beispielanwendung ändern möchten, können Sie die EAR-Datei in Ihre Entwicklungstool laden. Nähere Informationen finden Sie im Abschnitt „Integrierte Anwendung mit ObjectGrid und WPF erstellen" auf Seite 170.

- Machen Sie sich mit der Beispielanwendung vertraut. Eine Erläuterung des Codes in der Beispielanwendung finden Sie im Abschnitt „Beispiel: Programmierung mit ObjectGrid und dem Partitionierungs-Feature (WPF)" auf Seite 174.

Der Abschnitt Kapitel 6, „ObjectGrid mit WebSphere Application Server integrieren", auf Seite 145 beschreibt, wie Sie ObjectGrid mit anderen Features von WebSphere Application Server integrieren. Nähere Informationen zum Programmiermodell von ObjectGrid finden Sie im Abschnitt Kapitel 5, „Übersicht über die Anwendungsprogrammierschnittstelle von ObjectGrid", auf Seite 37.

ObjectGrid und WPF

ObjectGrid und das Partitionierungs-Feature (WPF) unterstützen in Zusammenarbeit die Zwischenspeicherung von Schlüssel/Wert-Paaren und die kontextbasierte Weiterleitung anhand von Objektmerkmalen.

Die Beispielanwendung ObjectGridPartitionCluster veranschaulicht die kombinierten Funktionen von ObjectGrid und WPF. ObjectGrid und WPF sind zwei Features im Produkt WebSphere Extended Deployment. Das Feature ObjectGrid bietet die Möglichkeit, Schlüssel/Wert-Paare nach Transaktion zwischenzuspeichern, und WPF unterstützt die kontextbasierte Weiterleitung anhand von Objektmerkmalen.

Neben den Features der Plug-ins Loader und TransactionCallback demonstriert diese Beispielanwendung auch, wie die Plug-ins ObjectGridEventListener, ObjectTransformer und OptimisticCallback verwendet werden. Die Beispielanwendung zeigt insbesondere, wie lokale ObjectGrid-Transaktionen weitergegeben und wie die geänderten Objekte von einem Server auf anderen mit und ohne das optimistische Versionsprüfprogramm ungültig gemacht werden.

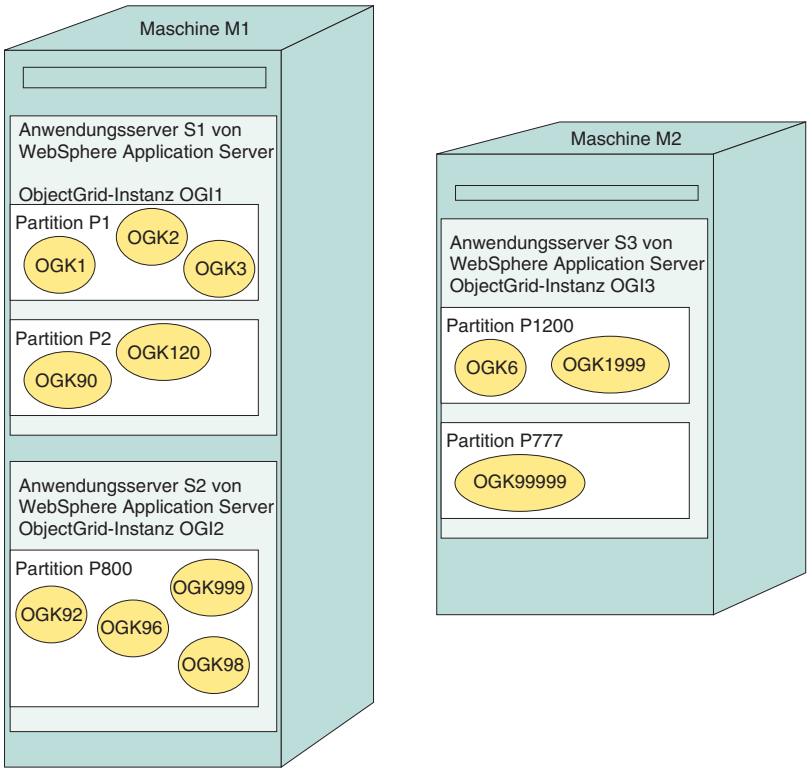
Sie müssen das WPF-Feature für kontextbasierte Weiterleitung verwenden, um sicherzustellen, dass Objektanforderungen des Typs update, insert und remove für denselben Schlüssel an dieselbe Java Virtual Machine (JVM) weitergeleitet und dass die Objektabrufanforderungen mit Workload-Management (WLM) an alle ObjectGrid-JVMs verteilt werden können. Durch die Verwendung von WPF wird die Datenintegrität in den verschiedenen ObjectGrid-Instanzen der Cluster-Member gewährleistet.

Zur Bewahrung der ObjectGrid-Konsistenz und -Integrität können Sie mit WPF ein großes ObjectGrid auf viele partitionierte ObjectGrids verteilen, und das WPF-Feature für kontextbasierte Weiterleitung leitet die Anforderungen anhand der ObjectGrid-Schlüssel weiter. Beispielsweise benötigen Sie ObjectGrid für die Verarbeitung vieler Objekte, für die das ObjectGrid einer JVM nicht ausreicht. Mit der WPF-Methode partitionLoadEvent können Sie Daten vorab in verschiedene Server laden, und das WPF-Feature für kontextbasierte Weiterleitung findet das richtige ObjectGrid automatisch.

Die Beispielanwendung erstellt eine Reihe von Hash-basierten Partitionen und die Routing-Kontext für den Partitions-Cluster:

- Unter Verwendung einer n:n-Strategie können Sie die ObjectGrid-Schlüssel partitionieren und den WPF-Partitionen zuordnen.
- Die WPF-Partitionen können mit einer n:n-Strategie im Application-Server-Cluster verteilt werden.

Die folgende Abbildung zeigt die typischen Einstellungen und die typische Konfiguration für die Beispielanwendung ObjectGridPartitionCluster:



In dieser Abbildung werden die Maschinen M1 und M2 für die Implementierung der Beispielanwendung ObjectGridPartitionCluster verwendet. Auf jeder physischen Maschine können mehrere WebSphere Application Server installiert sein. Beispielsweise kann die Maschine M1 zwei Application Server enthalten: S1 und S2. Die Maschine M2 hat einen Server, den S3. Jeder Server hat eine ObjectGrid-Instanz: OGI1ObjectGrid für den Application Server S1, OGI2ObjectGrid für den Application Server S2 und OGI3ObjectGrid für den Application Server S3.

Jeder Application Server kann viele Partitionen enthalten. Beispielsweise kann der Server S1 die Partitionen P1 und P2 und der Server S3 die Partitionen P1200 und P777 enthalten.

Jede Partition kann mehrere ObjectGrid-Schlüssel enthalten, die Partition P1 beispielsweise die ObjectGrid-Schlüssel OGK1, OGK2 und OGK3 und die Partition P800 die ObjectGrid-Schlüssel OGK92, OGK96, OGK98 und OGK9999.

Alle ObjectGrid-Anforderungen des Typs update, insert und remove werden auf der Basis der ObjectGrid-Schlüssel weitergeleitet. Sie können Objekte auf zwei Arten abrufen: mit einer WLM-Strategie von einem beliebigen Server oder von der bestimmten Serverpartition für diesen Schlüssel.

Beispielanwendung ObjectGridPartitionCluster installieren und ausführen

Verwenden Sie diese Task, um die Beispielanwendung ObjectGridPartitionCluster zu installieren und auszuführen, wenn Sie die Funktionalität zwischen ObjectGrid und WPF testen möchten.

Installieren Sie WebSphere Extended Deployment. Diesbezügliche Anweisungen finden Sie auf der Webseite WebSphere 'Library' zu Extended Deployment.

Eine geeignete Umgebung für die Ausführung der Beispielanwendung ObjectGridPartitionCluster ist eine Umgebung, in der WebSphere Extended Deployment auf zwei physischen Maschinen installiert ist oder in der zwei Knoten erstellt und zusammen mit dem Deployment Manager eingebunden werden.

1. Konfigurieren Sie einen Cluster mit drei oder mehr Cluster-Membern, um sich mit den Features dieser Beispielanwendung eingehend vertraut zu machen.
2. Installieren Sie die Datei `D_ObjectGridPartitionClusterSample.ear`. Das Partitionierungs-Feature (WPF), das in der Datei `D_ObjectGridPartitionClusterSample.ear` implementiert ist, kann installiert und ausgeführt werden. Wenn Sie den Quellcode der Beispielanwendung ändern, folgen Sie den Anleitungen zu den Anweisungen build und wpf-deploy, um Ihre EAR-Datei zu erstellen und zu implementieren.

   Normalerweise werden die EAR-Dateien für Anwendungen mit der Administrationskonsole installiert. Folgen Sie zum Installieren der Datei `D_ObjectGridPartitionClusterSample.ear` der Installationsprozedur für Enterprise-Anwendungen. Klicken Sie auf **Anwendungen > Neue Anwendung installieren**, um auf diesen Teil der Administrationskonsole zuzugreifen. Implementieren Sie die EAR-Datei nicht während der Installation. Verwenden Sie in allen Schritten mit Ausnahme des Schritts, in dem Sie zur Auswahl eines Installationsverzeichnisses aufgefordert werden, die Standardeinstellungen. Wählen Sie in diesem Schritt anstelle des Standardservers server1 den Cluster aus, den Sie definiert haben.
3. Führen Sie die Beispielanwendung ObjectGridPartitionClusterSample aus.
   a. Starten Sie den Cluster. Klicken Sie in der Administrationskonsole auf **Server > Cluster**. Wählen Sie den Cluster aus und klicken Sie auf **Starten**.
   b. Führen Sie den Script-Befehl **WAS_INSTALL_ROOT\bin\wpfadmin balance** aus. Vergewissern Sie sich durch Ausführen des Befehls **WAS_INSTALL_ROOT\bin\wpfadmin list**, dass die Partition aktiv ist. Nähere Informationen zum Script wpfadmin und seinen Befehlen finden Sie im Handbuch zum Partitionierungs-Feature im WebSphere Extended Deployment Information Center.
   c. Setzen Sie zum Ausführen der Beispielanwendung ObjectGridPartitionClusterSample den folgenden Befehl ab:

      ```
      WAS_INSTALL_ROOT/bin/launchClient.bat|sh \
      WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear \
      -CCBootstrapPort=PORT
      ```

      PORT steht hier für den RMI-Port des Servers, den Sie nach dem Starten des Servers in der Datei `SystemOut.log` finden. Normalerweise ist der Port-Wert einer der folgenden: `9810, 9811, 9812`.

      Beispielbefehl:

      ```
      WAS_INSTALL_ROOT/bin/launchClient.bat|sh
      WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear
        -CCBootstrapPort=9811
      ```

      Die fortgeschrittene Verwendung dieses Script wird im Abschnitt „Optionen für den Anwendungsclient ObjectGridPartitionClusterSample” auf Seite 168 beschrieben.
4. Ändern Sie die Anzahl der Partitionen. Ändern Sie die Anzahl der Partitionen, die von der Session-Enterprise-Bean ObjectGridPartitionCluster erstellt werden. Die Anzahl der Partitionen, die von der Session-Bean PFClusterObjectGridEJB erstellt werden, wird von der Umgebungsvariablen NumberOfPartitions in der Datei `META-INF\ejb-jar.xml` bestimmt. Der Standardwert ist `10`. Sie können den

Wert dieser Umgebungsvariablen ändern und die Anwendung erneut installieren, um ein mehr oder weniger Partitionen zu erstellen. Der Maximalwert sind `999999` Partitionen.

5. Ändern Sie die Optionen für den verteilten Listener. Sie können die folgenden Optionen für den verteilten Listener ändern:

Tabelle 8. Optionen für verteilten Listener

Variablenname	Beschreibung
enableDistribution,	Mit der Umgebungsvariablen enableDistribution im EJB-Deployment-Deskriptor können Sie einen verteilten ObjectGrid-Listener aktivieren. Der Standardwert ist `true`, d. h. aktiviert. Setzen Sie den Wert auf `false`, wenn Sie den verteilten Listener inaktivieren möchten.
propagationMode	Mit der Umgebungsvariablen propagationMode im EJB-Deployment-Deskriptor können Sie den Weitergabemodus ändern. Der Standardwert ist `update`. Sie können den Wert in `invalidate` ändern, falls Sie einen anderen als den Standardwert verwenden möchten.
propagationVersionOption	Mit der Variablen propagationVersionOption im EJB-Deployment-Deskriptor können Sie die Option für die Weitergabeversion ändern. Die Standardeinstellung ist `enable`. Sie können diesen Wert in `disable` ändern.
compressionMode	Mit der Umgebungsvariablen compressionMode im EJB-Deployment-Deskriptor können Sie den Komprimierungsmodus ändern. Die Standardeinstellung ist `enable`. Sie können diesen Wert in `disable` ändern.

Standardmäßig werden die Aktualisierungen mit Versionsprüfung weitergegeben. Sie können den Wert aber auch auf den Modus 'invalidate' ohne Versionsprüfung setzen.

Sie haben die Beispielanwendung ObjectGridPartitionCluster installiert und ausgeführt.

Optionen für den Anwendungsclient ObjectGridPartitionClusterSample

Verwenden Sie diese Optionen für eine erweiterte Ausführung der Datei `D_ObjectGridPartitionClusterSample.ear`.

Erweiterte Nutzung der Beispielanwendung

Nähere Informationen zum Installieren und Ausführen der Datei `D_ObjectGridPartitionClusterSample.ear` finden Sie im Abschnitt „Beispielanwendung ObjectGridPartitionCluster installieren und ausführen" auf Seite 166.

Informationen zur erweiterten Nutzung der Beispielanwendung finden Sie in den folgenden Anleitungen:

```
WAS_INSTALL_ROOT/bin/launchClient.bat|sh
WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear
-CCproviderURL=corbaloc::HOSTNAME:RMI-PORT_DES_SERVERS [-loop SCHLEIFE] [-threads
ANZAHL_THREADS] [-add ANZAHL_STOCKS_PRO_PARTITION] [-waitForPropagation
WARTEZEIT_IN_SEKUNDEN_FÜR_WEITERGABE] [-getIteration
ANZAHL_DER_ITERATIONEN_PRO_OGSCHLÜSSEL]
```

Ersetzen Sie die folgenden Variablen:

- HOSTNAME: Gibt den Hostnamen des aktiven Anwendungsservers an.
- RMI-PORT_DES_SERVERS: Gibt den Bootstrap-Port des Anwendungsserves an.
- SCHLEIFE: Gibt an, wie oft der Client in der Schleife ausgeführt wird. Dieser Parameter ist optional. Der Standardwert ist 1.
- ANZAHL_THREADS: Gibt an, wie viele Threads der Client ausführt. Dieser Parameter ist optional. Der Standardwert ist 1.
- ANZAHL_STOCKS_PRO_PARTITION: Gibt die Anzahl der Aktien (Stocks) für jede hinzuzufügende Partition an. Dieser Parameter ist optional. Die Standardeinstellung ist 3.
- WARTEZEIT_IN_SEKUNDEN_FÜR_WEITERGABE: Gibt an, wie lange (in Sekunden) auf neu hinzugefügte oder aktualisierte ObjectGrid-Objekte gewartet wird, die an andere Server weitergegeben werden sollen. Der Standardwert ist 2 (Sekunden).
- ANZAHL_DER_ITERATIONEN_PRO_OGSCHLÜSSEL: Gibt die Anzahl der Iterationen für den Abruf von ObjectGrid-Objekten mit Workload-Management an. Der Standardwert ist 6. Wenn mehr Iterationen angegeben werden, ist ein klares Muster für Objekte desselben Schlüssels in unterschiedlichen Servern von WebSphere Application Server erkennbar.

Beispielausgabe

Die Ausgabe dieses Befehls gleicht dem folgenden Beispiel:

```
C:\dev\xd6\bin>launchClient
D_ObjectGridPartitionClusterSample.ear -CCBootstrapPort=9812
IBM WebSphere Application Server, Release 6.0
J2EE Application Client Tool
Copyright IBM Corp., 1997-2004
WSCL0012I: Die Befehlszeilenparameter werden verarbeitet.
WSCL0013I: Die Umgebung des J2EE-Anwendungsclient wird initialisiert.
WSCL0035I: Die Initialisierung der Umgebung des J2EE-Anwendungsclient ist abgeschlossen.
WSCL0014I: Die Application-Client-Klasse
com.ibm.websphere.samples.objectgrid.partitioncluster.client.PartitionObjectGrid
wird aufgerufen.
ObjectGrid Partition Sample has 10 partitions
PARTITION: ObjectGridHashPartition000007->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000003->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000005->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000010->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000006->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000009->clusterdevNode01/s2
PARTITION: ObjectGridHashPartition000008->clusterdevNode01/s1
PARTITION: ObjectGridHashPartition000002->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000001->clusterdevNode02/s3
PARTITION: ObjectGridHashPartition000004->clusterdevNode01/s2
************** Partition=ObjectGridHashPartition000004****************
------ObjectGrid Operations: Stock Ticket=Stock000104 --------
get on partition for ticket: Stock000104->clusterdevNode02/s2
update: Stock000104->clusterdevNode02/s2
sleep 2 seconds.....
Iteration 1 : Stock000104->clusterdevNode01/s2
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 2 : Stock000104->clusterdevNode01/s1
```

```
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 3 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 4 : Stock000104->clusterdevNode01/s2
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 5 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
Iteration 6 : Stock000104->clusterdevNode02/s3
> ObjectGrid Stock000104 price=43.35478459674703 lastTransaction=1121137456584
------ObjectGrid Operations: Stock Ticket=Stock000114 --------
get on partition for ticket: Stock000114->clusterdevNode01/s2
update: Stock000114->clusterdevNode02/s2
sleep 2 seconds.....
Iteration 1 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 2 : Stock000114->clusterdevNode01/s2
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 3 : Stock000114->clusterdevNode01/s1
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 4 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 5 : Stock000114->clusterdevNode01/s2
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
Iteration 6 : Stock000114->clusterdevNode02/s3
> ObjectGrid Stock000114 price=39.70991373766818 lastTransaction=1121137458737
------ObjectGrid Operations: Stock Ticket=Stock000124 --------
get on partition for ticket: Stock000124->clusterdevNode02/s2
update: Stock000124->clusterdevNode01/s2
sleep 2 seconds.....
Iteration 1 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 2 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 3 : Stock000124->clusterdevNode01/s2
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 4 : Stock000124->clusterdevNode01/s1
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 5 : Stock000124->clusterdevNode02/s3
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
Iteration 6 : Stock000124->clusterdevNode01/s2
> ObjectGrid Stock000124 price=35.37356414423455 lastTransaction=1121137460940
C:\dev\xd6\bin>
```

Integrierte Anwendung mit ObjectGrid und WPF erstellen

Sie können die ObjectGrid-Beispielanwendung für Partitionierung öffnen, ändern und installieren.

Führen Sie die folgenden Schritte aus, um die Datei `ObjectGridPartitionSample.ear` in einer Umgebung mit WebSphere Extended Deployment zu ändern, zu exportieren und zu installieren. Sollten Sie keine Änderungen an der Beispieldatei vornehmen wollen, können Sie die implementierte und WPF-fähige Datei `D_ObjectGridPartitionClusterSample.ear` verwenden. Wenn Sie die Datei `D_ObjectGridPartitionClusterSample.ear` verwenden, können Sie die Datei installieren und ausführen, ohne die folgenden Schritte ausführen zu müssen. Beide EAR-Dateien befinden sich im Verzeichnis `WAS_INSTALL_ROOT/installableApps`.

1. Konfigurieren Sie die Datei `ObjectGridPartitionSample.ear` in Ihrer Erstellungsumgebung, wie z. B. IBM Rational Application Developer Version 6.0.x oder Application Server Toolkit Version 6.0.x. Nähere Informationen finden Sie im Abschnitt „Einführung in die Erstellung einer Anwendung mit ObjectGrid und WPF" auf Seite 171.
2. Ändern Sie die gewünschten Teile im Quellcode der Beispielanwendung.

3. Exportieren Sie die Anwendung ObjectGridPartitionClusterSample aus Ihrer Erstellungsumgebung als EAR-Datei. Nähere Informationen finden Sie im Abschnitt „Datei ObjectGridPartitionClusterSample.ear in IBM Rational Application Developer exportieren" auf Seite 173.
4. Implementieren Sie die Anwendung so, dass sie mit dem Partitionierungs-Feature (WPF) funktioniert. Nähere Informationen finden Sie im Abschnitt „Datei ObjectGridPartitionClusterSample.ear für das Partitionierungs-Feature (WPF) implementieren" auf Seite 173.
5. Installieren Sie die Datei `ObjectGridPartitionClusterSample.ear` in WebSphere Extended Deployment. Normalerweise werden die EAR-Dateien für Anwendungen mit der Administrationskonsole von WebSphere Application Server installiert. Folgen Sie zum Installieren der Datei `D_ObjectGridPartitionClusterSample.ear` in der Administrationskonsole dem Installationsverfahren für Enterprise-Anwendungen. Wählen Sie während der Installation nicht die Option zum Implementieren der Datei aus, sondern verwenden Sie die Standardeinstellungen. Übernehmen Sie in den einzelnen Installationsschritte die Standardeinstellungen. Die einzige Ausnahme bildet die Auswahl des Installationsverzeichnisses. Wählen Sie in diesem Schritt anstelle des Standardservers server1 den von Ihnen definierten Cluster aus.

Sie haben die Datei `ObjectGridPartitionClusterSample.ear` in einer Umgebung mit WebSphere Extended Deployment installiert.

Nähere Informationen zur Programmierung mit ObjectGrid, zum Partitionierungs-Feature (WPF) und zu den Beispielanwendungen finden Sie im Abschnitt „Beispiel: Programmierung mit ObjectGrid und dem Partitionierungs-Feature (WPF)" auf Seite 174.

Einführung in die Erstellung einer Anwendung mit ObjectGrid und WPF

Verwenden Sie Application Server Toolkit Version 6.0.x oder IBM Rational Application Developer Version 6.0.x, um die Beispielanwendung erneut zu erstellen.

Die Datei `ObjectGridPartitionClusterSample.ear` im Verzeichnis `WAS_INSTALL_ROOT/installableApps` enthält den gesamten Quellcode. Sie können zum erneuten Erstellen dieser Beispielanwendung Application Server Toolkit Version 6.0.x oder IBM Rational Application Developer Version 6.0.x verwenden. In dieser Task wird Rational Application Developer verwendet, um die Erstellungsumgebung für die Datei `ObjectGridPartitionClusterSample.ear` einzurichten. Sie können aber auch Application Server Toolkit verwenden, ein kostenloses Assembliertool, das mit WebSphere Application Server auf einer gesonderten CD bereitgestellt wird.

Die implementierte und WPF-fähige EAR-Datei `D_ObjectGridPartitionClusterSample.ear`, die ebenfalls im Verzeichnis `WAS_INSTALL_ROOT/installableApps` gespeichert ist, ist installations- und ausführbereit.

1. Importieren Sie die Datei `ObjectGridPartitionClusterSample.ear` in Rational Application Developer.
   a. Starten Sie Rational Application Developer.
   b. Optional: Öffnen Sie die J2EE-Perspektive, um J2EE-Projekte zu bearbeiten. Klicken Sie auf **Fenster > Perspektive öffnen > Andere > J2EE**.
   c. Optional: Öffnen Sie die Sicht "Projekt-Explorer". Klicken Sie auf **Fenster > Sicht anzeigen > Projekt-Explorer**. Eine weitere hilfreiche Sicht ist die Navigator-Sicht, die Sie durch Anklicken von **Fenster > Sicht anzeigen > Navigator** aufrufen.

d. Importieren Sie die Datei `ObjectGridPartitionClusterSample.ear`. Klicken Sie auf **Datei > Importieren > EAR-Datei** und klicken Sie anschließend auf **Weiter**.

e. Wählen Sie die Datei `ObjectGridPartitionClusterSample.ear` im Verzeichnis `WAS_INSTALL_ROOT/installableApps` aus.

f. Optional: Klicken Sie auf **Neu**, um den Assistenten ″Neue Serverlaufzeit″ zu öffnen, und folgen Sie anschließend den Anweisungen.

g. Wählen Sie im Feld ″Zielserver″ **WebSphere Application Server 6.0** als Typ für die Serverlaufzeit aus.

h. Klicken Sie auf **Fertig stellen**.

Die Projekte ObjectGridPartitionClusterSample, ObjectGridPartitionClusterSampleEJB und ObjectGridPartitionClusterSampleClient müssen erstellt werden und in der Sicht ″Projekt-Explorer″ sichtbar sein.

2. Konfigurieren Sie das Projekt ObjectGridPartitionClusterSampleEJB.

a. Klicken Sie in der Sicht ″Projekt-Explorer″ der J2EE-Perspektive unter ″EJB-Projekte″ mit der rechten Maustaste auf das Projekt **ObjectGridPartitionClusterSampleEJB** und wählen Sie **Eigenschaften** aus. Das Fenster ″Eigenschaften″ erscheint.

b. Klicken Sie in der linken Anzeige auf **Java-Erstellungspfad**, klicken Sie in der rechten Anzeige auf das Register **Bibliotheken** und wählen Sie anschließend **Variable hinzufügen** aus. Das Fenster Neuer Eintrag in Variable CLASSPATH erscheint.

c. Klicken Sie auf **Variablen konfigurieren**, um das Fenster **Vorgabe** zu öffnen.

d. Klicken Sie auf **Neu**, um das Fenster ″Neuer Variableneintrag″ zu öffnen.

e. Geben Sie `ObjectGridPartitionCluster_JAR` im Feld **Name** ein und klicken Sie anschließend auf **Datei**, um das Fenster JAR-Auswahl zu öffnen.

f. Navigieren Sie in das Verzeichnis `WAS_INSTALL_ROOT/lib` und wählen Sie **ObjectGrid.jar** aus. Klicken Sie auf **Öffnen**, um das Fenster ″JAR-Auswahl″ zu schließen.

g. Klicken Sie auf **OK** , um das Fenster Neuer Variableneintrag zu schließen. Die Variable ″ObjectGridPartitionCluster_JAR″ wird in der Liste der Klassenpfadvariablen angezeigt.

h. Klicken Sie auf **OK**, um das Fenster **Vorgabe** zu schließen.

i. Wählen Sie in der Variablenliste die Variable **ObjectGridPartitionCluster_JAR** aus und klicken Sie auf **OK**, um das Fenster Neuer Eintrag in Variable CLASSPATH zu schließen. Die Variable ″ObjectGridPartitionCluster_JAR″ erscheint in der Anzeige ″Bibliotheken″.

j. Wiederholen Sie diese Prozedur, um Ihrer Umgebung die Datei `wpf.jar` hinzuzufügen.

k. Vergewissern Sie sich, dass die Datei `wpf.jar` und die Datei `objectgrid.jar` im Erstellungsklassenpfad enthalten sind.

Nachdem Sie Ihre Erstellungsumgebung eingerichtet haben, können sie den Quellcode modifizieren und weitere Änderungen vornehmen. Nähere Informationen finden Sie im Abschnitt „Integrierte Anwendung mit ObjectGrid und WPF erstellen″ auf Seite 170.

Datei ObjectGridPartitionClusterSample.ear in IBM Rational Application Developer exportieren

Nachdem Sie Ihre Änderungen an der Beispieldatei vorgenommen haben, können Sie die Anwendung ObjectGridPartitionClusterSample exportieren, um eine EAR-Datei zu erstellen, die Sie in Servern von WebSphere Extended Deployment installieren können.

Sie müssen die Datei `ObjectGridPartitionSample.ear` in Ihre Entwicklungstools importieren, um Änderungen an der Quelle vornehmen zu können. Nähere Informationen finden Sie im Abschnitt „Einführung in die Erstellung einer Anwendung mit ObjectGrid und WPF" auf Seite 171. Nehmen Sie vor dem Export die gewünschten Änderungen an der Beispielanwendung vor.

Sie können die Datei `ObjectGridPartitionClusterSample.ear` aus dem Projekt ObjectGridPartitionClusterSample in Enterprise-Anwendungen in IBM Rational Application Developer exportieren. Nach der Implementierung des Partitionierungs-Feature können Sie die exportierte Datei `ObjectGridPartitionClusterSample.ear` in jedem Server von WebSphere Extended Deployment Version 6.0 installieren.

1. Klicken Sie in der Sicht "Projekt-Explorer" der J2EE-Perspektive unter "Unternehmensanwendungen" mit der rechten Maustaste auf die Anwendung **ObjectGridPartitionClusterSample**. Klicken Sie auf **Exportieren > EAR-Datei**. Das Fenster "Exportieren" wird angezeigt.
2. Klicken Sie auf **Durchsuchen**, um das Fenster **Speichern unter** zu öffnen. Suchen Sie das Zielausgabeverzeichnis, geben Sie als Dateinamen `ObjectGridPartitionClusterSample` ein und klicken Sie anschließend auf **Speichern**.
3. Klicken Sie auf **Durchsuchen**, um das Fenster "Speichern unter" zu öffnen. Suchen Sie das Zielausgabeverzeichnis, geben Sie als Dateinamen `ObjectGridPartitionClusterSample` ein. Klicken Sie auf **Speichern**.

Die Datei `ObjectGridPartitionClusterSample.ear` wird im angegebenen Zielausgabeverzeichnis erstellt.

Nachdem Sie die Datei `ObjectGridPartitionClusterSample.ear` für das Partitionierungs-Feature (WPF) implementiert haben, können Sie die Datei in WebSphere Extended Deployment ausführen. Nähere Informationen finden Sie im Abschnitt „Datei ObjectGridPartitionClusterSample.ear für das Partitionierungs-Feature (WPF) implementieren".

Datei ObjectGridPartitionClusterSample.ear für das Partitionierungs-Feature (WPF) implementieren

Wenn Sie die Datei `ObjectGridPartitionClusterSample.ear` in WebSphere Extended Deployment installieren möchten, müssen Sie eine Operation wpf-deploy für die Datei ausführen.

Die Datei `ObjectGridPartitionClusterSample.ear` muss vorhanden sein. Informationen zum Ändern der vorhandenen Datei finden Sie im Abschnitt „Integrierte Anwendung mit ObjectGrid und WPF erstellen" auf Seite 170.

Führen Sie die Operation wpf-deploy aus, um die EAR-Datei in einer Extended-Deployment-Umgebung vorzubereiten.

1. Erstellen Sie ein Verzeichnis mit dem Namen `DEST_DIR`.

2. Kopieren Sie die Datei ObjectGridPartitionClusterSample.ear in das Verzeichnis DEST_DIR. Benennen Sie die Datei ObjectGridPartitionClusterSample.ear in old_ObjectGridPartitionClusterSample.ear um.
3. Führen Sie den folgenden Befehl aus. WORKING_DIR steht für das Arbeitsverzeichnis des Tools ejbdeloy, z. B. c:\temp.

   ```
   WAS_HOME\bin\ejbdeploy.bat|ejbdeploy.sh
    DEST_DIR\old_ObjectGridPartitionClusterSample.ear WORKING_DIR
    DEST_DIR\ObjectGridPartitionClusterSample.ear
   ```

4. Führen Sie den folgenden Befehl aus. TEMP_DIR steht für ein temporäres Verzeichnis des Tools. Wenn Sie das Argument -keep angeben, werden die temporären Verzeichnisse, die das Dienstprogramm wpfStubUtil erstellt, nicht gelöscht.

   ```
   WAS_HOME\bin\wpfStubUtil.cmd|wpfStubUtil.sh
    DEST_DIR\ObjectGridPartitionClusterSample.ear
    ObjectGridPartitionClusterSampleEJB.jar com/ibm/websphere/samples/
    objectgrid/partitioncluster/ejb/PFClusterObjectGridEJB.class
    TEMP_DIR [-stubDebug|-keep]
   ```

Die Datei ObjectGridPartitionClusterSample.ear kann jetzt in einer Umgebung mit WebSphere Extended Deployment ausgeführt werden. Die mit WebSphere Extended Deployment bereitgestellte Beispieldatei D_ObjectGridPartitionClusterSample.ear ist bereits implementiert. Falls Sie den Quellcode nicht geändert haben, müssen Sie diese Datei nicht implementieren, um die Anwendung installieren zu können.

Installieren Sie die Datei ObjectGridPartitionClusterSample.ear mit der Administrationskonsole in der Extended-Deployment-Umgebung. Nähere Informationen finden Sie im Abschnitt „Integrierte Anwendung mit ObjectGrid und WPF erstellen” auf Seite 170.

Beispiel: Programmierung mit ObjectGrid und dem Partitionierungs-Feature (WPF)

Dieses Beispiel veranschaulicht die kombinierten Funktionen von ObjectGrid und dem Partitionierungs-Feature in einer J2EE-Umgebung in WebSphere Extended Deployment verwendet werden.

Zweck

Neben der Demonstration der kombinierten Funktionen von ObjectGrid und Partitionierungs-Feature zeigt dieses Beispiel die Weitergabe und Invalidierung von Anforderungen mit einem verteilten ObjectGrid-Listener.

Anforderungen zum Aktualisieren, Einfügen und Entfernen von Objekten werden an bestimmte Server weitergeleitet, die Partitionen für die entsprechenden ObjectGrid-Schlüssel enthalten. Anforderungen für die ObjectGrid-Methode get werden gleichmäßig auf alle Server verteilt.

Außerdem demonstriert dieses Beispiel, wie eine große ObjectGrid-Instanz in kleinere ObjectGrid-Instanzen partitioniert und Daten mit der Methode partitionLoadEvent vorher geladen werden können, so dass die partitionierte ObjectGrid-Instanz eine unbegrenzte Anzahl von Objekten aufnehmen kann.

Übersicht

Die Datei `ObjectGridPartitionClusterSampler.ear` erstellt ein Objekt stock, das veranschaulicht, wie ObjectGrid und das Partionierungs-Feature miteinander arbeiten. Das Objekt stock hat die folgenden Merkmale:

- ticket
- company
- serialNumber
- description
- lastTransaction
- price

Das Merkmal lastTransaction gibt an, wann das Objekt stock zuletzt geändert wurde. Mit dem Merkmal lastTransaction können Sie die Aktualität von Objekten in den ObjectGrid-Instanzen unterschiedlicher JVMs angeben.

In dem Beispiel wird die ObjectGrid-Instanz in einer EJB-Methode setContext mit der Klasse ObjectGridFactory erstellt.

Definieren Sie eine Reihe von Hash-basierten Partitionen. Standardmäßig werden 10 Partitionen definiert, aber Sie können die Anzahl der Partitionen ändern. Speichern Sie die Stock-Tickets mit der Datei `SampleUtility.java` in einem Hash-Verfahren in diesen Partitionen. Jede Partition kann viele ObjectGrid-Schlüssel/Wert-Paare enthalten.

Die Beispielanwendung veranschaulicht, wie die ObjectGrid-Anforderungen insert, update und remove an einen bestimmten partitionierten Server weitergeleitet und wie die ObjectGrid-Methodenanforderungen get an einen bestimmten Server für den zugehörigen Schlüssel oder an einen beliebigen Server in einem Cluster weitergeleitet werden. Die Beispielanwendung vergleicht Objektwerte für einen Schlüssel in verschiedenen Servern, wenn sich ein Wert aufgrund einer update-, insert- oder remove-Operation für diesen Schlüssel in einem bestimmten Server ändert.

Position

Verwenden Sie diese Beispielanwendung in einer Cluster-Umgebung, in der jeder Server viele Partitionen und jede Partition viele Objekte mit unterschiedlichen Schlüsseln enthalten kann.

Das Verzeichnis `<Installationsstammverzeichnis>\installableApps\` enthält zwei Beispieldateien für einen ObjectGrid-Partitions-Cluster:

- Die Datei `ObjectGridPartitionClusterSampler.ear` enthält den Quellcode. Wenn Sie sich den Quellcode anschauen möchten, erweitern Sie die EAR-Datei im Dateisystem oder importieren Sie die Quelle in eine Entwicklungsumgebung. Nähere Informationen finden Sie im Abschnitt „Integrierte Anwendung mit ObjectGrid und WPF erstellen” auf Seite 170.
- Die Datei `D_ObjectGridPartitionClusterSample.ear` ist für das Partitionierungs-Feature bereits implementiert. Lesen Sie die Readme-Datei und die Anleitungen, um möglichst schnell mit der Ausführung dieser Datei zu beginnen.

Erläuterung

Die folgenden Abschnitte enthalten Erläuterungen zur Beispielanwendung für den ObjectGrid-Partitions-Cluster:

ObjectGrid operation EJB Interface

Dieser Artikel beschreibt das EJB-Interface für ObjectGrid-Operationen, das Operationen get, get from partitioned server, insert, update und remove ausführt.

Zweck

Das EJB-Interface für ObjectGrid-Operationen führt Operationen get, get from partitioned server, insert, update und remove aus. Die Methode 'get from partitioned server' wird an eine Partition weitergeleitet, die dem angeforderten Schlüssel entspricht. Die Methode 'get' wird mit einer WLM-Strategie an einen der Server weitergeleitet.

Interface PFClusterObjectGridEJB

Im Folgenden sehen Sie den Inhalt des Interface PFClusterObjectGridEJB:

```
/**
* Remote-Interface für Enterprise-Beans: PFClusterObjectGridEJB
*/
public interface PFClusterObjectGridEJB extends javax.ejb.EJBObject {
 public String PARTITION_PREFIX = "ObjectGridHashPartition";
 /**
 * Alle Partitionen abrufen
 *
 * @return Array of Strings
 * @throws java.rmi.RemoteException
 */

 public String [] getAllPartitions() throws java.rmi.RemoteException;
 /**
 * Position der Partition abrufen
 *
 * @param partition
 * @return String
 * @throws java.rmi.RemoteException
 */

 public String getServer(String partition)
 throws java.rmi.RemoteException;
 /**
 * Stock-Objekt und die zugehörigen Serverinformationen
 * (ServerIDResult) für ein Stock-Ticket von einem
 * Server im Cluster (mit Workload-Management) abrufen
 *
 * @param ticket
 * @return
 * @throws java.rmi.RemoteException
 */

 public ServerIDResult getStock(String ticket)
 throws java.rmi.RemoteException;
 /**
 * Stock-Objekt und die zugehörigen Informationen
 * zum partitionierten Server für ein Stock-Ticket
 * von der Partition abrufen, in der dieser
```

```
 * Ticket-Schlüssel im Hash-Verfahren gespeichert wurde
 *
 * @param ticket
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
 */

 public ServerIDResult getStockOnPartitionedServer(String ticket)
 throws java.rmi.RemoteException;
 /**
 * Stock in einem bestimmten Server aktualisieren,
 * in dem die Partition für diesen Stock-Ticket-Schlüssel
 * aktiv ist
 *
 * @param stock
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
 */

 public ServerIDResult updateStock(Stock stock)
 throws java.rmi.RemoteException;
 /**
 * Stock in einem bestimmten Server entfernen, in dem die
 * Partition für diesen Stock-Ticket-Schlüssel aktiv ist
 *
 * @param ticket
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
 */

 public ServerIDResult removeStock(String ticket)
 throws java.rmi.RemoteException;
 /**
 * Stock in einem bestimmten Server einfügen, in dem
 * die Partition für diesen Stock-Ticket-Schlüssel aktiv ist
 *
 * @param stock
 * @return ServerIDResult
 * @throws java.rmi.RemoteException
 */

 public ServerIDResult insertStock(Stock stock)
 throws java.rmi.RemoteException;
 /**
 * Daten von allen Servern abrufen und Werte vergleichen
 *
 * @param server
 * @return ServerObjectGridVerification
 * @throws java.rmi.RemoteException
 */

 public ServerObjectGridVerification verifyObjectGrid(String server)
 throws java.rmi.RemoteException;
}
```

Klasse PartitionKey

Die Klasse PartitionKey steuert das Verhalten der kontextbasierten Weiterleitung von WPF.

Der folgende Code zeigt die Beispielklasse für Partitionsschlüssel. Wenn die Methode einen Wert `ungleich null` zurückgibt, wird sie mit dem WPF-Router weitergeleitet. Gibt die Methode `null` zurück, wird Sie an den WLM-Router weitergeleitet.

```
/**
* PartitionKey für Partitioned Stateless Session Bean WPFKeyBasedPartition
*
*/
public class PFClusterObjectGridEJB_PartitionKey {
 /**
 * Anzahl der Partitionen
 *
 * Die Standardeinstellung ist 10.
 *
 */
 static int numOfPartitions=10;
 /**
 * Einmal für getPartitionNumbers
 */
 static boolean getNumOfPartitions=true;
 /**
 * Anzahl der Partitionen abrufen
 *
 */
 static void getPartitionNumbers(){
 // einmaliger Abruf
  if (getNumOfPartitions){
   try {
    InitialContext ic = new InitialContext();
    PFClusterObjectGridEJBHome home =
    (PFClusterObjectGridEJBHome) PortableRemoteObject.narrow(
    ic.lookup("java:comp/env/ejb/PFClusterObjectGridEJB"),
    PFClusterObjectGridEJBHome.class);
    final PFClusterObjectGridEJB session = home.create();
    String[] PARTITIONS = session.getAllPartitions();
    numOfPartitions=PARTITIONS.length;
    getNumOfPartitions=false;
   }
   catch (ClassCastException e) {
    e.printStackTrace();
    numOfPartitions=10;
   }
   catch (RemoteException e) {
    e.printStackTrace();
    numOfPartitions=10;
   }
   catch (NamingException e) {
    e.printStackTrace();
    numOfPartitions=10;
   }
   catch (CreateException e) {
    e.printStackTrace();
    numOfPartitions=10;
   }
  }
 }
/**
* Partitionsschlüssel zurückgeben
*
* @param partition
* @return String
*/
public static String getStock(String key) {
 return null;
}
/**
* Partitionsschlüssel zurückgeben
*
* @param key
* @return String
*/
```

```
public static String getServer(String key) {
 return key;
}
/**
* ObjectGrid-Daten von einem partitionierten Server abrufen, in dem
* die Änderung der Daten vorgenomemn wird (höchste Qualität und Integrität).
*
* @param ticket
* @return Hash-Code des Stock-Ticket
*/
public static String getStockOnPartitionedServer(String ticket) {
 if (ticket==null){
  return null;
 }
 getPartitionNumbers();
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Partitionsschlüssel zurückgeben
*
* @param stock
* @return Hash-Code des Stock-Ticket
*/
public static String updateStock(Stock stock) {
 getPartitionNumbers();
 String ticket=null;
 if (stock!=null){
  ticket=stock.getTicket();
 }
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Partitionsschlüssel zurückgeben
*
* @param stock
* @return Hash-Code des Stock-Ticket
*/
public static String insertStock(Stock stock) {
 getPartitionNumbers();
 String ticket=null;
 if (stock!=null){
  ticket=stock.getTicket();
 }
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Partitionsschlüssel zurückgeben
*
* @param server
* @return String
*/
public static String verifyObjectGrid(String server) {
 return server;
}
/**
* Partitionsschlüssel zurückgeben
*
* @param stock
* @return Hash-Code des Stock-Ticket
*/
public static String removeStock(String ticket) {
if (ticket==null){
 return null;
}
getPartitionNumbers();
 return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
```

```
/**
* Partitionsschlüssel zurückgeben
*
* @param partition
* @return
*/
public static String getAllPartitions() {
 return null;
}
}
```

Jede Remote-Methode muss eine entsprechende Methode haben, die eine gültige Zeichenfolge oder null zurückgibt.

Klasse SampleUtility und Partitionszuordnung

Verwenden Sie die Datei `SampleUtility.java`, um Stock-Tickets, Hash und Partitionen zu bearbeiten. Sie können diese Datei auch verwenden, um ObjectGrid-Schlüssel Partitionen zuzuordnen. Auf ähnliche Weise können Sie eine Dienstprogrammklasse entwickeln, die ObjectGrid-Schlüssel Partitionen zuordnet, die Ihren Geschäftsanforderungen entsprechen. Wenn Sie das Partitionierungs-Feature mit ObjectGrid verwenden möchten, müssen Sie unterschiedliche Schlüssel in unterschiedlichen Partitionen verwenden.

Klasse SampleUtility

Im Folgenden sehen Sie die Dienstprogrammklasse für die Beispielanwendung ObjectGridPartitionCluster:

```
/**
* Dienstprogrammklasse für die Beispielanwendung ObjectGridPartitionCluster
*
*
*/
public class SampleUtility {
 /**
 * Container für die Aufzeichnung von Partitionen
 */
 static Map serverPartitions= new HashMap();
 /**
 * Präfix für Partitionsnamen
 */

 public static String PARTITION_PREFIX = "ObjectGridHashPartition";
 /**
 * Präfix für Stock-Namen
 */

 public static String STOCK_PREFIX="Stock";
 /**
 * Zahl im Partitionsnamen abrufen
 *
 * @param partition
 * @return int
 */

 public static int getIntFromPartition(String partition){
  int result=-1;
  int pre=PARTITION_PREFIX.length();
  int p=partition.length();
  String num=partition.substring(pre, p);
  result=Integer.parseInt(num);
  return result;
 }
 /**
```

```
* Zahl im Stock-Ticket abrufen
*
* @param ticket
* @return
*/
public static int getIntFromStockTicket(String ticket){
 int result=-1;
 int pre=STOCK_PREFIX.length();
 int p=ticket.length();
 String num=ticket.substring(pre, p);
 result=Integer.parseInt(num);
 return result;
}
/**
* Stock-Ticket in einer bestimmten Hash-Basis speichern
*
* @param ticket
* @param base
* @return int
*/
public static int hashTicket(String ticket, int base){
 if (base<1){
  return 0;
 }
 int hash=0;
 int num=getIntFromStockTicket(ticket);
 hash= num % base;
 return hash;
}

/**
* Stock-Schlüssel in einer Paritition speichern
*
* @param ticket
* @param base
* @return String - partition name
*/
public static String hashStockKeyToPartition(String ticket, int base){
 String p=null;
 int hashcode=hashTicket(ticket, base)+1;
 p=PARTITION_PREFIX+ padZeroToString(hashcode+"", 6);
 return p;
}
/**
* Server/Partition aufzeichnen
*
* @param server
* @param partition
*/
public static void addServer(String server, String partition){
 serverPartitions.put(server, partition);
}
/**
* Server/Partition entfernen
*
* @param server
*/
public static void removeServer(String server){
 serverPartitions.remove(server);
}
/**
* Alle Server mit aktiven Partitionen abrufen
*
* @return Iterator - String
*/
```

```
 public static Iterator getAllServer(){
  return serverPartitions.values().iterator();
 }
}
```

Sie müssen dieselbe globale Hash-Tabelle verwenden und Variablen so auswerten, dass sie im Wertebereich der Hash-Tabelle liegen. Schauen Sie sich das folgende Beispiel an:

```
myKey.hashCode % hashBase
```

Sie müssen myKey als Hash-Variable auswerten und dieselbe Hash-Tabelle für verschiedene Server verwenden. Im vorherigen Beispiel wird dieselbe Variable in der Java-Umgebung gesucht. `key1 % 100` kann im Gegensatz zu `key2 % 90` nicht verwendet werden.

ObjectGrid-Instanz in der EJB-Methode setContext erstellen

Sie können die ObjectGrid-Instanz in der EJB-Methode setContext wie in der Datei `PFClusterObjectGridEJBBean.java` erstellen und die vorher geladenen Daten abrufen.

```
/**
* setSessionContext
*
* mit ObjectGrid-Instanz
*/
public void setSessionContext(javax.ejb.SessionContext ctx) {
 mySessionCtx = ctx;
 try {
  InitialContext ic = new InitialContext();
  // PartitionManager abrufen
  ivManager = (PartitionManager)
  ic.lookup("java:comp/websphere/wpf/PartitionManager");
  // enableDistribution-Konfiguration abrufen
  boolean enableDistribution = ((Boolean)
  ic.lookup("java:comp/env/enableDistribution")).booleanValue();
  System.out.println("***** enableDistribution="+ enableDistribution);
  // propagationMode-Konfiguration abrufen
  String propagationMode = (String) ic.lookup("java:comp/env/propagationMode");
  System.out.println("***** pMode="+ propagationMode);
  String pMode=null;
  if (propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_DEFAULT_KEY)||
   propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_INVALID_KEY) ){
   pMode=propagationMode;
  }
  // propagationVersionOption-Konfiguration abrufen
  String propagationVersionOption = (String)
  ic.lookup("java:comp/env/propagationVersionOption");
  System.out.println("***** pVersionOption="+ propagationVersionOption);
  String pVersion=null;
  if (propagationVersionOption.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_VERS_KEY)||
   propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_MODE_NOVERS_KEY) ){
    pVersion=propagationVersionOption;
  }
  // compressionMode-Konfiguration abrufen
  String compressionMode = (String) ic.lookup("java:comp/env/compressionMode");
  System.out.println("***** compressMode="+ compressionMode);
  String compressMode=null;
  if (compressionMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_COMPRESS_DISABLED)||
   propagationMode.equals(com.ibm.ws.objectgrid.Constants.
   OBJECTGRID_TRAN_PROPAGATION_COMPRESS_ENABLED) ){
```

```
   compressMode=compressionMode;
  }
  // wenn vorheriges Laden aktiviert ist
  bPreload = ((Boolean)
  ic.lookup("java:comp/env/preload")).booleanValue();
  System.out.println("***** enablePreload="+ bPreload);
  // wenn Entfernen aktiviert ist
  bRemove = ((Boolean)
  ic.lookup("java:comp/env/remove")).booleanValue();
  System.out.println("***** enableRemove="+ bRemove);
  // wenn Loader aktiviert ist
  boolean bLoader = ((Boolean)
  ic.lookup("java:comp/env/loader")).booleanValue();
  System.out.println("***** enableLoader="+ bLoader);
  // Dateipfad und -namen abrufen
  String filePathandName = (String)
  ic.lookup("java:comp/env/filePathandName");
  System.out.println("***** fileName="+ filePathandName);
  // ObjectGrid-Instanz abrufen
  og=ObjectGridFactory.getObjectGrid(ogName,
  enableDistribution,
  pMode, pVersion,
  compressMode, bLoader,
  filePathandName);
  if (og==null){
   throw new RuntimeException
    ("ObjectGrid instance is null in ObjectGridPartitionClusterSample");
  }
  System.out.println("Bean Context, getObjectGrid="
  + og + " for name="+ ogName);
  if (bPreload && !lock){
   System.out.println("Preload data");
   PersistentStore store=PersistentStore.getStore(filePathandName);
   store.preload(10);
   store.verify(10);
   lock=true;
   preloadData=store.getAllRecords();
  }
 }
 catch (Exception e) {
  logger.logp(Level.SEVERE, CLASS_NAME,
  "setSessionContext", "Exception: " + e);
  throw new EJBException(e);
 }
}
```

Singleton-Klasse ObjectGridFactory

Eine ObjectGrid-Instanz wird mit einer benutzerdefinierten Factory erstellt, die die ObjectGrid-Instanz mit benutzerdefinierten Einstellungen zwischenspeichert.

Das folgende Beispiel zeigt, wie Sie eine ObjectGrid-Instanz über das Programm erstellen, das ObjectGridTransformer-Objekt setzen, den Listener für Weitergabeereignisse konfigurieren und diesen Listener für die ObjectGrid-Instanz definieren. Sie können für diese Konfiguration auch eine XML-Datei verwenden.

```
/**
*
* ObjectGrid-Instanz erstellen und konfigurieren.
*
*
*/
public class ObjectGridFactory {
 /**
 * ObjectGrid-Name
 */
 static String ogName="WPFObjectGridSample";
```

```
/**
* ObjectGrid-Instanz
*/
static ObjectGrid og=null;
/**
* ObjectGrid-Sitzung
*/
static Session ogSession=null;
/**
* Map-Name
*/
static String mapName="SampleStocks";
/**
* ObjectGrid-Cache
*/
static Map ogCache= new HashMap();
/**
* ObjectGrid-Instanz abrufen
*
* @param ogn
* @param enableDist
* @param pMode
* @param pVersion
* @param compressMode
* @return
*/

public static synchronized ObjectGrid getObjectGrid(String ogn,
 boolean enableDist,
 String pMode,
 String pVersion,
 String compressMode,
 boolean loader,
 String fileName){
  if (ogn!=null){
   ogName=ogn;
  }
  else {
   throw new IllegalArgumentException ("ObjectGrid name given is null");
  }
  if (ogCache.containsKey(ogName)){
   return (ObjectGrid) ogCache.get(ogName);
  }
  try {
   ObjectGridManager manager= ObjectGridManagerFactory.
   getObjectGridManager();
   og=manager.createObjectGrid(ogName);
   if (enableDist){
    TranPropListener tpl=new TranPropListener();
     if (pMode!=null){
      tpl.setPropagateMode(pMode);
     }
     if (pVersion!=null){
      tpl.setPropagateVersionOption(pVersion);
     }
     if (compressMode!=null) {
      tpl.setCompressionMode(compressMode);
     }
     og.addEventListener(tpl);
   }
   // BackingMap definieren und Loader festlegen
  BackingMap bm = og.defineMap(mapName);
  ObjectTransformer myTransformer=
  new MyStockObjectTransformer();
  bm.setObjectTransformer(myTransformer);
  OptimisticCallback myOptimisticCallback=
  new MyStockOptimisticCallback();
```

```
  if (loader){
   TransactionCallback tcb=new MyTransactionCallback();
   Loader myLoader= new MyCacheLoader(fileName, mapName);
   og.setTransactionCallback(tcb);
   bm.setLoader(myLoader);
  }
  og.initialize();
  ogCache.put(ogName, og);
 }
 catch (Exception e) {
 }
 return og;
 }
}
```

Vorheriges Laden einer ObjectGrid-Partition

Dieser Abschnitt beschreibt, wie Sie Daten vorab in eine ObjectGrid-Instanz laden können.

Mit der Methode partitionLoadEvent können Sie Objekte für die jeweilige Partition nur laden, wenn die Partition aktiviert ist. Wenn die Partitionierung aktiviert ist und Sie Objekte laden, wird ObjectGrid so partitioniert, dass ObjectGrid große Mengen von Objekten verarbeiten kann.

```
/**
* Diese Methode wird aufgerufen, wenn dem Serverprozess eine bestimmte
* Partition zugeordnet ist.
* @param partitionName
* @return
*/
public boolean partitionLoadEvent(String partitionName) {
 // Daten vorher laden
 preloadDataForPartition(partitionName);
 logger.logp(
  Level.FINER,
  CLASS_NAME,
  "partitionLoadEvent",
  "Loading "+ partitionName );
  return true;
}
/**
*
* Daten vorher laden
*
* @param partition
*/

private synchronized void preloadDataForPartition(String partition){
 if (bPreload && (preloadData!=null)){
  Iterator itr=preloadData.keySet().iterator();
  while (itr.hasNext()){
   String ticket= (String) itr.next();
   String p=SampleUtility.
   hashStockKeyToPartition(ticket, numOfPartitions);
   if (partition.equals(p)){
    Stock stock= (Stock) preloadData.get(ticket);
    System.out.println("preload in partition=" +
    partition + " with data ticket="+ ticket);
    insertStock(stock);
   }
  }
 }
}
```

Möglicherweise müssen Sie die verteilten Aktualisierungen inaktivieren, wenn Sie für die Partitionierung einer großen ObjectGrid-Instanz das vorherige Laden mit Partitionierung verwenden. Die aktuelle Version für verteilte Aktualisierungen kann nicht partitioniert werden. Das WPF-Feature für kontextbasierte Weiterleitung sucht die richtigen Daten in der richtigen Partition.

Kapitel 7. Best Practices für die Leistung von ObjectGrid

Sie können die Leistung einer ObjectGrid-Map mit den folgenden Best Practices verbessern. Diese Best Practices werden nur im Kontext der Anwendung und ihrer Architektur implementiert.

Jede Anwendung und jede Umgebung verwendet eine andere Lösung für die Leistungsoptimierung. ObjectGrid unterstützt integrierte Anpassungen zur Verbesserung der Leistung, aber Sie können Leistungsverbesserungen auch durch Änderungen in der Anwendungsarchitektur erzielen. Leistungsverbesserungen können in den folgenden Bereichen erzielt werden:

- „Best Practices für Sperrstrategien"

  Wählen Sie zwischen unterschiedlichen Sperrstrategien, die sich auf die Leistung Ihrer Anwendungen auswirken können.
- „Best Practices für die Methode copyMode" auf Seite 188

  Wählen Sie zwischen unterschiedlichen Kopiermodi, mit denen festgelegt werden kann, wie ObjectGrid Einträge verwaltet und kopiert.
- „Best Practices für das Interface ObjectTransformer" auf Seite 192

  Verwenden Sie das Interface ObjectTransformer, damit Callbacks für die Anwendung benutzerdefinierte Implementierungen gebräuchlicher und kostenintensiver Operationen wie Objektserialisierung und Deep Copy für ein Objekt bereitstellen können.
- „Best Practices für die Leistung des Plug-in-Evictor" auf Seite 194

  Wählen Sie zwischen der LFU- und der LRU-Bereinigungsstrategie.
- „Best Practices für Standard-Evictor" auf Seite 196

  Merkmale für den Standard-TTL-Evictor, den Standard-Evictor, der mit jeder BackingMap erstellt wird.

Best Practices für Sperrstrategien

Sperrstrategien können sich auf die Leistung Ihrer Anwendungen auswirken.

Nähere Einzelheiten zu den folgenden Sperrstrategien finden Sie im Abschnitt „Sperren" auf Seite 64.

Pessimistische Sperrstrategie

Sie können die pessimistische Sperrstrategie für Lese- und Schreiboperationen in Maps verwenden, wenn die Anzahl der Schlüsselkollisionen sehr hoch ist. Die pessimistische Sperrstrategie hat die größten Auswirkungen auf die Leistung.

Optimistische Sperrstrategie

Die optimistische Sperrstrategie ist die Standardkonfiguration. Diese Strategie bietet im Vergleich mit der pessimistischen Strategie eine höhere Leistung und Skalierbarkeit. Verwenden Sie diese Strategie, wenn Ihre Anwendungen einige optimistische Aktualisierungsfehler hinnehmen können, wenn die Leistung trotzdem höher ist als bei der pessimistischen Strategie. Diese Strategie eignet sich bestens für Anwendungen, die in einem höheren Umfang lesen als aktualisieren.

Strategie ohne Sperren

Die Strategie ohne Sperren (NONE) eignet sich für Anwendungen, die nur lesen. Diese Strategie fordert keine Sperren an. Deshalb bietet sie in Bezug auf die Anzahl gemeinsamer Zugriffe, Leistung und Skalierbarkeit die meisten Vorzüge.

Best Practices für die Methode copyMode

ObjectGrid erstellt basierend auf der Einstellung von CopyMode eine Kopie des Wertes. Mit der Methode setCopyMode(CopyMode, valueInterfaceClass) der API BackingMap können Sie den Kopiermodus auf eines der folgenden Felder des Typs ″final static″ setzen, die in der Klasse com.ibm.websphere.objectgrid.CopyMode definiert sind.

Wenn eine Anwendung das Interface ObjectMap verwendet, um eine Referenz auf einen Map-Wert abzurufen, wird empfohlen, diese Referenz nur in der ObjectGrid-Transaktion zu verwenden, die die Referenz abgerufen hat. Die Verwendung der Referenz in einer anderen ObjectGrid-Transaktion kann zu Fehlern führen. Wenn Sie beispielsweise die pessimistische Sperrstrategie für die BackingMap verwenden, fordert ein Aufruf der Methode get oder getForUpdate eine S- (gemeinsam genutzt) bzw. U-Sperre (Aktualisierung) an. Die Methode get gibt die Referenz auf den Wert, und die angeforderte Sperre wird freigegeben, wenn die Transaktion abgeschlossen wird. Die Transaktion muss die Methode get oder getForUpdate aufrufen, um den Map-Eintrag in einer anderen Transaktion zu sperren. Jede Transaktion muss ihre eigene Referenz auf den Wert anfordern, indem sie die Methode get oder getForUpdate aufruft, anstatt dieselbe Wertreferenz in mehreren Transaktionen wiederzuverwenden.

Verwenden Sie die folgenden Informationen, wenn Sie den Kopiermodus auswählen:

Modus COPY_ON_READ_AND_COMMIT

Der Modus COPY_ON_READ_AND_COMMIT ist der Standardmodus. Das Argument valueInterfaceClass wird in diesem Modus ignoriert. Dieser Modus stellt sicher, dass eine Anwendung keine Referenz auf das Wertobjekt in der BackingMap besitzt und stattdessen mit einer Kopie des Wertes arbeitet. Der Modus copy_on_read stellt sicher, dass die Anwendung die in der BackingMap zwischengespeicherten Daten nicht versehentlich beschädigt. Wenn eine Anwendungstransaktion eine Methode ObjectMap.get für einen bestimmten Schlüssel aufruft und es sich um den ersten Zugriff des ObjectMap-Eintrags für diesen Schlüssel handelt, wird eine Kopie des Wertes zurückgegeben. Beim Festschreiben der Transaktion werden alle von der Anwendung festgeschriebenen Änderungen in die BackingMap kopiert, um sicherzustellen, dass die Anwendung keine Referenz auf den festgeschriebenen Wert in der BackingMap enthält.

Modus COPY_ON_READ

Der Modus COPY_ON_READ bietet im Vergleich mit dem Modus COPY_ON-_READ_AND_COMMIT eine bessere Leistung, weil der Kopiervorgang beim Festschreiben einer Transaktion wegfällt. Das Argument valueInterfaceClass wird in diesem Modus ignoriert. Um die Integrität der BackingMap-Daten zu gewährleisten, stellt die Anwendung sicher, jede Referenz auf einen Eintrag zu löschen, nachdem die Transaktion festgeschrieben wurde. In diesem Modus gibt eine Methode ObjectMap.get eine Kopie des Wertes anstelle einer Referenz auf den Wert zurück, um sicherzustellen, dass die Änderungen, die von der Anwendung an dem Wert

vorgenommen werden, so lange keine Auswirkungen auf den BackingMap-Wert haben, bis die Transaktion festgeschrieben wird. Es wird jedoch keine Kopie der Änderungen erstellt, wenn die Transaktion festgeschrieben wird. Stattdessen wird die Referenz auf die Kopie, die von der Methode ObjectMap.get zurückgegeben wurde, in der BackingMap gespeichert. Die Anwendung löscht alle Einträge auf Map-Einträge, wenn die Transaktion festgeschrieben wird. Sollte die Anwendung dies nicht tun, ist es möglich, dass die Anwendung die in der BackingMap zwischengespeicherten Daten beschädigt. Wenn eine Anwendung diesen Modus verwendet und Probleme auftreten, wechseln Sie in den Modus COPY_ON_READ_AND_COMMIT, um zu prüfen, ob das Problem dort auch auftritt. Falls der Fehler nicht mehr auftritt, kann die Anwendung nicht alle Referenzen beim Festschreiben der Transaktion löschen.

Modus COPY_ON_WRITE

Der Modus COPY_ON_WRITE bietet im Vergleich mit dem Modus COPY_ON_READ_AND_COMMIT eine bessere Leistung, weil der Kopiervorgang beim ersten Aufruf der Methode ObjectMap.get für einen bestimmten Schlüssel wegfällt. Die Methode ObjectMap.get gibt einen Proxy anstelle einer direkten Referenz auf das Wertobjekt zurück. Der Proxy stellt sicher, dass nur dann eine Kopie des Wertes erstellt wird, wenn die Anwendung die Methode set im Value-Interface aufruft, die im Attribut valueInterfaceClass angegeben ist. Der Proxy verwendet eine Implementierung von *copy on write*. Wenn eine Transaktion festgeschrieben wird, prüft die BackingMap den Proxy, um festzustellen, ob beim Aufruf einer Methode set eine Kopie erstellt worden ist. Falls eine Kopie erstellt wurde, wird die Referenz auf diese Kopie in der BackingMap gespeichert. Dieser Modus hat den Vorteil, dass ein Wert bei einer read- oder commit-Operation nicht kopiert wird, wenn die Transaktion keine Methode set zum Ändern des Wertes aufruft.

In den Modi COPY_ON_READ_AND_COMMIT und COPY_ON_READ wird eine tiefe Kopie (Deep Copy) erstellt, wenn ein Wert aus der ObjectMap abgerufen wird. Wenn eine Anwendung nur einige Werte aktualisiert, die in einer Transaktion abgerufen werden, ist dieser Modus nicht optimal. Der Modus COPY_ON_WRITE unterstützt dieses Verhalten effizient, setzt aber voraus, dass die Anwendung ein einfaches Muster verwendet. Die Wertobjekte müssen ein Interface unterstützen. Die Anwendung muss die Methoden in diesem Interface verwenden, wenn sie mit dem Wert in einer ObjectGrid-Session interagiert. Wenn dies der Fall ist, erstellt die ObjectGrid-Instanz Proxys für die Werte, die an die Anwendung zurückgegeben werden. Der Proxy hat eine Referenz auf den echten Wert. Wenn die Anwendung nur Leseoperationen ausführt, wird hierfür immer die echte Kopie verwendet. Wenn die Anwendung ein Attribut im Objekt ändert, erstellt der Proxy eine Kopie des echten Objekts und nimmt anschließend die Änderung an der Kopie vor. Danach verwendet der Proxy die Kopie. Auf diese Weise kann die Kopieroperation für Objekte, die von der Anwendung nur gelesen werden, vollständig vermieden werden. Alle Änderungsoperationen müssen mit dem Präfix set beginnen. Enterprise-JavaBeans verwenden diese Art der Methodenbenennung normalerweise für Methoden, die Objektattribute ändern. Diese Konvention muss befolgt werden. Alle Objekte werden kopiert, wenn sie von der Anwendung geändert werden. Dies ist das effizienteste Szenario für Lese- und Schreiboperationen, das vom ObjectGrid unterstützt wird. Wenn eine Map den Modus COPY_ON_WRITE verwenden soll, konfigurieren Sie sie wie folgt: In diesem Beispiel möchte die Anwendung Person-Objekte mit dem Schlüssel name in der Map speichern. Schauen Sie sich das Person-Objekt im folgenden Code-Snippet an:

```
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;
 }
}
```

Die Anwendung verwendet das Interface IPerson nur, wenn sie mit Werten interagiert, die von einer ObjectMap-Instanz abgerufen wurden. Sie können das Objekt wie im folgenden Beispiel gezeigt so ändern, dass es ein Interface verwendet:

```
interface IPerson
{
 void setName(String n);
 String getName();
 void setAge(int a);
 int getAge();
}
// Person ändern, um Interface IPerson zu implementieren
class Person implements IPerson
{
 ...
}
```

Anschließend muss die Anwendung die BackingMap, wie im folgenden Beispiel gezeigt, so konfigurieren, dass sie den Modus COPY_ON_WRITE verwendet:

```
ObjectGrid dg = ...;
BackingMap bm = dg.defineMap("PERSON");
// COPY_ON_WRITE für diese Map mit
// IPerson als valueProxyInfo-Klasse verwenden
bm.setCopyMode(CopyMode.COPY_ON_WRITE,IPerson.class);
// Die Anwendung muss bei der Verwendung der Map
// PERSON das folgende Muster verwenden.
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
...
sess.begin();
// Die Anwendung setzt den zurückgegebenen Wert in IPerson und nicht in Person um
IPerson p = (IPerson)person.get("Billy");
p.setAge(p.getAge()+1);
...
// Neues Person-Objekt erstellen und zur Map hinzufügen
Person p1 = new Person();
p1.setName("Bobby");
p1.setAge(12);
person.insert(p1.getName(), p1);
sess.commit();
// Das folgende Snippet FUNKTIONIERT NICHT. Es führt zu einer ClassCastException
sess.begin();
```

```
// Der Fehler ist hier, dass Person anstelle von
// IPerson verwendet wird.
Person a = (Person)person.get("Bobby");
sess.commit();
```

Der erste Abschnitt zeigt, wie die Anwendung den Wert Billy aus der Map abruft. Die Anwendung setzt den zurückgegebenen Wert in ein IPerson-Objekt und nicht in das Objekt Person um, da der zurückgegebene Proxy zwei Interfaces implementiert:

- das im Methodenaufruf BackingMap.setCopyMode angegebene Interface
- das Interface com.ibm.websphere.objectgrid.ValueProxyInfo

Sie können den Proxy in zwei Typen umsetzen. Der letzte Teil des vorherigen Code-Snippets zeigt, was im Modus COPY_ON_WRITE nicht zulässig ist. Die Anwendung ruft den Datensatz Bobby ab und versucht ihn, in ein Objekt Person umzusetzen. Diese Aktion schlägt mit einer Ausnahme bei Klassenumsetzung fehl, weil der zurückgegebene Proxy kein Person-Objekt ist. Der zurückgegebene Proxy implementiert das Objekt IPerson und ValueProxyInfo.

Interface ValueProxyInfo und Unterstützung für Teilaktualisierung

Dieses Interface ermöglicht einer Anwendung, den festgeschriebenen schreibgeschützten Wert, den der Proxy referenziert, oder die Attribute abzurufen, die während der Transaktion geändert wurden.

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

Die Methode ibmGetRealValue gibt eine schreibgeschützte Kopie des Objekts zurück. Die Anwendung darf diesen Wert nicht ändern. Die Methode ibmGetDirtyAttributes gibt eine Liste mit Zeichenfolgen zurück, die die Attribute darstellen, die von der Anwendung während der Transaktion geändert wurden. Hauptsächlich wird die Methode ibmGetDirtyAttributes in einem JDBC- oder CMP-basierten Loader verwendet. Es müssen nur die in der Liste angegebenen Attribute in der SQL-Anweisung oder dem der Tabelle zugeordneten Objekt aktualisiert werden. Die vom Loader generierte SQL ist dann effizienter. Wenn eine Transaktion des Typs ″copy on write″ festgeschrieben wird oder ein Loader implementiert ist, kann der Loader die Werte der geänderten Objekte in das Interface ValueProxyInfo umsetzen, um diese Informationen abzurufen.

Handhabung der Methode equals bei der Verwendung von COPY_ON_WRITE oder Proxys.

Der folgende Code erstellt ein Objekt Person und fügt es anschließend in eine ObjectMap ein. Danach ruft er dasselbe Objekt mit der Methode ObjectMap.get ab. Der Wert wird in das Interface umgesetzt. Wenn der Wert in das Interface Person umgesetzt wird, hat dies eine Ausnahme des Typs ClassCastException zur Folge, weil der zurückgegebene Wert ein Proxy ist, der das Interface IPerson und kein Person-Objekt implementiert. Die Gleichheitsprüfung schlägt fehl, wenn Sie die Operation == verwenden, weil es sich nicht um dasselbe Objekt handelt.

```
session.begin();
// new the Person object
Person p = new Person(...);
personMap.insert(p.getName, p);
// Objekt erneut abrufen und daran denken, für die Umsetzung das Interface zu verwenden
```

```
IPerson p2 = personMap.get(p.getName());
if(p2 == p)
{
// die Objekte sind identisch
}
else
{
// die Objekte sind nicht identisch
}
```

Beim Überschreiben der Methode equals muss ein weiterer Punkt berücksichtigt werden. Wie im folgenden Code-Snippet gezeigt, muss die Methode equals prüfen, ob das Argument ein Objekt ist, das das Interface IPerson implementiert, und das Argument in ein IPerson-Objekt umsetzen. Da das Argument ein Proxy sein kann, der das Interface IPerson implementiert, müssen Sie die Methoden getAge und getName verwenden, wenn Sie Instanzvariablen vergleichen.

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

Modus NO_COPY

Im Modus `NO_COPY` kann eine Anwendung zusichern, dass sie ein Wertobjekt, das mit einer Methode `ObjectMap.get` angefordert wird, nicht ändert, um Leistungsvorteile zu erzielen. Das Argument valueInterfaceClass wird in diesem Modus ignoriert. Wenn Sie diesen Modus verwenden, wird der Wert nicht kopiert. Wenn die Anwendung Werte ändert, sind die Daten in der BackingMap beschädigt. Der Modus NO_COPY ist in erster Linie für schreibgeschützte Maps sinnvoll, in denen die Daten von der Anwendung nicht geändert werden. Wenn die Anwendung diesen Modus verwendet und Probleme auftreten, wechseln Sie in den Modus COPY_ON_READ_AND_COMMIT, um zu prüfen, ob das Problem dort auch auftritt. Sollte das Problem in diesem Modus nicht auftreten, ändert die Anwendung den von der Methode ObjectMap.get zurückgegebenen Wert während der Transaktion oder nach der Festschreibung der Transaktion.

Best Practices für das Interface ObjectTransformer

ObjectTransformer verwendet Callbacks für die Anwendung, um benutzerdefinierte Implementierungen gebräuchlicher und kostenintensiver Operationen wie Objektserialisierung und Deep Copy für ein Objekt bereitzustellen.

Spezielle Informationen zum Interface ObjectTransformer finden Sie im Abschnitt „Plug-in ObjectTransformer" auf Seite 115. Vom Leistungsstandpunkt aus betrachtet und wie auch aus den Informationen zur Methode CopyMode im Abschnitt „Best Practices für die Methode copyMode" auf Seite 188 hervorgeht, kopiert ObjectGrid die Werte in allen Fällen mit Ausnahme des Modus NO_COPY. Der Standardkopiermechanismus, der in ObjectGrid eingesetzt wird, ist die Serialisierung, eine bekanntermaßen kostenintensive Operation. In dieser Situation kann das Interface ObjectTransformer verwendet werden. Das Interface ObjectTransformer verwendet Callbacks für die Anwendung, um benutzerdefinierte Implementierungen gebräuchlicher und kostenintensiver Operationen wie Objektserialisierung und Deep Copy für ein Objekt bereitzustellen.

Eine Anwendung kann einer Map eine Implementierung des Interface ObjectTransformer bereitstellen. Die ObjectGrid-Instanz delegiert dann an die Methoden in diesem Objekt und erwartet von der Anwendung, dass sie eine optimierte Version jeder Methode im Interface bereitstellt. Schauen Sie sich das Interface ObjectTransformer an:

```
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);
}
```

Mit Code wie dem folgenden können Sie einem ObjectTransformer-Interface eine BackingMap-Instanz zuordnen:

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

Objektserialisierung und -erweiterung

Die Objektserialisierung ist normalerweise das größte Leistungsproblem in ObjectGrid. ObjectGrid verwendet den Standardserialisierungsmechanismus, wenn die Anwendung kein ObjectTransformer-Plug-in bereitstellt. Eine Anwendung kann Implementierungen des Serializable readObject und writeObject bereitstellen oder das Interface Externalizable von Objekten implementieren lassen, was in etwa 10 Mal schneller ist. Wenn die Objekte in der map nicht geändert werden können, kann eine Anwendung der ObjectMap einen ObjectTransformer zuordnen. Mit den Methoden serialize und inflate kann die Anwendung benutzerdefinierten Code bereitstellen, um diese Optionen im Hinblick auf ihre große Leistungsauswirkungen auf das System zu optimieren. Die serialize-Methoden serialisieren das Objekt und stellen einen Datenstrom bereit. Die Methode serialisiert die Methode in den bereitgestellten Datenstrom. Die inflate-Methoden stellen den Eingabedatenstrom bereit und erwarten, dass die Anwendung das Objekt erstellt, mit den Daten im Datenstrom erweitert und das Objekt anschließend zurückgibt. Die Implementierungen der serialize- und inflate-Methoden müssen sich gegenseitig spiegeln.

Deep-Copy-Operationen optimieren

Wenn eine Anwendung ein Objekt von einer ObjectMap empfängt, führt die ObjectGrid-Instanz eine Deep-Copy-Operation für den Objektwert aus, um sicherzustellen, dass die Kopie in der BaseMap gesichert ist. Die Anwendung kann den Objektwert anschließend sicher ändern. Beim Festschreiben der Transaktion wird die Kopie des Objektwertes in der BaseMap-Map mit dem neuen geänderten Wert aktualisiert, und die Anwendung verwendet diesen Wert danach nicht mehr. Sie könnten das Objekt in der Festschreibungsphase erneut kopiert haben, um eine private Kopie zu erstellen, aber in diesem Fall wurden die Leistungseinbußen dieser Aktion hingenommen und dem Anwendungsprogrammierer nicht mitgeteilt, den Wert nach der Transaktionsfestschreibung nicht mehr zu verwenden. Der Standardmechanismus für das Kopieren von Objekten versucht, einen Klon oder ein serialize/inflate-Paar zu verwenden, um eine Kopie zu generieren. Das

serialize/inflate-Paar ist im Hinblick auf die Leistung das ungeeignetste Szenario. Wenn bei einer Profilerstellung festgestellt wird, dass die Ausführung von serialize- und inflate-Operationen ein Problem für Ihre Anwendung darstellt, verwenden Sie ein ObjectTransformer-Plug-in und implementieren Sie die Methoden `copyValue` und `copyKey` mit einer effizienteren Objektkopie.

Best Practices für die Leistung des Plug-in-Evictor

Wenn Sie Plug-in-Evictor verwenden möchten, müssen Sie diese erstellen und anschließend der BackingMap mitteilen, dass sie diese verwenden soll. Halten Sie sich beim Erstellen von LFU- und LRU-Evictor an die folgenden Best Practices und Leistungstipps.

LFU-Evictor

Das Konzept eines LFU-Evictor sieht vor, dass die Einträge aus der Map entfernt werden, die nur selten verwendet werden. Die Einträge der Map sind über eine festgelegte Gruppe von binären Heap-Speichern verteilt. Mit zunehmender Verwendung eines bestimmten Cache-Eintrags wird dieser im Heap-Speicher höher eingestuft. Wenn der Evictor versucht, den Cache zu bereinigen, entfernt er nur die Cache-Einträge, die sich unterhalb einer bestimmte Stelle im binären Heap-Speicher befinden. So werden nur die am seltensten verwendeten Einträge gelöscht.

LRU-Evictor

Der LRU-Evictor verwendet bis auf wenige Ausnahmen dieselben Konzepte wie der LFU-Evictor. Der Hauptunterschied besteht darin, dass der LRU-Evictor eine FIFO-Warteschlange (First in, First out) anstelle binärer Heap-Speicher verwendet. Bei jedem Zugriff auf einen Cache-Eintrag wird dieser an den Anfang der Warteschlange verschoben. Deshalb sind am Anfang der Warteschlange die Map-Einträge, die zuletzt verwendet wurden, und am Ende der Warteschlange die Map-Einträge enthalten, die am längsten nicht verwendet wurden. Wenn der Cache-Eintrag A beispielsweise 50 Mal und der Cache-Eintrag B nur einmal und zwar direkt nach dem Cache-Eintrag A aufgerufen wird, befindet sich der Cache-Eintrag B am Anfang und der Cache-Eintrag A am Ende der Warteschlange, da der Cache-Eintrag B zuletzt verwendet wurde. Der LRU-Evictor löscht die Cache-Einträge, die sich am Ende der Warteschlange befinden, weil diese am längsten nicht verwendet wurden.

LFU- und LRU-Merkmale und Best Practices zur Verbesserung der Leistung

Anzahl der Heap-Speicher
Wenn Sie den LFU-Evictor verwenden, werden alle Cache-Einträge für eine bestimmte Map auf die angegebene Anzahl von Heap-Speichern verteilt. Dies verbessert die Leistung erheblich und verhindert, dass alle Bereinigungen mit nur einem binären Heap-Speicher synchronisiert werden, der die gesamte Eintragsreihenfolge für die Map enthält. Außerdem verringert sich bei der Verwendung von mehr Heap-Speichern die Zeit, die für die Neusortierung der Heap-Speicher benötigt wird, da jeder dieser Heap-Speicher weniger Einträge enthält als ein großer Heap-Speicher. Setzen Sie die Anzahl der Heap-Speicher auf 10 % der Anzahl von Einträgen in der Basis-Map.

Anzahl der Warteschlangen
Wenn Sie den LRU-Evictor verwenden, werden alle Cache-Einträge für eine bestimmte Map auf die angegebene Anzahl von Warteschlangen ver-

teilt. Dies verbessert die Leistung erheblich und verhindert, dass alle Bereinigungen mit nur einer Warteschlange synchronisiert werden, die die gesamte Eintragsreihenfolge für die Map enthält. Setzen Sie die Anzahl der Warteschlangen auf 10 % der Anzahl von Einträgen in der Basis-Map.

Merkmal MaxSize
Wenn ein LFU- oder LRU-Evictor mit dem Löschen von Einträgen beginnt, bestimmt er anhand des Evictor-Merkmals MaxSize, wie viele binäre Heap-Speicher oder LRU-Warteschlangenelemente gelöscht werden müssen. Angenommen, Sie definieren die Anzahl der Heap-Speicher oder Warteschlangen so, dass jeder Heap-Speicher bzw. jede Map-Warteschlange ungefähr 10 Map-Einträge enthält. Wenn das Merkmal MaxSize auf 7 gesetzt ist, löscht der Evictor drei Einträge aus jedem Heap-Speicher bzw. jedem Warteschlangenobjekt, um die Größe jedes Heap-Speichers bzw. jeder Warteschlange auf 7 zu verringern. Der Evictor löscht nur dann Map-Einträge aus einem Heap-Speicher oder einer Warteschlange, wenn der Heap-Speicher bzw. die Warteschlange mehr Elemente enthält, als der Wert des Merkmals MaxSize vorgibt. Setzen Sie das Merkmal MaxSize auf 70 % der Größe Ihres Heap-Speichers bzw. Ihrer Warteschlange. In diesem Beispiel wird der Wert 7 verwendet. Die ungefähre Größe eines Heap-Speichers bzw. einer Warteschlange können Sie ermitteln, indem Sie die Anzahl der BaseMap-Einträge durch die Anzahl der verwendeten Heap-Speicher bzw. Warteschlangen teilen.

Merkmal SleepTime
Ein Evictor entfernt Einträge nicht ständig aus der Map. Vielmehr wird er in einem bestimmten Intervall von *n* Sekunden aktiviert, wobei *n* für den Wert des Merkmals SleepTime steht. Dieses Merkmal wirkt sich positiv auf die Leistung aus. Wenn die Bereinigungsdurchläufe zu häufig ausgeführt werden, verringert sich die Leistung, weil für diese Verarbeitung Ressourcen benötigt werden. Wird der Evictor zu selten aktiviert, kann die Map zu viele nicht benötigte Einträge enthalten. Eine Map, die lauter nicht mehr benötigte Einträge enthält, kann sich negativ auf den Speicherbedarf und die erforderlichen Verarbeitungsressourcen für die Map auswirken. Für die meisten Maps empfiehlt es sich, das Intervall für die Bereinigungsdurchläufe auf 15 Sekunden einzustellen. Wenn häufig in die Map geschrieben wird und die Map eine hohe Transaktionsrate aufweist, kann es hilfreich sein, das Intervall zu verkürzen. Ein längeres Intervall eignet sich besser für Maps mit einer niedrigen Zugriffsrate.

Beispiel

Das folgende Beispiel definiert eine Map, erstellt einen neuen LFU-Evictor, setzt die Evictor-Merkmale und konfiguriert den Evictor für die Map:

```
// ObjectGridManager verwenden, um ObjectGrid zu erstellen/abzurufen. Lesen
// Sie den Abschnitt zu ObjectGridManger.
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");

// Merkmale setzen. Dabei von 50.000 Map-Einträgen ausgehen
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Der LRU-Evictor wird ähnlich verwendet wie ein LFU-Evictor. Es folgt ein Beispiel:

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");

// Merkmale setzen. Dabei von 50.000 Map-Einträgen ausgehen
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

Beachten Sie, dass sich nur zwei Zeilen vom LFUEvictor-Beispiel unterscheiden.

Best Practices für Standard-Evictor

Best Practices für Standard-*TTL*-Evictor.

Zusätzlich zu den Plug-in-Evictor, die im Abschnitt „Best Practices für die Leistung des Plug-in-Evictor" auf Seite 194 beschrieben sind, wird für jede BackingMap ein Standard-TTL-Evictor erstellt. Der Standard-Evictor entfernt Einträge nach einem Konzept, das auf der *Lebensdauer* (TTL, Time to Live) basiert. Dieses Verhalten wird mit dem Attribut ttlType definiert. Es sind drei ttlTypes-Attribute verfügbar:

- **None**: Gibt an, dass Einträge nicht verfallen und deshalb nicht aus der Map entfernt werden.
- **Creation time**: Gibt an, dass Einträge auf der Basis der Erstellungszeit entfernt werden.
- **Last accessed time**: Gibt an, dass Einträge basierend auf der letzten Zugriffszeit entfernt werden.

Standard-Evictor-Merkmale und Best Practices für die Leistung

Merkmal TimeToLive

Dieses Merkmal ist aus der Sicht der Leistung zusammen mit dem Merkmal ttlType das wichtigste Merkmal. Wenn Sie ttlType auf CREATION_TIME setzen, entfernt der Evictor einen Eintrag, wenn dessen Lebensdauer seit der Erstellungszeit dem Wert des Attributs TimeToLive entspricht. Wenn Sie das Attribut TimeToLive auf 10 Sekunden setzen, werden alle Einträge in der Map nach Sekunden entfernt. Wenn Sie diesen Wert für den den TTL-Typ CREATION_TIME festlegen, müssen Sie vorsichtig vorgehen. Dieser Evictor eignet sich, wenn dem Cache sehr viele Einträge hinzugefügt werden, die nur für bestimmte Zeit genutzt werden. Mit dieser Strategie werden alle erstellten Einträge nach einer bestimmten Zeit entfernt.

Im Folgenden wird ein Beispielszenario beschrieben, in dem die Verwendung des TTL-Typs CREATION_TIME sinnvoll ist. Sie verwenden eine Webanwendung, die Börsennotierungen abruft und die bezüglich der Aktualität der Notierungen nicht kritisch ist. In diesem Fall werden die Börsennotierungen 20 Minuten lang in einer ObjectGrid-Cache zwischengespeichert. Nach 20 Minuten verfallen die Einträge in der ObjectGrid-Map und werden daraufhin entfernt. Alle zwanzig Minuten verwendet die ObjectGrid-Map das Plug-in Loader, um die Map-Daten mit aktuellen Daten aus der Datenbank zu aktualisieren. Die Datenbank wird alle 20 Minuten mit den aktuellen Börsennotierungen aktualisiert. Für diese Anwendung ist somit ein TimeToLive-Wert von 20 Minuten ideal.

Wenn Sie den TTL-Typ LAST_ACCESSED_TIME verwenden, setzen Sie TimeToLive auf einen niedrigeren Wert als beim TTL-Typ CREATION_TIME, da das Attribut TimeToLive der Einträge bei jedem Zugriff

zurückgesetzt wird. Anders ausgedrückt, wenn das Attribut TimeToLive auf 15 Sekunden gesetzt ist, ein Eintrag 14 Sekunden alt ist und dann aufgerufen wird, verlängert sich seine Verfallszeit um weitere 15 Sekunden. Wenn Sie TimeToLive auf einen relativ hohen Wert setzen, werden viele Einträge möglicherweise nie entfernt. Bei einem Wert von ungefähr 15 Sekunden besteht jedoch die Möglichkeit, dass auch solche Einträge, auf die selten zugegriffen wird, entfernt werden.

Im Folgenden wird ein Beispielszenario beschrieben, in dem die Verwendung des TTL-Typs LAST_ACCESSED_TIME sinnvoll ist. In einer ObjectGrid-Map werden Sitzungsdaten eines Clients gespeichert. Die Sitzungsdaten müssen gelöscht werden, wenn der Client die Sitzungsdaten eine bestimmte Zeit lang nicht verwendet. Die Sitzungsdaten sollen beispielsweise verfallen, wenn der Client länger als 30 Minuten inaktiv ist. In diesem Fall ist der TTL-Typ LAST_ACCESSED_TIME mit einem TimeToLive-Wert von 30 Minuten genau die Einstellung, die für diese Anwendung benötigt wird.

Beispiel

Das folgende Beispiel erstellt eine BackingMap, definiert das Attribut ttlType für den Standard-Evictor ttlType und das Merkmal TimeToLive.

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
bMap.setTtlEvictorType(TTLType.LAST_ACCESSED_TIME);
bMap.setTimeToLive(15);
```

Die meisten Evictor-Einstellungen müssen vor der ObjectGrid-Initialisierung gesetzt werden. Ausführliche Informationen zu den Evictor-Plug-ins finden Sie im Abschnitt „Evictor” auf Seite 93.

Kapitel 8. Änderungen an Peer-JVMs verteilen

Die Objekte LogSequence und LogElement übertragen die Änderungen, die in einer ObjectGrid-Transaktion vorgenommen werden, mit einem Plug-in.

Nähere Informationen zur Verteilung von transaktionsorientierten Änderungen mit Java Message Service (JMS) finden Sie im Abschnitt „Java Message Service für die Verteilung von Transaktionsänderungen" auf Seite 202.

Mit den Objekten kann eine Anwendung Änderungen, die in einer ObjectGrid-Instanz vorgenommen werden, auf einfache Weise mit einem Messaging-Protokoll an Peer-ObjectGrids in fernen Java Virtual Machines (JVM) verteilen und diese Änderungen anschließend in dieser JVM anwenden. Die Klasse LogSequenceTransformer ist für die Aktivierung dieser Unterstützung von entscheidender Bedeutung. Dieser Artikel beschreibt, wie Sie mit einem JMS-Messaging-System einen Listener für die Weitergabe der Nachrichten schreiben.

ObjectGrid unterstützt die Übertragung von LogSequences, die aus dem Festschreiben einer ObjectGrid-Transaktion resultieren, an die Member eines Application-Server-Cluster mit einem von IBM bereitgestellten Plug-in. Diese Funktion ist standardmäßig nicht aktiviert, kann aber konfiguriert werden. Nähere Informationen finden Sie im Abschnitt „Verteilte Transaktionsweitergabe konfigurieren" auf Seite 159. Wenn der Konsument oder Produzent jedoch kein WebSphere Application Server ist, müssen Sie unter Umständen ein externes JMS-Messaging-System verwenden.

1. Plug-in initialisieren. Die ObjectGrid-Instanz ruft die Plug-in-Methode `initialize` (Teil des ObjectGridEventListener-Interface-Vertrags) auf, wenn ObjectGrid gestartet wird. Die Methode `initialize` muss ihre JMS-Ressourcen anfordern, wie z. B. Verbindungen, Sitzungen und Publisher, und den JMS-Listener-Thread starten. Das folgende Beispiel zeigt die Methode initialize:

```
public void initialize(Session session)
{
 mySession = session;
 myGrid = session.getObjectGrid();
 try
 {
  if(mode == null)
  {
   throw new ObjectGridRuntimeException("No mode specified");
  }
  if(userid != null)
  {
   connection = topicConnectionFactory.createTopicConnection(
   userid, password);
  }
  else
   connection = topicConnectionFactory.createTopicConnection();

  // Verbindung starten, um Nachrichten empfangen zu können
  connection.start();

  // Die JMS-Sitzung ist nicht transaktionsorientiert (false).
  jmsSession = connection.createTopicSession(false,
  javax.jms.Session.AUTO_ACKNOWLEDGE);
  if(topic == null)
   if(topicName == null)
   {
```

```
    throw new ObjectGridRuntimeException("Topic not specified");
   }
   else
   {
    topic = jmsSession.createTopic(topicName);
   }
  publisher = jmsSession.createPublisher(topic);
  // start the listener thread.
  listenerRunning = true;
  listenerThread = new Thread(this);
  listenerThread.start();
 }
 catch(Throwable e)
 {
  throw new ObjectGridRuntimeException("Cannot initialize", e);
 }
}
```

Der Code zum Starten des Thread verwendet einen J2SE-Thread (Java 2 Platform, Standard Edition). Wenn Sie einen Server von WebSphere Application Server Version 6.x oder WebSphere Application Server Version 5.x Enterprise ausführen, verwenden Sie zum Starten dieses Dämon-Thread die API für asynchrone Beans. Sie können auch die allgemeinen APIs verwenden. Das folgende Beispiel-Snippet zeigt dieselbe Aktion unter Verwendung eines Work Manager:

```
// Listener-Thread starten
listenerRunning = true;
workManager.startWork(this, true);
```

Das Plug-in muss das Interface Work anstelle des Interface Runnable implementieren. Außerdem müssen Sie eine Methode release hinzufügen, um die Variable listenerRunning auf false zu setzen. Das Plug-in muss mit einer WorkManager-Instanz in einem eigenen Konstruktur oder bei Verwendung eines IoC-Containers (Inversion of Control) durch Injection bereitgestellt werden.

2. Änderungen übertragen. Die folgende Beispielmethode transactionEnd veröffentlicht die lokalen Änderungen, die in einer ObjectGrid-Instanz vorgenommen wurden. Obwohl in diesem Beispiel JMS verwendet wird, können Sie jedes Messaging-Protokoll verwenden, das in der Lage ist, Nachrichten zuverlässig zu veröffentlichen und zu subskribieren.

```
// Diese Methode wird synchronisiert, um sicherzustellen,
// dass die Nachrichten in der Reihenfolge veröffentlicht werden,
// in der die Transaktion festgeschrieben wurde. Wenn die Veröffentlichung
// der Nachrichten parallel erfolgte, könnten die Empfänger die Map-Instanz
// beschädigen, wenn Löschanforderungen vor Einfügeanforderungen usw.
// ankommen.
public synchronized void transactionEnd(String txid,
boolean isWriteThroughEnabled,
boolean committed, Collection changes)
{
 try
 {
  // Durchschreiben und Festschreiben erforderlich
  if(isWriteThroughEnabled && committed)
  {
   // Sequenzen in ein Byte-Array [] schreiben
   ByteArrayOutputStream bos = new ByteArrayOutputStream();
   ObjectOutputStream oos = new ObjectOutputStream(bos);
   if (publishMaps.isEmpty()) {
    // vollständige Sammlung serialisieren
    LogSequenceTransformer.serialize(changes, oos, this, mode);
   }
   else {
    // LogSequences basierend auf dem publishMaps-Inhalt filtern
```

```
    Collection publishChanges = new ArrayList();
    Iterator iter = changes.iterator();
    while (iter.hasNext()) {
     LogSequence ls = (LogSequence) iter.next();
     if (publishMaps.contains(ls.getMapName())) {
     publishChanges.add(ls);
    }
   }
   LogSequenceTransformer.serialize(publishChanges, oos, this,
mode);
  }
  // Objektnachricht für die Änderungen erstellen
  oos.flush();
  ObjectMessage om = jmsSession.createObjectMessage(
  bos.toByteArray());
  // Merkmale definieren
  om.setStringProperty(PROP_TX, txid);
  om.setStringProperty(PROP_GRIDNAME, myGrid.getName());
  // Übertragen
  publisher.publish(om);
  }
 }
 catch(Throwable e)
 {
  throw new ObjectGridRuntimeException("Cannot push changes", e);
 }
}
```

Diese Methode verwendet mehrere Instanzvariablen:

- Variable **jmsSession**: Eine JMS-Sitzung, die für die Veröffentlichung von Nachrichten verwendet wird. Sie wird bei der Initialisierung des Plug-in erstellt.
- Variable **mode**: Der Verteilungsmodus.
- Variable **publishMaps**: Eine Gruppe, die die Namen der Map-Instanzen mit zu veröffentlichenden Änderungen enthält. Wenn die Variable leer ist, werden alle Map-Instanzen veröffentlicht.
- Variable **publisher**: Ein TopicPublisher-Objekt, das während der Initialisierung des Plug-in erstellt wird.

3. Aktualisierungsnachrichten empfangen und anwenden. Im Folgenden wird die Methode run gezeigt. Diese Methode wird so lange in einer Schleife ausgeführt, bis die Anwendung die Schleife stoppt. Jede Iterationsschleife versucht, eine JMS-Nachricht zu empfangen und auf die ObjectGrid-Instanz anzuwenden.

```
private synchronized boolean isListenerRunning()
{
 return listenerRunning;
}
public void run()
{
 try
 {
  System.out.println("Listener starting");
  // Eine JMS-Sitzung für den Empfang der Nachrichten abrufen.
  // Nicht transaktionsorientiert.
  TopicSession myTopicSession;
  myTopicSession = connection.createTopicSession(false,
  javax.jms.Session.AUTO_ACKNOWLEDGE);

  // Einen Subskribenten für das Topic abrufen. true zeigt an,
  // dass keine Nachrichten empfangen werden sollen, die mit
  // Publishern in dieser Verbindung übertragen wurden.
  // Sonst werden eigene Aktualisierungen empfangen.
  TopicSubscriber subscriber = myTopicSession.createSubscriber(topic,
```

```
 null, true);
 System.out.println("Listener started");
 while(isListenerRunning())
 {
  ObjectMessage om = (ObjectMessage)subscriber.receive(2000);
  if(om != null)
  {
   // Das Session-Objekt verwenden, das mit der Methode
   // initialize übergeben wurde...
   // Sehr wichtig: hier kein Durchschreiben verwenden
   mySession.beginNoWriteThrough();
   byte[] raw = (byte[])om.getObject();
   ByteArrayInputStream bis = new ByteArrayInputStream(raw);
   ObjectInputStream ois = new ObjectInputStream(bis);
   // LogSequences erweitern
   Collection collection = LogSequenceTransformer.inflate(ois,
    myGrid);
   Iterator iter = collection.iterator();
   while (iter.hasNext()) {
    // Die Änderungen jeder Map-Instanz entsprechend dem
    // Modus verarbeiten, der für die Serialisierung
    // der LogSequence verwendet wurde
    LogSequence seq = (LogSequence)iter.next();
    mySession.processLogSequence(seq);
   }
   mySession.commit();
  } // Wenn in der Schleife eine
 } // Nachricht empfangen wird,
 // Verbindung stoppen
 connection.close();
}
catch(IOException e)
{
 System.out.println("IO Exception: " + e);
}
catch(JMSException e)
{
 System.out.println("JMS Exception: " + e);
}
catch(ObjectGridException e)
{
 System.out.println("ObjectGrid exception: " + e);
 System.out.println("Caused by: " + e.getCause());
}
catch(Throwable e)
{
 System.out.println("Exception : " + e);
}
System.out.println("Listener stopped");
}
```

Die Klasse LogSequenceTransformer und die APIs ObjectGridEventListener, LogSequence und LogElement unterstützen die Verwendung jedes zuverlässigen Publish/Subscribe-Systems für die Verteilung der Änderung und für das Filtern der zu verteilenden Map-Instanzen. Die Snippets in dieser Task zeigen, wie diese APIs mit JMS verwendet werden, um ein Peer-to-Peer-ObjectGrid aufzubauen, das von mehreren Anwendungen auf verschiedenen Plattformen mit einem gemeinsamen Messaging-Protokoll genutzt werden kann.

Java Message Service für die Verteilung von Transaktionsänderungen

Mit dem Java Message Service (JMS) können Sie Änderungen an verschiedene Schichten oder in Umgebungen auf heterogenen Plattformen verteilen.

Java Message Service (JMS) ist das ideale Protokoll für die Verteilung von Änderungen an verschiedene Schichten oder in Umgebungen auf heterogenen Plattformen. Einige Anwendungen, die ObjectGrid verwenden, können beispielsweise in Gluecode oder Tomcat implementiert sein, während andere in WebSphere Application Server Version 6.0 ausgeführt werden. JMS eignet sich für die Verteilung von Änderungen an ObjectGrid-Peers in diesen unterschiedlichen Umgebungen. Das Messaging-Protokoll des High Availability Manager ist sehr schnell, kann aber nur Änderungen an JVMs verteilen, die derselben Stammgruppe angehören. JMS ist langsamer, lässt aber die gemeinsame Nutzung einer ObjectGrid-Instanz durch größere und komplexere Gruppen von Anwendungsclients zu. JMS eignet sich beispielsweise ideal, wenn ein komplexer Swing-Client und eine Anwendung, die in WebSphere Extended Deployment implementiert ist, die Daten in einer ObjectGrid-Instanz gemeinsam nutzen.

Übersicht

JMS wird für die Verteilung von Transaktionsänderungen mit einem Java-Objekt implementiert, das sich wie ein ObjectGridEventListener-Listener verhält. Dieses Objekt kann den Status mit den folgenden vier Methoden weitergeben:

Invalidate
Jeder bereinigte, aktualisierte oder gelöschte Eintrag wird in allen JVMs (Java Virtual Machines) entfernt, wenn diese die Nachricht empfangen.

Invalidate conditional
Der Eintrag wird nur bereinigt, wenn die lokale Version gleich oder älter ist als die Version des Publisher.

Push Jeder bereinigte, aktualisierte, gelöschte oder eingefügte Eintrag wird in allen Peer-JVMs hinzufügt oder überschrieben, wenn diese die JMS-Nachricht empfangen.

Push conditional
Der Eintrag wird auf der Empfängerseite nur aktualisiert oder hinzugefügt, wenn der lokale Eintrag älter ist als die Version, die veröffentlicht wird.

Auf zu veröffentlichende Änderungen warten

Das Plug-in implementiert das Interface ObjectGridEventListener, um das Ereignis transactionEnd abzufangen. Wenn die ObjectGrid-Instanz diese Methode aufruft, versucht das Plug-in, die LogSequence-Liste für jede Map-Instanz, die von der Transaktion berührt wird, in eine JMS-Nachricht zu konvertieren und anschließend zu veröffentlichen. Das Plug-in kann so konfiguriert werden, dass es Änderungen für alle Map-Instanzen oder nur eine bestimmte Gruppe von Map-Instanzen veröffentlicht. LogSequence-Objekte werden für die Map-Instanzen verarbeitet, in denen die Veröffentlichung aktiviert ist. Die ObjectGrid-Klasse LogSequenceTransformer serialisiert eine gefilterte LogSequence für jede Map-Instanz in einen Datenstrom. Nach der Serialisierung aller LogSequences in Datenströme wird eine JMS-ObjectMessage erstellt und in einem bekannten Topic veröffentlicht.

Auf JMS-Nachrichten warten und diese auf die lokale ObjectGrid-Instanz anwenden

Dasselbe Plug-in startet auch einen Thread, der eine Schleife startet und alle Nachrichten empfängt, die in dem bekannten Topic veröffentlicht werden. Wenn eine Nachricht ankommt, wird der Nachrichteninhalt für die Konvertierung in LogSequence-Objekte an die Klasse LogSequenceTransformer übergeben. Anschließend wird eine Transaktion ohne Durchschreiben gestartet. Jedes LogSequence-Objekt

wird an die Methode Session.processLogSequence übergeben, die die lokalen Map-Instanzen mit den Änderungen aktualisiert. Die Methode processLogSequence unterstützt den Verteilungsmodus. Die Transaktion wird festgeschrieben, und der lokale Cache spiegelt daraufhin die Änderungen wider.

Nähere Informationen zur Verwendung von JMS für die Verteilung von Transaktionsänderungen finden Sie im Abschnitt Kapitel 8, „Änderungen an Peer-JVMs verteilen”, auf Seite 199.

Kapitel 9. Injection-basierte Containerintegration

Sie können das IoC-Framework (Inversion of Control) verwenden, um die ObjectGrid-Instanz vollständig zu konfigurieren. Sie müssen also nicht das XML-Framework für die ObjectGrid-Konfiguration verwenden.

Injection-basierte Container

Injection-basierte Container, auch Inversion of Control (IoC) genannt, sind ein gebräuchliches Muster, das von Anwendungen auf der Clientseite und auf der Serverseite verwendet wird. Es sind verschiedene Open-Source-Implementierungen solcher Container verfügbar. Die neue Spezifikation Enterprise JavaBeans (EJB) Version 3.0 bedient sich auch einiger dieser Konzepte. Die meisten dieser Frameworks sind EJB-Container und für das Erstellen einer Instanz einer bestimmten Bean verantwortlich. Diese Frameworks können eine Bean auch mit einer Reihe von Merkmalen initialisieren und mit getter/setter-Paaren in EJB-Attributen eine Verbindung zu anderen erforderlichen Enterprise JavaBeans herstellen. Die Anwendungsprogrammierschnittstellen (API) von ObjectGrid sind für die Zusammenarbeit mit solchen Containern konzipiert. Mit den Methoden ObjectGridManager.createObjectGrid können Sie diese Container so konfigurieren, dass sie eine ObjectGrid-Instanz starten, damit die Anwendung eine Referenz auf eine funktionierende ObjectGrid-Instanz erhält oder den Container bei Bedarf auffordern kann, eine ObjectGrid-Sitzung bereitzustellen.

Unterstützte Muster

Die folgenden Abschnitte beschreiben, welche Schritte ausgeführt werden müssen, um sicherzustellen, dass die ObjectGrid-APIs ordnungsgemäß von einer Anwendung verwendet wird, die ein IoC-Framework nutzt.

ObjectGridManager zum Erstellen von ObjectGrids verwenden

Die ObjectGrid-APIs sind für die Zusammenarbeit mit IoC-Frameworks geeignet. Das Stamm-Singleton, das von einem solchen Framework verwendet wird, ist das Interface ObjectGridManager, das mehrere Factory-Methoden `createObjectGrid` besitzt, die eine Referenz auf eine benannte ObjectGrid-Instanz zurückgeben. Sie können diese ObjectGrid-Referenz als Singleton im IoC-Framework festlegen, so dass nachfolgende Anforderungen für die Bean dieselbe ObjectGrid-Instanz zurückgeben.

ObjectGrid-Plug-ins

ObjectGrid stellt die folgenden Plug-ins bereit:

- TransactionCallback
- ObjectGridEventListener
- SubjectSource
- MapAuthorization
- SubjectValidation

Diese Plug-ins sind einfache JavaBeans, die jeweils ein Interface implementieren. Sie können mit dem IoC-Framework diese Plug-ins erstellen und diesen die ent-

sprechenden Attribute in der ObjectGrid-Instanz zuweisen. Jedes dieser Plug-ins hat für eine problemlose Integration in das IoC-Framework eine entsprechende set-Methode im Interface ObjectGrid.

Maps definieren

Die Factory-Methode defineMap im Interface ObjectGrid kann für die Definition einer neu benannten Map verwendet werden. Alle von der BackingMap (dem von defineMap zurückgegeben Objekt) benötigten Plug-ins werden mit dem IoC-Framework erstellt und anschließend unter Verwendung des entsprechenden Attributnamens der BackingMap zugewiesen. Da die BackingMap von keinem anderen Objekt referenziert wird, wird sie von den IoC-Frameworks nicht automatisch erstellt. Dieses Problem können Sie umgehen, indem Sie jeder BackingMap ein Pseudoattribut in der Hauptanwendungs-Bean zuweisen.

BackingMap-Plug-ins

Die Plug-ins in der BackingMap verhalten sich auf dieselbe Weise wie die Plug-ins im ObjectGrid. Jedes Plug-in hat eine entsprechende `set`-Methode im Interface BackingMap. Die folgenden BackingMap-Plug-ins sind verfügbar:

- Loader
- ObjectTransformer
- OptimisticCallback
- Evictor
- MapEventListener

Verwendungsmuster

Nachdem das IoC-Framework seine Konfigurationsdatei für die Generierung einer ObjectGrid-Instanz definiert hat, erstellen Sie eine Enterprise-Bean mit dem Namen `gridName_Session` oder einem ähnlichen Namen. Definieren Sie diese Bean als Enterprise JavaBean, die mit dem Aufruf der Methode `getSession` in der ObjectGrid-Singleton-EJB abgerufen wird. Die Anwendung verwendet anschließend das IoC-Framework, um eine Referenz auf ein gridName_Session-Objekt abzurufen, sobald ein Objekt eine neue Sitzung erfordert.

Zusammenfassung

Der Einsatz von ObjectGrid in Umgebungen, die bereits ein IoC-Framework für die Bean-Instanzierung und -Konfiguration verwenden, ist problemlos. Mit dem IoC-Framework können Sie ObjectGrid vollständig konfigurieren und benötigen das XML-Konfigurations-Framework nicht. ObjectGrid integriert sich nahtlos in das vorhandene IoC-Framework.

Kapitel 10. Fehlerbehebung

Dieser Abschnitt beschreibt Szenario für die Behebung von Problemen, die auf einen Anwendungsfehler oder einen Fehler im Anwendungsdesign zurückzuführen sind. Wenn Sie einen Fehler in ObjectGrid vermuten, müssen Sie möglicherweise den Trace für ObjectGrid aktivieren. Nähere Informationen hierzu finden Sie im Abschnitt zum Interface ObjectGridManager.

Sporadisch auftretende und ungeklärte Fehler

Eine Anwendung versucht, ihre Leistung durch den Kopiermodus COPY_ON-_READ, COPY_ON_WRITE oder NO_COPY zu verbessern. Diese Kopiermodi sind im Abschnitt ″CopyMode″ beschrieben. In der Anwendung treten Probleme sporadisch auf, wenn sich das Fehlersymptom ändert und der Fehler unerwartet und ohne Erklärung auftritt. Sporadisch auftretender Fehler treten nicht auf, wenn der Kopiermodus COPY_ON_READ_AND_COMMIT verwendet wird.

Problem

Das Problem kann auf beschädigte Daten in der ObjectGrid-Map in Folge einer Verletzung des Programmiervertrags des verwendeten Kopiermodus zurückzuführen sein. Fehlerhafte können zu unvorhersehbaren Fehlern führen, die sporadisch oder in einer nicht erklärbaren oder unerwarteten Weise auftreten.

Lösung

Die Anwendung muss dem Programmiervertrag für den verwendeten Kopiermodus entsprechen. Für die Kopiermodi COPY_ON_READ und COPY_ON_WRITE bedeutet dies, dass die Anwendung eine Referenz auf ein Wertobjekt außerhalb des Transaktionsbereichs verwendet, aus dem die Wertreferenz abgerufen wurde. Wenn Sie diese Modi verwenden möchten, muss die Anwendung die Referenz auf das Wertobjekt löschen, wenn die Transaktion abgeschlossen ist, und eine neue Referenz auf das Wertobjekt in jeder Transaktion abrufen, die auf das Wertobjekt zugreifen muss. Im Modus NO_COPY darf die Anwendung das Wertobjekt nicht ändern. In diesem Fall muss die Anwendung so geändert werden, dass sie das Wertobjekt nicht ändert, oder die Anwendung muss einen anderen Kopiermodus verwenden. Ausführliche Einzelheiten zur Einstellung des Kopiermodus finden Sie im Abschnitt ″CopyMode″.

Allgemeines Verfahren bei der Ausnahmebehandlung

Die eigentliche Ursache für ein Throwable-Objekt zu kennen, ist für die Isolierung einer Fehlerquelle sehr hilfreich. Das folgende Beispiel zeigt eine Dienstprogrammmethode, die die Ausnahmebehandlungsroutine verwenden kann, um die eigentliche Ursache für ein Throwable-Objekt zu ermitteln.

Beispiel

```
static public Throwable findRootCause( Throwable t )
{
     // Mit dem aufgetretenen Throwable-Objekt als eigentliche Ursache
     // (root) beginnen.
     Throwable root = t;
```

```
        // Der Ursachenkette (cause) folgen, bis das letzte Throwable-Objekt
        // in der Kette gefunden ist.
        Throwable cause = root.getCause();
        while ( cause != null )
        {
           root = cause;
           cause = root.getCause();
        }

        // Letztes Throwable-Objekt in der Kette als eigentliche Ursache (root) zurückgeben
        return root;
    }
```

Spezielle Verfahren für die Ausnahmebehandlung

Mehrfacheinfügungen

Dieses Problem tritt in der Regel nur in einer Umgebung mit verteilter Transaktionsweitergabe und auch dann nur selten auf.

Nachricht

```
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl < processLogSequence Exit
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl > commit for:
TX:08FE0C67-0105-4000-E000-1540090A5759 Entry
[7/11/05 22:02:11:303 CDT] 00000032 SessionImpl > rollbackPMapChanges for:
TX:08FE0C67-0105-4000-E000-1540090A5759
as result of Throwable: com.ibm.websphere.objectgrid.plugins.
CacheEntryException:
Duplicate key on an insert!
Entry com.ibm.websphere.objectgrid.plugins.CacheEntryException:
Duplicate key on an insert!
at com.ibm.ws.objectgrid.map.BaseMap.applyPMap(BaseMap.java:528)
at com.ibm.ws.objectgrid.SessionImpl.commit(SessionImpl.java:405)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.commitPropagatedLogSequence
(TranPropWorkerThread.java:553)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.processCommitRequest
(TranPropWorkerThread.java:449)
at com.ibm.ws.objectgrid.plugins.TranPropWorkerThread.run
(TranPropWorkerThread.java:200)
at java.lang.Thread.run(Thread.java:568)
```

Problem

Wenn die gefilterte Protokollsequenz von einer JVM an eine andere weitergeben wird, wird die fremde Protokollsequenz in der zweiten JVM verarbeitet. Der Eintrag für diesen Schlüssel kann bereits vorhanden oder die beiden LogSequence-Operationscodes können verschieden sein. Dieses Problem tritt gelegentlich auf.

Auswirkung und Lösung

Wenn dieses Problem auftritt, wird der Eintrag in einer anderen JVM nicht aktualisiert. Dies kann zu Inkonsistenzen in ObjectGrid führen. Dieses Problem kann jedoch umgangen werden. Zusätzlich zu den Objektoperationen insert, update und remove, können Sie für den Objektabruf das Partitionierungs-Feature (WPF) verwenden. Nähere Informationen zu diesem Verfahren finden Sie im Abschnitt zur Integration von WPF und ObjectGrid.

OptimisticCollisionException

Sie können eine Ausnahme des Typs OptimisticCollisionException direkt oder in Verbindung mit einer Ausnahme des Typs ObjectGridException empfangen. Der folgende Beispielcode zeigt, wie die Ausnahme abgefangen und anschließend die zugehörige Nachricht angezeigt wird.

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

Ausnahmeursache

Eine Ausnahme vom Typ OptimisticCollisionException wird erstellt, wenn zwei Clients versuchen, denselben Map-Eintrag fast gleichzeitig zu aktualisieren. Die Sitzung eines Clients wird festgeschrieben und aktualisiert damit den Map-Eintrag. Der andere Client hat die Daten jedoch bereits vor der Festschreibung gelesen und enthält veraltete oder ungültige Daten. Wenn der andere Client versucht, die ungültigen Daten festzuschreiben, wird diese Ausnahme erstellt.

Schlüssel abrufen, der die Ausnahme ausgelöst hat

Bei der Behandlung einer solchen Ausnahme kann es hilfreich sein, den Schlüssel für den Eintrag abzurufen, der die Ausnahme ausgelöst hat. Die Ausnahme OptimisticCollisionException bietet den Vorteil, dass sie eine integrierte Methode getKey enthält, die das Objekt für diesen Schlüssel zurückgibt. Das folgende Beispiel zeigt, wie der Schlüssel beim Abfangen der Ausnahme OptimisticCollisionException abgerufen und ausgegeben wird:

```
try {
...
} catch (OptimisticCollisionException oce) {
 System.out.println(oce.getKey());
}
```

Eine Ausnahme des Typs OptimisticCollisionException kann die Ursache für eine Ausnahme des Typs ObjectGridException sein. In einem solchen Fall können Sie den folgenden Code verwenden, um den Ausnahmetyp zu bestimmen und den Schlüssel auszugeben. Der folgende Code verwendet die Dienstprogrammmethode findRootCause, die im Abschnitt "Allgemeines Verfahren zur Ausnahmebehandlung" beschrieben ist.

```
try {
...
}
catch (ObjectGridException oe) {
 Throwable Root = findRootCause( oe );
 if (Root instanceof OptimisticCollisionException) {
  OptimisticCollisionException oce = (OptimisticCollisionException)Root;
  System.out.println(oce.getKey());
 }
}
```

Ausnahme LockTimeoutException

Nachricht

Sie können eine Ausnahme des Typs LockTimeoutException direkt oder beim Abfangen einer Ausnahme des Typs ObjectGridException abfangen. Das folgende Code-Snippet zeigt, wie die Ausnahme abgefangen und die Nachricht angezeigt wird.

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

Das Ergebnis ist wie folgt:

```
com.ibm.websphere.objectgrid.plugins.LockTimeoutException: %Message
```

%Message steht für die Zeichenfolge, die als Parameter übergeben wird, wenn die Ausnahme eintritt und die Ausnahmemerkmale und -methoden zum Anzeigen einer genauen Fehlernachricht verwendet werden. Sie beschreibt in der Regel den Typ der anforderten Sperre und den Map-Eintrag, für den die Transaktion ausgeführt wurde.

Ausnahmeursache

Wenn eine Transaktion oder ein Client eine Sperre für einen bestimmten Map-Eintrag anfordert, muss sie bzw. er häufig warten, bis der aktuelle Client die Sperre freigibt. Falls die Sperrenanforderung für einen längeren Zeitraum blockiert wird und keine Sperre erteilt wird, wird eine Ausnahme des Typs `LockTimeoutException` erstellt. Auf diese Weise wird ein gegenseitiges Sperren, wie es im folgenden Abschnitt näher erläutert wird, verhindert. Diese Ausnahme wird eher bei einer pessimistischen Sperrstrategie angezeigt, weil dort die Sperre erst freigegeben wird, wenn die Transaktion festgeschrieben wird.

Weitere Einzelheiten zur Sperrenanforderung und Ausnahme abrufen

Die Ausnahme `LockTimeoutException` hat eine integrierte Methode mit dem Namen getLockRequestQueueDetails, die eine Zeichenfolge zurückgibt, die eine ausführliche Beschreibung Situation hält, die zur Ausnahme geführt hat. Das folgende Beispiel zeigt Code, der die Ausnahme abfängt und eine Fehlernachricht anzeigt:

```
try {
 ...
}
catch (LockTimeoutException lte) {
 System.out.println(lte.getLockRequestQueueDetails());
}
```

The output result is:

```
lock request queue
->[TX:163C269E-0105-4000-E0D7-5B3B090A571D, state =
 Granted 5348 milli-seconds ago, mode = U]
->[TX:163C2734-0105-4000-E024-5B3B090A571D, state =
```

```
 Waiting for 5348 milli-seconds, mode = U]
->[TX:163C328C-0105-4000-E114-5B3B090A571D, state =
 Waiting for 1402 milli-seconds, mode = U]
```

Wenn Sie die Ausnahme in einem catch-Block für `ObjectGridException` empfangen, bestimmt der folgende Code die Ausnahme und gibt Einzelheiten zur Warteschlange aus. Der Code verwendet die Dienstprogrammmethode findRootCause, die im Abschnitt ″Allgemeines Verfahren zur Ausnahmebehandlung″ beschrieben ist.

```
try {
...
}
catch (ObjectGridException oe) {
 Throwable Root = findRootCause( oe );
 if (Root instanceof LockTimeoutException) {
  LockTimeoutException lte = (LockTimeoutException)Root;
  System.out.println(lte.getLockRequestQueueDetails());
 }
}
```

Mögliche Lösung

Die Ausnahme `LockTimeoutException` soll Situationen mit gegenseitigem Sperren in Ihrer Anwendung verhindern. Eine Ausnahme dieses Typs wird ausgelöst, wenn die Anwendung eine festgelegte Wartezeit überschreitet. Die Wartezeit kann jedoch mit der Methode `setLockTimeout(int)` für die BackingMap definiert werden. Wenn Sie die Methode setLockTimeout verwenden, wird die Ausnahme `LockTimeoutException` nicht ausgelöst. Sollte keine Situation mit gegenseitigem Sperren in Ihrer Anwendung vorliegen, können Sie das Auslösen der Ausnahme LockTimeoutException durch Anpassen des Zeitlimits für Sperre verhindern.

Der folgende Code zeigt, wie ein ObjectGrid erstellt, eine Map definiert und der LockTimeout-Wert auf 30 Sekunden gesetzt wird:

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("MapName");
// Damit wird die Zeit festgelegt, die eine Sperrenanforderung wartet,
// bevor eine Ausnahme ausgelöst wird.
bMap.setLockTimeout(30);
```

Der vorherige Code kann verwendet werden, um die ObjectGrid- und Map-Merkmale fest zu codieren. Wenn Sie ObjectGrid mit einer XML-Datei konfigurieren, können Sie das Merkmal LockTimeout im Tag backingMap definieren. Das folgende Beispiel zeigt ein Tag backingMap, das einen LockTimeout-Wert von 30 Sekunden für die Map festlegt:

```
<backingMap name="MapName" lockStrategy="PESSIMISTIC" lockTimeout="30">
```

LockDeadlockException

Nachricht

Sie können eine Ausnahme des Typs LockDeadLockException direkt oder beim Abfangen einer Ausnahme des Typs ObjectGridException abfangen. Das folgende Codebeispiel zeigt, wie die Ausnahme abgefangen und anschließend eine Nachricht angezeigt wird.

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

Das Ergebnis ist wie folgt:

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: %Message
```

%Message steht für die Zeichenfolge, die als Parameter übergeben wird, wenn die Ausnahme erstellt und ausgelöst wird.

Ausnahmeursache

Ein gegenseitiges Sperren tritt am häufigsten auf, wenn eine pessimistische Sperrstrategie verwendet wird und zwei gesonderte Clients jeweils eine gemeinsame Sperre für ein bestimmtes Objekt besitzen. Daraufhin versuchen beide Clients, ihre Sperre in eine exklusive Sperre für das Objekt umzustufen. Die folgende Abbildung zeigt eine solche Situation, in der sich die Transaktionen gegenseitig blockieren und daraufhin die Ausnahme ausgelöst wird.

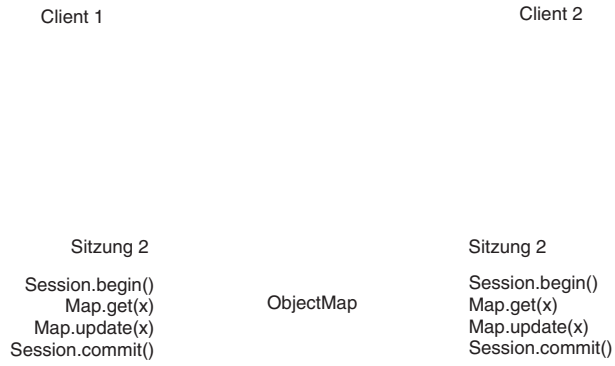


Abbildung 7. Beispiel für eine Situation mit potenziellem gegenseitigen Sperren

Dies ist eine abstrakte Sicht der Vorgänge in Ihrem Programm, wenn die Ausnahme eintritt. In einer Anwendung, in der viele Threads dieselbe ObjectMap-Instanz aktualisieren, ist das Auftreten einer solchen Situation möglich. Im Folgenden wird schrittweise beschrieben, was geschieht, wenn zwei Clients die Transaktionscodeblöcke ausführen, wie in der Abbildung gezeigt.

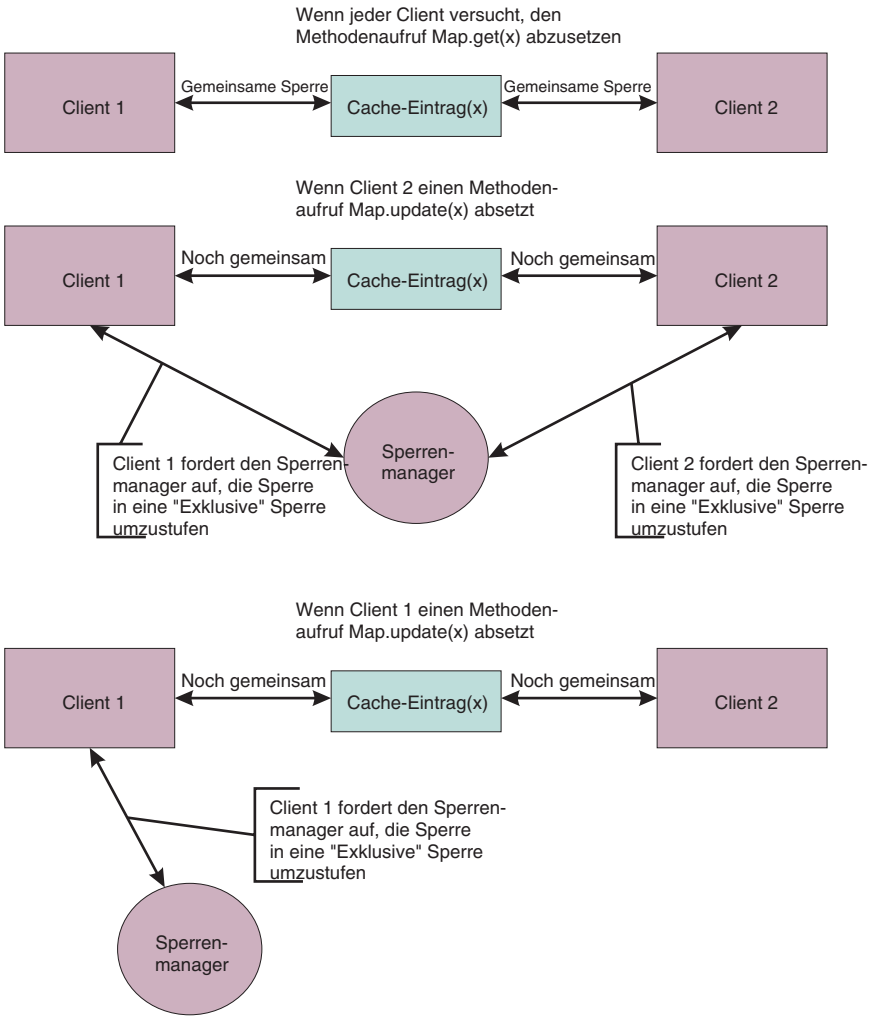


Abbildung 8. Situation mit gegenseitigem Sperren

Wenn beide Clients wie gezeigt versuchen, ihre Sperre in eine exklusive Sperre umzustufen und weiterhin die gemeinsamen Sperren halten, ist es unmöglich, dass einer der beiden Clients eine exklusive Sperre erhält. Beide warten darauf, dass der jeweils andere seine gemeinsame Sperre freigibt. Dies führt zu einer Ausnahme des Typs `LockDeadlockException`.

Mögliche Lösungen

Diese Ausnahme kann gelegentlich erwünscht sein. Wenn viele Threads vorhanden sind, die alle Transaktionen für eine bestimmte Map ausführen, kann die zuvor beschriebene Situation (Abbildung 1) eintreten. Diese Ausnahme wird ausgelöst, damit ihr Programm nicht blockiert. Indem Sie diese Ausnahme abfangen, können Sie sich selbst eine Benachrichtigung senden und gegebenenfalls dem catch-Block Code hinzufügen, um Einzelheiten zur Fehlerursache zu erhalten. Da diese Ausnahme nur bei der Verwendung einer pessimistischen Sperrstrategie angezeigt wird, ist die einfachste Lösung, eine optimistische Sperrstrategie zu verwenden. Sollte die pessimistische Strategie erforderlich sein, können Sie jedoch anstelle der Methode get die Methode getForUpdate verwenden. Damit verhindern Sie, dass Sie Ausnahmen für die zuvor beschriebene Situation erhalten.

Diagnose von XML-Konfigurationsproblemen

Auf eine nicht vorhandene Plug-in-Sammlung verweisen

Wenn Sie BackingMap-Plug-ins mit XML-Dateien definieren, muss das Attribut pluginCollectionRef des Elements backingMap auf ein Element backingMapPlugin-Collection verweisen. Das Attribut pluginCollectionRef muss mit der ID eines der backingMapPluginCollection-Elemente übereinstimmen.

Nachricht

Wenn das Attribut pluginCollectionRef keiner ID eines backingMapPluginConfiguration-Elements entspricht, wird im Protokoll eine Nachricht ähnlich der folgenden aufgezeichnet:

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E:
This is an English only Error message:
Invalid XML file. Line: 14; URI: null;
Message: Key 'pluginCollectionRef' with
value 'bookPlugins' not found for identity
constraint of element 'objectGridConfig'..
```

Die folgende Nachricht ist ein Auszug aus dem Protokoll mit aktiviertem Trace:

```
[7/14/05 14:02:01:971 CDT] 686c060e XmlErrorHandl E CWOBJ9002E: This is an
English only Error message:
Invalid XML file. Line: 14; URI: null; Message: Key
'pluginCollectionRef' with
value 'bookPlugins' not found for identity constraint
of element 'objectGridConfig'..
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R com.ibm.websphere.objectgrid.
ObjectGridException:
Invalid XML file: etc/test/document/bookstore.xml
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R at
com.ibm.ws.objectgrid.config.XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/14/05 14:02:01:991 CDT] 686c060e SystemErr R at
com.ibm.websphere.objectgrid.ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R Caused by: org.xml.sax.
SAXParseException: Key 'pluginCollectionRef' with value 'bookPlugins'
not found for identity
constraint of element 'objectGridConfig'.
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.error(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:001 CDT] 686c060e SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:011 CDT] 686c060e SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/14/05 14:02:02:011 CDT] 686c060e SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

Problem

Die folgende XML-Datei wurde verwendet, um diesen Fehler zu produzieren. Beachten Sie, dass das Attribut für die BackingMap-Instanz pluginCollectionRef auf bookPlugins gesetzt ist und das einzige Element backingMapPluginCollection die ID collection1 hat:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://ibm.com/ws/objectgrid/config
   ../objectGrid.xsd"  xmlns="http://ibm.com/ws/objectgrid/config">
   <objectGrids>
      <objectGrid name="bookstore">
         <backingMap name="book" pluginCollectionRef="bookPlugin" />
      </objectGrid>
   </objectGrids>
   <backingMapPluginCollections>
    <backingMapPluginCollection id="collection1">
     <bean id="Evictor"
       className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
     </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Lösung

Zur Behebung dieses Problems stellen Sie sicher, dass der Wert jedes Elements pluginCollectionRef mit der ID eines der backingMapPluginCollection-Elemente übereinstimmt. In diesem Beispiel können Sie den Fehler beheben, indem Sie einfach den Namen des Elements pluginCollectionRef in collection1 ändern. Weitere Möglichkeiten sind, die ID des vorhanden Elements backingMapPluginCollection so zu ändern, dass sie dem Element pluginCollectionRef entspricht, oder ein weiteres Element backingMapPluginCollection mit einer ID hinzuzufügen, die dem Element pluginCollectionRef entspricht.

Ein erforderliches Attribut fehlt

Viele Elemente in der XML-Datei haben mehrere optionale Attribute. Optionale Attribute können in die Datei eingefügt und aus dieser entfernt werden und haben keine Auswirkung auf die Validierung der XML-Datei. Es gibt jedoch diverse erforderliche Attribute. Wenn eines dieser erforderlichen Attribute nicht vorhanden ist, wenn das zugehörige Element verwendet wird, schlägt die XML-Validierung fehl.

Nachricht

Wenn ein erforderliches Attribut fehlt, wird eine Nachricht ähnlich der folgenden im Protokoll aufgezeichnet. In diesem Beispiel fehlt das Attribut type im Element property.

```
[7/15/05 13:41:41:267 CDT] 6873dcac XmlErrorHandl E CWOBJ9002E:
This is an English only
Error message: Invalid XML file.
Line: 12; URI: null; Message: cvc-complex-type.4:
Attribute 'type' must appear on element 'property'..
```

Die folgende Nachricht ist ein Auszug aus dem Protokoll mit aktiviertem Trace.

```
[7/15/05 14:08:48:506 CDT] 6873dff9 XmlErrorHandl E CWOBJ9002E: This is an English
only Error message: Invalid XML file.
Line: 12; URI: null; Message: cvc-complex-type.4: Attribute 'type'
must appear on element 'property'..
[7/15/05 14:08:48:526 CDT] 6873dff9 SystemErr R com.ibm.websphere.objectgrid.
ObjectGridException: Invalid XML file: etc/test/document/bookstore.xml
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R at com.ibm.ws.objectgrid.config.
XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R at com.ibm.websphere.objectgrid.
ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
[7/15/05 14:08:48:536 CDT] 6873dff9 SystemErr R Caused by: org.xml.sax.
SAXParseException: cvc-complex-type.4:
```

```
Attribute 'type' must appear on element 'property'.
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.util.
ErrorHandlerWrapper.error(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.
XMLErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/15/05 14:08:48:546 CDT] 6873dff9 SystemErr R at org.apache.xerces.impl.xs.
XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

Problem

Die folgende XML-Beispieldatei wurde verwendet, um den vorherigen Fehler zu produzieren. Beachten Sie, dass das Merkmal im Evictor-Abschnitt nur zwei der drei erforderlichen Attribute enthält. Die Attribute name und value sind vorhanden, aber das Attribut type fehlt. Das Fehlen des Attributs bewirkt, dass die XML-Validierung fehlschlägt.

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
   xmlns="http://ibm.com/ws/objectgrid/config">
   <objectGrids>
      <objectGrid name="bookstore">
         <backingMap name="book" pluginCollectionRef="collection1" />
      </objectGrid>
   </objectGrids>
   <backingMapPluginCollections>
    <backingMapPluginCollection id="collection1">
     <bean id="Evictor"
       className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
       <property name="maxSize" value="89" />
     </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

Lösung

Zur Behebung dieses Problems fügen Sie der XML-Datei das erforderliche Attribut hinzu. In der oben gezeigten XML-Beispieldatei müssten Sie das Attribut type hinzufügen und diesem den Wert integer zuweisen.

Ein erforderliches Element fehlt

Es gibt verschiedene XML-Elemente, die für ein Schema erforderlich sind. Wenn diese Elemente nicht vorhanden sind, schlägt die XML-Validierung fehl.

Nachricht

Wenn ein erforderliches Element fehlt, wird eine Nachricht ähnlich der folgenden im Protokoll aufgezeichnet. In diesem Fall fehlt das Element objectGrid.

```
[7/15/05 14:54:23:729 CDT] 6874d511 XmlErrorHandl E CWOBJ9002E:
This is an English only Error message: Invalid XML file.
Line: 5; URI: null; Message: cvc-complex-type.2.4.b: The content of
element 'objectGrids' is not complete.
One of '{"http://ibm.com/ws/objectgrid/config":objectGrid}' is expected..
```

Aktivieren Sie den Trace, um zusätzliche Informationen zu diesem Fehler zu erhalten. Der Abschnitt ″ObjectGridManager″ beschreibt, wie Sie den Trace aktivieren.

Problem

Die folgende XML-Beispieldatei wurde verwendet, um diesen Fehler zu produzieren. Beachten Sie, dass das Element objectGrids keine untergeordneten objectGrid-Elemente hat. Gemäß XML-Schema muss mindestens ein Element objectGrid die objectGrids-Tags eingeschlossen sein. Da dieses Element hier fehlt, schlägt die XML-Validierung fehl.

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

Lösung

Zur Behebung dieses Problems müssen Sie sicherstellen, dass die erforderlichen Elemente in der XML-Datei enthalten sind. Für das vorherige Beispiel müssten Sie mindestens ein Element objectGrid in die objectGrids-Tags einfügen. Sobald die erforderlichen Elemente vorhanden sind, können Sie die XML-Datei problemlos validieren.

Die folgende gültige XML-Datei enthält die erforderlichen Elemente.

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

XML-Wert des Attributs ist nicht gültig

Nachricht

Einigen Attributen in der XML-Datei dürfen nur bestimmte Werte zugeordnet werden. Die gültigen Werte für diese Attribute sind nach Schema aufgelistet. Die Attribute sind im Folgenden aufgeführt:

- `authorizationMechanism` - Attribut im Element objectGrid
- `copyMode` - Attribut im Element `backingMap`
- `lockStrategy` - Attribut im Element `backingMap`
- `ttlEvictorType` - Attribut im Element `backingMap`
- `type` - Attribut im Element `property`

Wenn einem dieser Attribute ein ungültiger Wert zugeordnet ist, schlägt die XML-Validierung fehl.

Im Protokoll wird die folgende Nachricht aufgezeichnet, wenn ein Attribut auf einen anderen Wert als die aufgelisteten gesetzt wird:

```
[7/19/05 16:45:40:992 CDT] 6870e51b XmlErrorHandl E CWOBJ9002E: This is an English
only Error message: Invalid XML file. Line: 6; URI: null; Message:
cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid with
respect to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ,
COPY_ON_WRITE, NO_COPY]'. It must be a value from the enumeration..
```

The following excerpt is from the log with trace enabled.

```
[7/19/05 16:45:40:992 CDT] 6870e51b XmlErrorHandl E CWOBJ9002E: This is an
  English only Error message: Invalid XML file. Line: 6; URI: null; Message:
     cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid with
     respect to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ,
     COPY_ON_WRITE, NO_COPY]'. It must be a value from the enumeration..
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R com.ibm.websphere.objectgrid
     .ObjectGridException: Invalid XML file: etc/test/document/backingMapAttrBad.xml
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R at com.ibm.ws.objectgrid.config
     .XmlConfigBuilder.<init>(XmlConfigBuilder.java:160)
[7/19/05 16:45:41:022 CDT] 6870e51b SystemErr R at com.ibm.websphere.objectgrid
     .ProcessConfigXML$2.run(ProcessConfigXML.java:99)...
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R Caused by:
  org.xml.sax.SAXParseException:
     cvc-enumeration-valid: Value 'INVALID_COPY_MODE' is not facet-valid
  with respect
     to enumeration '[COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE,
  NO_COPY]'.
     It must be a value from the enumeration.
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.util
     .ErrorHandlerWrapper.createSAXParseException(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.util
     .ErrorHandlerWrapper.error(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
     .XMLErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
     .XMLErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
     .xs.XMLSchemaValidator$XSIErrorReporter.reportError(Unknown Source)
[7/19/05 16:45:41:032 CDT] 6870e51b SystemErr R at org.apache.xerces.impl
     .xs.XMLSchemaValidator.reportSchemaError(Unknown Source)
...
```

Problem

Einem Attribut wurde ein Wert zugeordnet, der außerhalb des gültigen Wertebereichs liegt. In diesem Fall handelt es sich um das Attribut copyMode, für das keiner der aufgelisteten Werte definiert wurde. Das Attribut wurde auf INVALID_COPY_MODE gesetzt. Die folgende XML-Datei wurde verwendet, um diesen Fehler zu produzieren:

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

Lösung

In diesem Beispiel hat copyMode einen ungültigen Wert. Setzen Sie das Attribut auf einen der folgenden gültigen Werte: COPY_ON_READ_AND_COMMIT, COPY_ON_READ, COPY_ON_WRITE oder NO_COPY.

XML-Dateien ohne Unterstützung einer Implementierung validieren

IBM Software Development Kit (SDK) Version 1.4.2 enthält eine Implementierung diverser JAXP-Funktionen, mit der Sie XML-Dateien auf der Basis eines Schemas validieren können.

Nachricht

Bei der Verwendung eines SDK, das diese Implementierung nicht enthält, kann die XML-Validierung fehlschlagen. Wenn Sie XML-Dateien mit einem SDK validieren möchten, die diese Implementierung nicht enthält, laden Sie Xerces herunter und fügen Sie die JAR-Dateien dieses Produkts in den Klassenpfad ein.

Wenn Sie versuchen, XML-Dateien mit einem SDK zu validieren, das die erforderliche Implementierung nicht enthält, wird der folgende Fehler im Protokoll aufgezeichnet.

```
[7/19/05 10:50:45:066 CDT] 15c7850 XmlConfigBuil d XML validation is enabled
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R com.ibm.websphere
     .objectgrid[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at
   com.ibm.ws.objectgrid
     .ObjectGridManagerImpl.getObjectGridConfigurations
   (ObjectGridManagerImpl.java:182)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid
     .ObjectGridManagerImpl.createObjectGrid(ObjectGridManagerImpl.java:309)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid.test.
     config.DocTest.main(DocTest.java:128)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R Caused by: java.lang
     .IllegalArgumentException: No attributes are implemented
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at org.apache.crimson.jaxp.
     DocumentBuilderFactoryImpl.setAttribute(DocumentBuilderFactoryImpl.java:93)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.ws.objectgrid.config
     .XmlConfigBuilder.<init>(XmlConfigBuilder.java:133)
[7/19/05 10:50:45:086 CDT] 15c7850 SystemErr R at com.ibm.websphere.objectgrid
     .ProcessConfigXML$2.run(ProcessConfigXML.java:99)
...
```

Problem

Das verwendete SDK enthält keine Implementierung der JAXP-Funktionen, die für die Validierung von XML-Dateien auf der Basis eines Schemas erforderlich sind.

Lösung

Wenn Sie Apache Xerces herunterladen und die zugehörigen JAR-Dateien in den Klassenpfad einfügen, können Sie die XML-Dateien problemlos validieren.

ObjectGrid-Nachrichten

Diese Referenzinformationen enthalten zusätzliche Informationen zu den Nachrichten, die während der Verwendung von ObjectGrid angezeigt werden können. Nachrichten haben einen Nachrichtenschlüssel, und zu jeder Nachricht sind eine Erläuterung und eine empfohlene Benutzeraktion verfügbar. Der Typ der jeweiligen Nachricht ist am letzten Zeichen des Nachrichtenschlüssels erkennbar: I steht für Information, W für Warnung und E für Fehler. In der Erläuterung zur Nachricht wird beschrieben, warum die Nachricht ausgegeben wurde. In der Benutzeraktion zur Nachricht wird beschrieben, welche Maßnahmen ergriffen werden können, um auf eine Warnung oder eine Fehlernachricht zu reagieren.

CWOBJ0001E: Die Methode {0} wurde nach Abschluss der Initialisierung aufgerufen.
Erläuterung: Nach Abschluss der Initialisierung können bestimmte Methoden nicht mehr aufgerufen werden.
Benutzeraktion: Strukturieren Sie Ihren Code so um, dass die Konfiguration abgeschlossen ist, bevor die Laufzeitumgebung initialisiert wird.

CWOBJ0002W: Die Komponente ObjectGrid ignoriert die unerwartete Ausnahme: {0}
Erläuterung: CMSG0001
Benutzeraktion: CMSG0002

CWOBJ0004E: Der Wert {0} ist für {2} nicht gültig.
Erläuterung: Es wurde ein ungültiger Wert für die Variable angegeben.
Benutzeraktion: Definieren Sie einen gültigen Wert. Schlagen Sie im Dokument zur Komponente ObjectGrid die gültigen Werte für die Variablen und Merkmale nach.

CWOBJ0005W: Der Thread löst eine Ausnahme des Typs InterruptedException aus: {0}
Erläuterung: Es wird eine Ausnahme des Typs InterruptedException ausgelöst.
Benutzeraktion: Lesen Sie die Ausnahmenachricht und stellen Sie fest, ob es sich um eine erwartete Störung handelt.

CWOBJ0006W: Ausnahme abgefangen: {0}
Erläuterung: Zur Laufzeit wurde eine Ausnahme ausgelöst.
Benutzeraktion: Lesen Sie die Ausnahmenachricht und stellen Sie fest, ob es sich um eine erwartete Ausnahme handelt.

CWOBJ0010E: Der Nachrichtenschlüssel {0} fehlt.
Erläuterung: Im Nachrichtenressourcenpaket fehlt ein Nachrichtenschlüssel.
Benutzeraktion: CMSG0002

CWOBJ0011: Das Feld {0} in der Klasse {1} konnte nicht entserialisiert werden. Es wird der Standardwert verwendet.
Erläuterung: Während der Entserialisierung eines Objekts wurde ein erwartetes Feld nicht gefunden. Wahrscheinlich wird das Objekt gerade von einer anderen Version der Klasse entserialisiert, die das Objekt serialisiert hat.
Benutzeraktion: Diese Warnung weist auf ein potenzielles Problem hin, aber es ist keine Benutzeraktion erforderlich, sofern keine weiteren Fehler auftreten.

CWOBJ0012E: Der LogElement-Typencode {0} ({1}) wird für diese Operation nicht erkannt.
Erläuterung: Interner Fehler in der Laufzeitumgebung von ObjectGrid.
Benutzeraktion: CMSG0002

CWOBJ0013E: Es wurde eine Ausnahme beim Löschen von Einträgen aus dem Cache abgefangen: {0}
Erläuterung: Beim Löschen von Einträgen aus dem Cache ist ein Fehler aufgetreten.
Benutzeraktion: Lesen Sie die Ausnahmenachricht und stellen Sie fest, ob es sich um eine erwartete Ausnahme handelt.

CWOBJ0014E: Die Laufzeitumgebung von ObjectGrid hat den Versuch einer Transaktionsverschachtelung erkannt.
Erläuterung: Die Verschachtelung von Transaktionen ist nicht zulässig.
Benutzeraktion: Ändern Sie den Code so, dass keine Transaktionen verschachtelt werden.

CWOBJ0015E: Ausnahme beim Verarbeiten einer Transaktion abgefangen: {0}
Erläuterung: Beim Versuch, eine Transaktion zu verarbeiten, ist ein Fehler aufgetreten.
Benutzeraktion: Lesen Sie die Ausnahmenachricht und stellen Sie fest, ob es sich um eine erwartete Ausnahme handelt.

CWOBJ0016E: Es wurde keine aktive Transaktion für die aktuelle Operation gefunden.
Erläuterung: Für die Durchführung dieser Operation ist eine aktive Transaktion erforderlich.
Benutzeraktion: Ändern Sie den Code so, dass vor der Durchführung dieser Operation eine Transaktion gestartet wird.

CWOBJ0017E: Bei der Verarbeitung der ObjectMap-Operation wurde festgestellt, dass ein Schlüssel doppelt vorhanden ist: {0}
Erläuterung: Der Schlüssel für den Eintrag ist bereits im Cache vorhanden.
Benutzeraktion: Ändern Sie den Code so, dass ein Schlüssel stets nur einmal eingefügt wird.

CWOBJ0018E: Bei der Verarbeitung der ObjectMap-Operation wurde der Schlüssel nicht gefunden: {0}
Erläuterung: Der Schlüssel für den Eintrag ist nicht im Cache enthalten.
Benutzeraktion: Ändern Sie den Code so, dass vor der Durchführung dieser Operation sichergestellt wird, dass der Eintrag vorhanden ist.

CWOBJ0019W: Es wurden keine Daten in dem für {0} reservierten Cache-Eintragsbereich gefunden, die für den Cache-Namen {1} verwendet werden können.
Erläuterung: Interner Fehler in der Laufzeitumgebung von ObjectGrid.
Benutzeraktion: CMSG0002

CWOBJ0020E: Der Cache-Eintrag ist nicht im Sicherungs-Cache {0} vorhanden.
Erläuterung: Interner Fehler in der Laufzeitumgebung von ObjectGrid.
Benutzeraktion: CMSG0002

CWOBJ0021E: Bei der Entserialisierung der LogSequence für ObjectGrid {0} und ObjectMap {1} wurde keine verwendbare ObjectTransformer-Instanz gefunden.
Erläuterung: Die Konfiguration der Empfangsseite einer LogSequence bietet keine Unterstützung für die erforderliche ObjectTransformer-Instanz.
Benutzeraktion: Überprüfen Sie die Konfiguration der ObjectGrid-Instanzen für die Sender- und Empfangsseite der LogSequence.

CWOBJ0022E: Der Aufrufende ist nicht der Mutex-Eigner: {0}
Erläuterung: Interner Fehler in der Laufzeitumgebung von ObjectGrid.
Benutzeraktion: CMSG0002

CWOBJ0023E: CopyMode ({0}) wird für diese Operation nicht erkannt.
Erläuterung: Interner Fehler in der Laufzeitumgebung von ObjectGrid.
Benutzeraktion: CMSG0002

CWOBJ0024E: Das Feld {0} in der Klasse {1} konnte nicht entserialisiert werden. Die Entserialisierung ist fehlgeschlagen.
Erläuterung: Bei der Entserialisierung eines Objekts wurde ein erforderliches Feld nicht gefunden. Wahrscheinlich ist ein Fehler in der ObjectGrid-Laufzeitumgebung aufgetreten.
Benutzeraktion: CMSG0002

CWOBJ0025E: Die Serialisierung der LogSequence ist fehlgeschlagen. Die Anzahl der serialisierten LogElements ({0}) stimmt nicht mit der Anzahl der gelesenen LogElements ({1}) überein.
Erläuterung: Interner Fehler in der Laufzeitumgebung von ObjectGrid.
Benutzeraktion: CMSG0002

CWOBJ1207W: Das Merkmal {0} im Plug-in {1} wird nicht unterstützt.
Erläuterung: Es werden nur primitive Java-Typen und ihre java.lang-Entsprechungen unterstützt. java.lang.String wird auch unterstützt.
Benutzeraktion: Überprüfen Sie den Merkmaltyp und verwenden Sie einen der unterstützten Typen.

CWOBJ1208W: Das Plug-in {0} wird nicht unterstützt.
Erläuterung: Der Typ des Plug-in wird nicht unterstützt.
Benutzeraktion: Fügen Sie einen der unterstützten Plug-in-Typen hinzu.

CWOBJ1211E: Die PMI-Erstellung von {0} ist fehlgeschlagen. Die Ausnahme {0} wurde ausgelöst.
Erläuterung: Der Versuch, eine ObjectGrid-PMI zu erstellen, ist fehlgeschlagen.
Benutzeraktion: Untersuchen Sie die Ausnahmenachricht und das FFDC-Protokoll, um die Fehlerursache festzustellen.

CWOBJ1215I: Der Ereignis-Listener für die Weitergabe von ObjectGrid-Transaktionen wird initialisiert [ObjectGrid {0}].
Erläuterung: Der Ereignis-Listener für die Weitergabe von ObjectGrid-Transaktionen wird initialisiert.
Benutzeraktion: Keine. Es handelt sich hierbei um eine Informationsnachricht.

CWOBJ1216I: Der Ereignis-Listener für die Weitergabe von ObjectGrid-Transaktionen wurde initialisiert [ObjectGrid {0}].
Erläuterung: Der Ereignis-Listener für die Weitergabe von ObjectGrid-Transaktionen wurde initialisiert.
Benutzeraktion: Keine. Es handelt sich hierbei um eine Informationsnachricht.

CWOBJ1217I: Der Servicepunkt für die Weitergabe von ObjectGrid-Transaktionen wurde initialisiert [ObjectGrid {0}].
Erläuterung: Der Servicepunkt für die Weitergabe von ObjectGrid-Transaktionen wurde initialisiert.
Benutzeraktion: Keine. Es handelt sich hierbei um eine Informationsnachricht.

CWOBJ1218E: Fehler beim Ereignis-Listener für die Weitergabe von ObjectGrid-Transaktionen [ObjectGrid {0} Ausnahmenachricht {1}].
Erläuterung: Es ist ein Fehler bei der Weitergabe der ObjectGrid-Transaktionen aufgetreten.
Benutzeraktion: Untersuchen Sie die Ausnahme, um die Fehlerursache zu bestimmen.

CWOBJ1219E: Fehler beim Service-Endpoint für die Weitergabe von ObjectGrid-Transaktionen [ObjectGrid {0} Ausnahmenachricht {1}].
Erläuterung: Es ist ein Fehler beim Service-Endpoint für die Weitergabe von ObjectGrid-Transaktionen aufgetreten.
Benutzeraktion: Untersuchen Sie die Ausnahme, um die Fehlerursache zu bestimmen.

CWOBJ9000I: This is an English only Informational message: {0}
Erläuterung: Message was added after translation cutoff and so is not translated.
Benutzeraktion: See message for details.

CWOBJ9001W: This is an English only Warning message: {0}
Erläuterung: Message was added after translation cutoff and so is not translated.
Benutzeraktion: See message for details.

CWOBJ9002E: This is an English only Error message: {0}
Erläuterung: Message was added after translation cutoff and so is not translated.
Benutzeraktion: See message for details.

Bemerkungen

Hinweise auf IBM Produkte, Programme und Services in dieser Veröffentlichung bedeuten nicht, dass IBM diese in allen Ländern, in denen IBM vertreten ist, anbietet. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Anstelle der Produkte, Programme oder Dienstleistungen können auch andere ihnen äquivalente Produkte, Programme oder Dienstleistungen verwendet werden, solange diese keine gewerblichen oder andere Schutzrechte verletzen. Die Verantwortung für den Betrieb der Produkte, Programme oder Services in Verbindung mit Fremdprodukten und Fremdservices liegt beim Kunden, soweit solche Verbindungen nicht ausdrücklich von IBM bestätigt sind.

Für in diesem Dokument beschriebene Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York 10594 USA

Marken und Servicemarken

Folgende Namen sind in gewissen Ländern (oder Regionen) Marken der IBM Corporation:

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- MQSeries
- MVS
- OS/390
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java und alle Java-basierten Marken sind in gewissen Ländern Marken von Sun Microsystems, Inc.

LINUX ist in gewissen Ländern eine Marke von Linus Torvalds.

Microsoft, Windows, Windows NT und das Windows-Logo sind in gewissen Ländern Marken der Microsoft Corporation.

UNIX ist in gewissen Ländern eine eingetragene Marke von The Open Group.

Andere Namen von Unternehmen, Produkten und Dienstleistungen können Marken anderer Unternehmen sein.