WebSphere WebSphere Extended Deployment バージョン 6.0.x

ObjectGrid プログラミング・モデル・ガイド

ご注意

本書および本書で紹介する製品をご使用になる前に、 393 ページの『特記事項』に記載されている情報をお読みください。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： WebSphere Extended Deployment Version 6.0.x
ObjectGrid programming model guide

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2006.3

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、 平成角ゴシック体™W5、および平成角ゴシック体™ W7を使用しています。 この(書体*)は、(財）日本規格協会と使用契約を締結し使用しているものです。 フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

第 1 章 サンプル・アプリケーションの実行による ObjectGrid の開始

このトピックを使用して、オブジェクトをアプリケーションのセットに対して使用可能にする分散コンピューティング・フレームワーク ObjectGrid を開始します。

WebSphere Extended Deployment バージョン 6.0 以降および WebSphere Application Server バージョン 6.0.2 以降が、環境内の少なくとも 1 つのマシンにインストールされている必要があります。

制約事項: WebSphere Extended Deployment バージョン 6.0 で ObjectGrid を使用している場合は、Java 2 Platform, Standard Edition (J2SE) バージョン 1.4.2 以降の環境または WebSphere Application Server バージョン 6.02 以降の環境でも ObjectGrid を使用するためには追加のライセンス契約が必要です。詳しくは、営業担当員にお問い合わせください。

WebSphere Extended Deployment がインストールされているサーバー・マシンにアクセスしないで ObjectGrid アプリケーションを作成する場合、ローカル・マシンでアプリケーションを実行することができます。ローカル・マシンには、 IBM Software Developer Kit (SDK) または Eclipse のインストールが必要です。

ObjectGrid アプリケーションをローカル・マシン上で開発するには、インストール済み環境からローカル・マシンに次のディレクトリーをコピーします。

- WebSphere Extended Deployment バージョン 6.0.1 を使用する場合、/lib/wsobjectgrid.jar ファイルおよび /optionalLibraries/ObjectGrid/objectgridSamples.jar ファイルを作業ディレクトリーにコピーします。
- 混合サーバー環境のインストールを介して ObjectGrid をインストールした場合、/ObjectGrid/lib/objectgrid.jar ファイルおよび /ObjectGrid/samples/objectgridSamples.jar ファイルを作業ディレクトリーにコピーします。

ObjectGrid と一緒にインストールされた Java アーカイブ (JAR) ファイルについて詳しくは、『ObjectGrid のパッケージ化』を参照してください。

このタスクを使用して、ObjectGrid サンプル・アプリケーションを実行およびステップスルーします。このタスクのアプリケーションは、Java コマンド行、Eclipse、または Java 2 Platform, Enterprise Edition (J2EE) 環境で実行することができます。

- コマンド行で実行中の ObjectGrid サンプル・アプリケーションを取得するには、『コマンド行での ObjectGrid サンプル・アプリケーションの実行』を参照してください。
- ObjectGrid サンプル・アプリケーションを Eclipse で実行するには、『Eclipse における ObjectGrid サンプル・アプリケーションのインポートおよび使用』を参照してください。
- ObjectGrid サンプル・アプリケーションを WebSphere Extended Deployment で実行するには、『WebSphere Extended Deployment での ObjectGrid サンプル・アプリケーションのロードおよび実行』を参照してください。

サンプル・アプリケーションを実行し、サンプルを開発環境にロードすることによって、ObjectGrid を開始しました。

コマンド行での ObjectGrid サンプル・アプリケーションの実行

このトピックを使用し、ObjectGrid 対応アプリケーションを Java コマンド行で実行し、 ObjectGrid 構成をテストします。

このタスクの開始前に、スタンドアロン ObjectGrid を含む混合サーバー環境をインストールしてください。

Software Development Kit (SDK) がインストールされている必要があります。また、 ObjectGrid サンプル・アプリケーションへのアクセスを持っている必要があります。詳しくは、『ObjectGrid の開始』を参照してください。

このタスクを使用して、ObjectGrid が使用可能になったアプリケーションを即座に実行します。

1. Software Development Kit (SDK) のバージョンを確認します。 ObjectGrid には、 IBM SDK 1.4.2 以降が必要です。ObjectGrid サンプル・アプリケーションを実行する前に、Java 環境をテストするには、以下のステップを実行します。
   a. コマンド行プロンプトを開きます。
   b. 以下のコマンドを入力します。

      ```
      java -version
      ```

      コマンドが正しく実行されると、以下の例に似たテキストが表示されます。

      ```
      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))
      ```

      注: また、Java 2 Platform, Standard Edition (J2SE) バージョン 1.3.x Software Development Kit (SDK) を使用してもこれらのサンプルを実行できます。詳しくは、『ObjectGrid のパッケージ化』を参照してください。

      エラーが表示される場合は、 SDK がインストールされていて、ご使用の CLASSPATH 内にあることを確認します。
2. ObjectGrid サンプル・アプリケーションを実行します。 サンプル・アプリケーションは、従業員、オフィス、および勤務地に関する単純な事例を説明します。サンプル・アプリケーションは、各オブジェクト・タイプに対して、マップ付きの ObjectGrid インスタンスを作成します。各マップには、ObjectGrid キャッシング関数を示すために挿入および操作されるエントリーがあります。
   a. コマンド行を開き、作業ディレクトリーにナビゲートします。
      objectgrid.jar、asm.jar、および cglib.jar ファイルを /ObjectGrid/lib フォルダーから作業ディレクトリーにコピーします。/ObjectGrid/samples/objectgridSamples.jar を作業ディレクトリーにコピーします。
   b. 以下のコマンドを発行します。

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

システムに、以下のテキストに似た出力が表示されます。この出力は、パブリッシュの目的のために短縮されています。

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

3. 分散 ObjectGrid サンプル・アプリケーションを実行します。

com.ibm.websphere.samples.objectgrid.basic.ObjectGridSample プログラムは、データ・キャッシュとしてローカル ObjectGrid インスタンスを使用します。すべてのオブジェクトは、ローカルの Java 仮想マシン (JVM) 内にキャッシュされます。ObjectGrid クラスターにデプロイされた分散 ObjectGrid を使用するには、

com.ibm.websphere.samples.objectgrid.distributed.DistributedObjectGridSample プログラムを使用します。DistributedObjectGridSample プログラムは、objectgridSamples.jar 内に含まれています。

a. ObjectGrid クラスターを開始します。 スタンドアロン ObjectGrid クラスターを開始して、分散 ObjectGrid のサンプルで使用することについて詳しくは、『スタンドアロンのサンプル ObjectGrid クラスターの開始』を参照してください。

b. ObjectGrid サーバーを開始したら、以下のコマンドを使用して、分散 ObjectGrid のサンプル・アプリケーションを実行できます。

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

必要な ObjectGrid クラスターが開始したら、DistributedObjectGridSample プログラムは ObjectGridSample プログラムと同様の出力を出力します。

ObjectGrid サンプル・アプリケーションを、Java コマンド行で実行し、ObjectGrid 機能性をテストしました。

このサンプルのソースは、objectgridSamples.jar ファイルに入っています。具体的には com¥ibm¥websphere¥samples¥objectgrid¥basic¥ObjectGridSample.java および com¥ibm¥websphere¥samples¥objectgrid¥distributed¥DistributedObjectGridSample.java ファイルに入っています。

スタンドアロンのサンプル ObjectGrid クラスターの開始

分散 ObjectGrid のサンプルを実行するには、必要な ObjectGrid をホストする ObjectGrid クラスターを開始する必要があります。

WebSphere Extended Deployment for Mixed Server Environment、バージョン 6.0.x がインストールされていることを確認します。

このタスクを使用して、`cluster-config-1.xml` ファイルおよび `cluster-objectgrid-definition.xml` ファイルに基づく ObjectGrid サーバーを開始します。このタスクは、分散 ObjectGrid のサンプルを実行するのに必要です。詳しくは、『コマンド行での ObjectGrid サンプル・アプリケーションの実行』および『Eclipse における ObjectGrid サンプル・アプリケーションのインポートおよび使用』を参照してください。`cluster-config-1.xml` は、1 つの ObjectGrid サーバー定義のみを保持します。この ObjectGrid サーバーは、サンプルの ObjectGrid クラスターを表します。

1. `objectgridSamples.jar` ファイルを *`mse_install_root`*`/ObjectGrid/samples` ディレクトリーに配置します。
2. `META-INF/cluster-config-1.xml` ファイルおよび `META-INF/cluster-objectgrid-definition.xml` ファイルを `objectgridSamples.jar` ファイルから *`mse_install_root`*`/ObjectGrid/samples` ディレクトリーに抽出します。
3. JAVA_HOME 環境変数が設定されていること、および Java のバージョンが要件を満たしていることを確認します。 ObjectGrid サーバーには、Java 2 Platform, Standard Edition (J2SE) バージョン 1.4.2 以降の環境が必要です。Java 環境を確認するには、次のステップを実行します。

a. JAVA_HOME 環境変数を確認します。 コマンド行プロンプトで、次のコマンドを発行します。

```
echo %JAVA_HOME%
```

このコマンドは、Java へのパスを示します。JAVA_HOME 環境変数を設定する必要がある場合は、次のコマンドを実行します。

```
set JAVA_HOME=JDK_INSTALL_ROOT
```

JDK_INSTALL_ROOT を Java インストール・ディレクトリー (例: c:¥java など) に設定します。

b. Java のバージョンを確認します。 以下のコマンドを実行します。

```
java -version
```

ご使用のバージョンが Java 2 Platform, Standard Edition (J2SE) バージョン 1.4.2 以降であることを確認します。

4. ObjectGrid サーバーを開始します。 コマンド行プロンプトで、次のコマンドを発行します。

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

重要: クラスパス内で -jvmArgs オプションを使用して objectgridSamples.jar ファイルを指定する必要があります。objectgridSamples.jar ファイルには、サンプル ObjectGrid サーバーが、cluster-objectgrid-definition.xml ファイル内で定義されたプラグインのインプリメンテーションに必要とするクラスが含まれています。この JAR ファイルは、マップ内に保管されているオブジェクトのシリアライズおよびデシリアライズにも使用されます。

システムに、以下のテキストに似た出力が表示されます。この出力は、パブリッシュの目的のために短縮されています。

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

分散 ObjectGrid サンプル・アプリケーションの実行については、『コマンド行での ObjectGrid サンプル・アプリケーションの実行』または 『Eclipse における ObjectGrid サンプル・アプリケーションのインポートおよび使用』を参照してください。コマンド行でのスタンドアロン ObjectGrid サーバーの開始および停止について詳しくは、85 ページの『第 8 章 コマンド行サポート』を参照してください。

Eclipse における ObjectGrid サンプル・アプリケーションのインポートおよび使用

このタスクを使用して、Eclipse において ObjectGrid サンプル・アプリケーションをインポートおよび使用します。

このタスクの開始前に、スタンドアロン ObjectGrid を含む混合サーバー環境をインストールしてください。

このサンプル・アプリケーションの場合、Eclipse バージョン 3.1 以降を使用して、サンプルをインポートおよび実行します。Eclipse の入手は、WebSphere Application Server とともに組み込まれている Application Server Toolkit から、Rational Application Developer のインストールから、または Eclipse.org から直接ダウンロードします。

Eclipse を使用することにより、アプリケーションを容易にデバッグすることができます。サンプル・アプリケーションの段階的なウォークスルーを実行することができます。

1. プロジェクトを Eclipse にインポートします。
   a. Eclipse プログラムを実行します。 Eclipse インストール・ディレクトリー内の `eclipse.exe` ファイルを使用します。
   b. Eclipse を使用して、新規プロジェクトを作成します。
      1) 「**ファイル**」>「**新規作成**」 >「**プロジェクト**」>「**Java**」>「**Java プロジェクト**」をクリックします。「**次へ**」をクリックします。
      2) プロジェクト名を入力します。例えば、`ObjectGridSamples` と入力します。
      3) 「**Create new project in workspace**」を選択します。
      4) 「Project Layout」セクションで、「**Configure default**」をクリックします。
      5) ソースおよび出力フォルダーの場合は、「**プロジェクト**」を選択して「**OK**」をクリックします。
      6) 「**次へ**」をクリックします。
      7) 「**ライブラリー**」タブをクリックします。
      8) 「**外部 JAR の追加**」をクリックします。
      9) `/ObjectGrid/lib` フォルダーにナビゲートし、**objectgrid.jar**、**asm.jar**、および **cglib.jar** の各ファイルを選択します。「**JAR selection**」ウィザードで「**オープン**」をクリックします。
      10) 「**終了**」をクリックします。
2. `objectgridSamples.jar` ファイルを Java プロジェクトへインポートします。
   a. Java プロジェクトを右クリックし、「**インポート**」を選択します。
   b. 「**Select an import source**」の下の「**Zip ファイル**」を選択します。
   c. 「**次へ**」をクリックします。
   d. 「**参照**」をクリックし、「Import From Zip File wizard」を開きます。
   e. **objectgridSamples.jar** ファイルを開きます。 `/ObjectGrid/samples` ディレクトリーにナビゲートします。 **objectgridSamples.jar** ファイルを選択し、「**オープン**」をクリックします。
   f. ルート・ファイル・ツリーのチェック・ボックスが選択されていることを確認します。
   g. 「**フォルダー内**」に前のステップで作成した Java プロジェクト (例えば `ObjectGridSamples` プロジェクト) が含まれていることを確認します。

h. 「**終了**」をクリックします。

3. Java プロジェクトのプロパティーを確認します。
   a. Java パースペクティブを開きます。「**ウィンドウ**」>「**パースペクティブのオープン**」>「**Java**」をクリックします。
   b. コンソール・ビューに進みます。「**ウィンドウ**」>「**Show view**」>「**コンソール**」をクリックします。
   c. 「パッケージ・エクスプローラー」ビューが使用可能で、選択されていることを確認します。「**ウィンドウ**」>「**Show View**」>「**パッケージ・エクスプローラー**」をクリックします。
   d. Java プロジェクトを右クリックし、「**プロパティー**」を選択します。
   e. 左側のパネルで「**Java ビルド・パス**」をクリックします。
   f. 右側のパネルで「**ソース**」タブをクリックします。
   g. プロジェクト・ルートが、「Build path」パネル上のソース・フォルダーにリストされていることを確認します。
   h. 右側のパネルで「**ライブラリー**」タブをクリックします。
   i. `objectgrid.jar`、`asm.jar`、および `cglib.jar` ファイルおよび JRE システム・ライブラリーが「Build path」パネルの JAR およびクラス・フォルダーにリストされていることを確認します。
   j. 「**OK**」をクリックします。
4. ObjectGrid サンプルを実行します。
   a. 「パッケージ・エクスプローラー」ビューから、Java プロジェクトを展開します。
   b. com.ibm.websphere.samples.objectgrid.basic パッケージを拡張します。
   c. **ObjectGridSample.java** ファイルを右クリックします。「**実行**」>「**Java アプリケーション**」をクリックします。
   d. コンソールは、Java コマンド行でアプリケーションを実行するときに、類似した出力を表示します。 出力の例については、『コマンド行での ObjectGrid サンプル・アプリケーションの実行』を参照してください。
5. 分散 ObjectGrid のサンプルを実行します。 分散 ObjectGrid のサンプルを実行するには、ObjectGrid クラスターを構成する必要があります。このサンプルの実行には、objectgridSamples.jar ファイルで提供されている、定義済み XML 構成ファイルを使用できます。詳しくは、『スタンドアロンのサンプル ObjectGrid クラスターの開始』を参照してください。

   ObjectGrid サーバーが始動された後は、以下のステップで分散 ObjectGrid のサンプル・アプリケーションを実行できます。
   a. 「パッケージ・エクスプローラー」ビューから、Java プロジェクトを展開します。
   b. com.ibm.websphere.samples.objectgrid.distributed パッケージを展開します。
   c. **DistributedObjectGridSample.java** ファイルを右クリックします。「**実行**」>「**Java アプリケーション**」をクリックします。
   d. コンソールに ObjectGrid サンプルと類似した出力が表示されます。

プロジェクトをロードし、デバッガーを実行する手順は SamplesGuide.htm ファイルにもあります。 SamplesGuide.htm ファイルは、objectgridSamples.jar ファイルの doc ディレクトリーにあります。

関連資料

ObjectGrid のパッケージ化
ObjectGrid パッケージには、WebSphere Extended Deployment のインストールまたは混合サーバー環境のインストールの 2 つの方法でアクセスすることができます。

WebSphere Extended Deployment での ObjectGrid サンプル・アプリケーションのロードおよび実行

このタスクを使用して、WebSphere Extended Deployment 内で Java 2 Platform, Enterprise Edition (J2EE) ObjectGrid サンプルをロードおよび実行します。

WebSphere Application Server および WebSphere Extended Deployment がインストールされている必要があります。

このタスクは、WebSphere Extended Deployment における ObjectGrid の統合を理解し、テストするために使用します。詳しくは、303 ページの『第 10 章 ObjectGrid と WebSphere Application Server の統合』を参照してください。

1. ObjectGridSample.ear ファイルをインストールします。 エンタープライズ・アーカイブ (EAR) ファイルは、単一アプリケーション・サーバーまたはクラスターにインストールすることができます。管理コンソールで ObjectGridSample.ear ファイルをインストールするには、以下のステップを実行します。
   a. 管理コンソールで、「**アプリケーション**」>「**新規アプリケーションのインストール**」をクリックします。
   b. 「**Preparing for application installation**」ページで、 ObjectGrid サンプル・アプリケーションのロケーションを指定します。 例えば、<install_root>/installableApps/ObjectGridSample.ear を参照します。「**次へ**」をクリックします。
   c. 2 番目の「**Preparing for application installation**」ページで、デフォルト設定にして、「**次へ**」をクリックします。
   d. 「**Select installation options**」ページでデフォルト設定を採用し、「**次へ**」をクリックします。
   e. 「**Map modules to servers**」ページで、アプリケーションに含まれるモジュールをインストールするデプロイメント・ターゲットを指定します。 すべてのモジュールの「**クラスターおよびサーバー**」リストからターゲット・サーバーまたはクラスターを選択します。「**モジュール**」チェック・ボックスを選択し、すべてのアプリケーション・モジュールを選択するか、個々のモジュールを選択します。
   f. 以下のページで、デフォルト値を使用して、「**終了**」をクリックします。
   g. アプリケーションのインストールが終了した後、「**マスター構成に保管**」をクリックします。

h. 「**ノードと変更を同期化**」オプションをクリックします。 「**エンタープライズ・アプリケーション**」>「**保管**」ページで、「**保管**」をクリックします。

i. 「**OK**」をクリックします。

2. サーバーの default_host の HTTP ポートを確認し、ホスト別名を追加します。インストール中にホスト名を変更しない限り、デフォルトでは、Web モジュールは default_host 仮想ホスト名にバインドされます。クラスター上でアプリケーションをインストールしている場合は、各クラスター・メンバーの default_host の HTTP ポートに対して、少なくとも 1 つのホスト別名を構成する必要があります。また、各クラスター・メンバーの default_host の HTTP ポートを確認し、管理コンソールのホスト別名リストに、対応するホスト別名を追加する必要があります。 サーバーの default_host の HTTP ポートを確認するには、以下のステップを実行します。

   a. 管理コンソールで、「**サーバー**」>「**アプリケーション・サーバー**」>「***server_name***」をクリックします。

   b. 通信セクションで、ポートを展開します。**WC_defaulthost** ポートは、default_host 仮想ホスト・ポートです。

   ホスト別名を追加するには、以下のステップを実行します。

   a. 管理コンソールで、「**環境**」>「**仮想ホスト**」>「**default_host**」>「**ホスト別名**」>「**新規作成**」をクリックします。

   b. ホスト名のデフォルト値を使用し、ポートを指定します。

   c. 「**OK**」をクリックします。

3. ObjectGrid サンプル・アプリケーションを開始します。

   - サーバー上でアプリケーションを開始するには、「**サーバー**」>「**アプリケーション・サーバー**」をクリックします。 `ObjectGridSample.ear` ファイルがインストールされているサーバーを選択します。「**開始**」をクリックします。
   - クラスター上でアプリケーションを開始するには、「**サーバー**」>「**クラスター**」をクリックします。 `ObjectGridSample.ear` ファイルがインストールされているクラスターを選択します。「**開始**」をクリックします。

   サーバーまたはクラスター上でアプリケーションを開始した後、ホスト・サーバーまたはクラスターから独立してアプリケーションを停止および開始することができます。 ObjectGrid サンプル・アプリケーションを停止または開始するには、以下のステップを実行します。

   a. 管理コンソールで、「**アプリケーション**」>「**エンタープライズ・アプリケーション**」をクリックします。

   b. ObjectGrid サンプル・アプリケーションを選択します。

   c. 「**開始**」または「**停止**」をクリックします。

4. ObjectGrid サンプルにアクセスします。 `ObjectGridSample.ear` ファイルを単一サーバーまたはクラスターにインストールし、アプリケーションを開始した後、以下の Web アドレスで ObjectGrid サンプルにアクセスすることができます。

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

   例えば、ホスト名が `localhost` で、ポート値が `9080` である場合、 Web アドレス `http://localhost:9080/ObjectGridSample` を使用します。

5. WebSphere Application Server 環境で、分散 ObjectGrid の機能をテストします。 ObjectGridSample.ear ファイルには、WebSphere Application Server 環境における分散 ObjectGrid の使用を説明する DistributedObjectGridServlet サーブレットも含まれています。 DistributedObjectGridServlet サーブレットをホストするアプリケーション・サーバーは、必要な ObjectGrid クラスターのメンバーである ObjectGrid サーバーもホストする必要があります。
   - DistributedObjectGridServlet を実行するための ObjectGrid クラスターの構成について詳しくは、『WebSphere 環境でのサンプル ObjectGrid クラスターの開始』を参照してください。
   - アプリケーション・サーバーにおける ObjectGrid サーバーの始動について詳しくは、『アプリケーション・サーバーでの ObjectGrid サーバーの始動』を参照してください。

   ObjectGridSample.ear ファイルがインストールされたアプリケーション・サーバーでも必要な ObjectGrid サーバーをホストすると、DistributedObjectGridServlet サーブレットは他のサーブレットと同様に振る舞います。サーブレットには、次の Web アドレス http://hostname:port/ObjectGridSample/DistributedObjectGridServlet によりアクセスできます。 例えば、ホスト名が localhost で、ポート値が 9080 である場合、 Web アドレス http://localhost:9080/ObjectGridSample/DistributedObjectGridServlet を使用します。

   ObjectGrid のトレースは、次のトレース・ストリング ObjectGrid*=all=enabled を使用することにより可能です。

WebSphere Extended Deployment サーバー上に、ObjectGrid サンプル・アプリケーションおよび分散 ObjectGrid サンプル・アプリケーションをインストールし、構成しました。

サーバーまたはクラスター上にアプリケーションをインストールした後、以下の Web アドレスでアプリケーションを開始すると、サンプル文書にアクセスすることができます。

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

例えば、ホスト名が **localhost** で、ポート値が **9080** である場合、 Web アドレス http://localhost:9080/ObjectGridSample/docs/introduction.html を使用します。

WebSphere 環境でのサンプル ObjectGrid クラスターの開始

このタスクでは、単純な ObjectGrid クラスターを開始し、WebSphere Application Server 環境における分散 ObjectGrid の機能をテストします。

WebSphere Extended Deployment がインストールされている必要があります。アプリケーション・サーバーに ObjectGridSample.ear ファイルがインストールされている必要があります。 ObjectGridSample.ear ファイルのインストールについて詳しくは、『WebSphere Extended Deployment での ObjectGrid サンプル・アプリケーションのロードおよび実行』を参照してください。

このタスクを使用し、cluster-config-1.xml および cluster-objectgrid-definition.xml の各ファイルを基にした ObjectGrid サーバーをホストするアプリケーション・サーバーをセットアップします。

cluster-config-1.xml ファイルには、1 つの ObjectGrid サーバー定義のみがあります。この ObjectGrid サーバーは、サンプルの ObjectGrid クラスターを表します。1 つのスタンドアロン・アプリケーション・サーバー、または 1 つのクラスター・メンバーがあるクラスターのいずれかを使用して、サンプル ObjectGrid サーバーをホストできます。

1. META-INF/cluster-config-1.xml および META-INF/cluster-objectgrid-definition.xml の各ファイルの両方を、/optionalLibraries/ObjectGrid/objectgridSamples.jar ファイルから /optionalLibraries/ObjectGrid ディレクトリーに取り出します。
2. 必要な汎用 JVM 引数を定義します。
   a. 管理コンソールで、「**サーバー**」>「**アプリケーション・サーバー**」>「***server_name***」>「**プロセス定義**」>「**Java 仮想マシン**」をクリックします。
   b. 「汎用 JVM 引数」パネルで、以下のテキストを入力します。

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

      INSTALL_ROOT は、 WebSphere Application Server インストールのルート・ディレクトリーです。
   c. 「**保管**」をクリックします。
   d. 「**マスター構成に保管**」をクリックします。
   e. 「**ノードと変更を同期化**」オプションを選択します。「**保管**」をクリックします。
3. /optionalLibraries/ObjectGrid/objectgridSamples.jar ファイルを /classes または lib/ext ディレクトリーにコピーします。 objectgridSamples.jar には、cluster-objectgrid-definition.xml ファイル内で定義されたプラグインのインプリメンテーションのためにサンプル ObjectGrid サーバーが必要とするクラスが含まれています。この JAR ファイルは、マップ内に保管されているオブジェクトのシリアライズおよびデシリアライズにも使用されます。
4. サーバーを再始動し、加えた変更を有効にします。

アプリケーション・サーバーにおける ObjectGrid サーバーの始動および停止について詳しくは、『アプリケーション・サーバーでの ObjectGrid サーバーの始動』を参照してください。

アプリケーション・サーバーでの ObjectGrid サーバーの始動

ObjectGrid サーバーは、アプリケーション・サーバー内で開始するように構成できます。WebSphere Application Server は ObjectGrid コンポーネントを検出し、自動的に ObjectGrid サーバーを開始します。

ObjectGrid サーバーは、WebSphere Extended Deployment や WebSphere Business Integration Server などのアドオンをインストールする場合も含め、WebSphere

Application Server バージョン 6.0.2 以降で構成できます。旧バージョンの WebSphere Application Server (WebSphere Application Server バージョン 5.0.2 など) は、ObjectGrid をクライアントとして使用するアプリケーションを持つことができますが、ObjectGrid サーバー機能を旧バージョンのアプリケーション・サーバーと連結することはできません。

複製を可能にするクラスター構成を使用している場合は、HA マネージャーが必要です。ObjectGrid サーバーによる HA マネージャーの使い方は、通常のアプリケーション・サーバーとは異なります。ObjectGrid サーバーがアプリケーション・サーバー内にある場合、ObjectGrid サーバーは HA マネージャー・サービスを構成、初期化、または作成せずに、アプリケーション・サーバー内の既存の高可用性サービスを使用します。 ObjectGrid サーバー間で複製を行うには、同じコア・グループのメンバーであるアプリケーション・サーバー内で ObjectGrid サーバーが稼働している必要があります。

ObjectGrid サーバーのその他の機能はすべて、サーバーが WebSphere Application Server で稼働している場合と同じです。ObjectGrid クラスター仕様に 3 つのサーバーが含まれている場合、単一のコア・グループ内の任意の 3 つのアプリケーション・サーバーでこれらの ObjectGrid サーバーをホストできます。クラスターが同一のコア・グループに属している限り、アプリケーション・サーバーはクラスターをまたぐこともできます。最も重要なステップは、cluster.xml ファイル内でサーバーの TCP/IP ホスト名とポート情報を関連付けることです。

次のタスクを使用して、WebSphere Application Server 環境のアプリケーション・サーバー内で ObjectGrid サーバーを実行します。

1. Java 仮想マシン (JVM) で、必要なカスタム・プロパティーを追加します。 管理コンソールで、「**サーバー**」>「**アプリケーション・サーバー**」>「***server_name***」>「**Java およびプロセス管理**」>「**プロセス定義**」>「**Java 仮想マシン**」>「**カスタム・プロパティー**」をクリックします。「**新規作成**」をクリックします。以下のカスタム・プロパティーを作成します。

表 1. ObjectGrid サーバーの JVM カスタム・プロパティー

カスタム・プロパティーの名前	**説明**	**ERROR! SEGMENT DATA CORRUPTED, SEGDATA= 値の例**
objectgrid.server.name	このアプリケーション・サーバー内で使用される ObjectGrid サーバーの名前を指定します。提供する名前は、ObjectGrid クラスター XML ファイルで定義されているサーバー名の 1 つにする必要があります。	server1
objectgrid.xml.url	ObjectGrid XML ファイルの Universal Resource Locator (URL) を指定します。このプロパティーは必須です。	file:///d:/was/etc/test/objectGridMatch.xml

表 1. ObjectGrid サーバーの JVM カスタム・プロパティー (続き)

カスタム・プロパティーの名前	説明	ERROR! SEGMENT DATA CORRUPTED, SEGDATA= 値の例
objectgrid.cluster.xml.url	ObjectGrid クラスター XML ファイルの URL を指定します。このプロパティーは必須です。	file:///d:/was/etc/test/csCluster0.xml
objectgrid.security.server.props	ObjectGrid サーバー・セキュリティー・プロパティー・ファイルの URL を指定します。このプロパティーは、ObjectGrid クラスター XML ファイルでセキュリティーが使用可能になっている場合に限り必須です。クラスター XML ファイルでセキュリティーが使用可能になっているかどうかを判別するには、以下のテキストを探します。 `<cluster name="cluster1" securityEnabled="true" ....` securityEnabled 属性が false に設定されている場合は、このプロパティーを定義する必要はありません。 テンプレートとして security.ogserver.props ファイルを使用してください。このファイル内のこれらのプロパティーの意味、および使用方法については、 144 ページの『ObjectGrid セキュリティー』を参照してください。	file:///d:/was/optionalLibraries/ObjectGrid/properties/security.ogserver.props

これらの JVM プロパティーは、管理コンソールの「**Java 仮想マシン**」パネルの「**汎用 JVM 引数**」フィールドでも定義できます。以下に、「汎用 JVM 引数」フィールドの値の例を示します。

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

2. 変更を保管し、アプリケーション・サーバーを再始動します。 WebSphere Application Server は ObjectGrid コンポーネントを検出し、自動的に ObjectGrid サーバーを開始します。

   アプリケーション・サーバー内の ObjectGrid は、チャネル・フレームワークを使用して、ObjectGrid クライアント、特にクライアント・アクセス・ポートと呼

ばれる ObjectGrid クライアントと対話します。ObjectGrid サーバーは、開始すると、WebSphere Application Server との連結を検出し、アプリケーション・サーバー内ですでに稼働しているチャネル・フレームワークを使用します。ObjectGrid サーバーは、チャネル・フレームワークがアプリケーション・サーバーに作成されていない場合、または開始していない場合にのみ、独自のチャネル・フレームワークを作成し、開始します。

3. ObjectGrid サーバーを停止します。 関連するアプリケーション・サーバーを停止して、ObjectGrid サーバーを停止します。ObjectGrid システム管理コマンドを使用して ObjectGrid サーバーを停止することはできません。

WebSphere Application Server 環境内のアプリケーション・サーバーは、ObjectGrid サーバーを実行します。

第 2 章 ObjectGrid

ObjectGrid は、Java 2 Platform、Standard Edition (J2SE) および Java 2 Platform, Enterprise Edition (J2EE) アプリケーション用の拡張可能なトランザクション・オブジェクト・キャッシング・フレームワークです。

ObjectGrid API を使用して、ObjectGrid フレームワークのオブジェクトを検索、保存、削除、および更新するアプリケーションを作成することができます。ご使用の ObjectGrid アプリケーション環境でのキャッシュの更新モニター、データの検索および外部データ・ソースとのデータの保管、キャッシュからのエントリー撤去の管理、およびバックグラウンド・キャッシュ機能の操作を行うカスタマイズされたプラグインを実装することもできます。

Map ベースの API

ObjectGrid は、java.util.Map インターフェースに基づく API を提供します。 API は、トランザクションのブロック内で操作のグループ化をサポートするように拡張されています。このインターフェースは、java.util.Map インターフェースのスーパーセットで、バッチ操作、無効化、キーワードの関連付け、および明示的な挿入と更新のサポートを追加します。 Java Map セマンティクスは、以下の拡張機能を実装できるように、拡張ポイントで拡張されています。

- 微調整キャッシュ・エントリー存続期間に対するキャッシュ Evictor
- 慎重にトランザクション管理を制御し、オプションで J2EE 環境の WebSphere トランザクション・マネージャーと統合するトランザクション・コールバック・インターフェース
- アプリケーション・プログラマーが ObjectGrid Map の get および put 操作を使用する場合に、データベースとの間のデータの検索と配置を自動的に行うローダー・インプリメンテーション
- すべてのコミットされたトランザクションが起こるたびにその情報を提供でき、全体として ObjectGrid フレームワークに向けて適用されるか、または特定の Map インスタンスに適用されるリスナー・インターフェース
- キーと値のより効率的なコピーと直列化を可能にするオブジェクト変換プログラム・インターフェース

ObjectGrid 環境

ObjectGrid フレームワークを使用するには、次のいずれかの既存オファリングをインストールします。

- ObjectGrid は、WebSphere Extended Deployment バージョン 6.0.1 に統合されており、フルインストールの一部となっています。
- スタンドアロン ObjectGrid は、混合サーバー環境 (MSE) インストールの一部になります。

いずれのオファリングでも、ObjectGrid ではクライアント/サーバー・フィーチャーがサポートされています。サーバー・ランタイムでは、分散オブジェクト・キャッシュのフル・クラスタリング、複製、および区画化がサポートされています。クライアント・ランタイムでは、ニア・キャッシュの概念、およびリモート・クラスタ

ーへのワークロード管理ルーティング・ロジックがサポートされています。また、クライアント・ランタイムでは、ローカル・オブジェクト・マップ作成もサポートされています。

サポートのレベルは、クライアント・ランタイム、サーバー・ランタイム、統合済み ObjectGrid、またはスタンドアロン ObjectGrid のいずれを実行しているかによって異なります。

WebSphere Extended Deployment オファリングに統合されている ObjectGrid

サーバー・ランタイム: サーバー・ランタイムが統合されています。WebSphere Extended Deployment バージョン 6.0.1 に統合されているランタイムは、z/OS プラットフォームではサポートされていません。

クライアント・ランタイム: クライアント・ランタイムは、J2SE および J2EE の JDK レベル 1.3.1 以降でサポートされています (WebSphere Application Server バージョン 5.0.2 以降を含む)。クライアント・ランタイムは、z/OS プラットフォームで完全にサポートされています。

スタンドアロン ObjectGrid オファリング

サーバー・ランタイム: サーバー・ランタイムは、スタンドアロンの Java 仮想マシン (JVM) で単一サーバーまたはサーバー・クラスターとして稼働できます。スタンドアロン・サーバーは、ほとんどの J2SE および J2EE プラットフォームの JDK レベル 1.4.2 以降でサポートされています。スタンドアロン・サーバーは、WebSphere Application Server バージョン 6.0.2 以降でサポートされています。スタンドアロン・サーバー・ランタイムは、WebSphere Extended Deployment バージョン 6.0.1 の z/OS プラットフォームではサポートされていません。

クライアント・ランタイム: クライアント・ランタイムは、J2SE および J2EE プラットフォームの JDK レベル 1.3.1 以降でサポートされています (WebSphere Application Sever バージョン 5.0.2 以降を含む)。

セッション管理

HTTP セッション・オブジェクトを ObjectGrid に格納する、完全に分散された HTTP セッション管理インプリメンテーションが提供されています。

シンプルなインストール

簡単なステップをいくつか実行するだけで、ObjectGrid をインストールし、構成することができます。これらのステップでは、Java アーカイブ (JAR) ファイルをクラスパスにコピーしたり、いくつかの構成ディレクティブを定義したりします。

トランザクションの変更

プログラマチック・インターフェースを堅固にするために、トランザクションのコンテキストですべての変更を行います。トランザクションをアプリケーション内で明示的に制御することも、アプリケーションで自動コミット・プログラミング・モードを使用することもできます。これらのトランザクションの変更は、拡張が容易で耐障害性のあるアクセスを提供するために、非同期モードおよび同期モードの ObjectGrid クラスターで複製できます。

ObjectGrid は、単一の Java 仮想マシン (JVM) で稼働する単純なグリッドから、Java 仮想マシンの 1 つ以上の ObjectGrid クラスターを含むグリッドに拡張することができます。これらのサーバーによって、多数の ObjectGrid 対応クライアントでは、Map API を介してデータを利用できるようになります。 ObjectGrid クライアントでは、基本 Java Map API が使用されます。ただし、アプリケーション開発者は、Java TCP/IP およびリモート・メソッド呼び出し (RMI) API を開発する必要はありません。これは、ObjectGrid クライアントは、ネットワークを介して情報を保持しているその他の ObjectGrid サーバーにアクセスすることができるためです。単一の JVM に対するデータ・セットが大きすぎる場合は、ObjectGrid を使用してデータを区画化することができます。

また、ObjectGrid により、アプリケーション・ソリューションに高可用性機能が付け加えられます。オブジェクト共用は、プライマリー・サーバー、1 つ以上の複製サーバー、および 1 つ以上の待機サーバーが存在する複製モデルに基づいています。複製サーバーのこのクラスターは、複製グループと呼ばれます。複製グループへのアクセスが書き込み操作である場合は、要求はプライマリー・サーバーに送信されます。アクセスが読み取り操作の場合、またはマップが読み取り専用マップの場合、要求はプライマリー・サーバーまたは複製サーバーに送信できます。待機サーバーは、1 つのサーバーが失敗した場合の潜在的な複製サーバーとして定義されています。プライマリー・サーバーが失敗すると、停止時間を最小限に抑えるために複製サーバーがプライマリー・サーバーとなります。この振る舞いは、必要に応じて構成および拡張可能です。

より簡単なオブジェクト伝搬方法を使用する場合は、Extended Deployment バージョン 6.0 のときと同様にサービスの品質を低くしたピアツーピア・モデルも使用可能です。より簡単なこの分散トランザクション・サポートでは、メッセージ・トランスポートを使用して、ピアに変更を通知することができます。 WebSphere Application Server バージョン 6.0.2 以降を実行している場合、メッセージ・トランスポートが組み込まれています。 WebSphere Application Server バージョン 6.0.2 以降を実行していない場合は、Java Message Service (JMS) プロバイダーなどの別のメッセージ・トランスポートが提供される必要があります。

注入コンテナー互換性 API

単純な XML ファイルまたは Java API を使用してプログラマチックに ObjectGrid を構成します。 Java API は、インジェクション・ベースのフレームワークを使用して、アプリケーションを構成する環境で作業するようにも設計されています。API および ObjectGrid オブジェクトのインターフェースは、Inversion of Control (IoC) コンテナーによって呼び出すこともでき、キー ObjectGrid オブジェクトへの参照をアプリケーションにインジェクトできます。

拡張可能なアーキテクチャー

プラグインを作成することによって、ObjectGrid フレームワークのほとんどのエレメントを拡張することができます。 ObjectGrid を調整して、アプリケーションが一貫性とパフォーマンスの間の決定をトレードオフできるようにします。プラグインでカスタマイズしたコードは、次のアプリケーション固有の動作もサポートできます。

- 初期化、トランザクション開始、トランザクション終了、および破棄のために ObjectGrid インスタンス・イベントを listen します。

- トランザクション・コールバックを起動して、トランザクション固有の処理を可能にします。
- 汎用 ObjectGrid トランザクションで特定の共通トランザクション・ポリシーを実装します。
- 外部データ・ストアおよびその他の情報リポジトリーへの透過的で、共通なエントリー・ポイントおよびエグジット・ポイントのローダーを使用します。
- ObjectTransformer インターフェースを使用して、特定の方法で順序付け不可能なオブジェクトを処理します。

基本 ObjectGrid キャッシュ API インターフェースの使用に影響を及ぼすことなく、これらの動作それぞれを実装することができます。この透過性によって、キャッシュ・インフラストラクチャーを使用しているアプリケーションは、これらのアプリケーションに影響することなく、データ・ストアおよびトランザクション処理を大きく変更することができます。

1 次 API または第 2 レベルのキャッシュとして ObjectGrid を使用

ObjectGrid API は、ルックアサイド・キャッシュまたはライトスルー・キャッシュとしてアプリケーションによって直接使用できます。ライトスルー・モードでは、ObjectGrid が変更を適用し、アプリケーションにデータを直接かつ透過的に取り出せるように、アプリケーションはローダー・オブジェクトに接続します。
ObjectGrid はまた、アダプターを書き込むことによって、ポピュラーなオブジェクト・リレーショナル・マッパーの第 2 レベル・キャッシュとして使用することができます。アプリケーションはデータにアクセスする 1 次 API として、オブジェクト・リレーショナル・マッパーから API を使用するため、キャッシュはこのモードではアプリケーションに不可視です。

第 3 章 ObjectGrid の概要

ObjectGrid は、Java Map ベースのデータ・アクセス・モデルおよび分散キャッシング・テクノロジーを提供します。ObjectGrid を使用すると、可用性の高いクラスタリング環境を構成できます。ObjectGrid クライアントは、大規模な統合ソリューションのため、多数のさまざまな ObjectGrid クラスターに同時に接触することができます。また、ObjectGrid は、大量の正規化情報用の、豊富な、分散データ区分化ソリューションに、複数の Java 仮想マシン内のデータを提供します。基本的に、ObjectGrid は、正規化された Java API と、ローカルおよび分散キャッシングを可能にするネットワーク・サービスのセットです。ソリューションの範囲は、より豊富な Java Map ソリューションを必要とする単一の Java 仮想マシン (JVM) から、エンタープライズ全体内の複数の ObjectGrid クラスターから必要とされる、分散され拡張が容易なありとあらゆるデータ・サービスまでに渡ります。

単一の Java 仮想マシン (JVM) での ObjectGrid

ObjectGrid の最も基本的な使用法は、単一 JVM で使用することです。

ObjectGrid を使用して、ObjectGrid インスタンスのセットを作成することができます。各 ObjectGrid インスタンスには、1 つ以上の Java Map と互換性を持つインスタンスを組み込むことができます。 Java Map インスタンスには、Java プログラマーが使い慣れている get および put インターフェース、および現在の Java Map インターフェースと機能にはない追加のフィーチャーが備えられています。以下の図は、最も基本的な ObjectGrid の使用法を示しています。

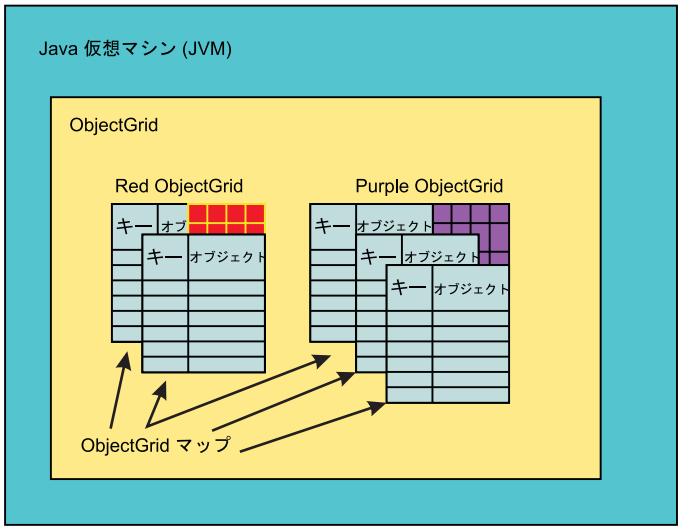


図 1. ObjectGrid JVM 使用法

ObjectGrid および ObjectGrid Map には、標準の Java Map インターフェースには現在備えられていない数多くのフィーチャーが備えられています。これらのフィーチャーには、トランザクション・アクセス、さまざまな種類のロック・ストラテジー (ロックなしストラテジー、オプティミスティック・ロック・ストラテジー、ペシミスティック・ロック・ストラテジー)、プラグ・アンド・プレイ除去機能、デー

タベースとのシームレスな相互作用 (get および put API の使用による副次作用として)、およびその他の多くの機能があります。また、ObjectGrid への独自の拡張機能を開発することもできます。例えば、所定の Map インスタンスに対してコミットされた各トランザクションの結果を提供する Map リスナーを開発することができます。ユーザーは、トランザクションが失われることのないように事業所のファイルなどに変更内容を記録しておいたり、Java Message Service (JMS) やその他のインフラストラクチャーを使用して変更内容を伝搬することができます。

前述の図では、JVM には 2 つの ObjectGrid インスタンスがあり、1 つには使用する 2 つの Java Map のようなオブジェクトが含まれ、もう 1 つには 3 つの Map オブジェクトが含まれます。 Map オブジェクトは 2 次元であるため、通常の Java Map のように操作するためにキーとオブジェクトをペア化することができます。単一の ObjectGrid インスタンスでは、数多くの特定のマップ・インスタンスをサポートすることができます。

以下の構成は、Red ObjectGrid インスタンスと Purple ObjectGrid インスタンスの基本的な ObjectGrid 構成を示しています。

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

分散 ObjectGrid

単一の JVM 内での ObjectGrid Java アーカイブ (JAR) ファイルの使用に加えて、分散環境内で ObjectGrid を使用できます。この環境では、ObjectGrid クラスターを作成できます。ObjectGrid クラスターは、それぞれが独自の単一の JVM を持った ObjectGrid サーバーのセットにより構成されます。

19 ページの『単一の Java 仮想マシン (JVM) での ObjectGrid』のトピックでは、ObjectGrid が Java Map の概念をサポートすることについて説明しています。この概念は、単一の JVM および 1 つ以上のリモート ObjectGrid キャッシュ・クラスターに接続する Java クライアントでもローカルにサポートされています。ObjectGrid サーバーは、単一の JVM の事例で前述した基本機能を配布する機能を提供します。例えば、複数のクライアントは、ロック・ストラテジー (ロックなしストラテジー、オプティミスティック・ロック・ストラテジー、ペシミスティック・ロック・ストラテジー) を使用して、同じ ObjectGrid インスタンス Map を共用できます。さらに、ObjectGrid クラスター・サーバー内の Evictor は、サーバー・サイドの Map インスタンス・データの除去を管理できます。すべてのクライアントは、共通の get および put のセマンティクスを使用することができます。また、ObjectGrid クラスター・サーバーで構成された Loader は、各クライアント上

で Java Database Connectivity (JDBC) ドライバーをデプロイおよび管理するのではなく、データベースとのすべての対話を実行します。

次の図では、JVM には、2 つの ObjectGrid インスタンスがあります。 1 つのインスタンスには、使用する 2 つの Java Map のようなオブジェクトがあり、もう 1 つのインスタンスには、3 つの Java Map のようなオブジェクトがあります。それぞれのマップは、キーおよびオブジェクトにできる 2 次元オブジェクトです。単一の ObjectGrid インスタンスは、主にアプリケーションの要件に応じて、大量のマップをサポートできます。この事例の違いは、マップが ObjectGrid クラスター・サーバーに配置されているということです。クライアントは、通常の Java アプリケーションまたは Java 2 Platform, Enterprise Edition (J2EE) アプリケーション・サーバーにできます。

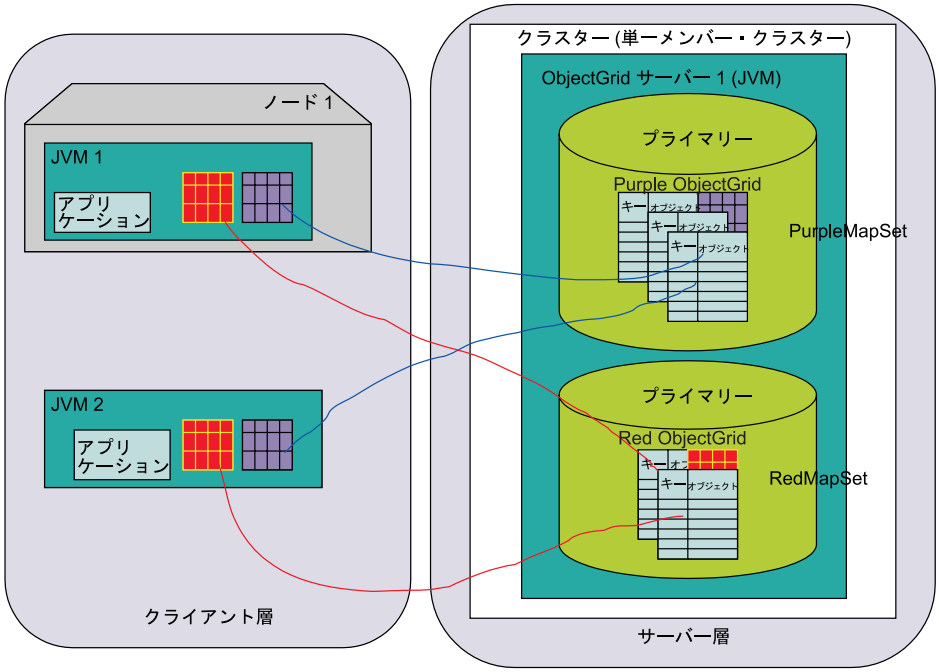


図 2. 分散 ObjectGrid の単一サーバーのトポロジー (2 つの MapSet)

ObjectGrid クライアント

ObjectGrid クライアントは、API のセットにより構成され、ObjectGrid クラスターに接続し、ObjectGrid クラスターの幅の広い構成を介してブートストラップし、次に実際に分散された ObjectGrid マップ操作を実行します。ObjectGrid クライアントは、分散方式で ObjectGrid を使用する独自の JVM インスタンス内の Java アプリケーションです。分散 ObjectGrid クライアントは、同じ Java 仮想マシン内で非分散機能を使用することもできます。ObjectGrid クライアントの用法は、複数の並列の ObjectGrid 接続 (それぞれセキュリティーが有効で、別のユーザーのために動作する) を持つ全体のアプリケーション・サーバー同様に複雑になる場合があります。

分散動作を有効にするには、ObjectGrid クラスター (ObjectGrid ソリューションのサーバー・サイドのサービス) を作成する必要があります。追加して構成する必要があるのは、ObjectGrid 構成ファイルに加えて、ObjectGrid クラスター XML ファイルです。

以下は、前の図の ObjectGrid Network Deployment を構成する ObjectGrid クラスター XML です。

```
<?xml version="1.0" encoding="UTF-8" ?>
<clusterConfig xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="cluster1">
<!– single server ->
<serverDefinition name="server1" host="localhost" clientAccessPort="12053"
peerAccessPort="12500" />
</cluster>
<objectGridBinding ref="Red">
<mapSet name="RedMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstRedMap" />
<map ref="SecondRedMap" />
</mapSet>
</objectGridBinding>
<objectGridBinding ref="Purple">
<mapSet name="PurpleMapSet" partitionSetRef="ColorMapsPartitioningSet">
<map ref="FirstPurpleMap" />
<map ref="SecondPurpleMap" />
<map ref="ThirdPurpleMap" />
</mapSet>
</objectGridBinding>
<partitionSet name="ColorMapsParitioningSet">
<partition name="partition1" replicationGroupRef="ColorMapsReplicationGroup" />
</partitionSet>
<replicationGroup name="ColorMapsReplicationGroup">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

この構成は、server1 サーバーを含む単一のクラスター「cluster1」を記述します。server1 サーバーは、2 つの ObjectGrid、「Red」および「Purple」をホストします。構成ファイルは、分割化および複製の情報を同様に指定します。ObjectGrid での ObjectGrid クライアント/サーバーのサポートは、プログラマーが、ObjectGrid クラスターで定義された 1 つのサーバーに接続する必要があります。接続処理中に、ObjectGrid および ObjectGrid クラスターの構成は、動的にクライアントにダウンロードされます。これにより、使用するためのクライアントの準備、およびクライアント側の構成内容を管理する必要性が大幅に簡素化されます。「接続」操作を実行する ObjectGrid 以外のクライアントの場合、ローカル JVM および実際に ObjectGrid クラスター内でホストされている JVM に有効範囲を設定した ObjectGrid を使用するプログラミング API および概念は通常同じです。

ObjectGrid クラスターの初期化

ObjectGrid に提供されているコマンド行ツールを使用して、クラスター内で ObjectGrid サーバーを開始することができます。ObjectGrid アプリケーションは、ObjectGrid クライアントを含むことができ、その他の Java API ライブラリーをアプリケーション開発フレームワークに組み込まれるように統合することができます。ただし、両方の場合で ObjectGrid の使用法を初期化する必要があります。

ローカル Java 仮想マシン (JVM) の使用シナリオ、または分散 ObjectGrid クラスターで作業するには、管理可能なアプローチを介してブートストラップ用の有効な構成を取得する必要があります。ObjectGrid クライアントおよび ObjectGrid クラスター・サーバーは、同一の構成を使用する必要があります。プログラマーとして、非常に単純な構成 (JVM の単一の Java アプリケーション内で結合されたもの) で開始することができます。

次に、マルチクライアント、同時ユーザー・テストの開始を準備する場合、第 1 の単一サーバー ObjectGrid クラスターを作成します。初期のクライアント/サーバー・ベースのテストが完了したら、管理スタッフとの作業が可能になり、複製およびその他の高可用性ソリューションの要件を試験することができます。開発要件におけるこれらの通常の進行には、より詳細な構成ファイルが必要です。

説明した各開発ステージのこれらの各高度機能を使用可能にする構成ファイルの変更は、比較的緩やかなものですが、ソリューション開発の各ステージでは、異なるバージョンの構成ファイルが必要です。すなわち、変更は互いに行われます。ソリューションを開発するデータ量が単一システムを圧迫しない場合、または開発目的のために人為的に制約できる場合、単一マシン上で複製されたソリューションの単体テストを行うことができます。

XML を使用した ObjectGrid 構成

1 つ以上のクライアントおよび 1 つ以上の ObjectGrid サーバーのある分散 ObjectGrid 構成では、XML 構成が必要です。ベースの ObjectGrid XML 構成ファイルに加えて、 ObjectGrid クラスター XML 記述を作成する必要があります。

単一の ObjectGrid XML 構成記述および ObjectGrid クラスター XML 構成記述は、単一の ObjectGrid クラスターのクライアントおよびサーバーに、期待どおりに機能するために必要な情報を提供します。環境内では任意の数の ObjectGrid クラスターを持つことができますが、クラスター固有の ObjectGrid クラスター XML 文書は、特定のクラスターを記述する必要があります。

ObjectGrid の開始に必要な構成ファイルは、通常の URL アプローチを介して獲得できます。例えば、クライアントおよびサーバーは、物理ファイルまたは HTTP URL を使用して XML ファイルを獲得できます。

次図で示す分散 ObjectGrid 環境内では、ファイル URL は複雑な場合があるが、ファイル・システム上でシンプルなファイルであるため、ObjectGrid サーバーの初期設定を、コマンド行を使用して構成し URL を介して構成を取得できます。ただし、より良い方法は、クラスター内で既に作動可能な他のサーバーから、同じ ObjectGrid クラスター内の後続のサーバーをブートストラッピングして開始することです。この方法は、管理者が ObjectGrid クライアントまたはサーバーをホストする各マシン上で構成ファイルを追跡する必要がないため、はるかに管理が容易です。さらに、ブートストラッピングにより開始されるサーバーは、XML 構成エラーが削減されており、XML が既に正常に処理されたことが保証されます。

ブートストラッピング

ObjectGrid サーバーのブートストラップ

次の図では、同じ ObjectGrid 構成をホストする通常の ObjectGrid クラスター環境のブートストラッピングを表していますが、より複雑な複製クラスター構成も掲載しています。この場合、最初のサーバーは、HTTP URL を介してブートストラップし、第 2、第 3 のサーバーは最初のサーバーから開始します。また、第 2 または第 3 のサーバーは、最初のサーバーと同じ URL からも開始できます。

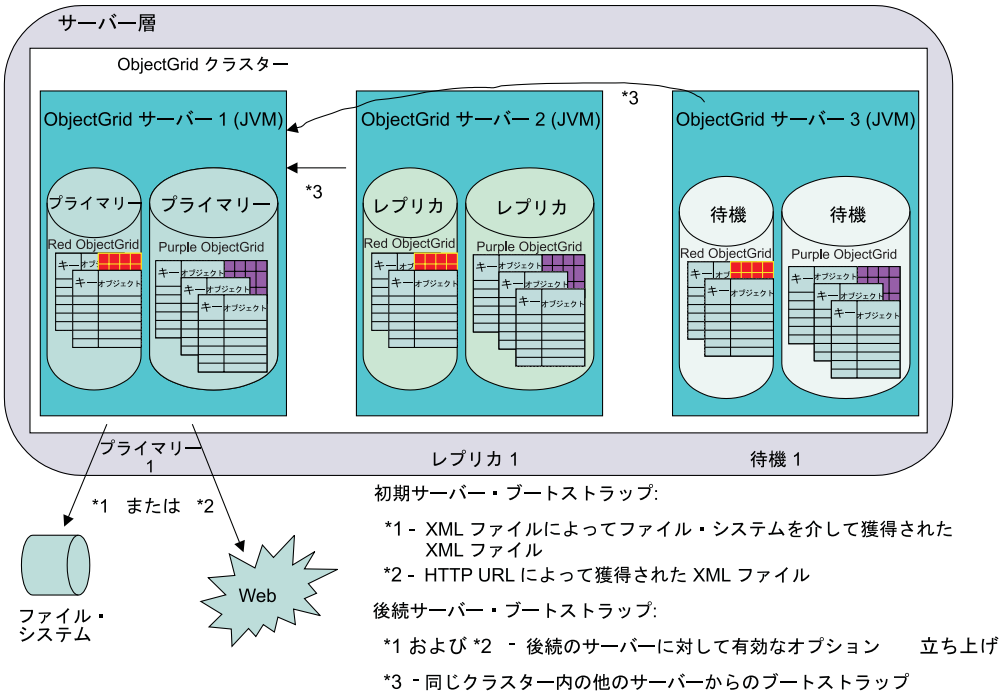


図 3. XML ファイル構成を介した、または既存のサーバーからの初期サーバー・ブートストラップ

前の図で示したように、cluster1 クラスターの server1 サーバーがブートストラップする初期サーバーです。server1 サーバーは、ファイル・システム上の XML ファイルを介して、またはローカル・ファイル、リモート HTTP サーバー、またはその他の有効な URL オプションへの URL を介してブートストラップできます。server2 サーバーおよび server3 サーバーは、これらの方法で、または構成ブートストラップ・ホストとして server1 サーバーをターゲットにすることで開始できます。通常、後続のサーバーを他のサーバーからブートストラッピングすると、クラスター・メンバー間での構成の一貫性が確保されます。

この特定のシナリオでは、server1 サーバーが失敗し、server2 および server3 が作動可能の場合、server1 は、server2 または server3 から、または再度ファイルまたは URL を介してブートストラップできます。ブートストラッピングおよび特定の構成オプションについて詳しくは、103 ページの『ObjectGrid クライアント接続 API』を参照してください。

ObjectGrid クライアントのブートストラップ

ObjectGrid クラスター・サーバー・メンバー・サービスを使用する ObjectGrid クライアントは、クラスター内の ObjectGrid サーバーの 1 つからブートストラップする必要があります。各クライアントは、クラスターのアクティブなメンバーに「接続」できます。管理者は、特定のサーバーを構成してこのサービスを実行できます。大規模なクライアントのデプロイメントの場合、構成済み ObjectGrid クラスター・サーバーの唯一の目的は、クライアント・ブートストラップ・サポートの提供です。この方法は、クライアント数が多く、接続と切断を頻繁に行う場合に便利です。クライアントが「接続」すると、クラスター構成で定義された ObjectGrids への分散参照を入手できます。詳しくは、95 ページの『ObjectGridManager インターフェース』を参照してください。

クライアントは、標準的な構成を ObjectGrid クラスターから獲得します。そのため、管理者は、クライアント・コミュニティーの XML を管理する必要はありません。ObjectGrid クライアントは、ObjectGrid クラスター・サーバーが立ち上げ用に実行するようにリモート URL を使用して、クライアント固有の特定の設定をオーバーライドできます。

分散 ObjectGrid 環境の ObjectGrid クライアント

ObjectGrid クライアントは、並行して複数の ObjectGrid クラスターと接続できます。Java 仮想マシン (JVM) 内の単一の Java アプリケーションは、同じリモート・クラスターに複数回接続することができます。また、このアプリケーションは、同時に別のリモート・クラスターに接続することもできます。この機能は、1 つ以上の ObjectGrid クラスターを介してエクスポートされた情報の多くのさまざまなリソースにアクセスするクライアントの機能を使用可能にするため重要です。

同じ ObjectGrid クライアントが同じ ObjectGrid サーバーに接触できる最初のケースは、クライアントがアプリケーション・サーバーになり、アプリケーション・サーバーからリモート ObjectGrid クラスターへの各接続が異なるセキュリティー・クリデンシャルを使用する、セキュアな環境で重要です。別の例は、単一の目的のため、さまざまな ObjectGrid クラスターからのデータの相関を必要とする ObjectGrid クライアントです。

次の図は、Web アプリケーションを介した企業の Web ベースのクライアント・ユーザーが、企業の 3 つの異なる部署からの報告書を生成するシナリオを表します。サーブレット・エンジンは、アプリケーション・サーバーの ObjectGrid クライアント機能を使用して、企業の各部署により管理されている 3 つの異なる ObjectGrid クラスターに接触します。多くの企業で、データを収集できます。ObjectGrid の主な目標は、簡単な方法で情報をさらに活用できるようにすることです。情報が外部化されたら、興味とセキュリティー・クリデンシャルを有する他のユーザーは、新たな方法で情報を獲得および使用することができます。データは、読み取り専用モードで提供できます。必要に応じて読み取り/書き込み更新のシナリオもあります。

このシナリオでは、データをセキュアな方法で獲得できます。このシナリオの ObjectGrid キャッシングは、企業の各部署内で共通の方針に基づいた方法でフレキシブルなデータ共有を可能にするだけでなく、多くの Java 開発者がよく使用する非

常にセキュアで、シンプルで、クリーンなプログラミング・モデルを介して獲得した情報に対する部署間のデータ・アクセスも可能にします。

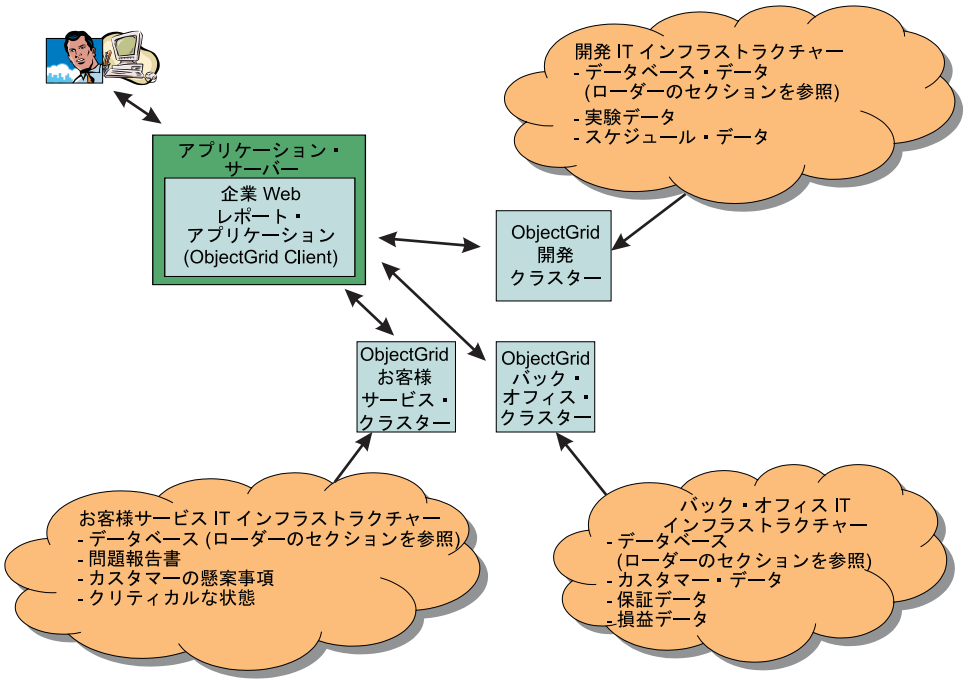


図 4. Web ベースのクライアント・ユーザーは、企業の異なる 3 つの部署からの報告書を生成します。

ObjectGrid クラスタリングの概念

分散 ObjectGrid という用語には、1 つ以上の ObjectGrid クラスターと対話できるクライアントという概念が含まれます。ObjectGrid クラスターは、1 対多の ObjectGrid サーバーにより構成されます。

ObjectGrid クライアント

ObjectGrid クライアントは、2 つの方法で考えることができます。クライアントを、ObjectGrid API を使用して ObjectGrid クラスターに接続する Java 仮想マシン (JVM) と考え、そのクラスターに対して Java Map 操作を実行することができます。クライアントについてのもう 1 つの、より公式な考え方は、同じ JVM 内の複数のクライアントの概念を考慮することです。提供された ObjectGrid 機能を完全に使用する場合、同じ JVM 内で複数のクライアントを使用できます。

プログラマーが JVM 内で ObjectGrid クライアント接続操作を実行するときはいつでも、クラスター・コンテキストが戻されます。このコンテキストは、実際に 1 つのクライアントのインスタンスです。目に見えないところで、非同期スレッドは、コンテキストごとにキャッシングのさまざまな局面を処理しています。各コンテキストに対して、ObjectGridManager を使用して、特定のリモート ObjectGrid クラスター・インスタンスでホストされている ObjectGrids を獲得できます。そのため通常は、同じ JVM 内の 3 つのリモート・クラスターに接続する場合、同じ JVM に 3 つのクライアント・ソリューションをインプリメントします。

このシナリオの重要な考慮事項は次のとおりです。単一トランザクション・セッションは、同じクラスター内のマップ・セットに及びません。同じ、または異なる ObjectGrid クラスターに接続された異なるクライアント間で単一トランザクションを持つことはできません。ただし、情報の格納場所を統合しようとするユーザーの場合、ユーザーは、トランザクションを使用して各リモート ObjectGrid クラスターから情報を引き出すこと、統合レポートを印刷したり情報を結合して ObjectGrid を介して別の ObjectGrid クラスターにトランザクション・データを送信すること、また、カスタマー固有の方法で個別の ObjectGrid クラスターを簡単に更新することができます。これは主に、ObjectGrid は、個別のトランザクション・マネージャーが通常提供する 2 局面トランザクション・サポートではなく、単一局面のトランザクション・サポートを提供するためです。このトピックの詳細については、39 ページの『ObjectGrid トランザクション区分』を参照してください。

複製

同じ ObjectGrid クラスター内にある ObjectGrid サーバー間で複製が可能です。複製を使用すると、ユーザーが必要とする特定の情報を持つプライマリーの ObjectGrid サーバーに障害が発生したり、保守のためにシャットダウンしている場合に、障害からより迅速にリカバリーすることができます。次の図では、Red ObjectGrid および Purple ObjectGrid は、2 つの異なる ObjectGrid 複製グループ・メンバーに入っています。ObjectGrid では、各 MapSet、ObjectGrid のサブセットは、1 つの単位として複製できます。PartitionSet は、この規則の例外です。このことについては、以下のセクションで説明します。記述された単一サーバー構成は、次の図で変更され、複製を記述します。

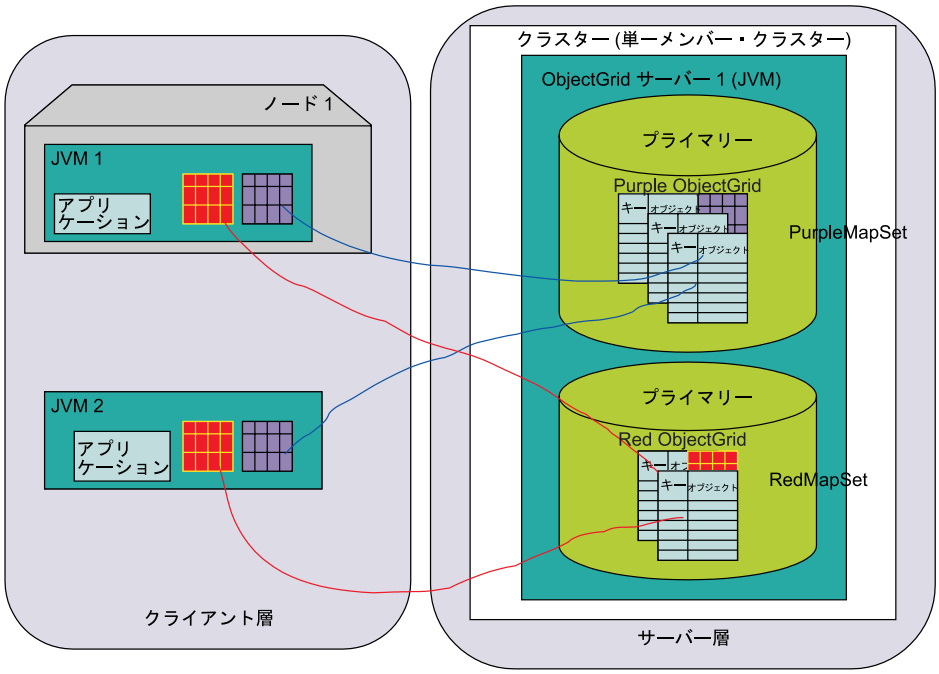


図 5. 分散 ObjectGrid の単一サーバーのトポロジー (2 つの MapSet)

図では、アプリケーション・サーバーをクライアント・アプリケーションおよびスタンドアロンの Java アプリケーションとして示しています。両方のクライアント

は、2 つの ObjectGrid インスタンス (単一サーバー ObjectGrid クラスター内の Red および Purple インスタンス) へのアクセスを必要とします。これらのインスタンスのそれぞれは、実際に複製グループ・メンバー内に含まれています。複製グループ・メンバーは、主要な概念であり、ObjectGrid トランザクション区分の境界です。トランザクションは、単一の複製グループ・メンバーに対しての変更のみをコミットできます。

ObjectGrid クラスターでは、Java クライアントは、ObjectGrid トランザクション (セッション) を開始することができ、単一の複製グループ・メンバー内でデータを更新することができます。各複製グループ・メンバーは、同期または非同期で、要件に完全に依存するわけではなく、単位として複製できます。各 ObjectGrid のクライアント要求は、ObjectGrid クラスター・サーバー内の特定の複製グループ・メンバーにルートされます。要求を受信中の複製グループ・メンバー内の ObjectGrid インスタンスは、要求を処理し、クライアントに結果を戻します。*同期* 複製の場合、各要求は、クライアントに戻される前に、レプリカ (次図の ObjectGrid サーバー 2) に送信され、レプリカ複製グループ・メンバーが適正に更新を適用していることを確認し、次にクライアントに結果を戻します。*非同期* モードでは、ObjectGrid クライアントは、変更を適用することができ、ObjectGrid サーバーのプライマリーの複製グループ・メンバーがクライアントに結果を戻し、変更が受信されたことおよび適正に適用されたことをレプリカが確認するのを待機しません。非同期モードでは、更新はプライマリーの複製グループ・メンバー上でトランザクションが正常にコミットされた後、リモート・サーバーのレプリカ複製グループ・メンバーに送信されます。

次の図は、ブートストラップ例の別バージョンです。この場合、3 つのサーバーは、それぞれユーザーが対話しようとする 2 つの ObjectGrid インスタンスの複製で固有の役割を持っています。ObjectGrid クラスターは、3 つのサーバーにより構成され、それぞれが 2 つの複製グループ・メンバーをホストします。server1 サーバーは、2 つのプライマリーをホストし、 server2 サーバーは、2 つのレプリカをホストし、server 3 サーバーは、2 つの待機をホストします。

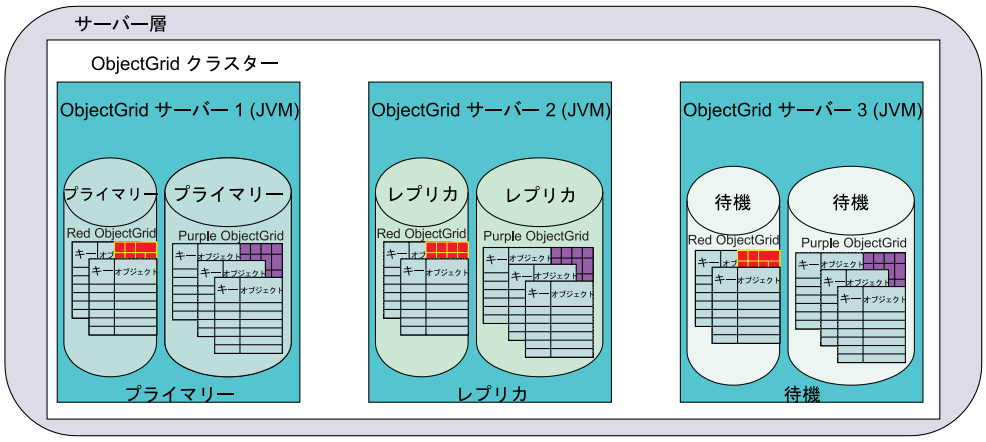


図 6. 基本構成例の複製

30 ページの『高可用性の概要』では、これらの概念を説明していますが、理解しておくべき主な概念は、単一サーバーから前の図で説明した 3 つのサーバーが複製されたソリューションに移動するのに必要な構成の違いです。

マルチサーバー複製構成の概要

次の構成は、Red および Purple の ObjectGrid インスタンスの基本の ObjectGrid 構成です。

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

この構成を分散 ObjectGrid クラスターに変換するには、追加の構成ファイル、Cluster XML ファイルが必要です。単一サーバー上で、Red および Purple Object のインスタンスの元の構成を変換するには、次の例で示す追加のみが必要です。具体的には、2 つのサーバー参照のみが追加されます。複製グループは既に初期構成ファイルから示されています。これは、次の例で説明するように ColorMapsReplicationGroup 複製グループに対して相互参照されています。

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

前の例では、両方の MapSet (以下に説明) は、ColorMapsReplicationGroup ReplicationGroup を参照します。これは、複製グループに含めるサーバーを定義します。お客様の要件を満たすように各 MapSet が異なる順序の同じサーバー、または異なるサーバーを持つようにして、別の ReplicationGroup を含むように構成ファイルを展開できる場合があります。ObjectGrid クラスター構成は、スタンザの再使用をサポートします。デフォルトでは、 MapSet 複製属性は設定されておらず、複製グループには複数のサーバーがあるため、複製は使用可能でありモードは非同期です。

高可用性の概要

複製により、ObjectGrid クラスター内で高可用性が使用可能になります。

複製および高可用性を理解するには、ObjectGrid 複製グループ・メンバー・タイプを理解する必要があります。ObjectGrid がサポートする複製グループ・メンバー・タイプには、*プライマリー*、*レプリカ*、および*待機*があります。これらのタイプのそれぞれには、高可用性構成における特定の役割があります。

ObjectGrid 複製グループ・メンバーのタイプ

プライマリー複製グループ・メンバー

プライマリー複製グループ・メンバーは、使用されているデータのクライアントの最新のビューを保持します。データが更新されると、データはレプリカに伝搬されます。プライマリーは、ObjectGrid ローダー・インターフェースを介して接続データベースと通信し、コミットを同期的、非同期的、または複製構成に完全に依存するわけではなく伝搬するインスタンスです。

レプリカ複製グループ・メンバー

レプリカ複製グループ・メンバーは、プライマリーから伝搬されたデータのバージョンを保持します。プライマリーを構成して、さまざまな方法で変更を送信することができます。複製グループは、プライマリーおよびレプリカを保持するものとしてリストされた 2 つ以上のサーバーを保持する必要があります。そうではない場合、複製は使用可能になりません。

待機複製グループ・メンバー

待機複製グループ・メンバーは、プライマリーに変更が加えられた場合にレプリカと異なり更新を受信しません。これはシンプルに構成されており、プライマリーまたはレプリカが失敗した場合に更新を受信するようになっています。プライマリーが失敗した場合、レプリカは新規のプライマリーになり、待機はレプリカに変換される必要があります。

高可用性のシナリオ

通常、複製により、ObjectGrid クラスター内で高可用性が使用可能になります。次の 2 つの図では、プライマリーが失敗した場合のシナリオおよびリカバリーを示します。プライマリー複製グループ・メンバーが複製され、失敗が発生すると、レプリカの 1 つが選択され、新規のプライマリーになります。このシナリオでは、1 つのレプリカが存在しています。

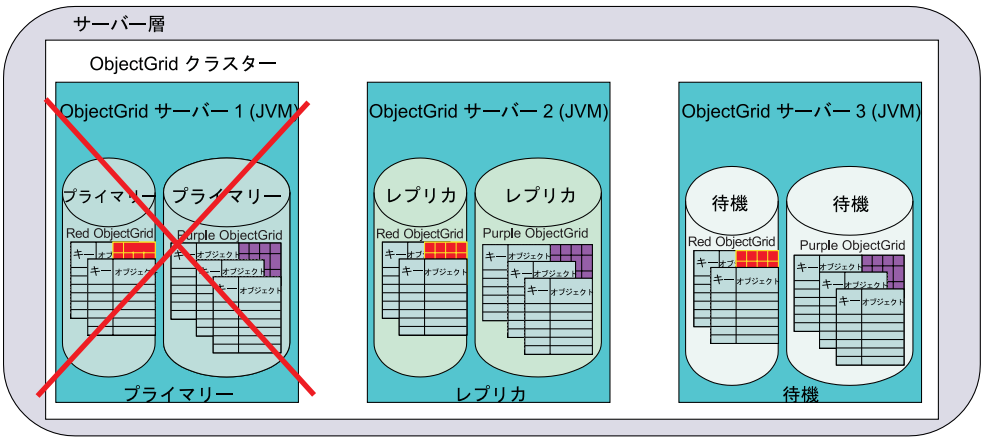


図 7. ObjectGrid 高可用性のシナリオ

失敗が検出されると、プライマリーは使用不可になります。レプリカは、プライマリーになります。待機が存在する場合、次図のリカバリー例と同様にレプリカになります。

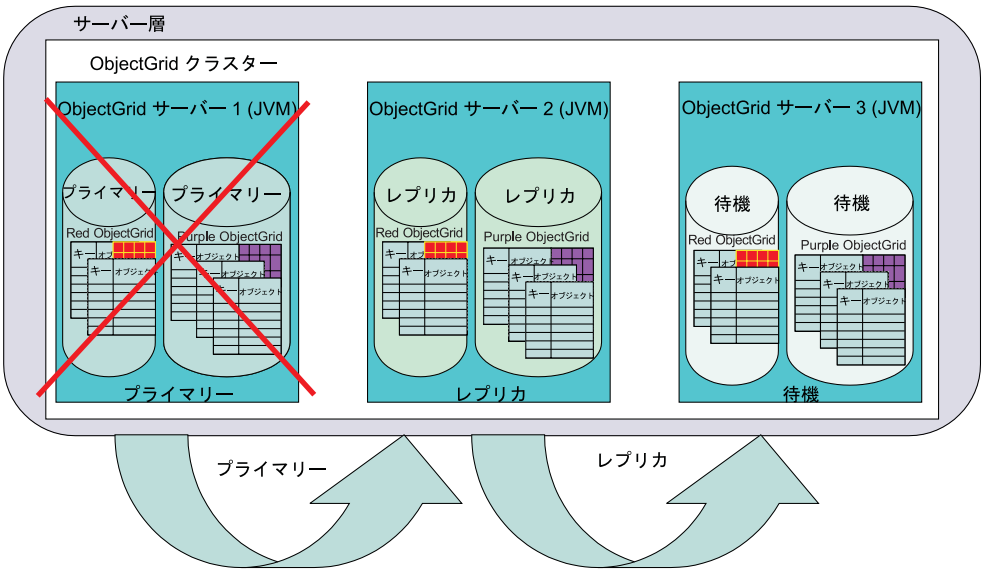


図 8. ObjectGrid のフェイルオーバー

ObjectGrid クライアントは、影響を受けたいずれかのサーバーへの次の接続の間に、この調整を認識します。失敗したサーバーに接触するクライアントは、ランタイム構成を使用し、動的にクラスター内の別のサーバーを試すことができます。クライアントは、構成内の次のサーバーに接触します。サーバーがアップしており、操作可能でない場合、クライアントはタイムアウト期間を待機します。クライアントは、複製された複製グループ・メンバーが失敗からリカバリーしていることを前提とします。時間が経過すると、クライアントは再試行し、現在 2 つのメンバーを保持している複製グループが操作可能になり、新規のルーティング・テーブルがクライアントに提供されます。ルーティング・テーブルは、現在のプライマリーが配置されている場所、そのレプリカの場所、および現在待機しているこのグループの複製グループ・メンバーを記述します。

ObjectGrid クラスタリング構成のセット

ObjectGrid クラスタリングを構成する場合、ObjectGrid を MapSet に分離することができます。この分離は、ObjectGrid に多くのマップを含めることができるため重要です。MapSet は、PartitionSet により分割でき、ReplicationGroup により複製できます。これらの構成オプションのそれぞれは、ObjectGrid サーバーの開始中に作成される複製グループ・メンバーの数に影響を与えます。セットの各タイプの概要は、各タイプの役割の説明に役立ちます。

ObjectGrid MapSet

各 ObjectGrid のマップは、異なる用法および可用性の要件を持つことがありますが、通常のアプリケーションの用法と相関しています。例えば、あるマップは、読み取り専用でプリロード完了後は変更できませんが、別のマップは、スケーラビリティー目的のため読み取り/書き込みが可能で分割されている場合があります。この場合、各マップは固有の MapSet 内に含まれます。前の例では、PurpleMapSet および RedMapSet は、指定された ObjectGrid それぞれのすべてのマップを保持します。これは最もシンプルなオプションです。

MapSet は、ObjectGrid サーバー間で複製が可能な単位であり、分割されていない複製グループ・メンバーに相関しています。PartitionSet を介して MapSet に関連した各複製グループ・サーバーは、必要に応じて複製グループ・メンバーをホストして、要求された構成をサポートします。複製グループ・メンバーは、ObjectGrid クラスター内の固有のエンドポイントであり、構成内で指定された MapSet が指示するすべてのマップをホストします。

例えば、前の図では、server1 サーバーはプライマリーを持っており、server2 サーバーはレプリカを持っています。 server3 サーバーは、複製リカバリー・シナリオに応じてレプリカまたはプライマリーにできる待機ユニットを持っています。 Mapset は、3 つのサーバーを記述する PartitionSet に相関しています。そのため、MapSet は両方とも、それぞれが異なる数のマップを持っていたとしても、PartitionSet および ReplicationGroup のスタンザが同じであるため、同じサーバーにマップされます。

ObjectGrid PartitionSet

通常は、MapSet および列挙された複製グループ・サーバーにより、ObjectGrid クラスター内の特定の MapSet をサポートするサーバーの数が決定されます。複製グループ・メンバーは、ホストする各複製グループ・サーバー内で作成されます。ただし、分割化は、MapSet の複製グループ・メンバーの数に影響を与える場合があります。分割化は、MapSet および ReplicationGroup 間の PartitionSet 関係を介して構成ファイル内で管理されます。

PartitionSet は、1 つの Java 仮想マシン (JVM) が単一のプライマリー複製グループ・メンバー内に MapSet 全体を保持しないように MapSet を部分に分割します。例えば、1,000,000 キーのデータベースがあるとします。各キーが参照する各オブジェクトが大規模である場合、単一の 32 ビット JVM が単一のプライマリー、レプリカ、または待機複製グループ・メンバーのメモリー内にマップを保持できなくなることが十分に考えられます。しかし、大規模なデータ・セットは頻繁に必要とされます。Purple ObjectGrid の最初のマップ・インスタンスを PurpleFirstMapMap1、PurpleFirstMap2、PurpleFirstMapN マップに手動で分割するな

ど、データを自ら人為的に分割することを避け、それぞれを異なる Mapset に配置するため、ObjectGrid はこの作業の大部分を実行できます。

この資料の後半で、PartitionKey の概念が定義されています。これにより、ObjectGrid が呼び出して、挿入中の特定のエントリー用のキー・ハッシュ・コードを決定する API が効果的に得られます。MapSet に 2 つの区画がある場合、2 つの複製グループ・メンバーがその MapSet に作成されます。また、データが大規模であるため、これらの複製グループ・メンバーは、異なるサーバーに配置されることがよくあります。開発者については、これらのメンバーを、初期プロトタイピング中は同じサーバーに配置する必要がある場合があります。各複製グループ・メンバーは、使用可能な分割された複製グループ・メンバー間で同じ値にハッシュするキーを保持します。簡単な例として、MapSet が 0、1、および 2 の 3 つに分割されたとします。3 つのプライマリー複製グループ・メンバーが確立され、そのうちの 1 つが、指定された値の係数にプライマリー複製グループ・メンバーの数をハッシュするすべてのキーを保持します。キーのハッシュ値が 7 とすると、7 の係数 3 が 1 です。そのため、1 の分割インデックスを持つプライマリー複製グループ・メンバーがインスタンスを含みます。

PartitionSet の例

次の図では、purple マップの 2 つの区画への分離について説明します。マップの各キーは、整数にハッシュされ、適切な複製グループ・メンバーへの挿入で特定の区画セットに割り当てられます。各区画は、異なる JVM に存在する場合があり、ObjectGrid によりプログラマーは通常 PurpleFirstMap を分割されていない論理的な単一のマップ・インスタンスとして扱うため、別々の複製グループ・メンバーにあります。ObjectGrid クライアントおよびサーバーのサポートは、複製グループ・メンバー間で要求のルーティングを適正に管理します。

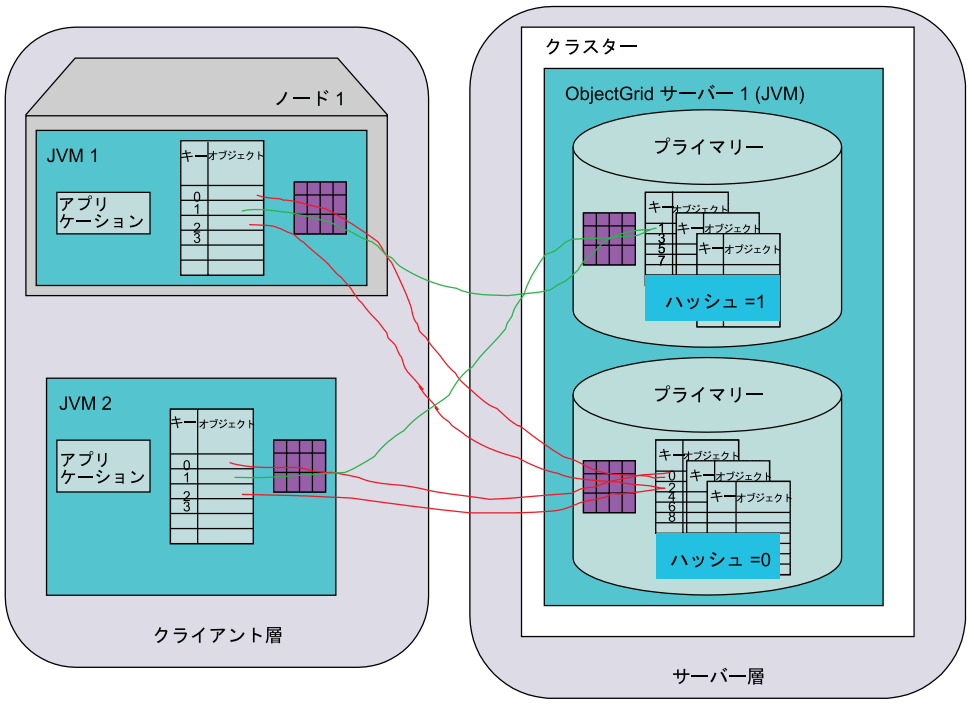


図 9. 分散 ObjectGrid のトポロジー: 分割化された単一のサーバー

区分セットの構成は、変更され次のようにこの構成を使用可能にします。

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

構成例は、複製され、分割された Purple ObjectGrid の確立方法を反映しています。次の場合では、3 つのサーバーが存在し、プライマリー複製グループ・メンバーのそれぞれは、同じ方法で 3 つのサーバーのセットにマップされています。別の ReplicationGroup スタンザが使用されている場合、これらは、別の方法で簡単にマップされることがあります。

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

再度、前の例での注意点として、構成ファイルに対する小さな変更のみでも新しいレベルの機能が発生し、結果を実現するためにアプリケーションの変更が必要となります。次の図では、ObjectGrid クラスターのビュー、および複製グループ・メンバーが前の分割構成をサポートするための配置方法について説明します。

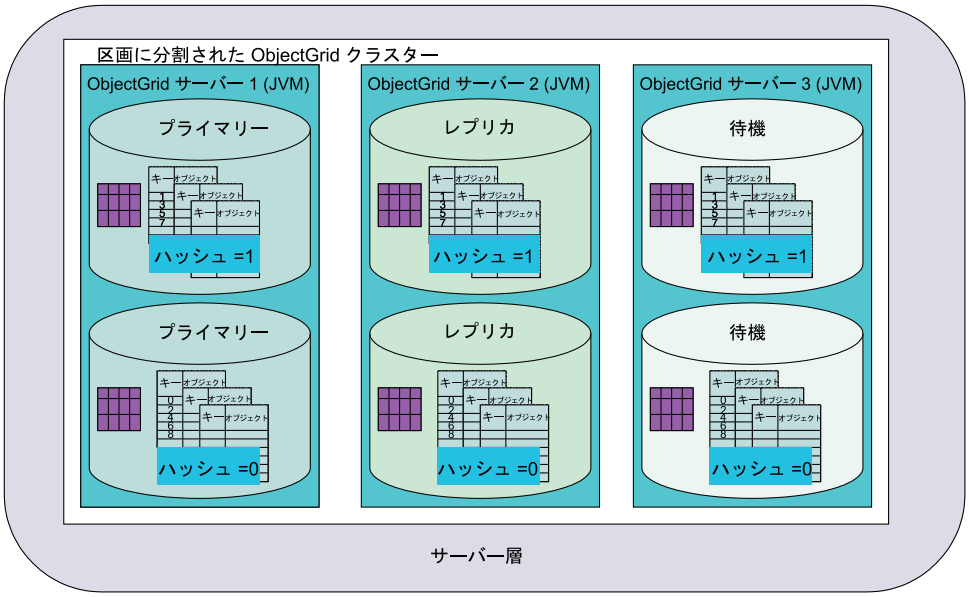


図 10. 分散 ObjectGrid のトポロジー: 分割化されたマルチサーバー

以下は、PartitionSet の説明を完成するための、前の例の別のバリエーションです。この事例では、第 2 の ReplicationGroup が作成されました。各 PartitionSet は、現在、別個のサーバー上の独自の複製グループに含まれています。

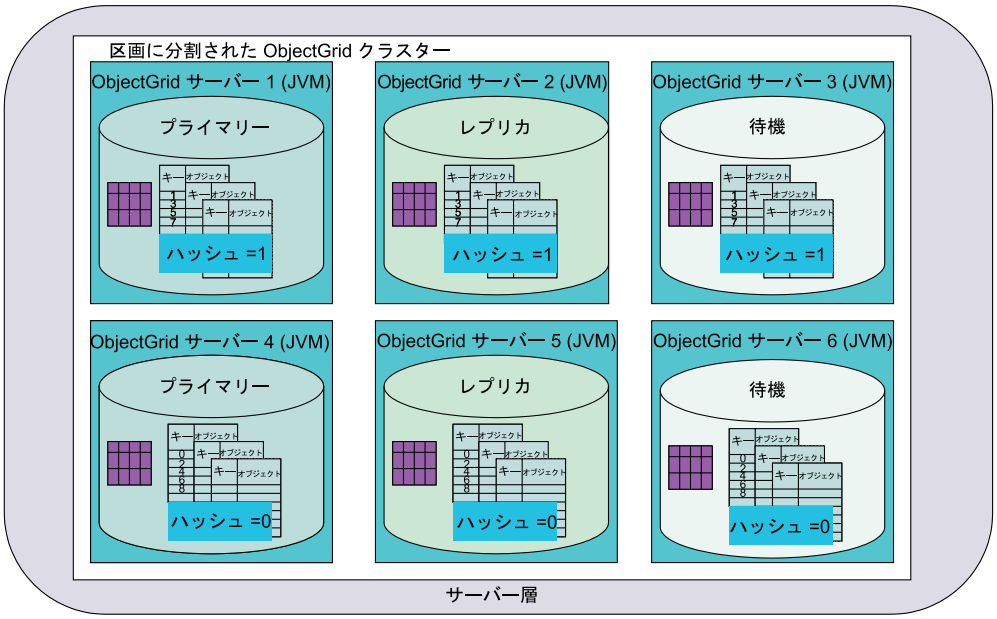


図 11. 分散 ObjectGrid のトポロジー: 分割化されたマルチサーバー

このトポロジーの構成は、前の例に非常によく似ています。変更点には、3 つの追加のサーバー・インスタンス、および第 2 の PartitionSet により参照される新規の ReplicationGroup があります。

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

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

この構成では、結果的に同じ数の複製グループ・メンバーになりますが、第 2 の複製グループが独自の XML スタンザを介して特別に構成されることが強制され、同時に、それぞれのインスタンスは、前の例のような連結ではなく異なるサーバー・インスタンスに帰することになります。

この構成の利点は、2 つの複製グループ・メンバーが独自の JVM インスタンスに入っているため、この場合、各区画が 1.5 ギガバイト以上のデータ (合計で 3 ギガバイト以上) を保持できるという点です。これにより、アドレス可能なメモリー・スペースの 32 ビットの JVM 2 ギガバイトを最大限に活用できます。

複数の ObjectGrid クラスターと接触する ObjectGrid クライアント

ObjectGrid は、特に、Java 仮想マシン (JVM) 間の分割サポートを拡張するだけではなく、通常の Java Map インターフェースを取得して情報が獲得できる範囲を拡張するために設計されました。単一のクライアントで多くの ObjectGrid クラスターに接触できます。

次のシナリオでは、サーバー層には 2 つの ObjectGrid クラスターが含まれています。 Java アプリケーション・クライアントの 1 つおよびアプリケーション・サーバーの 1 つが、複数のクラスターに接触する必要があります。これは強力な機能であり、高いスケーラビリティーを実現できます。

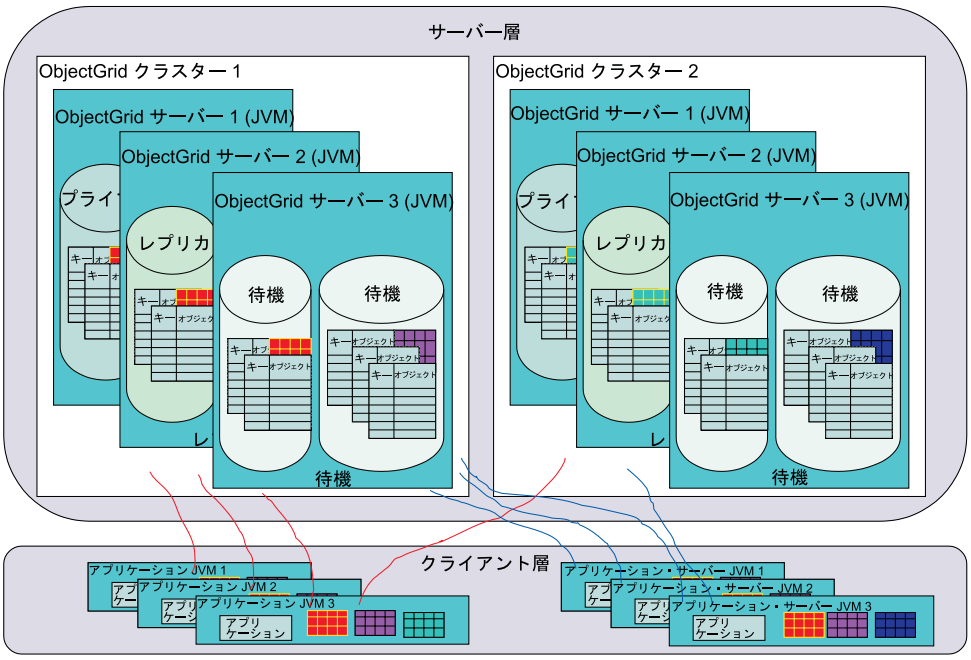


図 12. 複数の ObjectGrid クラスターと相互作用する ObjectGrid クライアント

ObjectGrid クラスターは、それぞれが多くの MapSet 構成および PartitionSet 構成を含む多くの ObjectGrid をサポート可能です。各 ObjectGrid クラスターは、1 つ以上の JVM (可能な限り多数) により作成できます。大規模なエンタープライズの場合、ObjectGrid クライアントが同時に 1 つだけでなく複数の ObjectGrid クラスターに接触できる機能は、非常に価値があります。

ただし、注意すべき点として、ObjectGrid クライアントは、単一のトランザクション内で複数のクラスターからのデータを参照できません。ObjectGrid クライアント・アプリケーションは、 1 つ以上のトランザクションを実行して、Java 仮想マシンのインスタンスにデータをキャッシュし、取得した情報を Java オブジェクトとして相関させる必要があります。また、情報に基づいた更新も、各クラスターの単一のトランザクション内に含める必要があります。この問題について詳しくは、39 ページの『ObjectGrid トランザクション区分』を参照してください。

ObjectGrid クライアント・ニア・キャッシング・サポート

ObjectGrid クライアントは、実際にはキャッシング層です。リモート・サーバーから以前獲得したデータが不整合でないかどうかがわかっている場合は、ローカルのキャッシング機能を利用するようアプリケーションを設計することができます。例えば、データが ObjectGrid クラスターで更新されたが、このクライアントでは更新されなかった場合 (新規 get(...) 要求が行われます)、クライアントでは、ObjectGrid クラスターと整合する (すべてのアプリケーションでこの状態が必要となるわけではありません) ようにローカル・キャッシュを更新する必要があります。

get 操作が実行された後、同じキーとオブジェクト・ペアに対して次回 get 操作を要求すると、ObjectGrid クライアントによって、データが既に検索されていることが検出され、ネットワークを介して ObjectGrid クラスターに移動してデータにアクセスするのではなく、「Java 仮想マシン (JVM)」にキャッシュされているバージョンが使用されます。ネットワークを介してデータが検索されると、その後、ローカ

ル・エントリーが手動または通常構成された Evictor によって除去されるまで、ローカル・キャッシュからデータが提供され続けます。

例えば、サーバー上のデータが 6 時間ごとに更新されることについて理解していると、ローカル・キャッシュ、またはニア・キャッシュ・クライアントが更新される時期を制御することができます。ユーザーは、ニア・キャッシュ・エントリーを無効化してから、get 要求を発行します。get 要求はサーバーにアクセスし、すべてが正常に行われている場合、情報を獲得します。オブジェクトがイメージ・ファイルであるとします。更新時期後、初めてイメージがダウンロードされると、後続の各要求では、サーバーへのイメージ取得のためのリモート・プロシージャー・コールは実行されません。

クライアントでは、要求されたロック・ストラテジーを実行するために、ObjectGrid クラスター・データをロックする必要があることがあるため、ペシミスティック・モードではニア・キャッシング・サポートは適用されません。削除する必要のあるニア・キャッシュ・エントリーを、ObjectGrid クラスターの情報ビューを変更することなくクリアまたは無効化する方法について詳しくは、beginNoWriteThrough() メソッドを参照してください。

ObjectGrid トランザクション区分

プログラマーとして念頭に置いておく必要のある主な概念に、トランザクション区分の概念があります。 ObjectGrid では、ObjectGrid クラスター内の複製グループ・メンバーに渡るトランザクション・コミット処理の 2 フェーズ・コミット・プロトコルがサポートされていません。単一の ObjectGrid クラスター・ベースのセッションでは、単一のプライマリー複製グループ・メンバーに読み取り/書き込み更新が適用される必要があります。複数の複製グループ・メンバーが関連する場合は、トランザクション・コミット処理中に更新をアトミックにすることができません。このことは、単一の ObjectGrid 内のすべてのマップに対するコミットが可能なローカルの ObjectGrid JAR プログラミング・モデルとは若干異なります。

注意する必要のある特殊なケースとして、複数の区画が定義されている PartitionSet があります。これらのシナリオでは、キー 1 はサーバー 1 上に、キー 2 をサーバー 2 上になどのように配置することができます。1 つのトランザクション内にキー 1 とキー 2 に対する更新がある場合、ObjectGrid セッションでは 2 つの複製グループ・メンバーに対してコミットを行うことができないため、これらの更新は失敗します。前述したとおり、2 つ以上の区画をサポートする PartitionSet は、アプリケーション内で慎重に使用する必要があります。トランザクション・シーケンスでは、複数のトランザクションからのデータが更新されないようにする必要があります。

データベースに対する ObjectGrid の関係

この概要は、他の Java クライアントとは異なる場所から初期化済みデータを獲得する ObjectGrid サーバーについて特に言及したものではありません。ObjectGrid が開始すると、ローダーは、MapSet の各マップに対して初期化されます。このローダーにより、Java Map の get または put の形式のユーザー要求を、必要に応じてデータベースから取得したり、データベースに書き込んだりすることができます。データベース操作は、Java Map のユーザーには見えません。また、これらのアプリケーションでは特別なコーディングは必要ありません。ただし、ローダーの機能は、エ

ンド・ユーザー・プログラマーがこの機能を使用する前に、プログラマーが開発し、ObjectGrid 内で使用するために構成する必要があります。詳しくは、 210 ページの『ローダー』を参照してください。

さらに、ObjectGrid クラスター初期化の後、データベースからプリロードするためにプリロードのサポートがあります。同様に分割プリロードがあり、指定された MapSet の特定の複製グループ・メンバー区画に正しいデータが確実に読み込まれるようにします。ユーザーは、ロックなしストラテジー、オプティミスティック・ロック・ストラテジー、ペシミスティック・ロック・ストラテジーなどのロック・ストラテジーを変更することで、読み取り/書き込み情報にアクセスし、更新することができます。読み取り専用データ・アクセスがサポートされています。複数の最適化が提供されるため、これが最も高速なモデルです。

第 4 章 ObjectGrid 解説: アプリケーション・プログラミング・モデル

このタスクを使用して、ObjectGrid アプリケーション・プログラミング・モデルについて学習します。

ObjectGrid アプリケーションを実行するために、環境を準備します。Java アーカイブ (JAR) ファイルの場所、Java 要件、および単純なファイルを実行して環境が正しくセットアップされていることを確認する方法について学習するには、1 ページの『第 1 章 サンプル・アプリケーションの実行による ObjectGrid の開始』を参照してください。

このタスクで使用するプログラミング環境を決定します。 Eclipse などの統合開発環境 (IDE) を使用することができますが、コマンド行 Java 環境も機能します。ObjectGrid について知識を深めたら、 ObjectGrid をエンタープライズ Bean およびサーブレットに取り込みます。解説の例では、特定の Java 環境を想定していないため、よく理解している任意の環境を使用することができます。

ほとんどの基本定義では、ObjectGrid はオブジェクトのキャッシュおよびメモリー内のリポジトリーです。オブジェクトを保管およびアクセスするための java.util.Map マップの使用は、 ObjectGrid の使用に似ています。同時に、ObjectGrid は、単なるキャッシュではありません。このタスクでさまざまな機能およびプラグインを検討することにより、 ObjectGrid は非常に拡張可能であり、柔軟であることが分かります。ObjectGrid は、単純なルックアサイド・キャッシュ、またはリソース・マネージャーによって支持される複雑なキャッシュとして使用することができます。

この解説の例は、完全なプログラムではありません。インポート、例外処理、および一部の変数は、すべての例で完全に宣言されているわけではありません。サンプルを使用して、独自のプログラムを書き込むことができます。

このタスクを使用して、 Java プログラムから ObjectGrid を使用します。

1. ObjectGrid API および例外を見つけます。 すべてのパブリック ObjectGrid API および例外は、`com.ibm.websphere.objectgrid` パッケージに含まれています。拡張システムまたは構成トピックの場合、`com.ibm.websphere.objectgrid.plugins` パッケージ内で追加 API および例外を参照します。ここでは、プラグインのインプリメンテーションが提供されており、 `com.ibm.websphere.objectgrid.plugins.builtins` パッケージ内にこれらのクラスを配置します。ObjectGrid セキュリティー機能の場合、`com.ibm.websphere.objectgrid.security`、`com.ibm.websphere.objectgrid.security.plugins`、および `com.ibm.websphere.objectgrid.security.plugins.builtins` など、名前に **security** があるパッケージを探します。

   このタスクは、`com.ibm.websphere.objectgrid` パッケージ内にある API に焦点を合わせます。 ObjectGrid の完全な JavaDoc は、`<install_root>/web/xd/apidocs` にあります。

```
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. ObjectGrid インスタンスを取得または作成します。 ObjectGridManagerFactory を使用して、 ObjectGridManager singleton インスタンスを取得します。次に、以下のステートメントを使用して、ObjectGrid インスタンスを作成します。

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

ObjectGridManager インターフェースは、 ObjectGrid インスタンスを作成、取得、および除去するためのいくつかのメソッドを持ちます。状態に対して変数を選択する場合は、 95 ページの『ObjectGridManager インターフェース』のトピックを参照してください。トレース設定は、 ObjectGridManager インターフェースを使用して設定することもできます。WebSphere Extended Deployment または WebSphere Application Server 内で稼働している場合、トレースは組み込み機能によって管理されるため、これらのメソッドは必要ありません。 WebSphere Application Server の外側で稼働している場合、これらのメソッドが役立ちます。これらのメソッドに関する完全な情報については、 102 ページの『ObjectGrid のトレース』のトピックを参照してください。

3. ObjectGrid を初期化します。
   a. `create` メソッドを使用して ObjectGrid に名前を設定しなかった場合は、その名前を設定します。
   b. 初期アプリケーションの BackingMap のデフォルト構成を使用して、BackingMaps を定義します。
   c. BackingMaps を定義した後、ObjectGrid を初期設定します。 ObjectGrid の初期化は、すべての構成が完了し、ObjectGrid の使用を開始することを示します。
   d. ObjectGrid を初期化した後、Session オブジェクトを取得します。詳しくは、 109 ページの『ObjectGrid インターフェース』および JavaDoc を参照してください。

このステップのガイダンスとして、以下の例を使用します。

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

4. セッションを使用して、トランザクション操作を管理します。 ObjectGrid キャッシュへのすべてのアクセスはトランザクションです。複数のアクセス、挿入、更新、およびキャッシュからの Objects の除去は、セッションと呼ばれる単一の作業単位内に含まれています。セッションの最後に、この作業単位内ですべての変更をコミットするか、または作業単位内のすべての変更をロールバックして忘れることができます。

キャッシュに対する単一アトミック操作のために自動コミットを使用することもできます。アクティブ・セッションのコンテキストがない場合、キャッシュ・コンテンツへの個々のアクセスは、自動的にコミットされる独自のセッションに囲まれます。

Session インターフェースに関するその他の重要な面は、 ObjectMap インターフェースを持つ BackingMap へのトランザクション・アクセスまたはハンドルを取得することです。 getMap メソッドを使用して、事前定義 BackingMap への ObjectMap ハンドルを作成することができます。挿入、更新、削除などの、キャッシュに対するすべての操作は、ObjectMap インスタンスを使用して完了します。詳しくは、 119 ページの『Session インターフェース』のトピックを参照してください。 以下の例を使用して、セッションを取得および管理します。

```
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. ObjectMap インターフェースを使用して、キャッシュにアクセスし、更新します。 ObjectMap インターフェースを参照するときに、キャッシュへのアクセスおよび更新用のいくつかのメソッドに注意してください。ObjectMap インターフェースは、マップのようなインターフェースとしてモデル化されます。ただし、チェックあり例外は、Eclipse などの IDE を持つ開発 ObjectGrid アプリケーションでの補助機能として導入されます。チェックあり例外なしで java.util.Map インターフェースを使用する場合、 getJavaMap メソッドを使用することができます。詳しくは、 123 ページの『ObjectMap および JavaMap インターフェース』を参照してください。

   明示的な insert および update メソッドは、はっきりしない PUT 操作を回避します。依然として put メソッドを使用できますが、明示的な insert および update メソッドを使用すると、意志がさらに明確に伝わります。 put メソッドの使用は、insert メソッドとしての先行 get 操作なしで put メソッドを定義することによって明確になります。先行 get 操作が PUT 操作の前に試行される場合、エントリーがキャッシュ内に存在するかどうかによって、PUT 操作は挿入または更新として処理されます。

   基本的な ObjectMap 操作 (get、put、insert、update、remove、touch、invalidate、および containsKey) を実行できます。さまざまな詳細および差異については、45 ページの『システム・プログラミング・モデルの概要』のトピックまたは ObjectMap API の資料を参照してください。 以下の例は、 ObjectMap を使用したキャッシュの変更を示します。

```
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);
```

6. 索引を使用して、キャッシュされたオブジェクトを検索します。 アプリケーションは、索引を使用して、特定の値または値の範囲によってオブジェクトを見つけることができます。アプリケーションで索引機能を使用するには、BackingMap マップに索引プラグインが構成されている必要があります。アプリケーションは、ObjectMap インターフェースの getIndex() メソッドから索引オブジェクトを取得して、それを正しい索引インターフェース (MapIndex インターフェース、MapRangeIndex インターフェース、カスタム索引インターフェースなど) にキャストする必要があります。

   現在のところ、索引付けフィーチャーは、ローカル・キャッシュでしかサポートされていません。索引付けフィーチャーは、分散キャッシュではサポートされていません。索引付け操作を分散キャッシュに対して行おうとすると、UnsupportedOperationException 例外が発生します。

   以下の例に、索引の使用方法を示します。

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

このセクションを読み終え、コード例で実験すると、必要な ObjectGrid プログラミング・モデルをより理解できます。

より具体的な情報については、 95 ページの『第 9 章 ObjectGrid アプリケーション・プログラミング・インターフェースの概要』を参照してください。

リモート ObjectGrid 入門

ここに簡略説明を挿入してください。最初の段落と要約に使用します。

41 ページの『第 4 章 ObjectGrid 解説: アプリケーション・プログラミング・モデル』では、「ローカル」またはアプリケーション内での ObjectGrid の使用方法を一般的に解説しています。そこでは、アプリケーションが ObjectGrid のインスタンスを作成し、そのインスタンスを使用しました。アプリケーションの Java 仮想マシン (JVM) が終了すると、ObjectGrid キャッシュも終了しました。リモート ObjectGrid は、その名前が示しているように、異なる JVM にある ObjectGrid へのアクセスを

許可します。複数のクライアントが、同じ API を透過的に使用して、リモートの ObjectGrid に接続し、ObjectGrid にアクセスすることができます。

1. 初めての場合は、以下のセクションを参照してください。
   - 19 ページの『第 3 章 ObjectGrid の概要』
   - 270 ページの『ObjectGrid 構成』
   - 103 ページの『ObjectGrid クライアント接続 API』
   - 85 ページの『第 8 章 コマンド行サポート』
2. サーバーを開始するには、ObjectGrid XML ファイルおよびクラスター XML ファイルを定義する必要があります。 285 ページの『分散 ObjectGrid 構成』を参照してください。このトピックでは、`university.xml` ファイルと `universityCluster.xml` ファイルを参照します。これらのファイルを例として使用して、サーバーを起動または開始するホストとポートを変更することができます。ObjectGrid サーバーの起動についての詳細は、85 ページの『第 8 章 コマンド行サポート』を参照してください。
3. サーバーが稼働している場合、クライアントはこの稼働中のサーバーに接続できます。 クライアントが接続して、ObjectGrid 操作を実行するための詳細な方法については、103 ページの『ObjectGrid クライアント接続 API』を参照してください。

システム・プログラミング・モデルの概要

システム・プログラミング・モデルは、ObjectGrid のいくつかの追加フィーチャーおよび拡張ポイントを提供します。

以下の図は、システム・プログラミング・モデルがいくつかの追加フィーチャーおよび拡張ポイントを提供する方法を示しています。

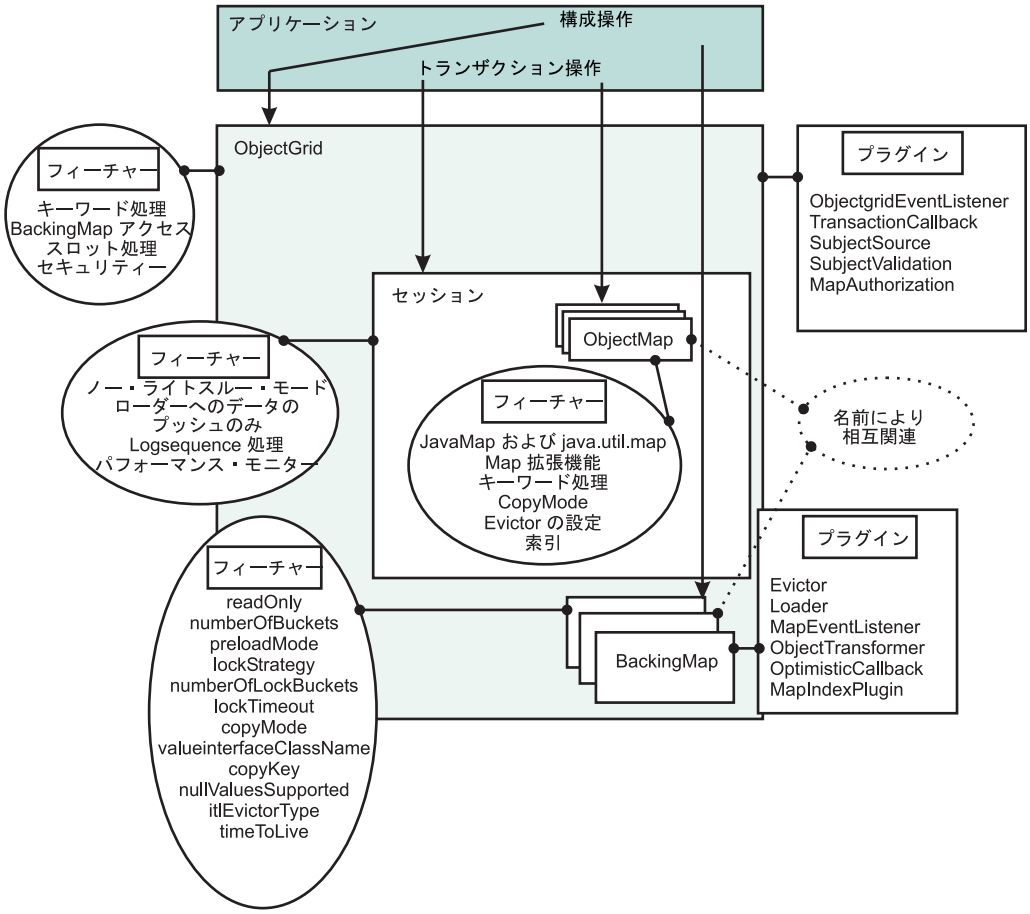


図 13. ObjectGrid の概要

ObjectGrid のプラグインは、ObjectGrid および BackingMap を含むプラグ可能な ObjectGrid コンポーネントに特定のタイプの機能を提供するコンポーネントです。これは、ObjectGrid、Session、BackingMap、ObjectMap など、ObjectGrid コンポーネントの特定の機能または特性を表すフィーチャーです。フィーチャーが機能を表す場合、特定のコンピューティング・オブジェクトを取得するために使用することができます。フィーチャーが特性である場合、ObjectGrid コンポーネントの振る舞いを調整するために使用できます。

以下の各セクションでは、前の図で示した機能および拡張機能の一部を説明しています。

- 47 ページの『システム・プログラミング・モデルの概要: ObjectGrid インターフェース・プラグ・ポイントおよびフィーチャー』

  ObjectGrid インターフェースには、 ObjectGrid との対話をさらに拡張するためのプラグ・ポイントおよびフィーチャーが幾つかあります。

- 50 ページの『システム・プログラミング・モデルの概要: BackingMap インターフェースのプラグ・ポイントとフィーチャー』

  BackingMap インターフェースには、 ObjectGrid による拡張可能な対話のためのいくつかのプラグ・ポイントおよびフィーチャー (オプション) があります。

- 58 ページの『システム・プログラミング・モデルの概要: Session インターフェース・フィーチャー』

  Session インターフェースには、ObjectGrid との対話をさらに拡張するためのフィーチャーが幾つかあります。このトピックの各セクションでは、このフィーチャーについて説明し、使用シナリオの一部の簡潔なコードの断片を提供します。
- 59 ページの『システム・プログラミング・モデルの概要: ObjectMap インターフェース・フィーチャー』

  ObjectMap インターフェースには、ObjectGrid との対話をさらに拡張するためのフィーチャーが幾つかあります。このトピックの各セクションでは、このフィーチャーについて説明し、使用シナリオの一部の簡潔なコードの断片を提供します。

個々のフィーチャーおよびプラグインの詳細情報については、 95 ページの『第 9 章 ObjectGrid アプリケーション・プログラミング・インターフェースの概要』を参照してください。

システム・プログラミング・モデルの概要: ObjectGrid インターフェース・プラグ・ポイントおよびフィーチャー

ObjectGrid インターフェースには、 ObjectGrid との対話をさらに拡張するためのプラグ・ポイントおよびフィーチャーが幾つかあります。

次の各セクションではフィーチャーを説明し、使用法のシナリオについて簡単なコードの断片を提供します。該当する場合は、XML の断片が提供されて代替の XML 構成を示します。詳しくは、 109 ページの『ObjectGrid インターフェース』および 270 ページの『ObjectGrid 構成』のトピックを参照してください。

キーワード処理

ObjectGrid インターフェースはキーワードを基にした柔軟な無効化機構を提供します。キーワード は、任意の直列化可能オブジェクトの非ヌル・インスタンスです。キーワードはどんな方法を使って BackingMap エントリーに関連付けてもかまいません。ほとんどのキーワード処理は ObjectMap レベルで行われますが、 1 つのキーワードを別のキーワードに関連付けてキーワードの階層ツリーを形成する場合は ObjectGrid レベルで行われます。

associateKeyword (親の java.io.Serializable、子の java.io.Serializable) メソッドは方向関係で 2 つのキーワードをリンクします。親が無効化されている場合、子も無効化されています。子を無効にしても親には影響しません。例えば、このメソッドを使用してアメリカ・マップ・エントリーの子としてニューヨーク・マップ・エントリーを追加します。そのためアメリカ・エントリーが無効化されると、ニューヨーク・エントリーもすべて無効化されます。以下はコードのサンプルです。

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
// associate several cities with "USA" keyword
objectGrid.associateKeyword("USA", "New York");
objectGrid.associateKeyword("USA", "Rochester");
objectGrid.associateKeyword("USA", "Raleigh");
:
// insert several entries with various keywords
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");
:
// invalidate all entries associated with "USA" keyword, leaving
// "key2" and "key6" entries
objectMap.invalidateUsingKeyword("USA", true);
:
```

詳しくは、 128 ページの『キーワード』を参照してください。

BackingMap アクセス

ObjectGrid は BackingMap オブジェクトへのアクセスを実行します。defineMap または getMap メソッドで BackingMap にアクセスできます。詳しくは、 114 ページの『BackingMap インターフェース』を参照してください。次の例は 2 つの BackingMap 参照を作成します。

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

スロット処理

トランザクション ID オブジェクト (TxID) またはデータベース接続オブジェクト (Connection) など、トランザクションの過程で使用されたオブジェクトを保管するスロットを予約することができます。保管されたこれらのオブジェクトは、 reserveSlot メソッドで提供された特定のインデックスで参照されます。スロット使用についての詳細は 210 ページの『ローダー』および 226 ページの『TransactionCallback プラグイン』のトピックを参照してください。以下のコードの断片はスロット処理の例です。

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
int index = objectGrid.reserveSlot
(com.ibm.websphere.objectgrid.TxID.SLOT_NAME);
:
// Use the index later when storing or retrieving objects from
//the TxID object ...
TxID tx = session.getTxID();
tx.putSlot(index, someObject);
:
Object theTxObject = tx.getSlot(index);
:
```

セキュリティー処理

マップはセキュリティー機構を使用して保護することができます。セキュリティー・フィーチャーの構成と使用に、ObjectGrid で次のメソッドが使用可能です。

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

使用可能なセキュリティー機構についての詳細は 144 ページの『ObjectGrid セキュリティー』を参照してください。

ObjectGridEventListener

ObjectGridEventListener リスナーは、トランザクションが開始またはコミットした場合、アプリケーションが通知を受け取る方法を提供します。ObjectGridEventListener のインスタンスを ObjectGrid に設定できます。詳しくは 194 ページの『リスナー』のトピックを参照してください。次に示すのは、ObjectGridEventListener インターフェースをプログラマチックにインプリメントする方法の例です。

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

XML を使用して同じ構成を作成することもできます。

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

TransactionCallback プラグイン

セッションの呼び出しメソッドが、TransactionCallback プラグインに、対応イベントを送信します。1 つの ObjectGrid はゼロまたは 1 つの TransactionCallback プラグインを持ちます。 TransactionCallback プラグインにより ObjectGrid で定義された BackingMaps は対応するローダーを持たなければなりません。詳しくは、 226 ページの『TransactionCallback プラグイン』を参照してください。以下のコードの断片は、 TransactionCallback プラグインをプログラマチックにインプリメントする方法を示します。

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

XML を使用して同じ構成を作成することができます。

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

システム・プログラミング・モデルの概要: BackingMap インターフェースのプラグ・ポイントとフィーチャー

BackingMap インターフェースには、ObjectGrid とより拡張性のある対話を行うためのオプションのプラグ・ポイントが複数あります。

次の各セクションではフィーチャーを説明し、使用法のシナリオについて簡単なコードの断片を提供します。該当する場合は、XML の断片が提供されて代替の XML 構成を示します。詳しい情報については、114 ページの『BackingMap インターフェース』および 270 ページの『ObjectGrid 構成』の各トピックまたは API の資料を参照してください。

構成属性

複数の構成項目が BackingMaps と関連付けられます。

- **ReadOnly** (デフォルトは `false`): この属性を `true` に設定するとバックアップ・マップが読み取り専用になります。 `false` に設定するとバックアップ・マップを読み書きできます。値を指定しない場合には、デフォルトで読み書きできるようになります。
- **NullValuesSupported** (デフォルトは `true`): ヌル値をサポートするということは、ヌル値をマップに入れることができるという意味です。この属性を true に設定すると、ObjectMap でヌル値がサポートされます。そうでない場合は、ヌル値はサポートされません。ヌル値がサポートされる場合は、ヌルを戻す `get` 操作は、値がヌルか、または渡されたキーがマップに含まれていないことを意味します。
- **NumberOfBuckets** (デフォルトは `503`): この BackingMap によって使用されるバケットの数を指定します。BackingMap インプリメンテーションは、その実装のためにハッシュ・マップを使用します。 BackingMap に多くのエントリーが存在する場合は、バケットが多い方がパフォーマンスが上がります。バケット数が増えるほど、衝突のリスクが低減するからです。また、バケットが多いと並行性も高くなります。
- **NumberOfLockBuckets** (デフォルトは `383`): この BackingMap についてロック・マネージャーによって使用されるロック・バケットの数を指定します。lockStrategy 属性が OPTIMISTIC または PESSIMISTIC に設定されている場合、BackingMap についてロック・マネージャーが作成されます。ロック・マネージャーは、ハッシュ・マップを使用して、1 つ以上のトランザクションによってロックされるエントリーを追跡します。ハッシュ・マップ内に多くのエントリーが存在する場合は、ロック・バケットが多い方がパフォーマンスが上がります。バケット数が増えるほど、衝突のリスクが低減するからです。また、ロック・バケットが多いと並行性も高くなります。 lockStrategy が NONE の場合、この BackingMap によって使用されるロック・マネージャーはありません。この場合は、numberOfLockBuckets 属性を設定しても何の影響もありません。

プログラマチック構成の例

以下の例では、バックアップ・マップのプロパティーを構成します。

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// override default of read/write
```

```
backingMap.setReadOnly(true);
// override default of allowing Null values
backingMap.setNullValuesSupported(false);
// override default (prime numbers work best)
backingMap.setNumberOfBuckets(251);
// override default (prime numbers work best)
backingMap.setNumberOfLockBuckets(251);
:
```

XML 構成の例

次の XML 構成の例では、前述のプログラマチックの例で示されたものと同じプロパティーを構成します。

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

ロック・ストラテジー

ロック・ストラテジーが OPTIMISTIC または PESSIMISTIC に設定されている場合は、BackingMap についてロック・マネージャーが作成されます。デッドロック発生を回避するために、ロック・マネージャーには、ロックの認可を待機するためのデフォルトのタイムアウト値があります。このタイムアウト限度を超えると、LockTimeoutException 例外が起こります。ほとんどのアプリケーションではデフォルト値を 15 秒に設定すれば十分ですが、負荷が高いシステムでは、実際のデッドロックが存在していないときにタイムアウトが発生する場合もあります。その場合は、setLockTimeout メソッドを使用してロックのタイムアウト値をデフォルトから必要なだけ変更し、誤ったタイムアウト例外が起こらないようにします。ロック・ストラテジーが NONE である場合、この BackingMap はロック・マネージャーを使用しません。この場合は、lockTimeout 属性を設定しても何の影響もありません。詳しくは、 134 ページの『ロック』のトピックを参照してください。

プログラマチック構成の例

次の例ではロック・ストラテジーを設定します。

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// override default value of OPTIMISTIC
backingMap.setLockStrategy(LockStrategy.PESSIMISTIC);
backingMap.setLockTimeout(30); // sets lock timeout to 30 seconds
:
```

XML 構成の例

次の例では、前述のプログラマチックの例で定義したロック・ストラテジーと同じものを設定します。

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

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

キーと値をコピーする

キーと値のコピーは、リソースの面でもパフォーマンスの面でも高価な操作になる場合があります。これらのコピー機能がない場合、見慣れない、デバッグの難しい問題が起こることがあります。ObjectGrid では、キーと値のコピーを行うかどうか、およびいつ行うかを構成する機能を提供しました。通常、キーは不変であると見なされるので、キー・オブジェクトをコピーする必要はありません。デフォルト・モードでは、キー・オブジェクトのコピーをしません。Value オブジェクトは、アプリケーションによって変更される可能性がより高くなります。Value オブジェクトのコピーを行う時期か、Value オブジェクトの実際の参照かを、オプションで構成できます。CopyKey および CopyMode の設定について詳しくは、345 ページの『第 11 章 ObjectGrid パフォーマンスのベスト・プラクティス』のトピックおよび JavaDoc を参照してください。

プログラマチック構成の例

以下は、コピー・モードの設定およびキーのコピーの設定の例です。

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
backingMap.setCopyKey(true); // make a copy of each new key
backingMap.setCopyMode(NO_COPY); // Most efficient - trust the application
:
```

XML 構成の例

以下の例は前述のプログラマチック構成例と同じ結果になります。

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

Evictor

Evictor は、マップ内の不要なエントリーを定期的にクリーンアップするために使用します。除去するエントリーは、Evictor によって定義します。組み込み型 Evictor は時間ベースなので、除去ストラテジーはマップ内でエントリーが存続している時間の長さに基づきます。ほかの除去ストラテジーには使用法に基づくもの、サイズに基づくもの、または複数の要因の組み合わせに基づくものがあります。

- **組み込み型存続時間 (TTL) Evictor**: 組み込み型存続時間 Evictor は、setTtlEvictorType メソッドおよび setTimeToLive メソッドによって BackingMap に設定される、2、3 の構成項目を提供します。 デフォルトでは、この組み込み型存続時間 Evictor はアクティブではありません。3 つの値、CREATION_TIME、LAST_ACCESS_TIME、または NONE (デフォルト) のいずれか 1 つによって setTtlEvictorType メソッドを呼び出すことにより、これをアク

ティブにすることができます。次に、選択した存続時間 Evictor のタイプに応じて setTimeToLive メソッドの値を使用し、各マップ・エントリーの存続時間を設定します。

- **Evictor プラグイン:** 組み込み型存続時間 Evictor に加えて、アプリケーションは独自の Evictor 実装プラグインを提供できます。ユーザーは任意のアルゴリズムを定期的に使用して、マップ・エントリーを無効にできます。

プログラマチック構成

次のクラスで Evictor を作成します。

```
class MyEvictor implements com.ibm.websphere.objectgrid.plugins.Evictor { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
BackingMap backingMap = objectGrid.getMap("someMap");
// timer starts when entry is first created
backingMap.setTtlEvictorType(CREATION_TIME);
// Allow each map entry to live 30 seconds before invalidation
backingMap.setTimeToLive(30);
// Both builtin and custom Evictors will be active
backingMap.setEvictor(new MyEvictor()); :
```

XML 構成

次の XML コードは、前述のプログラマチック構成と同じ構成を作成します。

```
:
<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>
:
```

詳しくは、 199 ページの『Evictor』を参照してください。

ローダー

ObjectGrid ローダーは、ObjectGrid マップを、通常同じシステムまたは別のシステム上の永続ストアに保存されているデータのメモリー・キャッシュとして振る舞えるようにする、プラグ可能なコンポーネントです。通常、データベースまたはファイル・システムは永続ストアとして使用されます。ローダーは永続ストアからデータを読み取ったり、永続ストアにデータを書き込んだりするためのロジックを有しています。

ローダーは、ObjectGrid バックアップ・マップのオプションのプラグインです。指定されたバックアップ・マップとは 1 つのローダーしか関連付けることができず、また、各バックアップ・マップには独自のローダー・インスタンスがあります。バックアップ・マップは、自身が所有していないデータをそのローダーから要求しま

す。マップに対するすべての変更は、ローダーにプッシュ・アウトされます。ローダー・プラグインは、マップとその永続ストア間でバックアップ・マップがデータを移動する手段を提供します。

プログラマチック構成

以下はローダー実装の例です。

```
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 構成

次の XML のサンプルは、前述のプログラマチックの例と同じ構成結果になります。

```
:
<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>
:
```

詳しくは、 210 ページの『ローダー』のトピックを参照してください。

MapEventListener インターフェース

MapEventListener コールバック・インターフェースは、マップ・エントリーの除去やデータ・プリロードの完了などの、マップに関するイベントを受け取りたい場合に、アプリケーションによって実装されます。以下のコード例では、BackingMap インスタンス上で MapEventListener インスタンスを設定する方法を示します。

プログラマチック構成

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

XML 構成

以下の例は前述のプログラマチックの例と同じ構成になります。

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

詳しくは、 194 ページの『リスナー』のトピックを参照してください。

ObjectTransformer インターフェース

ObjectTransformer は、直列可能と定義されていないキャッシュ・エントリーのキーと値を直列化し、直接 Serializable インターフェースを拡張したり実装したりすることなく、独自の直列化スキームを定義できるようにするために使用できます。このインターフェースはキーおよび値に対するコピー機能を実行するメソッドも提供します。以下は ObjectTransformer インターフェースを実装するクラスです。

プログラマチック構成

```
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 構成

以下の XML 例は、前述のプログラマチックのサンプルと同じ構成になります。

```
:
<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>
:
```

詳しくは、 222 ページの『ObjectTransformer プラグイン』のトピックを参照してください。

OptimisticCallback インターフェース

OptimisticCallback インターフェースは、指定された Value オブジェクトと関連付けられるバージョン・フィールドの作成と処理に使用できます。多くの場合、Value オブジェクトを直接使用して、値が検索されてから別のキャッシュ・クライアントがその値を変更したかどうかを判別するのは、非常に非効率的であり、エラーが起こりやすくなります。この代替として、Value オブジェクトの状態を表す別のフィールドを提供します。 Value オブジェクトを表す、代替の Versioned Value オブジェクトを提供することが、この OptimisticCallback インターフェースの意図です。以下は、OptimisticCallback インターフェースの構成サンプルです。

プログラマチック構成

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

XML 構成

以下の例は前述のプログラマチックの例と同じ構成になります。

```
:
<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>
:
```

索引

MapIndexPlugin プラグイン (短縮名は Index) は、BackingMap が、格納されているオブジェクトの指定された属性に基づいて索引をビルドする場合に使用するオプションです。索引によって、アプリケーションは、特定の値または値の範囲を使用してオブジェクトを検索することができます。索引を使用するには、アプリケーションは、ObjectMap インターフェースの getIndex() メソッドから索引オブジェクトを取得し、MapIndex や MapRangeIndex などの正しい索引インターフェースまたはカスタム索引インターフェースに、その索引オブジェクトをキャストする必要があります。

現在のところ、索引付けフィーチャーはローカル・キャッシュにおいてのみサポートされており、分散キャッシュではサポートされていません。分散キャッシュに対して索引付け操作を実行しようとすると、UnsupportedOperationException が発生します。

索引には、静的索引と動的索引の 2 つのタイプがあります。静的索引は、プログラマチック構成と XML 構成の両方で作成できます。動的索引は、プログラマチック構成でのみ作成できます。

プログラマチック構成

以下のコード例は、BackingMap インスタンスに静的索引を追加する方法を示します。

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("indexSampleGrid");
BackingMap personBackingMap= objectGrid.getMap("person");
//use the builtin com.ibm.websphere.objectgrid.plugins.index.HashIndex
//class as the index plugin class.
HashIndex mapIndexPlugin = new HashIndex();
mapIndexPlugin.setName("CODE");
mapIndexPlugin.setAttributeName("EmployeeCode");
mapIndexPlugin.setRangeIndex(true);
personBackingMap.addMapIndexPlugin(mapIndexPlugin);
//Note: the previous Index configuration assumes that the stored object has
// an attribute named EmployeeCode and a method named getEmployeeCode()
//that returns the value of the EmployeeCode attribute.
:
```

以下のコード例は、BackingMap インスタンス上に動的索引を作成する方法を示します。

```
class DynamicIndexCallbackImpl implements
com.ibm.websphere.objectgrid.plugins.index.DynamicIndexCallback { ... }
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("indexSampleGrid");
BackingMap personBackingMap= objectGrid.getMap("person");
objectGrid.initiallize();
:
//insert, update, or remove data
//Dynamic index can be created after the containing ObjectGrid instance has
//been initialized
//If there is a need to create a dynamic index, create it without
//DynamicIndexCallback
personBackingMap.createDynamicIndex("CODE2", true, "employeeCode", null);
:
//Another option is to create dynamic index with DynamicIndexCallback
//Assuming there is a DynamicIndexCallbackImpl class implements
//DynamicIndexCallback interface
personBackingMap.createDynamicIndex("CODE3", true, "employeeCode",
new DynamicIndexCallbackImpl());
:
```

XML 構成

以下の例は、前述の静的索引のプログラマチックの例と同じ構成になります。

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

```
description="true for MapRangeIndex />
<property name="AttributeName" type="java.lang.String"
value="employeeCode" description="attriubte name" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
:
```

詳しくは、『索引付け』のトピックを参照してください。

システム・プログラミング・モデルの概要: Session インターフェース・フィーチャー

Session インターフェースには、ObjectGrid との対話をさらに拡張するためのフィーチャーが幾つかあります。次の各セクションではフィーチャーを説明し、使用法のシナリオについて簡単なコードの断片を提供します。

Session インターフェースについての詳細は、119 ページの『Session インターフェース』を参照してください。

ノー・ライトスルー・モード

アプリケーションが、基本マップを変更することがありますが、ローダーを変更することはありません。Session インターフェースの beginNoWriteThrough メソッドは、この目的を実現するよう設計されています。Session インターフェースの isWriteThroughEnabled メソッドを使用して、現行セッションがバックエンド・ローダーに書き込みを行っているかどうかを検査できます。このセッション・オブジェクトの別のユーザーがこれを使用して、現在どのタイプのセッションが処理されているかを知ることができます。以下はノー・ライトスルー・モードを使用可能にする例です。

```
:
ObjectGrid objectGrid = objectGridManager.createObjectGrid("someGrid");
objectGrid.defineMap("someMap");
objectGrid.initialize();
Session session = objectGrid.getSession();
session.beginNoWriteThrough();
boolean isWriteThroughEnabled = session.isWriteThroughEnabled();
// make updates to the map ...
session.commit();
:
```

ローダーにのみデータをプッシュしてください。

次の例にあるように、アプリケーションは、フラッシュ・メソッドの起動により、変更を永久にはコミットしないで、セッションでローダーを局所的に変更することができます。

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

processLogSequence メソッド

processLogSequence メソッドを使用して LogSequence を処理します。LogSequence 内の各 LogElement が検査され、 insert、update、invalidate オペレーションなどの適切なオペレーションが、 LogSequence MapName によって識別された BackingMap に対して実行されます。このメソッドが起動する前に、ObjectGrid セッションをアクティブにしなければなりません。呼び出し元は、次に適切なコミットまたはロールバック・コールを発行して、セッションを完了する必要があります。自動コミット処理は、このメソッド呼び出しには使用できません。

このメソッドは主に、リモート JVM が受信した LogSequence の処理に使用されます。例えば、分散コミット・サポートを使用して、コミット済み指定セッションに関連した LogSequences がこの後、別の Java 仮想マシン (JVM) で listen している別の ObjectGrids に配布されます。リスナーはリモート JVM で LogSequences を受信後、 beginNoWriteThrough メソッドを使用してセッションを開始し、この processLogSequence メソッドを起動して、次にセッションで commit メソッドを行います。以下に例を示します。

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

パフォーマンス・モニター

WebSphere Application Server での実行中にパフォーマンス・モニターを行うために、オプションでマップを装備できます。パフォーマンス・モニター・フィーチャーを構成および使用するためのセッションで、 setTransactionType メソッドを使用できます。詳しくは、 308 ページの『WebSphere Application Server Performance Monitoring Infrastructure (PMI) を使用した ObjectGrid パフォーマンスのモニター』を参照してください。

システム・プログラミング・モデルの概要: ObjectMap インターフェース・フィーチャー

ObjectMap インターフェースには、ObjectGrid との対話をさらに拡張するためのフィーチャーが幾つかあります。

ObjectMap インターフェースについての詳細は 123 ページの『ObjectMap および JavaMap インターフェース』を参照してください。

JavaMap インターフェースおよび java.util.Map インターフェース

java.util.Map インターフェースを使用するアプリケーションのために、ObjectMap は getJavaMap を持ち、このためアプリケーションは ObjectMap に支持された java.util.Map インターフェースのインプリメンテーションを取得できます。戻されたマップ・インスタンスは JavaMap インターフェースにキャストされ、

java.util.Map インターフェースを拡張します。JavaMap インターフェースは、ObjectMap と同じメソッド・シグニチャーを持ちますが、例外処理の方法は異なります。JavaMap インターフェースは java.util.Map インターフェースを拡張します。このためすべての例外は java.lang.RuntimeException クラスのインスタンスです。JavaMap インターフェースは java.util.Map インターフェースを拡張するので、オブジェクト・キャッシングに java.util.Map を使用する既存アプリケーションによって ObjectGrid をすぐに容易に使用できます。以下はコードの断片です。

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

マップ拡張子

ObjectMap インターフェースは、チェックあり例外機能の他に、追加の機能も提供します。例えば、ユーザーは getForUpdate メソッドにより特定のマップ・エントリーを更新するよう指定できます。このメソッドは、ObjectGrid ランタイムおよびローダー・プラグインに対して、適切な場合は処理中にエントリーがロックされることがあると指示します。バッチ処理は、getAll、putAll、removeAll メソッドによるもう 1 つの追加機能です。これらのメソッドについての詳細は API 文書を参照してください。

キーワード処理

ほとんどのマップ操作には、 insert、get、getForUpdate、put、remove、invalidate などのキーワード・パラメーターのバージョンがあります。簡単に利用するため、setDefaultKeyword メソッドも提供されています。このメソッドは、マップ操作のキーワード・バージョンを使用せずに、エントリーをキーワードに関連付けます。以下にキーワードの例を示します。

```
:
// setDefaultKeyword
session.begin();
objectMap.setDefaultKeyword("New York");
Person p = (Person) objectMap.get("Billy"); // "Billy" entry has "New York" keyword
p = (Person) objectMap.get("Bob", "Los Angeles"); // "Bob" entry
//has "Los Angeles" keyword
objectMap.setDefaultKeyword(null);
p = (Person) objectMap.get("Jimmy"); // "Jimmy" entry has no keyword
session.commit();
:
// keyword parameter version of insert operation
session.begin();
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person, "Rochester"); // "BillyBob" has
//"Rochester" keyword
session.commit();
:
```

詳しくは、 128 ページの『キーワード』を参照してください。

コピー・モード・メソッド

setCopyMode メソッドにより、マップ用のコピー・モードはこのセッションまたはトランザクション用のこのマップでのみオーバーライドできます。このメソッドは、アプリケーションがセッションごとに必要に応じて最適のコピー・モードを使用できるようにします。コピー・モードは、アクティブ・セッションの間は変更で

きません。対応する clearCopyMode メソッドがあり、これはコピー・モードを BackingMap で定義した状態にリセットします。このメソッドはアクティブ・セッションが存在しないときにのみ呼び出しが可能です。コピー・モード設定の例を以下に示します。

```
:
objectMap.setCopyMode(CopyMode.COPY_ON_READ, null);
session.begin();
// modify objectMap ...
session.commit();
objectMap.clearCopyMode(); // reset CopyMode to BackingMap setting
session.begin();
// modify objectMap ...
session.commit();
:
```

詳しくは 222 ページの『ObjectTransformer プラグイン』および 345 ページの『第 11 章 ObjectGrid パフォーマンスのベスト・プラクティス』のトピックを参照してください。

Evictor の設定

ObjectMap レベルの組み込み型存続時間 Evictor に応じて TimeToLive タイムアウト値をオーバーライドできます。 setTimeToLive メソッドは特定のキャッシュ・エントリーが存続する時間を秒数で設定します。変更を行うと、前の TimeToLive 値が戻されます。この TimeToLive 値は、エントリーが除去と判断されるまでキャッシュに存続する最小時間であり、最後のアクセス後にエントリーが存続する時間を組み込み型存続時間 Evictor に指示します。新規 TimeToLive 値は、トランザクションがアクセスした ObjectMap エントリーにのみ適用されますが、このトランザクションは ObjectMap インスタンスを取得するために使用したセッション・オブジェクトにより開始されたトランザクションです。新規 TimeToLive 値は、セッションについて進行中のすべてのトランザクション、およびセッションによって実行される将来のトランザクションに適用されます。新規 TimeToLive 値は、他のセッションによって開始されたトランザクションがアクセスした ObjectMap インスタンスのエントリーに影響しません。ObjectMap でこのメソッドを呼び出すことによって、BackingMap で setTimeToLive メソッドが設定した前の値はこの ObjectMap インスタンスのためにオーバーライドされます。以下に例を示します。

```
:
session.begin();
int oldTTL = objectMap.setTimeToLive(60); // set TTL to 60 seconds
Person person = new Person("Joe", "Bloggs", "Manhattan");
objectMap.insert("BillyBob", person); // "BillyBob" entry will have a TTL
//of 60 seconds
session.commit();
:
objectMap.setTimeToLive(oldTTL); // reset TTL to original value
Person person2 = new Person("Angelina", "Jolie", "somewhere");
objectMap.insert("Brad", person2); // "Brad" entry will use original TTL value
:
```

詳しくは、 199 ページの『Evictor』のトピックを参照してください。

第 5 章 ObjectGrid のサンプル

このトピックでは、WebSphere Extended Deployment 製品をインストールする際に提供される ObjectGrid のサンプルを説明します。

概要

いくつかの ObjectGrid のサンプルは、Java 2 Platform, Enterprise Edition (J2EE) の各アプリケーションおよび区画化機能 (WPF) との統合を示します。このトピックでは各サンプル、各サンプルが示すフィーチャー、各サンプルのロケーション、およびサンプルが実行される環境を説明します。

また、WebSphere Extended Deployment をインストールする際に提供されるサンプルを説明します。 Java Message Service (JMS) 統合の使用、および ObjectGrid と別のオープン・ソース・フレームワークとの統合に関連するほかのサンプルは、次の Web アドレスで提供されています。
http://www-1.ibm.com/support/docview.wss?uid=swg27006432

サンプル

- **ObjectGridSamplesSA :** このサンプルは、ObjectGrid の機能を示すために `objectgridSamples.jar` ファイルにパッケージされた、一連の Java 2 Platform, Standard Edition (J2SE) の例です。これらの J2SE のサンプルは、J2SE 環境で実行できます。`objectgridSamples.jar` ファイルには、`SamplesGuide.htm` ファイルが含まれ、これらのサンプルの実行に関する指示が含まれています。
- **ObjectGridSample :** このサンプルは、各サーブレットおよび Session Enterprise Bean による ObjectGrid 機能の使用法を示す、J2EE の例です。このサンプルは `ObjectGridSample.ear` エンタープライズ・アーカイブ (EAR) ファイルに同梱されています。`ObjectGridSample.ear` ファイルには、`readme.txt` ファイルが含まれ、このサンプルの設定と実行に関する指示が含まれています。
- **ObjectGridPartitionCluster :** このサンプルは、WPF および ObjectGrid の連動方法、ObjectGridEventListener を使用してオブジェクトの変更を伝搬する方法および、ObjectGrid の保全性と整合性を維持するためのコンテキスト・ベースのルーティング方法を示す、J2EE のサンプルです。このサンプルは `D_ObjectGridPartitionClusterSample.ear` EAR ファイルに同梱されています。`D_ObjectGridPartitionClusterSample.ear` ファイルには、`readme.txt` ファイルが含まれ、このサンプルの設定と実行に関する指示が含まれています。
- **ObjectGridJMSSamples:** これは、JMS 機能を使用して、単一の JVM またはクラスター環境内にある ObjectGrid インスタンスから別の ObjectGrid インスタンスに変更を転送する方法を示す、ObjectGridJMSSamples.zip ファイルにパッケージされている一連の J2EE サンプルです。これらの J2EE サンプルは、Web でのみ使用できます。Web アドレスは次のとおりです。
  http://www-1.ibm.com/support/docview.wss?uid=swg27006432

サンプル機能

表 2. サンプルの機能

機能領域	ObjectGrid SamplesSA のサンプル	ObjectGrid Sample のサンプル	ObjectGridPartition クラスターのサンプル	ObjectGrid JMSSamples のサンプル
ObjectGrid EventListener			x	x
トランザクション・コールバック	x	x	x	
ローダー	x	x	x	
MapEvent リスナー	x			
オブジェクト変換プログラム	x	x	x	x
オプティミスティック・コールバック	x	x	x	
BackingMap コピー・モード	x	x		
分散無効化			x	x
分散更新			x	x
LogSequence 処理				x
区画化機能 (WPF)			x	
Java Message Service (JMS)				x
Map 索引	x			
ObjectGrid クラスター	x	x		
ObjectGrid ClusterClient コンテキスト	x	x		
分散 ObjectGrid	x	x		
ObjectGrid 管理	x			
ObjectGrid セキュリティー	x		x	

ロケーション

WebSphere Extended Deployment がインストールされた後に、次の .jar ファイルが次のディレクトリーに配置されます。

表 3. サンプル・ロケーション

サンプル	ロケーション
ObjectGridSamplesSA	*install_root*¥optionalLibraries¥ObjectGrid¥objectgridSamples.jar
ObjectGridSample	*install_root*¥installableApps¥ObjectGridSample.ear
ObjectGridPartitionCluster	*install_root*¥installableApps¥D_ObjectGridPartitionClusterSample.ear

リストされている同梱のサンプルの更新バージョンと ObjectGridJMSSamples などの追加サンプルは、Web 上にあります。Web アドレスは次のとおりです。http://www-1.ibm.com/support/docview.wss?uid=swg27006432 次の Web アドレスで、関心があるトピックを説明している IBM DeveloperWorks で、項目を検索することもできます。http://www.ibm.com/developerworks ここで、**ObjectGrid** を検索してください。

サンプル環境

サンプルの中には J2SE 環境で実行されるものもありますが、J2EE 環境で実行しなければならないものもあります。単一のサーバー・インスタンスで実行できるものもあれば、クラスターで実行しなければならないものもあります。次のテーブルは各サンプルの実行環境を示しています。

制約事項: WebSphere Extended Deployment バージョン 6.0 環境で ObjectGrid を使用している場合は、追加のライセンス契約を手配すると、Java 2 Platform, Standard Edition (J2SE) バージョン 1.4.2 以降の環境または WebSphere Application Server バージョン 6.02 以降の環境でも ObjectGrid を使用することができます。詳しくは、営業担当員にお問い合わせください。

表 4. サンプル実行環境

		ObjectGrid SamplesSA	ObjectGrid サンプル	ObjectGrid 区画クラスター	ObjectGrid JMSSamples
J2SE	Eclipse	x			
	コマンド行	x			
WebSphere Application Sever バージョン 6.0.*x*	単一サーバー		x		x
	クラスター		x		x
	Rational Application Developer 単体テスト環境 (UTE)		x		
WebSphere Application Server バージョン 5.0.2.*x* およびバージョン 5.1.*x*	単一サーバー		x		
	クラスター		x		

表 4. サンプル実行環境 (続き)

		ObjectGrid SamplesSA	ObjectGrid サンプル	ObjectGrid 区画クラスター	ObjectGrid JMSSamples
WebSphere Extended Deployment バージョン 6.0.*x*	単一サーバー		x		x
	クラスター		x	x	x

第 6 章 ObjectGrid のパッケージ化

ObjectGrid パッケージには、WebSphere Extended Deployment のインストールまたは混合サーバー環境のインストールの 2 つの方法でアクセスすることができます。

WebSphere Extended Deployment バージョン 6.0.1 ObjectGrid パッケージ

WebSphere Extended Deployment バージョン 6.0.1 以降をインストールする場合、以下のランタイム・ファイルがインストールされます。

表 5. WebSphere Extended Deployment ObjectGrid ランタイム・ファイル

ファイル名	ランタイム環境	説明
/lib/asm.jar /lib/cglib.jar	ローカル、クライアント、およびサーバー	これらの jar は、Copy-on-Write コピー・モードを使用している場合の cglib ユーティリティー機能用です。
/lib/wsobjectgrid.jar	ローカル、クライアント、およびサーバー	この Java アーカイブ (JAR) ファイルには、WebSphere Extended Deployment バージョン 6.0.1 以降の環境で使用する ObjectGrid のローカル、クライアント、およびサーバー・ランタイムが含まれています。

WebSphere Extended Deployment for Mixed Server Environment バージョン 6.0.1 ObjectGrid パッケージ

WebSphere Extended Deployment for Mixed Server Environment をインストールする場合、以下のランタイム・ファイルがインストールされます。

表 6. WebSphere Extended Deployment for Mixed Server Environment ObjectGrid ランタイム・ファイル

ファイル名	ランタイム環境	説明
/ObjectGrid/lib/asm.jar /ObjectGrid/lib/cglib.jar	ローカル、クライアント、およびサーバー	これらの JAR ファイルは、Copy-on-Write コピー・モードを使用している場合の cglib ユーティリティー機能用です。Copy-on-Write 書き込みコピー・モードを使用しているときに、cglib プロキシー機能を使用する場合は、これらの JAR ファイルを CLASSPATH に組み込んでください。これらの JAR ファイルは、サーバー・ランタイムに自動的に組み込まれます。これらのファイルをクライアントまたはローカル ObjectGrid ランタイムに追加してください。
/ObjectGrid/lib/mx4j.jar /ObjectGrid/lib/mx4j-remote.jar /ObjectGrid/lib/mx4j-tools.jar	管理ゲートウェイ・クライアントおよびサーバー	これらの JAR ファイルは、ObjectGrid 管理ゲートウェイ・サーバーおよび管理ゲートウェイ・クライアント・プログラムで使用される mx4j ユーティリティー機能用です。管理ゲートウェイ・サーバーに接続する場合は、これらの JAR ファイルを、管理ゲートウェイ・クライアントの CLASSPATH に追加してください。
/ObjectGrid/lib/objectgrid.jar	ローカル、クライアント、およびサーバー	この JAR ファイルは、Java 2 Platform, Standard Edition (J2SE) バージョン 1.4.2 以降のスタンドアロン・サーバー・ランタイムで使用されます。また、この JAR ファイルは、J2SE バージョン 1.3 以降のクライアントおよびローカル・ランタイムにも使用できます。

表 6. WebSphere Extended Deployment for Mixed Server Environment ObjectGrid ランタイム・ファイル (続き)

ファイル名	ランタイム環境	説明
/ObjectGrid/lib/ogclient.jar	ローカルおよびクライアント	WebSphere プロセス外で実行している場合、この JAR ファイルには、ローカルおよびクライアント ObjectGrid ランタイムのみが含まれています。占有スペースが小さいため、objectgrid.jar よりもこの JAR ファイルを使用するほうが適切であることがあります。この jar は、J2SE バージョン 1.3 以降で使用できます。
/ObjectGrid/lib/wsobjectgrid.jar	ローカル、クライアント、およびサーバー	この JAR ファイルは、WebSphere Application Server バージョン 6.0.2 以降で使用します。この JAR ファイルは、WebSphere Extended Deployment でインストールされる JAR ファイルと同じものです。
/ObjectGrid/lib/wsogclient.jar	ローカルおよびクライアント	この JAR ファイルは、WebSphere Application Server バージョン 5.0.2 以降で使用します。この JAR ファイルには、ローカルおよびクライアント ObjectGrid ランタイムのみが含まれています。

J2SE バージョン 1.3 で ObjectGrid を使用する場合の考慮事項

J2SE バージョン 1.3.x 環境で ogclient.jar ファイルまたは objectgrid.jar ファイルを使用する場合、ObjectGrid で機能するように J2SE 1.3.x 環境に以下の要件を追加する必要があります。

- **Java 認証・承認サービス (JAAS) インプリメンテーション。** J2SE バージョン 1.3 には、JAAS 仕様の一部である javax.security.Subject オブジェクトが含まれていませんでした。 ObjectGrid および Session インターフェースでは、このオブジェクトが必要となります。JAAS インプリメンテーションを、jre/lib/ext Java 拡張ディレクトリーに配置します。
- **Java API for XML Processing (JAXP) インプリメンテーション。**XML ファイルを ObjectGrid ランタイムに渡す場合、ObjectGrid ランタイムが XML ファイルを解析するために JAXP インプリメンテーションが必要となります。ObjectGrid では、XML スキーマ定義の構文妥当性検査が使用されるため、スキ

ーマ妥当性検査をサポートするインプリメンテーションが必要となります。スキーマ妥当性検査をサポートするインプリメンテーションの例として、Apache Xerces 製品を挙げることができます。

- **Java Secure Socket Extension (JSSE) インプリメンテーション**。クライアント・ランタイムを使用する場合、JSSE インプリメンテーションが必要となります。セキュリティーを使用可能にして実行する場合、使用される JSSE インプリメンテーションが、ObjectGrid サーバー Java Development Kit (JDK) インプリメンテーションと互換性があるかどうかを検証します。

ローカルまたはクライアント ObjectGrid ランタイムが、J2SE バージョン 1.3 を使用する J2EE バージョン 1.3 互換環境に含まれている場合、必要な仕様インプリメンテーションはすべて J2EE バージョン 1.3 の一部として必要とされるため、これらすべての要件が満たされます。

第 7 章 システム管理の概要

WebSphere Extended Deployment バージョン 6.0.1 のリリースでは、ObjectGrid に備えられているシステム管理インフラストラクチャーによって、ObjectGrid 環境をモニターおよび管理することができます。システム管理アーキテクチャーは 3 層アプローチによるものであり、ユーザー・クライアントが管理ゲートウェイ・サーバーに接続し、管理ゲートウェイ・サーバーが ObjectGrid クラスターへの ObjectGrid クライアント接続を行います。

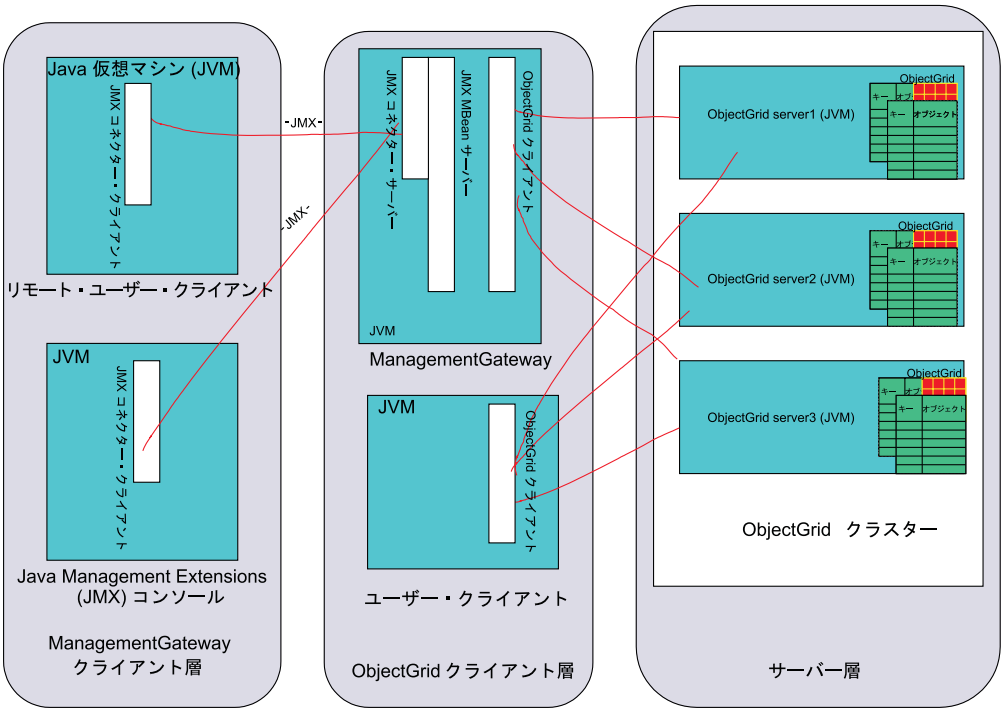


図 14. システム管理図

管理ゲートウェイ・クライアント層 には、Java Management Extensions (JMX) を使用して管理ゲートウェイ・サーバーに接続するすべてのプログラムが含まれています。また、すべてのサード・パーティー JMX コンソール、および MX4J API を使用するクライアント・プログラムも含まれています。 *ObjectGrid クライアント層* は、管理ゲートウェイ・サーバーで構成されています。管理ゲートウェイは、管理ゲートウェイ・クライアント層のサーバー、およびサーバー層 で機能する ObjectGrid クラスターのクライアントとして機能します。また、ObjectGrid クライアント・プログラムでは、ユーザーが JMX を使用しない場合、管理ゲートウェイ・サーバーが呼び出すものと同じ API を呼び出すことができます。サーバー層は、1 つの ObjectGrid クラスターで構成されています。

管理ゲートウェイには、管理 Bean (MBean) のセットが含まれ、ObjectGrid 環境の管理とモニターに JMX を使用し、MX4J オープン・ソース・プロジェクトがインプリメントされています。MX4J は、ObjectGrid に同梱されています。

ObjectGrid JMX および MBean 管理モデルは、JMX 環境の管理に使用可能なさまざまな JMX コンソールを利用するために作成されました。 JMX コンソールを選択して使用し、ダッシュボードをまとめることができます。コンソールは、ManagementGateway Java 仮想マシン (JVM) 上で実行されている MBean に接続でき、ダッシュボードはこれらの MBean を使用してアセンブルすることができます。コンソールには、グラフィカル・ヒストリーまたは数値やストリング値の図表が示されます。

システム管理コマンドを実行するには、2 つのオプションがあります。

- ObjectGridAdministrator インターフェースを使用して、現在配備されているクライアント/サーバー・インフラストラクチャーによって任意のコマンドを呼び出します。
- JMX を使用して同じコマンドを呼び出します。この場合、ObjectGrid MBean が、ObjectGridAdministrator のラッパーとして機能します。

ManagementGateway プロセスの開始

クラスター (または単一サーバー) が開始したら、ManagementGateway プロセスを開始できます。ManagementGateway は、ユーザー・クライアント要求に対するサーバー、および接続しているクラスターに対する ObjectGrid クライアントのように動作します。

オプション

以下は、ManagementGateway プロセスに渡すことのできるオプションのリストです。

- **connectorPort** (必須) - JMX コネクターのポート番号を指定します。
- **clusterHost** (必須) - ObjectGrid クラスター内のサーバーの 1 つのホスト名を指定します。
- **clusterPort** (必須) - ObjectGrid クラスター内のサーバーの 1 つのクライアント・アクセス・ポートを指定します。
- **clusterName** (必須) - ObjectGrid クラスターの名前を指定します。
- **traceEnabled** - トレースが ManagementGateway プロセスに対して有効になっている場合に指定します。
- **traceSpec** - ManagementGateway のトレース指定を示します。
- **traceFile** - トレース出力の出力先のファイルを指定します。
- **sslEnabled** - SSL が ManagementGateway 上で使用可能である場合に指定します。
- **csConfig** - セキュアな ManagementGateway の ClientSecurityConfiguration オブジェクトを指定します。
- **refreshInterval** - 管理ゲートウェイが MBean 属性を更新する時間間隔を指定します。

ManagementGateway インターフェース

MBeans を使用可能にするには ManagementGateway プロセスを開始する必要があります。ManagementGateway インターフェースは、ManagementGateway の開始時に渡すことのできるオプションを示します。

```
public interface ManagementGateway {
/**
* Start the JMX MBean connector server
*/
void startConnector();
/**
* Stop the JMX MBean connector server
*/
void stopConnector();
/**
* @param JMX connector port
*/
void setConnectorPort(int port);
/**
* @return JMX connector port
*/
int getConnectorPort();
/**
* @param a {@link com.ibm.websphere.objectgrid.security.config.
ClientSecurityConfiguration} object.
*/
void setCsConfig(ClientSecurityConfiguration csConfig);
/**
* @return a {@link com.ibm.websphere.objectgrid.security.config.
ClientSecurityConfiguration} object.
*/
ClientSecurityConfiguration getCsConfig();
/**
* @param port of server to which gateway client connects
*/
void setPort(String port);
/**
* @return port of server to which gateway client connects
*/
String getPort();
/**
* @param host of server to which gateway client connects
*/
void setHost(String host);
/**
* @return host of server to which gateway client connects
*/
String getHost();
/**
* @param boolean true if SSL enabled on gateway
*/
void setSSLEnabled(boolean sslEnabled);
/**
* @return boolean true if SSL enabled on gateway
*/
boolean getSSLEnabled();
/**
* @param cluster to which gateway client connects
*/
void setClusterName(String clusterName);
/**
* @return cluster to which gateway client connects
*/
String getClusterName();
/**
```

```
* @param true if trace is enabled on gateway
*/
void setTraceEnabled(boolean traceEnabled);
/**
* @return true if trace is enabled on gateway
*/
boolean getTraceEnabled();
/**
* @param trace specification on gateway
*/
void setTraceSpec(String traceSpec);
/**
* @return trace specification on gateway
*/
String getTraceSpec();
/**
* @param trace output file for gateway trace
*/
void setTraceFile(String traceFile);
/**
* @return trace output file for gateway trace
*/
String getTraceFile();
/**
* @param interval (in seconds) to refresh cluster MBean attributes
*/
void setRefreshInterval(int refreshInterval);
/**
* @return interval (in seconds) to refresh cluster MBean attributes
*/
int getRefreshInterval();
}
```

ManagementGateway プロセス開始用のオプション

プログラマチックな ManagementGatewayFactory の使用

このオプションの使用のためのコード例を次に示します。

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

接続しようとしている ObjectGrid クラスターが開始した後に実行されるユーザー・プログラム内にこのコードを配置する必要があります。

startManagementGateway バッチ・ファイルを指定したコマンド行

以下に例を示します。

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

startManagementGateway スクリプトについての詳細は、 90 ページの『管理ゲートウェイ・サーバーの始動』を参照してください。

ManagementGateway は、JMX 呼び出しを行うクライアント・プロセスのサーバーとして、また、ユーザーが接続するクラスターに対する ObjectGrid クライアントとして動作します。ManagementGateway が開始すると、クラスターに対する接続が確立し、JMX コネクター・サービスが使用可能になります。その後、MX4J または Java 2 Platform, Standard Edition (J2SE) バージョン 5 API を介して JMX コネクター・サービスにアクセスできます。

例

以下は、コネクター・ポート 1 の ManagementGateway を介して Server 1 というサーバーから MapStatsModule を取得するコード例です。

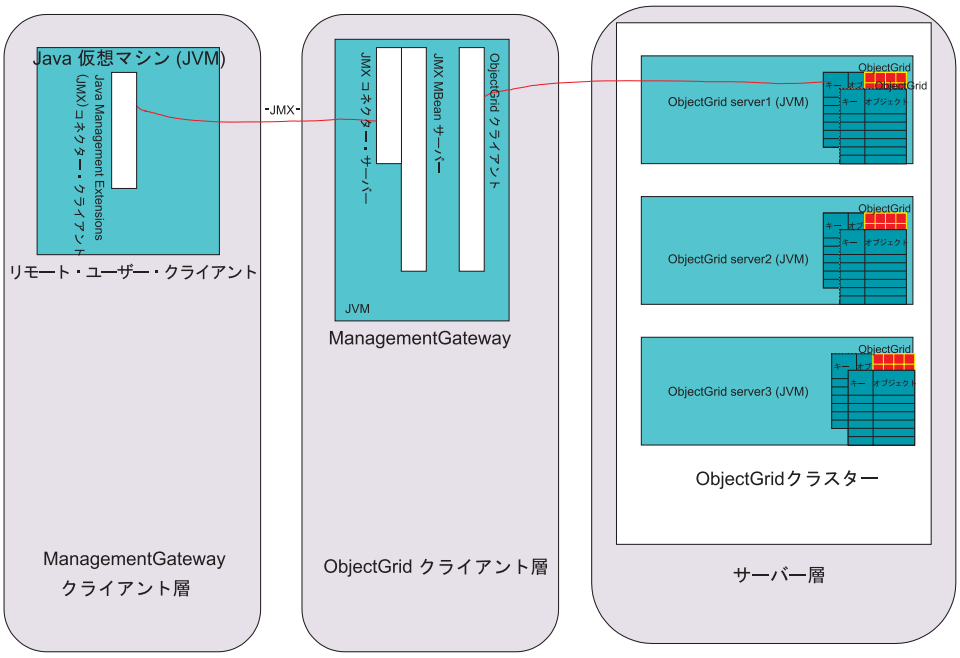


図 15. server1 サーバーからマップ統計を取得

前の図のリモート・ユーザー・クライアント・セクションで実行されるユーザー・プログラム内で次のコードを実行します。

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

ManagementGateway を介して server1 サーバーを停止するには、以下のようにします。

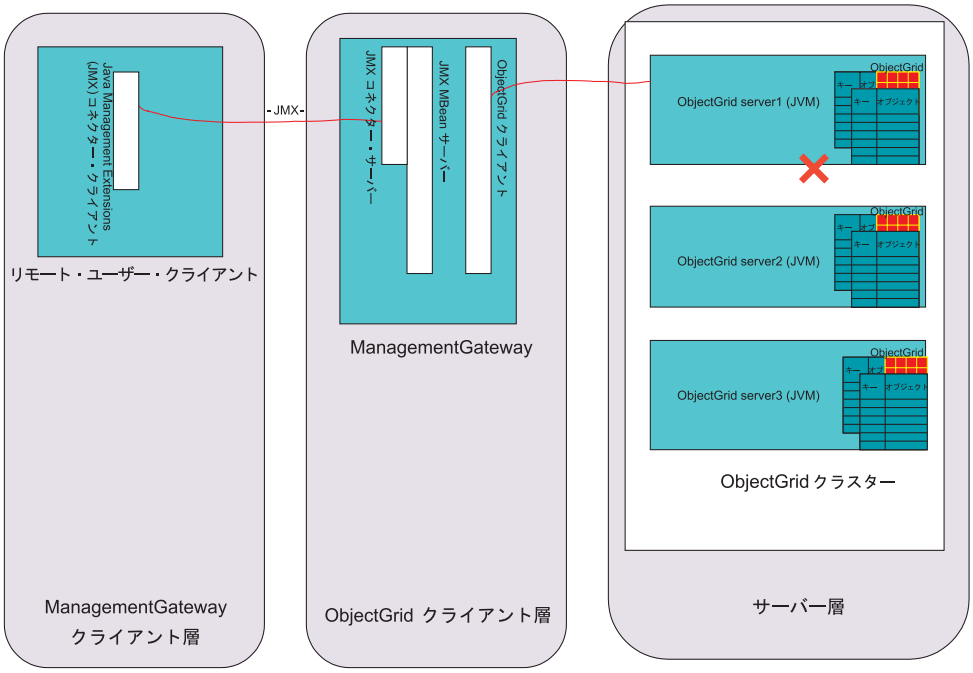


図 16. server1 サーバーの停止

前の図のリモート・ユーザー・クライアントで実行されるユーザー・プログラム内で次のコードを実行します。

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

前のコード例の実行の後、server1 サーバーは停止します。 server1 サーバーが停止したら、このサーバーは ManagementGateway を使用して再始動できません。サーバーは、コマンド行を使用して再始動できます。詳しくは、 89 ページの『ObjectGrid サーバーの停止』を参照してください。

ObjectGrid 管理 Bean (MBean)

ObjectGrid 環境には、5 つのタイプの MBean が存在します。各 MBean は、マップ、オブジェクト・グリッド、サーバー、複製グループ、または複製グループ・メンバーなどの特定のエンティティーを参照し、属性と操作を持っています。

ObjectGrid 内の各 MBean は、属性値を表す getxxx メソッドを持っています。これらの getxxx メソッドは、ユーザー・プログラムから直接呼び出すことができません。これは、Java Management Extensions (JMX) 仕様での属性と操作の扱いが異な

るためです。属性は、任意のサード・パーティー JMX コンソールから表示でき、操作は、ユーザー・プログラムまたはサード・パーティー JMX コンソールのいずれかから実行できます。

MapMbean Mbean

MapMBean を使用すると、ユーザーは、クラスターに定義されている各マップの統計をモニターできます。各マップには、以下の統計が関連付けられています。

- バッチ更新時間 (最小/最大/平均/合計)
- 数
- ヒット率

また、マップは複数のサーバーにわたって区画化できるため、マップ統計を特定のサーバーや複製グループ・メンバーに範囲設定することができます。クラスター全体をマップ統計の対象とすることもできます。MapMBean の ObjectName は、以下のいくつかの方法で指定できます。

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

複製グループ RG1 に 2 つのサーバー server1 と server2 がある ObjectGrid OG1 とマップ Map1 の構成例を見てみます。また、server1 サーバーがプライマリーで、server2 サーバーがレプリカであるとします。プライマリーの Map1 マップの統計を取得するには、これらの ObjectName のいずれかを使用します。

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

ObjectGrid MBean の ObjectName では、IDX=0 の場合、複製グループのプライマリーを参照します。IDX=1-10 の場合は、複製グループのレプリカを参照します。

MapMBean インターフェースのリストは、以下のとおりです。

```
public interface MapMBean {
/**
* Operation to get MapStatsModule associated with the MBean.
*
* @return MapStatsModule
*/
MapStatsModule retrieveStatsModule();
/**
* Operation will only go to server to get StatsModule if the
* StatsModule is not cached in the ObjectGridAdministrator.
*
*/
void refreshStatsModule();
/**
* Map.
*
* @return name of map
*/
String getMapName();
/**
* ObjectGrid containing the map.
*
* @return name of the object grid
```

```
*/
String getObjectGridName();
/**
* Server name of the replication group member for the map.
*
* @return name of server of the replication group member
*/
String getServerName();
/**
* Name of replication group for the map.
*
* @return name of replication group
*/
String getReplicationGroup();
/**
* Index of replication group member for the map.
*
* @return index of replication group member
*/
int getIndex();
/**
* MapStatsModule attribute loaded up by the
* retrieveStatsModule call.
*
* @return String form of MapStatsModule
*/
String getMapStatsModule();
/**
* Map count attribute loaded up by the
* retrieveStatsModule call.
*
* @return number of entries in map
*/
long getMapCountStatistic();
/**
* Hit rate attribute loaded up by the
* retrieveStatsModule call.
*
* @return hit rate for map
*/
double getMapHitRateStatistic();
/**
* Mean batch update time attribute loaded up by the
* retrieveStatsModule call.
*
* @return mean batch update time for map
*/
double getMapBatchUpdateMeanTime();
/**
* Maximum batch update time attribute loaded up by the
* retrieveStatsModule call.
*
* @return maximum batch update time for map
*/
double getMapBatchUpdateMaxTime();
/**
* Minimum batch update time attribute loaded up by the
* retrieveStatsModule call.
*
* @return minimum batch update time for map
*/
double getMapBatchUpdateMinTime();
/**
* Total batch update time attribute loaded up by the
* retrieveStatsModule call.
*
```

```
* @return total batch update time for map
*/
double getMapBatchUpdateTotalTime();
}
```

ObjectGridMBean MBean

ObjectGridMBean MBean を使用すると、ユーザーは、クラスターに定義されている各 ObjectGrid のすべてのマップの統計をモニターできます。各 ObjectGrid には、以下の統計が関連付けられています。

- トランザクション時間 (最小/最大/平均/合計)
- 数

また、ObjectGrid は複数のサーバーにわたって区画化できるため、ObjectGrid 統計を特定のサーバーや複製グループ・メンバーに範囲設定することができます。クラスター全体の ObjectGrid 統計を取得することもできます。ObjectGridMBean の ObjectName は、以下のいくつかの方法で指定できます。

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

ObjectGridMbean インターフェースのリストは、以下のとおりです。

```
public interface ObjectGridMBean {
/**
* Operation to get OGStatsModule associated with the MBean.
*
* @return OGStatsModule
*/
OGStatsModule retrieveStatsModule();
/**
* Operation will only go to server to get StatsModule if the
* StatsModule is not cached in the ObjectGridAdministrator.
*
*/
void refreshStatsModule();
/**
* ObjectGrid.
*
* @return name of the object grid
*/
String getObjectGridName();
/**
* Server name of the replication group member for the ObjectGrid.
*
* @return name of server of the replication group member
*/
String getServerName();
/**
* Name of replication group for the ObjectGrid.
*
* @return name of replication group
*/
String getReplicationGroup();
/**
* Index of replication group member for the ObjectGrid.
*
* @return index of replication group member
*/
```

```
int getIndex();
/**
* OGStatsModule attribute loaded up by the retrieveStatsModule call.
*
* @return String form of OGStatsModule
*/
String getOGStatsModule();
/**
* ObjectGrid count attribute loaded up by the retrieveStatsModule call.
*
* @return number of transactions
*/
long getOGCount();
/**
* Maximum transaction time attribute loaded up by the retrieveStatsModule call.
*
* @return maximum transaction time for the ObjectGrid
*/
long getOGMaxTranTime();
/**
* Minimum transaction time attribute loaded up by the retrieveStatsModule call.
*
* @return minimum transaction time for the ObjectGrid
*/
long getOGMinTranTime();
/**
* Mean transaction time attribute loaded up by the retrieveStatsModule call.
*
* @return mean transaction time for the ObjectGrid
*/
double getOGMeanTranTime();
/**
* Total transaction time attribute loaded up by the retrieveStatsModule call.
*
* @return total transaction time for the ObjectGrid
*/
long getOGTotalTranTime();
}
```

ServerMBean MBean

ServerMBean Mbean を使用すると、ユーザーは、クラスター内のサーバーに対して操作を実行できます。ServerMBean の ObjectName は、以下の方法で指定できます。

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

ServerMBean インターフェースのリストは、以下のとおりです。

```
public interface ServerMBean {
/**
* Operation to load the replication status for the server.
*
*/
void retrieveReplicationStatus();
/**
* Return the name of the server.
*
* @return server name
*/
String getServerName();
/**
* Operation to get the status of the server.
*
* @return status of server (true if running, false if not)
*/
```

```
boolean retrieveServerStatus();
/**
* Operation to stop the server.
*
* @return true if server was stopped, false if not
*/
boolean stopServer();
/**
* Operation to force the server stop.
*
* @return true if server was stopped, false if not
*/
boolean forceStopServer();
/**
* Operation to stop the cluster the server is a part of.
*
* @param determines if servers are stopped with force
* @return true if cluster was stopped, false if not
*/
boolean stopCluster(Boolean force);
/**
* Operation to modify the trace spec for all servers in
* the cluster the server is a part of.
*
* @param trace specification
*/
void modifyClusterTraceSpec(String spec);
/**
* Operation to modify the trace spec for the server.
*
* @param trace specification
*/
void modifyServerTraceSpec(String spec);
}
```

ReplicationGroupMBean Mbean

ReplicationGroupMBean を使用すると、特定の複製グループに関連付けられているすべての複製グループ・メンバーの状況 (プライマリーになっているサーバー、10 個までのレプリカなど) をモニターすることができます。 ReplicationGroupMBean の ObjectName は、以下のように指定できます。

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

ReplicationGroupMBean インターフェースのリストは、以下のとおりです。

```
public interface ReplicationGroupMBean {
/**
* Operation to load up the status of the replication group attributes.
*
*/
String[] retrieveReplicationGroupStatus();
/**
* ReplicationGroupName attribute.
*
* @return name of the ReplicationGroup
*/
String getReplicationGroupName();
/**
* Primary attribute.
*
* @return name of the Primary
*/
String getPrimary();
/**
```

```
* Replica1 attribute.
*
* @return server name of Replica1
*/
String getReplica1();
/**
* Replica2 attribute.
*
* @return server name of Replica2
*/
String getReplica2();
/**
* Replica3 attribute.
*
* @return server name of Replica3
*/
String getReplica3();
/**
* Replica4 attribute.
*
* @return server name of Replica4
*/
String getReplica4();
/**
* Replica5 attribute.
*
* @return server name of Replica5
*/
String getReplica5();
/**
* Replica6 attribute.
*
* @return server name of Replica6
*/
String getReplica6();
/**
* Replica7 attribute.
*
* @return server name of Replica7
*/
String getReplica7();
/**
* Replica8 attribute.
*
* @return server name of Replica8
*/
String getReplica8();
/**
* Replica9 attribute.
*
* @return server name of Replica9
*/
String getReplica9();
/**
* Replica10 attribute.
*
* @return server name of Replica10
*/
String getReplica10();
/**
* All replicas for this replication group comma delimited
*
* @return server names of all replicas
*/
String getReplicas();
}
```

ReplicationGroupMemberMBean Mbean

ReplicationGroupMemberMBean を使用すると、複製グループ・メンバーの以下の統計をモニターできます。

- 複製グループ・メンバーの状況。プライマリーまたはレプリカ・メンバーをモニターできます。
- レプリカのウェイト比率。この統計は、レプリカである複製グループ・メンバーにのみ適用されます。この比率は、レプリカのマップがプライマリーのマップにどれだけ近く同期されているかを定量化するものです。この比率が高いほど、レプリカがより最新のプライマリーの情報を持っていることになります。

ReplicationGroupMemberMBean の ObjectName は、以下の方法で指定できます。

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

IDX=0 を指定すると、複製グループのプライマリーが返され、IDX=1-10 を指定するとレプリカが返されます。ReplicationGroupMBean インターフェースのリストは、以下のとおりです。

```
public interface ReplicationGroupMemberMBean {
/**
* Operation to load up the status of the replication group member attributes.
*
*/
void retrieveReplicationGroupMemberStatus();
/**
* Operation to load up the status of the replication group member
* attributes.
* Will use the cache as opposed to the retrieveReplicationGroupMemberStatus
* method which will go to the server to get status.
*
*/
void refreshReplicationGroupMemberStatus();
/**
* ReplicationGroupName attribute.
*
* @return name of the ReplicationGroup this member belongs to
*/
String getReplicationGroupName();
/**
* Status of the ReplicationGroupMember: primary/replica/standby.
*
* @return status of the ReplicationGroupMember
*/
String getStatus();
/**
* Statistic representing the percentage how close a replica is
* to being up to date with the primary maps.
*
* @return Replica statistic of the ReplicationGroupMember
*/
double getReplicaWeightRatio();
/**
* Name of server on which this ReplicationGroupMember resides.
*
* @return server name
*/
String getServerName();
/**
```

```
* Index of ReplicationGroupMember.
*
* @return index of replica
*/
int getIndex();
}
```

第 8 章 コマンド行サポート

コマンド行スクリプトを使用して、ObjectGrid サーバーを管理します。

混合サーバー環境インストールの /ObjectGrid/bin ディレクトリーに一連のスクリプト・ファイルがあります。このスクリプトを使用して、ObjectGrid サーバーの開始と停止、管理ゲートウェイ・サーバーの開始、およびプロパティー・ファイルのパスワードのエンコードを行うことができます。このスクリプトを使用する場合は、JAVA_HOME 環境変数が設定されており、その値が ObjectGrid でサポートされているバージョンの Java であることを検証します。環境変数をグローバルに変更しない場合は、setupCmdLine.bat|sh ファイルで JAVA_HOME を更新して、適切なバージョンの Java を指すことができます。

コマンド行スクリプトについての詳細は、以下のトピックを参照してください。

- 『ObjectGrid サーバーの始動』
- 89 ページの『ObjectGrid サーバーの停止』
- 90 ページの『管理ゲートウェイ・サーバーの始動』
- 92 ページの『パスワードのエンコード』

ObjectGrid サーバーの始動

ObjectGrid サーバーを始動するために、startOgServer スクリプトが用意されています。

使用法

Windows マシン上のサーバーを始動するには、startOgServer.bat ファイルを使用します。 Linux および Unix プラットフォーム上の ObjectGrid サーバーを始動するには、startOgServer.sh ファイルを使用します。

XML ファイルの使用

ObjectGrid サーバーを正常に始動するには、有効な ObjectGrid XML ファイルを有効なクラスター XML ファイルとペアにしておく必要があります。 XML ファイルは、通常のファイル名または URL を使用して、startOgServer スクリプトに渡すことができます。 URL オプションでは、http、ftp、または jarfile など、ファイル・プロトコル以外のさまざまなプロトコルを使用できます。

XML ファイルを使用してサーバーを始動するための startOgServer スクリプトの引数は、以下のとおりです。

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

例

以下に、server1 ObjectGrid サーバーを始動するいくつかの例を示します。これらの例では、startOgServer.bat ファイルを利用しています。

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

この例では、universityCluster3Servers.xml ファイルを使用しています。 server1 サーバーが始動するサーバーとして指定されているため、universityCluster3Servers.xml ファイルでは、serverDefinition 値に名前 server1 が指定されている必要があります。

universityCluster3Servers.xml ファイルが続きます。 server1 serverDefinition と、そのホストが lion.ibm.com であることに注意してください。 server1 サーバーは、lion.ibm.com ホスト上で始動される必要があります。また、このファイルでは、server2 および server3 サーバーも定義されます。これらのサーバーはそれぞれ tiger.ibm.com および bear.ibm.com ホスト上で始動される必要があります。

universityCluster3Servers.xml ファイル

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

ブートストラッピング

クラスター内の 1 つの ObjectGrid サーバーが使用可能になった後、クラスター内の他のサーバーは、使用可能サーバーをブートストラップすることができます。既

に使用可能になっているサーバーをブートストラップするには、startOgServer スクリプトに、そのサーバーのホストとクライアント・アクセス・ポートを指定する必要があります。

クラスター内の最初のサーバーは XML を使用して起動されているため、そのサーバーは、クラスター内のすべてのサーバーの構成情報を既に持っています。ブートストラッピングによって、起動側サーバーは、使用可能サーバーに接続して構成をダウンロードすることができます。

使用可能サーバーをブートストラップすることによってサーバーを始動するための startOgServer スクリプトの引数は、以下のとおりです。

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

以下に、別のサーバーをブートストラップすることによってサーバーを始動するいくつかの例を示します。最初の例では、universityCluster3Servers.xml ファイルの server1 サーバーが、XML ファイルを使用して始動され、使用可能であると想定しています。この例は、server2 サーバーを始動するために server1 サーバーをブートストラップする方法を示しています。

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

次の例では、server2 サーバーが正常に始動されたが、server1 サーバーが使用不可になったことを想定しています。別のサーバーをブートストラップする場合、ホスト:ポートの組み合わせをコンマで区切ったリストを使用できます。使用可能サーバーが見つかるまで、リストの各ホストとポートへの接続が試行されます。以下の例では、server3 が、server1 サーバーのホストとポートに接続することを想定しています。ただし、このシナリオでは server1 サーバーが使用不可であるため、接続に失敗します。リストの次の項目に進み、server3 サーバーが server2 サーバーのホストとポートをブートストラップしようとします。 server2 サーバーは使用可能なため、このブートストラップの試行は成功します。

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

オプションの引数

startOgServer スクリプトに渡すことのできるオプションの引数がいくつかあります。有効な startOgServer 引数が続きます。

オプション:

- -traceSpec <トレース仕様>
- -traceFile <トレース・ファイル>
- -serverSecurityFile <サーバー・セキュリティー・プロパティー・ファイル>
- -timeout <秒>
- -script <スクリプト・ファイル名>
- -jvmArgs <JVM 引数>

-traceSpec

-traceSpec 引数を使用すると、サーバー始動時にほぼ即時に有効になるトレース仕様を設定することができます。通常のサーバー始動時には、クラスター XML ファイルまたはブートストラップされた構成から読み取られるま

では、トレース仕様は設定されません。サーバー始動時に問題が発生した場合、以前にトレース仕様を設定しておくと有用であることがあります。

以下に、-traceSpec オプションを設定する方法の例を示します。

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

-traceFile

-traceFile 引数を使用すると、サーバー始動時に出力されるトレースのロケーションを指定できます。このサーバーの構成が読み取られた後、クラスター XML ファイルに指定されているとおりのトレース設定が有効になります。

以下に、-traceFile オプションを設定する方法の例を示します。

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

-serverSecurityFile

-serverSecurityFile 引数を使用すると、サーバーにそのセキュリティー・プロパティー・ファイルを渡すことができます。このオプションは、サーバーでセキュリティーが使用可能になっている場合は必須です。以下に、-serverSecurityFile オプションを設定する方法の例を示します。

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

-timeout

-timeout 引数を使用すると、サーバーの起動がアボートされるまでに許可されている時間 (秒単位) を指定できます。デフォルトでは、サーバーには、起動されてから使用可能になるまでに 90 秒が許可されています。特定のシナリオでこの時間が短すぎる場合は、-timeout 引数を使用してより適切な値に設定します。以下に、timeout 引数を使用する方法の例を示します。

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

-script

-script 引数を使用すると、ObjectGrid サーバー・プロセスを起動するスクリプトを作成したり、出力を現在のコマンド・プロンプトに保持したりできます。通常の環境では、ObjectGrid サーバーが起動されると、startOgServer スクリプトによって、サーバーが使用可能になるまで、サーバー・プロセスからの出力がコマンド・プロンプトに表示されます。サーバーが使用可能になった後、startOgServer はサーバー・プロセスからの出力の表示を停止して終了します。場合によっては、現在のコマンド・プロンプトに出力するサーバー・プロセスを起動したいことがあります。

スクリプトにファイ名を指定する場合は、ファイルにパスを指定しないでください。ファイルは、OBJECTGRID_HOME パスの bin ディレクトリーに配置されます。ファイルの名前を指定します。作成されるスクリプト・ファイルには、startOgServer スクリプトに渡される引数が含まれるため、作成されたスクリプトを実行するときに、同じ引数を指定する必要がありません。

以下に、-script オプションを設定する方法の例を示します。

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

この例では、OBJECTGRID_HOME/bin ディレクトリーに universityClusterServer1.bat スクリプトが作成されます。新規に作成したスクリプトを実行するには、コマンド・プロンプトで適切なディレクトリーにナビゲートし、スクリプトの名前を入力して **Enter** キーを押します。

-jvmArgs

-jvmArgs 引数を使用すると、起動中の ObjectGrid サーバー Java 仮想マシン (JVM) に引数を送信できます。JVM に渡すことができるすべての引数は、通常、-jvmArgs 引数を使用してサーバーに渡すことができます。

-jvmArgs 引数は、startOgServer スクリプトの引数として指定される最後の ObjectGrid オプションの引数である必要があります。 -jvmArgs 引数より後にあるものは、JVM 引数としてサーバー JVM に渡されます。以下に、-jvmArgs 引数を設定する方法の例を示します。

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

-jvmArgs 引数に -classpath または -cp JVM 引数が含まれる場合、指定されたクラスパスが ObjectGrid クラスパスの後に付加されます。以下に、-jvmArgs 引数を使用して、ObjectGrid サーバーを起動する場合に使用されるクラスパスに Xerces Java アーカイブ (JAR) ファイルを組み込む例を示します。

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

ObjectGrid サーバーの停止

ObjectGrid サーバーを停止するには、stopOgServer スクリプトを使用します。

使用法

Windows マシン上のサーバーを停止するには、stopOgServer.bat ファイルを使用します。 Linux および Unix プラットフォーム上の ObjectGrid サーバーを停止するには、stopOgServer.sh ファイルを使用します。stopOgServer スクリプトによって、クラスター内の任意の使用可能サーバーに接続することによってサーバーを停止できるクライアントが作成されます。このスクリプトの振る舞いは、別のサーバーを始動するための使用可能サーバーへのブートストラッピングと似ています。以下に、サーバーを停止するための stopOgServer スクリプト引数を示します。

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

例

以下に、別の ObjectGrid サーバーを停止するいくつかの例を示します。これらの例では、stopOgServer.bat ファイルを利用しています。これらの例では、 85 ページの『ObjectGrid サーバーの始動』の universityCluster3Servers.xml ファイルで定義されている server1、server2、および server3 の 3 つのサーバーが稼働中であることを想定しています。

最初の例では、ホストとクライアント・アクセス・ポートにブートストラップすることによって、server1 サーバーを停止します。

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

server1 サーバーが正常に停止します。次の例では、まず server1 サーバーにブートストラップすることによって、server2 を停止します。 server1 サーバーが既に停止されているため、ブートストラップは成功しません。リスト内の次のホストとポートは、server3 サーバーに属しています。server3 サーバーは使用可能であるため、server3 サーバーへのブートストラップは成功し、server2 が停止します。

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

オプションの引数

stopOgServer スクリプトに渡すことのできるオプションの引数がいくつかあります。このセクションでは、これらのオプションの引数の使用方法をそれぞれ示します。有効な stopOgServer 引数の後にオプションの引数が続きます。

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

オプション:

- -traceSpec <トレース仕様>
- -traceFile <トレース・ファイル>
- -clientSecurityFile <クライアント・セキュリティー・プロパティー・ファイル>

-traceSpec

-traceSpec 引数を使用すると、ObjectGrid サーバーを停止しようとするクライアントにトレース仕様を設定することができます。以下に、-traceSpec 引数を設定する方法の例を示します。

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

-traceFile

-traceFile 引数を使用すると、サーバー・シャットダウン時に出力されるクライアント・トレースのロケーションを指定できます。以下に、traceFile 引数を設定する方法の例を示します。

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

-clientSecurityFile

-clientSecurityFile 引数を使用すると、クライアントにそのセキュリティー・プロパティー・ファイルを渡すことができます。この引数は、セキュリティーが使用可能になっているサーバーに接続する場合は必須です。

以下に、-clientSecurityFile 引数を設定する方法の例を示します。

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

管理ゲートウェイ・サーバーの始動

Java Management Extensions (JMX) を使用して ObjectGrid クラスターをモニターおよび管理するには、コマンド行スクリプトを使用するか、またはプログラマチックに管理ゲートウェイを開始する必要があります。

目的

コマンド行を使用して管理ゲートウェイを開始するには、startManagementGateway スクリプトを使用します。Windows マシンで ManagementGateway サーバーを始動するには、startManagementGateway.bat ファイルを使用します。Linux および UNIX プラットフォームで ManagementGateway サーバーを始動するには、startManagementGateway.sh ファイルを使用します。管理ゲートウェイ機能、ObjectGrid MBean、および JMX についての詳細は、71 ページの『第 7 章 システム管理の概要』を参照してください。

startManagementGateway スクリプトは、JMX コネクター・サーバー、およびサーバーを停止する ObjectGrid クラスターに接続される ObjectGrid クライアントを作成し、状況と統計を収集して、さらに機能を実行します。

以下に、管理ゲートウェイ・サーバーを始動する startManagementGateway スクリプトの引数を示します。

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

オプションの引数

startManagementGateway スクリプトには、オプションの引数を数個渡すことができます。有効な startManagementGateway 引数の後に、オプションの引数を指定します。

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

オプション

- -traceEnabled <true/false trace enabled>
- -traceSpec <trace specification>
- -traceFile <trace file>
- -refreshInterval <MBean attribute refresh interval>
- -sslEnabled <true/false SSL enabled for management gateway>
- -clientSecurityFile <path to client security file>

-traceEnabled

-traceEnabled 引数を使用すると、管理ゲートウェイ・サーバーのトレースをオンにするかどうかを設定できます。デフォルトは false のため、ObjectGrid トレースを確認するには、-traceEnabled を ″true″ に設定し、有効な -traceSpec 値および -traceFile 値を指定して、トレースを使用可能にする必要があります。

-traceSpec

-traceSpec 引数を使用すると、管理ゲートウェイ・サーバーのトレース仕様を設定できます。

-traceFile

-traceFile 引数を使用すると、管理ゲートウェイ・トレース出力のロケーションを指定できます。以下に、traceEnabled、traceSpec、および traceFile 引数の設定例を示します。

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

-refreshInterval

-refreshInterval 引数を使用すると、MBean 属性値の更新と更新の間に管理ゲートウェイが待機する時間 (秒単位) を渡すことができます。デフォルト値は 120 秒です。以下に、refreshInterval 引数の設定例を示します。

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

-sslEnabled

-sslEnabled 引数を使用すると、管理ゲートウェイで SSL が使用可能かどうかを設定できます。この引数の値が `true` の場合、管理ゲートウェイ・サーバーに接続されているユーザー・クライアントでは、SSL プロパティーを指定する必要があります。

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

-sslEnabled 引数が指定されていない場合のデフォルト値は、″false″ です。

-clientSecurityFile

-clientSecurityFile 引数を使用すると、管理ゲートウェイ・サーバーと ObjectGrid クラスターの間のセキュア・クライアント・アクセスのためのクライアント・セキュリティー・プロパティーが含まれているファイルの名前を渡すことができます。この引数は、セキュリティーが使用可能な状態でクラスターに接続する場合に必要となります。ObjectGrid には、クライアント・セキュリティー・プロパティー・ファイル・テンプレート `security.ogclient.props` が同梱されています。

以下に、sslEnabled および clientSecurityFile プロパティーの設定例を示します。

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

パスワードのエンコード

パスワードのエンコードによって、ObjectGrid セキュリティー・プロパティー・ファイルのパスワードが不用意に漏れるのを防ぐことができます。

使用法

ObjectGrid には、暗号化されていないエンコード・パスワードが複数含まれています。ObjectGrid には、これらのパスワードをエンコードするために使用する FilePasswordEncoder ユーティリティーがあります。Windows マシンでパスワードをエンコードする場合は、`FilePasswordEncoder.bat` ファイルを使用します。Linux および UNIX プラットフォームでパスワードをエンコードする場合は、`FilePasswordEncoder.sh` ファイルを使用します。

コマンド構文は、以下のとおりです。

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

オプション

FilePasswordEncoder コマンドには、以下のオプションを使用することができます。

file_name

file_name を使用すると、エンコードされるパスワードを持つファイルの名前を指定できます。例えば、`security.ogserver.props` とします。

password_prop_list

password_prop_list は、″trustStorePassword,keyStorePassword″ のようにコンマで区切られたパスワード・プロパティー名のリストです。

file_type

この引数はオプションです。file_type は xml またはプロパティー値となり、指定されているファイルがプロパティー・ファイルなのか XML ファイルなのかを示します。デフォルト値は、`property` です。現行では、ObjectGrid は XML ファイルにパスワードを保存しないため、このオプションは必要ありません。以下に正しい構文の例を示します。

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

この FilePasswordEncoder ユーティリティーは、WebSphere Extended Deployment に同梱されていません。 WebSphere Application Server の PropFilePasswordEncoder ユーティリティーを使用すると、これらのパスワードをエンコードできます。詳しくは、「PropFilePasswordEncoder command reference」を参照してください。

第 9 章 ObjectGrid アプリケーション・プログラミング・インターフェースの概要

このセクションでは、XML を使用して、またはプログラマチック・インターフェースを介して ObjectGrid を構成する方法について説明します。さらに、ObjectGrid が提供する外部インターフェースを実装するための情報が含まれています。すべての場合において、概要、API インターフェース、および例についての説明があります。

API 資料

ObjectGrid の JavaDoc は、API に関する情報ソースの決定版です。 JavaDoc は、ご使用の WebSphere Extended Deployment インストールのディレクトリー *install_root*¥web¥xd¥apidocs にあります。

ObjectGridManager インターフェース

ObjectGridManagerFactory クラスと ObjectGridManager インターフェースは、ObjectGrid インスタンスの作成、アクセス、およびキャッシュを行うメカニズムを提供します。 ObjectGridManagerFactory クラスは、ObjectGridManager インターフェースにアクセスする静的ヘルパー・クラスであり、singleton クラスです。ObjectGridManager インターフェースには、 ObjectGrid オブジェクトのインスタンスを作成するいくつかの便利なメソッドがあります。また、ObjectGridManager は、複数のユーザーがアクセス可能な ObjectGrid インスタンスの作成とキャッシングも容易にします。

createObjectGrid メソッド

このトピックを使用して、ObjectGridManager インターフェース内の 7 つの createObjectGrid メソッドについて学習します。

createObjectGrid メソッド

ObjectGridManager インターフェースには 7 つの createObjectGrid メソッドがあります。以下に、単純なシナリオを示します。

デフォルト構成による単純なケース

以下に、ObjectGrid を作成して多くのユーザー間で共用する単純なケースを示します。

```
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
final ObjectGridManager oGridManager=
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees = oGridManager.createObjectGrid("Employees",true);
employees.initialize();
employees.
/*sample continues..*/
```

前の Java コードの断片によって、Employees ObjectGrid が作成され、キャッシュに入れられます。 Employees ObjectGrid は、デフォルト構成によって初期化され、すぐに使用できる状態になります。 createObjectGrid メソッド内の 2 番目のパラメーターは、true に設定されます。これにより、ObjectGridManager は、作成した ObjectGrid インスタンスをキャッシュに入れるよう指示されます。このパラメーターが false に設定されている場合、インスタンスはキャッシュに入れられません。各 ObjectGrid インスタンスには name があり、その名前に基づき、多くのクライアントまたはユーザー間でそのインスタンスを共用できます。

objectGrid インスタンスがピアツーピア共用で使用されている場合は、キャッシングを true に設定する必要があります。ピアツーピア共用について詳しくは、357 ページの『第 12 章 ピア Java 仮想マシン間の変更の配布』を参照してください。

XML 構成

ObjectGrid は高度に構成可能です。前の例では、構成を伴わない単純な ObjectGrid を作成する方法を示しました。この例では、XML 構成ファイルに基づいて事前構成された ObjectGrid インスタンスを作成できます。 ObjectGrid インスタンスは、プログラマチックに構成するか、または XML ベースの構成ファイルを使用して構成することができます。これら 2 つの方法を組み合わせて ObjectGrid を構成することもできます。

ObjectGridManager インターフェースを使用すると、 XML 構成に基づいて ObjectGrid インスタンスを作成できます。 ObjectGridManager インターフェースには、URL を引数として取るいくつかのメソッドがあります。 ObjectGridManager 内に渡される各 XML ファイルについて、スキーマに対する妥当性検査を行う必要があります。 XML の妥当性検査は、前にファイルの妥当性検査が行われており、最後の妥当検査以降、そのファイルに対する変更が行われていない場合に限り、使用不可にすることができます。妥当性検査を使用不可にすると、少量のオーバーヘッドが節約されますが、無効な XML ファイルが使用される可能性が生じます。 IBM Java Developer Kit (JDK) 1.4.2 は、XML の妥当性検査をサポートします。これをサポートしない JDK を使用すると、 Apache Xerces で XML を妥当性検査しなければならない場合があります。

次の Java コードの断片は、ObjectGrid を作成するために XML 構成ファイルに渡す方法を示しています。

```
import java.net.MalformedURLException;
import java.net.URL;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
boolean validateXML = true; // turn XML validation on
boolean cacheInstance = true; // Cache the instance
String objectGridName="Employees"; // Name of Object Grid URL
allObjectGrids = new URL("file:test/myObjectGrid.xml");
final ObjectGridManager oGridManager=
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid employees = oGridManager.createObjectGrid(objectGridName,
allObjectGrids,
validateXML,
cacheInstance);
```

この XML ファイルには、いくつかの ObjectGrids の構成情報が含まれています。前のコードの断片は、具体的に ObjectGrid「Employees」を戻します。この場合、「Employees」構成は、そのファイルに定義されていることを前提とします。 XML の構文については、 270 ページの『ObjectGrid 構成』を参照してください。

7 つの createObjectGrid メソッドがあります。メソッドは、次のコード・ブロック内に文書化されています。

```
/**
* A simple factory method to return an instance of an
* Object Grid. A unique name is assigned.
* The instance of ObjectGrid is not cached.
* Users can then use {@link ObjectGrid#setName(String)} to change the
* ObjectGrid name.
*
* @return ObjectGrid an instance of ObjectGrid with a unique name assigned
* @throws ObjectGridException any error encountered during the ObjectGrid creation
* @ibm-api
*/
public ObjectGrid createObjectGrid() throws ObjectGridException;
/**
* A simple factory method to return an instance of an ObjectGrid with the
* specified name. The instances of ObjectGrid can be cached. If an ObjectGrid
* with the this name has already been cached, an ObjectGridException
* will be thrown.*
* @param objectGridName the name of the ObjectGrid to be created.
* @param cacheInstance true, if the ObjectGrid instance should be cached
* @return an ObjectGrid instance
* @this name has already been cached or
* any error during the ObjectGrid creation.
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, boolean cacheInstance)
throws ObjectGridException;
/**
* Create an ObjectGrid instance with the specified ObjectGrid name. The
* ObjectGrid instance created will be cached.
* @param objectGridName the Name of the ObjectGrid instance to be created.
* @return an ObjectGrid instance
* @throws ObjectGridException if an ObjectGrid with this name has already
* been cached, or any error encountered during the ObjectGrid creation
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName)
throws ObjectGridException;
/**
* Create an ObjectGrid instance based on the specified ObjectGrid name and the
* XML file. The ObjectGrid instance defined in the XML file with the specified
* ObjectGrid name will be created and returned. If such an ObjectGrid
* cannot be found in the xml file, an exception will be thrown.
*
* This ObjecGrid instance can be cached.
*
* If the URL is null, it will be simply ignored. In this case, this method behaves
* the same as {@link #createObjectGrid(String, boolean)}.
*
* @param objectGridName the Name of the ObjectGrid instance to be returned. It
* must not be null.
* @param xmlFile a URL to a wellformed xml file based on the ObjectGrid schema.
* @param enableXmlValidation if true the XML is validated
* @param cacheInstance a boolean value indicating whether the ObjectGrid
* instance(s)
* defined in the XML will be cached or not. If true, the instance(s) will
* be cached.*
* @throws ObjectGridException if an ObjectGrid with the same name
```

```
* has been previously cached, no ObjectGrid name can be found in the xml file,
* or any other error during the ObjectGrid creation.
* @return an ObjectGrid instance
* @see ObjectGrid
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, final URL xmlFile,
final boolean enableXmlValidation, boolean cacheInstance) throws
ObjectGridException;
/**
* Process an XML file and create a List of ObjectGrid objects based
* upon the file.
* These ObjecGrid instances can be cached.
* An ObjectGridException will be thrown when attempting to cache a
* newly created ObjectGrid
* that has the same name as an ObjectGrid that has already been cached.
*
* @param xmlFile the file that defines an ObjectGrid or multiple
* ObjectGrids
* @param enableXmlValidation setting to true will validate the XML
* file against the schema
* @param cacheInstances set to true to cache all ObjectGrid instances
* created based on the file
* @return an ObjectGrid instance
* @throws ObjectGridException if attempting to create and cache an
* ObjectGrid with the same name as
* an ObjectGrid that has already been cached, or any other error
* occurred during the
* ObjectGrid creation
* @ibm-api
*/
public List createObjectGrids(final URL xmlFile,
final boolean enableXmlValidation,
boolean cacheInstances)
throws ObjectGridException;
/*** Create all ObjectGrids that are found in the XML file. The XML file will be
* validated against the schema. Each ObjectGrid instance that is created will
* be cached. An ObjectGridException will be thrown when attempting to cache a
* newly created ObjectGrid that has the same name as an ObjectGrid that has
* already been cached.
* @param xmlFile The XML file to process. ObjectGrids will be created based
* on what is in the file.
* @return A List of ObjectGrid instances that have been created.
* @throws ObjectGridException if an ObjectGrid with the same name as any of
* those found in the XML has already been cached, or
* any other error encounterred during ObjectGrid creation.
* @ibm-api
*/
public List createObjectGrids(final URL xmlFile) throws ObjectGridException;
/**
* Process the XML file and create a single ObjectGrid instance with the
* objectGridName specified only if an ObjectGrid with that name is found in
* the file. If there is no ObjectGrid with this name defined in the XML file,
* an ObjectGridException
* will be thrown. The ObjectGrid instance created will be cached.
* @param objectGridName name of the ObjectGrid to create. This ObjectGrid
* should be defined in the XML file.
* @param xmlFile the XML file to process
* @return A newly created ObjectGrid
* @throws ObjectGridException if an ObjectGrid with the same name has been
* previously cached, no ObjectGrid name can be found in the xml file,
* or any other error during the ObjectGrid creation.
* @ibm-api
*/
public ObjectGrid createObjectGrid(String objectGridName, URL xmlFile)
throws ObjectGridException;
```

getObjectGrid メソッド

getObjectGrid メソッドを使用して、キャッシュ付きインスタンスを検索します。

キャッシュ付きインスタンスの検索

Employees ObjectGrid インスタンスは、 ObjectGridManager によってキャッシュに入れられるため、その他のすべてのユーザーは、以下のコードの断片を使ってアクセス可能です。

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

キャッシュ付き ObjectGrid インスタンスを戻す 2 つの getObjectGrid メソッドを次に示します。

```
/**
* Get a List of the ObjectGrid instances that have been previously cached.
* Returns null if no ObjectGrid instances have been cached.
* @return a List of ObjectGrid instances that have been previously cached
* @ibm-api
*/
public List getObjectGrids();
/**
* Use this if a ObjectGrid already exists. It returns a cached
* ObjectGrid instance by name. This method returns null if no
* ObjectGrid with this objectGridName has been cached.
*
* @param objectGridName the cached objectgrid name.
* @return a cached ObjectGrid which currently exists.
*
* @since WAS XD 6.0
* @ibm-api
*
*/
public ObjectGrid getObjectGrid(String objectGridName);
```

removeObjectGrid メソッド

このトピックは、2 つの removeObjectGrid メソッドの使用法について説明します。

ObjectGrid インスタンスの除去

キャッシュから ObjectGrid インスタンスを除去するには、 removeObjectGrid メソッドの 1 つを使用します。ObjectGridManager は、除去されたインスタンスの参照は保持しません。 2 つの 除去メソッドが存在します。 1 つのメソッドはブール値パラメーターを取ります。ブール値パラメーターが **true** に設定されている場合は、破棄メソッドが ObjectGrid に対して呼び出されます。 ObjectGrid に対して破棄メソッドが呼び出されると、 ObjectGrid はシャットダウンし、使用しているリソースをすべて解放します。以下に、2 つの removeObjectGrid メソッドの使用法について説明します。

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

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

getObjectGridAdministrator メソッド

クラスターの ObjectGridAdministrator インスタンスを返します。

```
public ObjectGridAdministrator getObjectGridAdministrator(ClientClusterContext ctx)
/**
* Return an ObjectGridAdministrator instance for this cluster. Each
* cluster will require the use of a different ObjectGridAdministrator.
*
* @param clientClusterContext. A unique cluster context, with which the client
* needs to interact.
* @param objectGridName the cached objectgrid name.
* @return an ObjectGrid
*
* @since WAS XD 6.0.1
* @ibm-api
*
*/
public ObjectGridAdministrator getObjectGridAdministrator(ClientClusterContext
ontext);
```

このメソッドについての詳細は、 71 ページの『第 7 章 システム管理の概要』を参照してください。

ObjectGridManager インターフェースを使用した ObjectGrid インスタンスのライフ・サイクルの制御

このトピックでは、ObjectGridManager インターフェースを使用して、スタートアップ Bean およびサーブレットを使用している ObjectGrid インスタンスのライフ・サイクルを制御する方法を説明します。

スタートアップ Bean での ObjectGrid インスタンスのライフ・サイクルの管理

スタートアップ Bean は、ObjectGrid インスタンスのライフ・サイクルの制御に使用できます。スタートアップ Bean はアプリケーションの開始時にロードします。スタートアップ Bean では、アプリケーションが予想通りに開始または停止するときにはいつでもコードを実行できます。スタートアップ Bean を作成するために、ホームの com.ibm.websphere.startupservice.AppStartUpHome インターフェースを使用し、また、リモート側の com.ibm.websphere.startupservice.AppStartUp インターフェースを使用します。 Bean で start メソッドおよび stop メソッドを実行します。 start メソッドは、アプリケーションの始動時に必ず起動されます。stop メソッドは、アプリケーションのシャットダウン時に必ず起動されます。start メソッドを使用して、ObjectGrid インスタンスを作成できます。 stop メソッドを使用して、ObjectGrid インスタンスを破棄できます。以下は、スタートアップ Bean の ObjectGrid のライフ・サイクル管理を示すコードの断片です。

```
public class MyStartupBean implements javax.ejb.SessionBean {
private ObjectGridManager objectGridManager;
/*
* The methods on the SessionBean interface have been
* left out of this example for the sake of brevity
*/
public boolean start(){
// Starting the startup bean
// This method is called when the application starts
objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
try {
// create 2 ObjectGrids and cache these instances
ObjectGrid bookstoreGrid =
objectGridManager.createObjectGrid("bookstore", true);
bookstoreGrid.defineMap("book");
ObjectGrid videostoreGrid =
objectGridManager.createObjectGrid("videostore", true);
// within the JVM,
// these ObjectGrids can now be retrieved from the
//ObjectGridManager using the getObjectGrid(String) method
} catch (ObjectGridException e) {
e.printStackTrace();
return false;
}
return true;
}
public void stop(){
// Stopping the startup bean
// This method is called when the application is stopped
try {
// remove the cached ObjectGrids and destroy them
objectGridManager.removeObjectGrid("bookstore", true);
objectGridManager.removeObjectGrid("videostore", true);
} catch (ObjectGridException e) {
e.printStackTrace();
}
}
}
```

start メソッドが呼び出された後、新規に作成された ObjectGrid インスタンスを、ObjectGridManager から取得できます。例えば、サーブレットがアプリケーションに含まれる場合、サーブレットは以下のコードの一部を使用する ObjectGrids にアクセスできます。

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

サーブレットの ObjectGrid ライフ・サイクルの管理

サーブレットで ObjectGrid のライフ・サイクルを管理する場合に、 init メソッドで ObjectGrid インスタンスを作成し、destroy メソッドで ObjectGrid を破棄する方法をとることができます。 ObjectGrid インスタンスがキャッシュされた場合、サーブレット・コードで検索および操作が行えます。以下は、サーブレット内での ObjectGrid の作成、操作、および破棄を示すサンプル・コードの一部です。

```
public class MyObjectGridServlet extends HttpServlet implements Servlet {
private ObjectGridManager objectGridManager;
public MyObjectGridServlet() {
super();
}
public void init(ServletConfig arg0) throws ServletException {
super.init();
```

```
objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
try {
// create and cache an ObjectGrid named bookstore
ObjectGrid bookstoreGrid =
objectGridManager.createObjectGrid("bookstore", true);
bookstoreGrid.defineMap("book");
} catch (ObjectGridException e) {
e.printStackTrace();
}
}
protected void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
ObjectGrid bookstoreGrid = objectGridManager.getObjectGrid("bookstore");
BackingMap bookMap = bookstoreGrid.getMap("book");
// perform operations on the cached ObjectGrid
// ...
}
public void destroy() {
super.destroy();
try {
// remove and destroy the cached bookstore ObjectGrid
objectGridManager.removeObjectGrid("bookstore", true);
} catch (ObjectGridException e) {
e.printStackTrace();
}
}
}
```

ObjectGrid のトレース

このトピックでは、ObjectGrid のトレースのセットアップ方法について説明します。

Java 2 Platform, Standard Edition (J2SE) 環境

デバッグ情報を IBM に送る必要がある場合は、デバッグ・トレースを入手するためにトレース機構を使用します。 J2SE 環境におけるデバッグ・トレースの入手方法の例を以下に示します。

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

前述の例には、ObjectGrid 用組み込み Evictor プラグインのトレースは含まれません。 ObjectGrid によって提供される 1 つ以上の Evictor プラグインを使用しており、除去に関連する可能性のある問題がある場合は、以下の例で示すように ObjectGrid と ObjectGrid の Evictor の両方に対するトレースを使用可能にします。

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

WebSphere Application Server 環境

WebSphere Application Server 環境では、ObjectGridManager を使用してトレースを設定する必要はありません。管理コンソールを使用してトレースの指定を設定します。

ObjectGrid クライアント接続 API

基本原則

クライアントは、クラスター内のアクティブ (実行中の)・サーバー・プロセスに接続します。クライアントは、最低限でも接続先のサーバーのホスト名とポート番号を必要とします。ホスト名およびポート情報は、サーバーを最初に始動するときに使用されたクラスター定義 XML ファイル内にあります。 XML 構成について詳しくは、 270 ページの『ObjectGrid 構成』を参照してください。以下に、このセクションのサンプルとして使用されるクラスター XML 定義の断片を示します。クライアントは、このセクションで説明する API を使用して、クライアント要求を受信して処理するよう構成されているリモート ObjectGrid に接続します。クライアントは、ObjectGrid とクラスター XML 定義の両方を「ダウンロード」し、クライアント自体をブートストラップすることに注意してください。クライアント構成は、接続先のサーバー構成に基づいています。

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

このクラスター定義は完全ではありませんが、ここで説明する例としては十分です。 cluster1 クラスター内には、server1、server2、および server3 サーバーという 3 つのサーバーがあります。clientAccessPort 属性は、サーバーが listen するリスナー・ポート、およびクライアントが最初に接続を確立するポートを指定します。前述の XML の断片では、server1、server2、および server3 サーバーのポートは、それぞれ 12503、12504、および 12504 になります。

接続 API

ObjectGridManager インターフェースには、以下の例で示されている接続メソッドがあります。以下の接続 API は、ObjectGridManager インターフェースから使用することができます。これらのメソッドについての詳細は、API 文書を参照してください。

```
/**
* This allows a client to connect to a remote ObjectGrid
* The remote ObjectGrid is hosted as specified by the parameters:
* @param clusterName: The name of the cluster to which this client
* attaches itself
* @param host: The host on which to connect
* @param port: The clientAceess port that is listening
* @param ClientSecurityConfiguration: Security configuration,can be
* null if security is not configured
* @param overRideObjectGrid xml. This parameter can be null. If it is not null,
* the client side configuration of ObjectGrid plug-in is overridden.
* Not all plug-ins can be overridden. For details see the
* ObjectGrid documents
* @throws ConnectException
*
*/
public ClientClusterContext connect(String clusterName,
String host,
String port,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
/**
*
* @param clusterName
* @param attributes Host and Port pair attributes that are tried in sequential
* order to connect. If an attempt to connect fails to one server, the next pair
* of host and port attributes is picked to retry the connect.
* @param ClientSecurityConfiguration: Security configuration. It can be null if
* security is not configured.
* @param overRideObjectGrid xml. This parameter can be null. If it is not null,
* the client side configuration of ObjectGrid plug-in is overridden.
* Not all plug-ins can be overridden. For details see the ObjectGrid documents.
* @return ClientClusterContext
* @throws ConnectException
*
*
*/
public ClientClusterContext connect(String clusterName,
HostPortConnectionAttributes[] attributes,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
/**
* This method can be used only if a client is colocated with
* an ObjectGrid server, especially in a Java 2 Platform,
* Enterprise Edition (J2EE) environment with IBM WebSphere
* Application Server, which supports the
* embedded ObjectGrid server.
* This method connects the client to the Server which is running
* in the same Java virtual machine (JVM).
* @param securityProps. It can be null if not running in secure mode.
* @param overRideObjectGrid xml. This parameter can be null. If it is
* not null, the client side configuration of ObjectGrid plug-in is overridden.
* Not all plug-ins can be overridden. For details see the
* ObjectGrid documents
* @return ClientClusterContext
* @throws ConnectException
*
*/
public ClientClusterContext connect(ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException;
/**
* This allows a client to connect to a Remote ObjectGrid
* @param clusterConfigFile A URL to the clusterConfig File. This is the
* same file that is used to start servers.
* This is used to retrieve host port information. It cannot be null. If it is
* null a IllegalArgumentException exception results.
* @param serverName A String, the name of the specific server to connect to.
```

```
*If the server name is not in the configuration, IllegalArgumentException
* results.
* This parameter can be null, in which case an attempt is made to connect
* to one of servers specified in the cluster
* XML file. If an attempt fails to connect to one, another server is picked,
* It is done, until such time the list is exhausted.
* @param securityProps
* @param overRideObjectGrid xml. This parameter can be null. If it is not
* null, the client side configuration of ObjectGrid plug-in is overridden.
* Not all plug-ins can be overridden. For details, see the ObjectGrid
* documents
* @return ClientClusterContext
* @throws ConnectException
*
* @ibm-api
*/
public ClientClusterContext connect(URL clusterConfigFile,
String serverName,
ClientSecurityConfiguration securityProps,
URL overRideObjectGrid) throws ConnectException ;
```

ホストおよびポート・パラメーターを使用した例

以下のコードでは、103 ページの『基本原則』で説明されているクラスター XML が使用されます。このクライアントは、ホスト s1.myco.com にポート 12503 で接続します。

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

1. ObjectGridManagerFactory から ObjectGridManager singleton オブジェクトを取得します。
2. 接続 API を呼び出します。

3. objectGrid 従業員がリモート ObjectGrid に存在すると想定し、ClientClusterContext パラメーターを渡して getObjectGrid メソッドを呼び出します。
4. 切断メソッドを呼び出します。作業が完了した場合、最後のステップとして、すべてのクライアントが切断を呼び出す必要があります。これは非常に重要なステップです。

ConnectException 例外の場合に接続を自動的に再試行するための複数のホストの提供

この例では、HostPortConnectionAttributes 属性を使用して、クライアントが接続できるホスト・ポート属性の配列を指定します。 API では、このホストとポートのペア属性を順番に使用して接続を行います。 1 つのサーバーに対する接続が失敗すると、次のホストとポート属性のペアが選択されて、接続が再試行されます。

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

同一プロセスにおけるクライアントとサーバー

クライアントがサーバーと同じ JVM 内にある場合、以下の接続メソッドを使用することができます。

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

クラスター XML の指定

クライアントがクラスター XML ファイルへのアクセス権を持っている場合は、ホスト名とポート番号を指定する必要がありません。この API は、サーバー名とポー

ト番号を検索し、接続に使用します。サーバー名はオプションであり、NULL にすることができます。NULL の場合、API は、クラスター XML ファイルに定義されているサーバーのいずれかに対して接続を試行します。

```
ctx=oGridManager.connect(urlToClusterxml,"server1",null,null);
// connect to server1
// or
ctx=oGridManager.connect(urlToClusterxml,null,null,null);
//connect to any server in the cluster
```

接続 API でのクライアント・セキュリティー

すべての例において、ClientSecurityConfiguration パラメーターは NULL になっていました。NULL 値を引き渡すと、セキュリティーは使用不可になります。セキュリティーが使用可能の場合は、ClientSecurityConfiguration オブジェクトを引数として渡します。詳しくは、144 ページの『ObjectGrid セキュリティー』を参照してください。

ObjectGrid XML 構成のオーバーライド

クライアントは、サーバーから ObjectGrid 定義を「ダウンロード」し、クライアント自体を構成します。ObjectGrid に定義されているすべてのプラグインが、クライアントに対して使用可能になります。基本的に、サーバー・サイド ObjectGrid と通信するローカル ObjectGrid がクライアント・サイドに存在します。接続 API にオーバーライド XML ファイルを指定すると、クライアントでのみ使用されるプラグイン構成を「オーバーライド」することができます。このようなプラグインには、以下のようなものがあります。

ObjectGrid プラグイン:

- TransactionCallback プラグイン
- ObjectGridEventListener プラグイン

BackingMap プラグイン:

- Evictor プラグイン
- MapEventListener プラグイン

オーバーライド XML に定義されているその他のプラグインは、無視されます。

例

クライアントが特定の BackingMap に対する Evictor 構成をオーバーライトする必要があるとします。つまり、クライアント・サイドの Evictor が、サーバー・サイドで構成されている Evictor とは異なるものである必要があります。

サーバー・サイドの Evictor は、以下のように使用されるものとします。組み込み Evictor の LFUEvictor が使用されます。

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

クライアント・サイドでの要件は異なります。クライアント・サイドでは、ユーザー定義の myco.og.MyEvictor Evictor を使用することが必要です。オーバーライド

XML には、以下のような断片を組み込むことができます。LFUEvictor を使用するように構成されているすべての backingMaps が、以下のユーザー定義を使用します。

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

完全な XML

以下の XML コードは、サーバー・サイドで使用される XML ファイルとクライアント用の XML ファイルの 2 つの XML ファイルを示しています。この構成では、クライアントは、BackingMap の Evictor 構成をオーバーライドできます。

サーバー・サイド ObjectGrid XML ファイル

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

接続中にオーバーライドするクライアント・サイド ObjectGrid XML ファイル

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

アプリケーションの設計上の考慮事項

クライアントの connect() API は、高価な操作です。作業内容に応じて、クライアントは、単一サーバーへの 1 つ以上の物理接続を確立します。クライアント接続数は、クラスター構成 XML 定義のクラスター・エレメント内に定義されている、tcpMinConnections 属性と tcpMaxConnections 属性で指定されている値の間で変化します。この定義は、サーバーの始動に使用されたものと同じクラスター XML 定義になります。接続マネージャーでは、これらの物理接続をプールし、ObjectGrid は必要に応じてこれらの接続を再利用します。tcpMinConnections および tcpMaxConnections 属性は、単一サーバーのみに対するクライアント接続数を指定します。クライアントが複数のサーバーに接続する場合、クライアント接続の最大数は、tcpMaxConnections 属性にクライアントが接続するサーバー数を掛けた値以下になります。例えば、クライアントが 3 つのサーバーに接続する場合、tcpMaxConnections が 5 に指定されていると、クライアントの最大接続回数は (5*3)=15 となり、tcpMinConnection が 1 に設定されていると、最小接続回数は 3 回となります。接続は、すべてのクライアント間で共有されます。

threadsPerClientConnect 属性は、ワーカー・スレッド数を指定します。これらのワーカー・スレッドによって、物理接続中に作業がディスパッチされます。ワーカー・スレッドでは、構成データ、クライアント要求、サーバー応答、およびシステム管理要求が処理されます。ワーカー・スレッド数のデフォルト値は 5 です。この属性は、最初の iFix 内で使用可能です。iFix が使用不可の場合は、-Dthreads Java 仮想マシン (JVM)・システム・プロパティーを使用して、ワーカー・スレッド数を指定します。アプリケーションによっては、この数を増やすとパフォーマンスが向上することがあります。一般的に、この数は、クライアントが使用する物理接続数以上にする必要があります。

ご使用のアプリケーション設計で許可されている場合、クライアントは、*ClientClusterContext* メソッドを再利用することにより、複数のスレッドを作成して 1 回の接続呼び出しで作業を完了することができます。

ObjectGrid インターフェース

このトピックは、 ObjectGrid の変更に必要なメソッドを参照する目的に使用してください。

概要

ObjectGrid は、Java Map インターフェースに基づいた拡張可能なトランザクションのオブジェクト・キャッシング・フレームワークです。 ObjectGrid API の操作は、トランザクションの作業単位にグループ化され、カスタム設計のプラグイン・サポートを通じて拡張できるようになっています。 ObjectGrid は、多数の BackingMap を含む名前付き論理コンテナーです。バッキング・マップについての詳細は、 114 ページの『BackingMap インターフェース』を参照してください。

作成と初期化

ObjectGrid インスタンスの作成に必要なステップについては、 ObjectGridManager インターフェースのトピックを参照してください。 ObjectGrid を作成する方法としては、プログラマチックな方法と、 XML 構成ファイルを使用した方法の 2 つがあります。詳しくは、 95 ページの『ObjectGridManager インターフェース』を参照してください。

get メソッドまたは set メソッドとファクトリー・メソッド

重要: どの *set* メソッドも、ObjectGrid インスタンスを初期化する前に呼び出す必要があります。初期設定メソッドを呼び出した後に set メソッドを呼び出すと、`java.lang.IllegalStateException` が発生します。 ObjectGrid インターフェースの各 getSession メソッドでも初期設定メソッドが暗黙的に呼び出されます。このため、いずれの getSession メソッドを呼び出す前にも、set メソッドを呼び出す必要があります。ただし、EventListener オブジェクトの設定、追加、および除去の場合のみは、このルールの例外となります。これらのオブジェクトは、「初期化」処理が完了した後に処理することができます。

ObjectGrid インターフェースには、以下のようなメソッドがあります。

表 7. ObjectGrid インターフェースのメソッド

メソッド	説明
BackingMap defineMap(String name);	*defineMap*: 一意的に名付けた BackingMap を定義するファクトリー・メソッドです。バッキング・マップについての詳細は、114 ページの『BackingMap インターフェース』を参照してください。
BackingMap getMap(String name);	*getMap*: *defineMap* を呼び出して、以前に定義された BackingMap を戻します。このメソッドを使用すると、XML 構成でまだ構成されていない場合に BackingMap を構成することができます。
BackingMap createMap(String name);	*createMap*: BackingMap を作成しますが、この ObjectGrid での使用のために BackingMap をキャッシュしません。このメソッドは、この ObjectGrid での使用のために BackingMaps をキャッシュする、ObjectGrid インターフェースの setMaps(List) メソッドと共に使用してください。これらのメソッドは、Spring Framework を使用して ObjectGrid を構成する場合に使用します。
void setMaps(List mapList);	*setMaps*: 以前この ObjectGrid で定義されたすべての BackingMaps をクリアし、提供されている BackingMaps のリストに置き換えます。
public Session getSession() throws ObjectGridException, TransactionCallbackException;	*getSession*: セッションを戻します。このセッションは、作業単位の開始、コミット、ロールバックの機能を提供します。セッション・オブジェクトについての詳細は、119 ページの『Session インターフェース』を参照してください。
Session getSession(CredentialGenerator cg);	*getSession(CredentialGenerator cg)*: CredentialGenerator オブジェクトを使用してセッションを取得します。このメソッドは、クライアント/サーバー環境の ObjectGrid クライアントによってのみ呼び出すことができます。

表 7. *ObjectGrid* インターフェースのメソッド (続き)

メソッド	説明
Session getSession(Subject subject);	*getSession(Subject subject)*: Session を取得するために、ObjectGrid で構成されている Subject オブジェクトではなく、特定の Subject オブジェクトの使用を許可します。
void initialize() throws ObjectGridException;	*initialize*: ObjectGrid は初期化され、汎用可能です。このメソッドは、ObjectGrid が初期化済み状態にない場合、 `getSession` メソッドが呼び出されると暗黙的に呼び出されます。
void destroy();	*destroy*: このメソッドが呼び出された後は、フレームワークは分解されて使用できません。
void setTxTimeout(int timeout);	*setTxTimeout*: この ObjectGrid インスタンスが作成した Session によって開始されたトランザクションに許可されている完了までの時間 (秒単位) を設定するとき、このメソッドを使用します。指定された時間内にトランザクションが完了しなかった場合は、トランザクションを開始した Session が「タイムアウト」としてマークされます。 Session がタイムアウトとしてマークされると、タイムアウトになった Session によって起動される次の ObjectMap メソッドで、 `com.ibm.websphere.objectgrid.TransactionTimeoutException` 例外が発生します。Session が「*ロールバックのみ*」としてマークされると、`TransactionTimeoutException` 例外がアプリケーションによってキャッチされた後に、アプリケーションがロールバック・メソッドではなくコミット・メソッドを呼び出した場合でも、トランザクションはロールバックされます。 タイムアウト値 `0` は、トランザクションの完了までに許可される時間が無制限であることを示します。タイムアウト値 `0` が使用されると、トランザクションはタイムアウトになりません。このメソッドが呼び出されない場合、このインターフェースの getSession メソッドによって返されるすべての Session のトランザクション・タイムアウト値は、デフォルトで `0` に設定されます。アプリケーションは、com.ibm.websphere.objectgrid.Session インターフェースの setTransactionTimeout メソッドを使用して、Session ごとにトランザクション・タイムアウト設定をオーバーライドできます。

表 7. *ObjectGrid* インターフェースのメソッド (続き)

メソッド	説明
int getTxTimeout();	*getTxTimeout*: トランザクション・タイムアウト値 (秒単位) を返します。このメソッドは、setTxTimeout メソッドのタイムアウト・パラメーターとして渡されるものと同じ値を返します。 setTxTimeout メソッドが呼び出されなかった場合、このメソッドは `0` を返し、トランザクションの完了までに許可されている時間が無制限であることを示します。
//Keywords.	
void associateKeyword(Serializable parent, Serializable child);	*associateKeyword*: ObjectGrid キーワードは、キーワードに基づいた柔軟な無効化手段を提供します。キーワードの詳細については、128 ページの『キーワード』を参照してください。このメソッドは、方向関係で 2 つのキーワードを結び付けます。親が無効な場合は、その子も無効になります。子を無効にしても親には影響しません。
//Security	
void setSecurityEnabled()	*setSecurityEnabled*: セキュリティーを使用可能にします。セキュリティーは、デフォルトでは使用不可です。
void setPermissionCheckPeriod(long period);	*setPermissionCheckPeriod*: このメソッドは、アクセス権を検査する頻度を示す単一パラメーターです。アクセス権は、クライアント・アクセスの許可に使用されます。パラメーターが `0` である場合、すべてのメソッドは、許可機構 (JAAS 許可、カスタム許可のどちらか) に問い合わせて、現行サブジェクトがアクセス権を持っているかどうかを確認します。この方法は、許可の実装に依存するので、パフォーマンスに問題を生じる可能性があります。しかし、このタイプの許可は、必要に応じて有効です。あるいは、パラメーターが `0` 未満である場合は、許可機構に戻して更新を行うまでに、アクセス権のセットをキャッシュに入れておくミリ秒数を示しています。このパラメーターはパフォーマンスをかなり向上させますが、その間にバックエンド・アクセス権が変更されると、バックエンドのセキュリティー・プロバイダーが変更されていても、 ObjectGrid はアクセスを許可されたり、防止されたりします。
void setAuthorizationMechanism(int authMechanism);	*setAuthorizationMechanism*: 許可機構を設定します。デフォルトは `SecurityConstants.JAAS_AUTHORIZATION` です。

表 7. *ObjectGrid* インターフェースのメソッド *(続き)*

メソッド	説明
setMapAuthorization(MapAuthorization ma);	*setMapAuthorization*: この ObjectGrid インスタンスに対する MapAuthorization プラグインを設定します。このプラグインを使用すると、Subject オブジェクトに含まれているプリンシパルに対する ObjectMap または JavaMap アクセスを許可することができます。このプラグインの通常の実装は、Subject オブジェクトからプリンシパルを検索し、指定されたアクセス権がプリンシパルに付与されているかどうかを確認することです。
setSubjectSource(SubjectSource ss);	*setSubjectSource*: SubjectSource プラグインを設定します。このプラグインを使用すると、ObjectGrid クライアントを表す Subject オブジェクトを取得できます。このサブジェクトは、ObjectGrid 許可に使用されます。ObjectGrid.getSession メソッドを使用してセッションを取得するとき、セキュリティーが使用可能になっている場合は、ObjectGrid ランタイムによって SubjectSource.getSubject メソッドが呼び出されます。このプラグインは、既に認証済みのクライアントの場合に役立ちます。つまり、このプラグインは、認証済み Subject オブジェクトを検索して、ObjectGrid インスタンスに渡すことができます。他の認証は必要ありません。
setSubjectValidation(SubjectValidation sv);	*setSubjectValidation*: この ObjectGrid インスタンスに対する SubjectValidation プラグインを設定します。このプラグインを使用すると、ObjectGrid に渡される javax.security.auth.Subject サブジェクトが、改ざんされていない有効なサブジェクトであるかどうかを検証できます。 Subject オブジェクトが改ざんされたかどうかは作成者のみが知っているため、このプラグインの実装には、Subject オブジェクト作成者からのサポートが必要です。ただし、サブジェクト作成者が、Subject が改ざんされたかどうかを関知していない場合もあります。この場合、このプラグインは使用しないようにしてください。

ObjectGrid インターフェース: プラグイン

ObjectGrid インターフェースには、対話をさらに拡張可能にするための、いくつかのオプショナル・プラグイン・ポイントがあります。

```
void addEventListener(ObjectGridEventListener cb);
void setEventListeners(List cbList);
void removeEventListener(ObjectGridEventListener cb);
void setTransactionCallback(TransactionCallback callback);
```

```
int reserveSlot(String);
// Security related plug-ins
void setSubjectValidation(SubjectValidation subjectValidation);
void setSubjectSource(SubjectSource source);
void setMapAuthorization(MapAuthorization mapAuthorization);
```

- *ObjectGridEventListener*: ObjectGridEventListener インターフェースは、 ObjectGrid で重大なイベントが発生したときに通知を受け取るために使用します。イベントの例として、ObjectGrid の初期化、トランザクションの開始、トランザクションの終了、および ObjectGrid の破棄などがあります。これらのイベントを listen するには、 ObjectGridEventListener インターフェースを実装するクラスを作成して、ObjectGrid に追加します。これらのリスナーは、各 Session に関連付けられています。詳しくは、 194 ページの『リスナー』および 119 ページの『Session インターフェース』を参照してください。
- *TransactionCallback*: TransactionCallback リスナー・インターフェースを使用すると、開始、コミット、およびロールバック・シグナルなどのトランザクション・イベントは、このインターフェースを送信することができます。一般に、TransactionCallback リスナー・インターフェースは、ローダーと併用されます。詳しくは、 226 ページの『TransactionCallback プラグイン』と 210 ページの『ローダー』を参照してください。これらのイベントは、外部リソースと一緒に、または複数のローダー内で、トランザクションを調整する目的で使用できます。
- *reserveSlot*: この ObjectGrid のプラグインが、TxID のようなスロットを持つオブジェクト・インスタンスでの使用のためにスロットを予約できるようにします。
- *SubjectValidation*: セキュリティーが使用可能である場合、このプラグインは、ObjectGrid に渡される `javax.security.auth.Subject` クラスの妥当性検査に使用できます。
- *MapAuthorization*: セキュリティーが使用可能である場合、このプラグインは、サブジェクト・オブジェクトによって表されるプリンシパルへの ObjectMap アクセスを許可する目的で使用できます。
- *SubjectSource*: セキュリティーが使用可能である場合、このプラグインは、ObjectGrid クライアントを表すサブジェクト・オブジェクトを取得する目的で使用できます。そうすると、このサブジェクトは ObjectGrid 許可に使用されます。

BackingMap インターフェース

各 ObjectGrid インスタンスは、BackingMap オブジェクトのコレクションを含みます。

各 BackingMap は、ObjectGrid インターフェースの defineMap メソッドを使用することによって指定され、ObjectGrid インスタンスに追加されます。これらのメソッドは BackingMap インスタンスを戻し、このインスタンスは個々の Map の振る舞いを定義するために使用されます。詳しくは、 109 ページの『ObjectGrid インターフェース』を参照してください。

Session インターフェースを使用してトランザクションを開始し、アプリケーションと BackingMap オブジェクト間のトランザクション対話を実行するのに必要な ObjectMap または JavaMap を取得します。ただし、トランザクションの変更は、トランザクションがコミットされるまで、BackingMap オブジェクトに適用されません。 BackingMap は、個々の Map のためにコミットされたデータのメモリー内キ

ャッシュとして構成することができます。 Session インターフェースについての詳細は、『Session インターフェース』を参照してください。

com.ibm.websphere.objectgrid.BackingMap インターフェースは、 BackingMap 属性を設定するためのメソッドを提供します。一部の set メソッドを使用して、いくつかのカスタム設計されたプラグインを介して BackingMap を拡張できます。以下は、属性を設定し、カスタム設計されたプラグイン・サポートを提供するための、set メソッドのリストです。

```
// For setting BackingMap attributes.
public void setReadOnly(boolean readOnlyEnabled);
public void setNullValuesSupported(boolean nullValuesSupported);
public void setLockStrategy( LockStrategy lockStrategy );
public void setCopyMode(CopyMode mode, Class valueInterface);
public void setCopyKey(boolean b);
public void setNumberOfBuckets(int numBuckets);
public void setNumberOfLockBuckets(int numBuckets);
public void setLockTimeout(int seconds);
public void setTimeToLive(int seconds);
public void setTtlEvictorType(TTLType type);
// For setting an optional custom plug-in provided by application.
public abstract void setObjectTransformer(ObjectTransformer t);
public abstract void setOptimisticCallback(OptimisticCallback checker);
public abstract void setLoader(Loader loader);
public abstract void setPreloadMode(boolean async);
public abstract void setEvictor(Evictor e);
public void setMapEventListeners( List /*MapEventListener*/ eventListenerList );
public void addMapEventListener(MapEventListener eventListener );
public void removeMapEventListener(MapEventListener eventListener );
public void addMapIndexPlugin(MapIndexPlugin index);
public void setMapIndexPlugins(List /* MapIndexPlugin */ indexList );
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb);
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback cb);
public void removeDynamicIndex(String name);
```

リストされている各 set メソッドには、対応する get メソッドが存在します。

BackingMap 属性

各 BackingMap には、BackingMap の振る舞いを変更または制御するために設定可能な以下の属性があります。

- *ReadOnly* 属性。この属性は、Map が読み取り専用 Map であるか、読み取り/書き込み Map であるかを示します。この属性が Map に設定されていない場合は、Map は読み取り/書き込み Map にデフォルトで設定されています。 BackingMap が読み取り専用で設定されている場合、ObjectGrid は可能な場合のみ読み取りのパフォーマンスを最適化します。
- *NullValuesSupported* 属性。この属性は、Map にヌル値を設定可能であるかどうかを示します。この属性が設定されていない場合は、Map はヌル値をサポートしません。 Map がヌル値をサポートする場合は、ヌルを戻す get 操作は、値がヌルであること、またはマップが get 操作によって指定されたキーを含まないことを意味します。
- *LockStrategy* 属性。この属性は、この BackingMap がロック・マネージャーを使用するかどうかを判別します。ロック・マネージャーを使用している場合は、LockStrategy 属性を使用して、マップ・エントリーのロッキングにオプティミスティック・ロッキングまたはペシミスティック・ロッキングのいずれの方法が使用されているか示します。この属性が設定されていない場合は、オプティミステ

ィック LockStrategy が使用されます。サポートされるロック・ストラテジーに関する詳細については、 134 ページの『ロック』のトピックを参照してください。

- *CopyMode* 属性。この属性は、値オブジェクトのコピーが、値がマップから読み込まれる際に BackingMap によって作成されるか、トランザクションのコミット・サイクル中に BackingMap へ設定されるかどうかを判別します。さまざまなコピー・モードがサポートされて、アプリケーションがパフォーマンスとデータ保全性の間でトレードオフを行うことを可能にします。この属性が設定されていない場合は、COPY_ON_READ_AND_COMMIT コピー・モードが使用されます。このコピー・モードでは、最良のパフォーマンスが上げられませんが、データ保全性の問題に対しては、最大限の保護が得られます。コピー・モードについての詳細は、『copyMode メソッドのベスト・プラクティス』を参照してください。
- *CopyKey* 属性。この属性は、マップに初めてエントリーが作成されたときに、BackingMap がキー・オブジェクトのコピーを作成するかどうかを判別します。キーは通常変更不可能なオブジェクトであるため、デフォルトのアクションでは、キー・オブジェクトのコピーは作成されません。
- *NumberOfBuckets* 属性。この属性は、BackingMap の使用するハッシュ・バケット数を示します。 BackingMap インプリメンテーションは、その実装のためにハッシュ・マップを使用します。 BackingMap に多くのエントリーがある場合、バケットが多いほどパフォーマンスが良好であることを意味します。同じバケットを持つキーの数は、バケット数が増えるにつれて少なくなります。また、バケットを増やすことによって、並行性も増大します。この属性は、パフォーマンスを微調整する際に便利です。アプリケーションによって NumberOfBuckets 属性が設定されない場合は、デフォルト値 503 が使用されます。
- *NumberOfLockBuckets* 属性。この属性は、この BackingMap のロック・マネージャーの使用するロック・バケットの数を示します。 LockStrategy が OPTIMISTIC または PESSIMISTIC に設定されると、BackingMap にロック・マネージャーが作成されます。ロック・マネージャーは、ハッシュ・マップを使用して、1 つ以上のトランザクションによってロックされるエントリーを追跡します。ハッシュ・マップに多くのエントリーがある場合は、同じバケットについて衝突するキーの数がバケット数が増えるほど減少するため、ロック・バケットの数が多いほどパフォーマンスが良好になります。また、ロック・バケットが多いと並行性も高くなります。 LockStrategy 属性が NONE に設定されると、この BackingMap はロック・マネージャーを使用しません。この場合、numberOfLockBuckets を設定しても効果はありません。この属性が設定されていない場合は、デフォルト値 383 が使用されます。
- *LockTimeout* 属性。BackingMap がロック・マネージャーを使用していると、この属性が使用されます。 LockStrategy 属性が OPTIMISTIC または PESSIMISTIC のいずれかに設定されている場合、BackingMap はロック・マネージャーを使用します。属性値は、ロック・マネージャーがロックが与えられるまで待機する時間を秒単位で決定します。この属性が設定されない場合は、LockTimeout 値として 15 秒が使用されます。発生するロック待機タイムアウト例外についての詳細は、『ペシミスティック・ロック』を参照してください。
- *TtlEvictorType* 属性。各 BackingMap に、時間ベースのアルゴリズムを使用して除去対象となるマップ・エントリーを判別する、組み込み型存続時間 Evictor があります。デフォルトでは、組み込み型存続時間 Evictor はアクティブではありません。 3 つの値 (CREATION_TIME、LAST_ACCESS_TIME、または NONE) のうちの 1 つを使用して、 setTtlEvictorType メソッドを呼び出すと、存続時間 Evictor をア

クティブにできます。 CREATION_TIME の値は、Evictor が、TimeToLive 属性を、マップ・エントリーが BackingMap で作成された時間に追加して、 Evictor が BackingMap からマップ・エントリーを除去する時点を判別することを意味します。 LAST_ACCESS_TIME の値は、Evictor が、TimeToLive 属性をアプリケーションが稼働している何らかのトランザクションによってマップ・エントリーが最後にアクセスを受けた時間に追加して、 Evictor がマップ・エントリーを除去する時点を判別することを意味します。 TimeToLive 属性によって指定される期間に、マップ・エントリーがトランザクションのアクセスを受けない場合にのみ、マップ・エントリーは除去されます。値 NONE は、Evictor が非アクティブなままで、マップ・エントリーを除去しないことを示します。この属性が設定されないと、NONE はデフォルトとして使用され、存続時間 Evictor はアクティブになりません。組み込み型存続時間 Evictor についての詳細は、『Evictor』を参照してください。

- *TimeToLive* 属性。この属性を使用して、組み込み型存続時間 Evictor が、TtlEvictorType 属性に関して説明したように、それぞれのエントリーの作成または最後のアクセス時に追加する必要がある秒数を指定します。この属性が設定されていない場合は、特殊値ゼロを使用して、存続時間が無限大であることを示します。この属性が無限大に設定されると、マップ・エントリーは Evictor によって除去されません。

以下の例は、someGrid ObjectGrid インスタンスでの someMap BackingMap の定義および BackingMap インターフェースの set メソッドを使用した BackingMap のさまざまな属性の設定を示しています。

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

BackingMap プラグイン

BackingMap インターフェースには、ObjectGrid とより拡張性のある対話を行うためのオプションのプラグ・ポイントが複数あります。

- **ObjectTransformer プラグイン**: 一部のマップ操作の場合、BackingMap は、BackingMap 内のエントリーのキーまたは値をシリアライズ、デシリアライズ、またはコピーする必要があります。 BackingMap は、ObjectTransformer インターフェースのデフォルトのインプリメンテーションを提供することによって、これらのアクションを実行することができます。アプリケーションは、BackingMap が、BackingMap 内のキーまたは値をシリアライズ、デシリアライズ、またはコピーするために使用する、カスタム設計された ObjectTransformer プラグインを提供することで、パフォーマンスを改善できます。詳しくは、 222 ページの『ObjectTransformer プラグイン』を参照してください。

- **Evictor プラグイン** : 組み込み型存続時間 Evictor は、時間ベースのアルゴリズムを使用して、BackingMap 内のエントリーを除去する時点を判別します。一部のアプリケーションは、BackingMap 内のエントリーを除去する時点を判別するために、別のアルゴリズムを使用する場合があります。 Evictor プラグインは、カスタム設計された Evictor を有効にして、BackingMap が使用できるようにします。Evictor プラグインは、組み込み型存続時間 Evictor に追加されます。このプラグインは、存続時間 Evictor を置き換えません。 ObjectGrid は、「Least Recently Used (LRU)」や「Least Frequently Used (LFU)」のような、既知のアルゴリズムを実装するカスタム Evictor プラグインを提供します。アプリケーションは、提供される Evictor プラグインのいずれか 1 つをプラグインすることも、独自の Evictor プラグインを提供することもできます。詳しくは、 199 ページの『Evictor』を参照してください。
- **MapEventListener プラグイン**: アプリケーションは、マップ・エントリーの除去や BackingMap 完了のプリロードなどの BackingMap イベントを認識する必要がある場合があります。 BackingMap は、MapEventListener プラグインでメソッドを呼び出し、アプリケーションに BackingMap イベントを通知します。アプリケーションは、setMapEventListener メソッドを使用して、さまざまな BackingMap イベントの通知を受け取り、 BackingMap に 1 つ以上のカスタム設計された MapEventListener プラグインを提供することができます。アプリケーションは、addMapEventListener メソッドまたは removeMapEventListener メソッドを使用して、リストされた MapEventListener オブジェクトを変更することができます。詳しくは、 197 ページの『MapEventListener インターフェース』を参照してください。
- **Loader プラグイン** : BackingMap は Map のメモリー内キャッシュです。Loader プラグインは、BackingMap がメモリー間でデータを移動させるために使用するオプションであり、 BackingMap の永続ストアに使用されます。例えば、Java Database Connectivity (JDBC) Loader を使用して、 BackingMap とリレーショナル・データベースの 1 つ以上のリレーショナル・テーブルとの間でデータを入れたり取り出したりすることができます。リレーショナル・データベースは、BackingMap の永続ストアとして使用する必要はありません。この Loader はまた、BackingMap と 1 つのファイル間、BackingMap と Hibernate マップ間、BackingMap と Java 2 Platform, Enterprise Edition (J2EE) Entity Bean 間、および BackingMap と他のアプリケーション・サーバー間などで、データを移動させるために使用できます。アプリケーションは、使用されるすべての技術に関して、BackingMap と永続ストアの間でデータを移動させるために、カスタム設計された Loader プラグインを提供する必要があります。 Loader が提供されない場合は、BackingMap は単純なメモリー内キャッシュになります。このプラグインについての詳細は、 210 ページの『ローダー』を参照してください。
- **OptimisticCallback プラグイン**: BackingMap の LockStrategy 属性は、OPTIMISTIC に設定され、BackingMap または Loader プラグインはマップの値に関する比較操作を実行する必要があります。 BackingMap および Loader は OptimisticCallback プラグインを使用して、オプティミスティック・バージョン管理の比較演算を実行します。詳しくは、 234 ページの『OptimisticCallback インターフェース』を参照してください。
- **MapIndexPlugin プラグイン**: MapIndexPlugin プラグイン (短縮名は Index) は、BackingMap が、格納されているオブジェクトの指定された属性に基づいて索引をビルドする場合に使用するオプションです。索引によって、アプリケーション

は、特定の値または値の範囲を使用してオブジェクトを検索することができます。索引には、静的と動的の 2 つのタイプがあります。詳しくは、 248 ページの『索引付け』を参照してください。

Session インターフェース

このセクションでは、アプリケーションが Session インターフェースを使用してトランザクションを開始および終了する方法について説明します。 Session インターフェースは、 ObjectMap および JavaMap インターフェースをベースにしたアプリケーションにもアクセスできます。

概要

各 ObjectMap または JavaMap インスタンスは、特定のセッション・オブジェクトに直接結合しています。ObjectGrid にアクセスする各スレッドは、最初に ObjectGrid オブジェクトからセッションを取得しなければなりません。セッション・インスタンスは、スレッド間で同時に共有することはできません。 ObjectGrid はスレッドのローカル・ストレージをまったく使用しませんが、プラットフォームの制約事項により、スレッド間でセッションを受け渡しする機会が制限されることがあります。

メソッド

以下のメソッドが Session インターフェースで使用できます。以下のメソッドについての詳細は、API 文書を参照してください。

```
public interface Session {
ObjectMap getMap(String cacheName)
throws UndefinedMapException;
void begin()
throws TransactionAlreadyActiveException, TransactionException;
void beginNoWriteThrough()
throws TransactionAlreadyActiveException, TransactionException;
public void commit()
throws NoActiveTransactionException, TransactionException;
public void rollback()
throws NoActiveTransactionException, TransactionException;
public void flush()
throws TransactionException; ObjectGrid getObjectGrid();
TxID getTxID()
throws NoActiveTransactionException;
boolean isWriteThroughEnabled();
void setTransactionType(String tranType);
public void processLogSequence(LogSequence logSequence)
throws NoActiveTransactionException, UndefinedMapException, ObjectGridException;

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

get メソッド

アプリケーションは、ObjectGrid.getSession メソッドを使用して ObjectGrid オブジェクトからセッション・インスタンスを取得します。以下のコード・スニペットは、セッション・インスタンスの取得方法を示しています。

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

セッションを取得した後、スレッドはそのセッションへの参照を専用に保持します。 getSession メソッドを複数回呼び出すと、その度に新規セッション・オブジェクトが戻されます。

トランザクション・メソッドとセッション・メソッド

セッションは、トランザクションの開始、コミット、またはロールバックに使用できます。 ObjectMaps と JavaMaps を使用した BackingMaps に対する操作は、セッション・トランザクション内では非常に効率よく実行されます。トランザクションが開始された後は、そのトランザクションの有効範囲にある 1 つ以上の BackingMaps に対するすべての変更は、そのトランザクションがコミットされるまで、特別のトランザクション・キャッシュに保管されます。トランザクションがコミットされると、保留になっている変更内容は BackingMaps とローダーに適用され、その ObjectGrid のその他のクライアントから見えるようになります。

ObjectGrid は、トランザクションを自動的にコミットする機能 (自動コミットともいう) もサポートします。すべての ObjectMap オペレーションがアクティブ・トランザクションのコンテキストの外部で実行される場合は、暗黙のトランザクションはそのオペレーションの前に開始され、そのトランザクションはアプリケーションに制御が戻される前に自動的にコミットされます。

```
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
```

Session.flush メソッド

Session.flush メソッドは、ローダーが BackingMap に関連付けられているときにのみ意味があります。 flush メソッドは、トランザクション・キャッシュ内の変更内容の現行セットを使用してローダーを呼び出します。ローダーは、変更内容をバックエンドに適用します。これらの変更内容は、flush が呼び出されるときはコミットされません。 flush 呼び出しの後、セッション・トランザクションがコミットされると、 flush 呼び出しの後で発生する更新のみがローダーに適用されます。 flush 呼び出しの後、セッション・トランザクションがロールバックされると、フラッシュされた変更内容はトランザクション内のその他すべての保留している変更内容と一緒に廃棄されます。 flush メソッドは、ローダーに対するバッチ操作の機会を制限するので、慎重に使用してください。以下は、Session.flush メソッドの使用例です。

```
Session session = objectGrid.getSession();
session.begin();
// make some changes
```

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

ノー・ライトスルー・メソッド

一部の ObjectGrid マップは、ローダーによって戻されます。ローダーはマップ内のデータに、永続ストレージを提供します。ObjectGrid マップのみにデータをコミットし、ローダーにデータをプッシュしないことが有益な場合があります。 Session インターフェースは、この目的のために beginNoWriteThough メソッドを提供します。 beginNoWriteThrough メソッドは、begin メソッドのようなトランザクションを開始します。 beginNoWriteThrough メソッドでは、トランザクションがコミットされると、データは ObjectGrid のメモリー内のマップにのみコミットされ、ローダーが提供する永続ストレージにはコミットされません。このメソッドが非常に役立つのは、データがマップにプリロードされるときです。

分散 ObjectGrid インスタンスを使用する場合、サーバーで遠くのキャッシュは変更せずに近くのキャッシュのみを変更するには、beginNoWriteThrough メソッドが役立ちます。近くのキャッシュでデータが不整合であると認識されている場合は、beginNoWriteThrough メソッドを使用すると、エントリーをサーバーでは無効にせずに、近くのキャッシュで無効にすることができます。

Session インターフェースは、現在活動中のトランザクション・タイプを判別する isWriteThroughEnabled メソッドも提供します。

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

TxID オブジェクト・メソッドの取得

TxID オブジェクトは、内部が見えないオブジェクトで、活動中のトランザクションを識別します。以下の目的には、TxID オブジェクトを使用します。

- ある特定のトランザクションを検索している場合の比較用
- TransactionCallback とローダーのオブジェクト間で共用データを保管するため

オブジェクト・スロット・フィーチャーについての追加情報は、 226 ページの『TransactionCallback プラグイン』と 210 ページの『ローダー』を参照してください。

パフォーマンス・モニター・メソッド用トランザクション・タイプの設定

WebSphere Application Server のアプリケーション・サーバー内で ObjectGrid を使用している場合、パフォーマンス・モニター用にトランザクション・タイプをリセットする必要があることがあります。トランザクション・タイプの設定には、setTransactionType メソッドを使用できます。 setTransactionType メソッドについての詳細は、 308 ページの『WebSphere Application Server Performance Monitoring Infrastructure (PMI) を使用した ObjectGrid パフォーマンスのモニター』を参照してください。

完全な LogSequence メソッドの処理

ObjectGrid は、1 つの Java 仮想マシン (JVM) から別のマシンへマップを配布する手段として、マップ変更のセットをほかの ObjectGrid リスナーに広めることができます。 リスナーが受信済み LogSequences を処理するのを容易にするために、Session インターフェースは processLogSequence メソッドを提供します。このメソッドは LogSequence 内で 各 LogElement を検査し、 LogSequence MapName によって識別される BackingMap に対して適切なオペレーション (例えば、挿入、更新、無効化など) を実行します。 ObjectGrid セッションは、processLogSequence メソッドが呼び出される前に、活動中でなければなりません。アプリケーションは、セッションを完了するために適切な commit または rollback 呼び出しを実行する役割があります。自動コミット処理は、このメソッド呼び出しには使用できません。

リモート JVM で受信している ObjectGridEventListener による通常の処理は、beginNoWriteThrough メソッドを使用してセッションを開始することになります。これにより、変更内容のエンドレスな伝搬が防止され、この processLogSequence メソッドに対する呼び出しが続き、トランザクションのコミットまたはロールバックが実行されます。

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

markRollbackOnly メソッド

このメソッドを使用して、現行トランザクションに ″rollbackOnly″ とマークを付けます。トランザクションに ″rollbackOnly″ とマークを付けると、アプリケーションで commit メソッドが呼び出された場合でも、トランザクションはロールバックされます。このメソッドは、通常、トランザクションのコミットが許可されている場合にデータ破壊が発生する可能性があると認識されているとき、ObjectGrid 自体またはアプリケーションで使用されます。

このメソッドが呼び出された後に、このメソッドに渡される Throwable オブジェクトが、以前に ″rollbackOnly″ とマーク付けされたセッションで commit メソッドが呼び出された場合の結果である `com.ibm.websphere.objectgrid.TransactionException` 例外にチェーニングされます。すでに ″rollbackOnly″ とマーク付けされているトランザクションのこのメソッドに対する以降の呼び出しは、無視されます。つまり、ヌル以外の Throwable 参照を渡す最初の呼び出しのみが使用されます。マークされたトランザクションが完了すると、″rollbackOnly″ マークは除去されるため、セッションで開始される次のトランザクションはコミットされます。

isMarkedRollbackOnly メソッド

セッションが現在 ″rollbackOnly″ とマークされている場合に戻されます。markRollbackOnly メソッドが以前このセッションで呼び出されており、セッション

で開始されたトランザクションがアクティブな場合、かつこの場合に限り、このメソッドによってブール値 true が戻されます。

setTransactionTimeout メソッド

このセッションで開始される次のトランザクションのトランザクション・タイムアウトを特定の秒数に設定します。このメソッドは、このセッションで以前に開始されたトランザクションのトランザクション・タイムアウトには影響を与えません。このメソッドが呼び出された後に開始されたトランザクションにのみ影響を与えます。このメソッドが呼び出されない場合は、com.ibm.websphere.objectgrid.ObjectGrid メソッドの setTxTimeout メソッドに渡されたタイムアウト値が使用されます。

getTransactionTimeout メソッド

このメソッドは、トランザクション・タイムアウト値 (秒単位) を戻します。タイムアウト値として setTransactionTimeout メソッドに渡された最後の値は、このメソッドによって戻されます。 setTransactionTimeout メソッドが呼び出されない場合は、com.ibm.websphere.objectgrid.ObjectGrid メソッドの setTxTimeout メソッドに渡されたタイムアウト値が使用されます。

transactionTimedOut

このメソッドは、このセッションで開始された現行トランザクションがタイムアウトになると、ブール値 true を戻します。

isFlushing メソッド

このメソッドは、呼び出されたセッション・インターフェースの flush メソッドの結果として、すべてのトランザクション変更がローダー・プラグインにフラッシュされる場合、かつこの場合に限り、ブール値 true を戻します。ローダー・プラグインでは、batchUpdate メソッドが呼び出された理由を確認する必要がある場合にこのメソッドが役立ちます。

isCommitting メソッド

このメソッドは、呼び出されたセッション・インターフェースの commit メソッドの結果として、すべてのトランザクション変更がコミットされる場合、かつこの場合に限り、ブール値 true を戻します。ローダー・プラグインでは、batchUpdate メソッドが呼び出された理由を確認する必要がある場合にこのメソッドが役立ちます。

ObjectMap および JavaMap インターフェース

このトピックでは、アプリケーションが ObjectMap および JavaMap インターフェースを使用して ObjectGrid と対話する方法を説明しています。この 2 つのインターフェースは、アプリケーションと BackingMaps との間のトランザクション対話のために使用されます。

ObjectMap インターフェース

ObjectMap インスタンスが、現行スレッドと対応するセッション・オブジェクトから獲得されます。 ObjectMap インターフェースは、BackingMap 内のエントリーを変更するためにアプリケーションが使用するメイン媒体です。

ObjectMap インスタンス

アプリケーションは、Session.getMap(String) メソッドを使用して、セッション・オブジェクトから ObjectMap インスタンスを入手します。以下のコードの断片は、ObjectMap インスタンスの獲得方法を示すものです。

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

各 ObjectMap インスタンスは、特定のセッション・オブジェクトと対応しています。特定のセッション・オブジェクトで同じ BackingMap 名を使用して getMap メソッドを複数回呼び出すと、常に同じ ObjectMap インスタンスを戻します。

自動コミット・トランザクション

前述のとおり、ObjectMaps および JavaMaps を使用する BackingMaps に対する操作が、セッション・トランザクション内で最も効果的に実行されます。 ObjectGrid は、ObjectMap および JavaMap インターフェース上のメソッドがセッション・トランザクションの外部で呼び出されたときに、autocommit サポートを提供します。メソッドは、暗黙のトランザクションを開始し、要求された操作を実行し、その暗黙のトランザクションをコミットします。

メソッド・セマンティクス

以下で、ObjectMap インターフェースおよび JavaMap インターフェース上の各メソッドの背後にあるセマンティクスについて説明します。 setDefaultKeyword メソッド、invalidateUsingKeyword メソッド、およびシリアライズ可能な引数を持つメソッドについては、 128 ページの『キーワード』のトピックで解説しています。
setTimeToLive メソッドについては、 199 ページの『Evictor』のトピックで解説しています。これらのメソッドの詳細については、API 資料を参照してください。

containsKey メソッド
: キーが BackingMap または Loader に値を持っているかどうかを判別します。アプリケーションが NULL 値をサポートしている場合は、このメソッドは get 操作から戻された NULL 参照が NULL 値を参照しているのか、BackingMap および Loader がキーを含んでいないことを示すのかを判別するために使用できます。

flush メソッド
: このメソッドのセマンティクスは、Session インターフェース上の flush メソッドと類似しています。注意すべき相違点は、セッション・フラッシュが、現行セッション内で変更されたすべてのマップの現行の保留変更点を適用するということです。このメソッドでは、この ObjectMap における変更点のみがローダーにフラッシュされます。

get メソッド

BackingMap からエントリーを取り出します。 BackingMap 内でエントリーが検出されず、Loader が BackingMap と関連付けられている場合は、Loader からエントリーを取り出そうとします。 getAll メソッドは、バッチ取り出し処理を許可するために提供されます。

getForUpdate メソッド

getforUpdate メソッドは get メソッドと同じですが、getForUpdate メソッドを使用すると、BackingMap および Loader に対し、エントリーを更新することが目的であることを指示します。Loader はこのヒントを使用して、データベース・バックエンドに「SELECT for UPDATE」照会を発行できます。ペシミスティック LockingStrategy が BackingMap に定義されている場合は、ロック・マネージャーがエントリーをロックします。 getAllForUpdate メソッドは、バッチ取り出し処理を許可するために提供されます。

insert メソッド

エントリーを BackingMap および Loader に挿入します。このメソッドを使用して、BackingMap および Loader に、これまで存在していないエントリーを挿入したいということを知らせます。既存のエントリー上でこのメソッドを起動すると、メソッドが起動されるとき、あるいは現行のトランザクションがコミットされるときに例外が発生します。

invalidate メソッド

invalidate メソッドのセマンティクスは、このメソッドに渡される **isGlobal** パラメーターに依存します。 invalidateAll メソッドは、バッチ無効化処理を許可するために提供されます。

invalidate メソッドの **isGlobal** パラメーターとして値 *false* が渡されるとき、ローカル無効化が指定されます。ローカル無効化は、トランザクション・キャッシュ内のエントリーへのいかなる変更も破棄します。アプリケーションが get メソッドを発行した場合、エントリーは BackingMap 内でコミットされた最後の値から取り出されます。 BackingMap 内にエントリーがない場合は、ローダー内で最後にフラッシュされたかまたはコミットされた値から、エントリーが取り出されます。トランザクションがコミットされているとき、ローカルに無効化されているとマークされたエントリーはいずれも BackingMap に影響を与えません。ローダーにフラッシュされたすべての変更は、エントリーが無効化された場合であってもコミットされます。

invalidate メソッドの **isGlobal** パラメーターとして *true* が渡されるとき、グローバル無効化が指定されます。グローバル無効化は、トランザクション・キャッシュ内のエントリーに対するすべての保留中の変更を破棄し、エントリー上で実行された以降の操作で BackingMap 値をバイパスします。トランザクションがコミットされているとき、グローバルに無効化されているとマークされたエントリーはいずれも BackingMap から除去されます。

以下の無効化のユース・ケースを例として考えます。 BackingMap が自動増分列を持つデータベース表から戻されます。増分列はレコードに固有の番号を割り当てるために有効です。アプリケーションはエントリーを挿入します。挿入の後で、アプリケーションは挿入された行のシーケンス番号を認識しておく必要があります。オブジェクトのコピーが古いことが分かると、グローバル無効化を使用して Loader から値を入手します。以下のコードはこのユース・ケースを説明しています。

```
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();
```

このサンプル・コードは、*Billy* にエントリーを追加します。 Person のバージョン属性が、データベースの自動増分列を使用して設定されます。アプリケーションは、最初に挿入コマンドを実行します。次にフラッシュを発行して、挿入を Loader およびデータベースに送信します。データベースはこのバージョン列をシーケンスの次の番号に設定します。これによりトランザクション内の Person オブジェクトが期限切れになります。このオブジェクトを更新するために、アプリケーションはグローバル無効化を実行します。発行される次の get メソッドは、トランザクションの値を無視するローダーからエントリーを入手します。エントリーは、更新されたバージョン値を持つデータベースから取り出されます。

put メソッド

put メソッドのセマンティクスは、前の get メソッドがキーに対するトランザクション内で起動されたかどうかに依存します。アプリケーションが BackingMap またはローダー内の既存のエントリーを戻す get 操作を発行する場合、put メソッドの起動は更新として解釈され、トランザクション内の前の値を戻します。前に get メソッドが起動されていなかった場合の put メソッドの起動や、前に get メソッドが起動されたけれどもエントリーが見つからなかった場合の put メソッドの起動は、挿入と解釈されます。挿入および更新メソッドのセマンティクスが、put 操作がコミットされると適用します。バッチ挿入と更新処理を使用可能にするために、putAll メソッドが提供されます。

remove メソッド

接続された場合、BackingMap および Loader からエントリーを除去します。除去されたオブジェクトの値は、このメソッドによって戻されます。そのオブジェクトが存在していない場合、このメソッドはヌル値を戻します。戻り値なしでバッチ削除処理を使用可能にするために、removeAll メソッドが提供されます。

setCopyMode メソッド

この ObjectMap の CopyMode を指定します。このメソッドを使用して、アプリケーションは、BackingMap 上で指定された CopyMode をオーバーライドすることができます。指定された CopyMode は、clearCopyMode メソッドが起動されるまで有効です。いずれのメソッドも、トランザクションの境界の外側で起動されます。 CopyMode はトランザクションの途中で変更することはできません。

touch メソッド

エントリーの最後のアクセス時間を更新します。このメソッドは、BackingMap からの値を検索しません。このメソッドは、自身のトランザクション内で使用します。無効化または除去のために BackingMap 内に提供されたキーが存在しない場合は、コミット処理中に例外が発生します。

update メソッド

BackingMap および Loader 内のエントリーを明示的に更新します。このメソッドを使用して、BackingMap および Loader に、既存のエントリーを更新したいことを示します。存在していないエントリー上でこのメソッドを起動すると、メソッドが起動されるとき、あるいはコミット処理中に例外が発生します。

getIndex メソッド

BackingMap に作成されている名前付き索引を取得しようとします。この索引は、スレッド間で共用することができず、セッションと同じ規則に基づいて機能します。戻された索引オブジェクトは、MapIndex インターフェース、MapRangeIndex インターフェース、またはカスタム索引インターフェースなどの正しいアプリケーション索引インターフェースにキャストする必要があります。

JavaMap インターフェース

JavaMap インスタンスは ObjectMap オブジェクトから獲得されます。 JavaMap インターフェースは、 ObjectMap と同じメソッド・シグニチャーを持ちますが、例外処理の方法は異なります。JavaMap は java.util.Map インターフェースを拡張します。したがって、すべての例外は java.lang.RuntimeException クラスのインスタンスになります。 JavaMap は java.util.Map インターフェースを拡張するので、オブジェクト・キャッシュ用に java.util.Map インターフェースを使用する既存のアプリケーションで簡単に ObjectGrid を使用できます。

JavaMap インスタンスの取得

アプリケーションは ObjectMap.getJavaMap メソッドを使用して ObjectMap オブジェクトから JavaMap インスタンスを入手します。以下のコードの断片は、JavaMap インスタンスの獲得方法を示すものです。

```
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;
```

JavaMap は、JavaMap の獲得元である ObjectMap によって戻されます。特定の ObjectMap を使用して複数回 getJavaMap を呼び出すと、常に同じ JavaMap インスタンスを戻します。

サポートされるメソッド

JavaMap インターフェースは java.util.Map インターフェース上のメソッドのサブセットのみをサポートします。 java.util.Map インターフェースは以下のメソッドをサポートします。

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

java.util.Map インターフェースから継承されたその他のすべてのメソッドは、java.lang.UnsupportedOperationException 例外を生じます。

キーワード

ObjectGrid はキーワードに基づいた柔軟な無効化の機構を提供します。キーワードは、任意の直列化可能オブジェクトの非ヌル・インスタンスです。選択した任意の方法で、キーワードを BackingMap エントリーと関連付けることができます。

キーワードとエントリーの関連付け

一連のエントリーはゼロ以上のキーワードに関連付けることができます。 get メソッド、update メソッド、put メソッド、insert メソッド、および touch メソッド等を含むエントリーを操作する ObjectMap と JavaMap のすべてのメソッドは、単一のキーワードを、メソッドにより変更されるすべてのエントリーに関連付けることができるバージョンを持っています。トランザクションがコミットされるまでは、新規キーワードの関連付けは現行のトランザクション内でのみ見ることができます。コミットの後、新しい関連付けは BackingMap に適用され、別のトランザクションでも見ることができるようになります。コミット処理中にエラーが発生してロールバックが起こったり、ユーザーがアクティブなトランザクションをロールバックしたりした場合、新規キーワードの関連付けはロールバックされます。次のコードは、新規エントリーがキーワードに関連付けられる方法を示しています。

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

前のサンプル・コードでは、新規エントリーを BackingMap に挿入し、それを、キーワード "New York" と関連付けています。エントリーを挿入するアプリケーションも、エントリーが取得されたときにキーワードと関連付けられる必要があります。アプリケーションは、キーワードを得るたびにエントリーと関連付ける必要があります。次のサンプル・コードで検討します。

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

前のサンプル・コードでは、取得されたエントリーがキーワード "New York" に関連付けられることを保証します。アプリケーションは複数のキーワードをエントリーに関連付けることができますが、各メソッド呼び出しにつき関連付けられるのは 1 つのキーワードのみです。さらに多くのキーワードと関連付けるため、次のサンプルのように、他のメソッド呼び出しを行います。

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

デフォルトのキーワード

ObjectMap および JavaMap の setDefaultKeyword メソッドは、エントリーを特定のキーワードに、get、 insert、put、update、または touch メソッドのキーワー

ド・バージョンを使用することなく関連付ける方法を提供します。そのメソッドのキーワード・バージョンを使用した場合、デフォルトのキーワードは無視され、提供されたキーワード・オブジェクトが使用されます。

```
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();
```

前の例で、Billy はデフォルトのキーワード ″New York″ に関連付けられています。 Bob は、デフォルトのキーワードには関連付けられていません。これは、明示的キーワードが get 呼び出しに渡され、エントリー Bob を取得するためです。″Jimmy″ と関連付けられたキーワードはありません。これは、デフォルトのキーワードがリセットされ、明示的なキーワードの引数が get メソッドの呼び出しに渡されたからです。

キーワードによるエントリーの無効化

ObjectMap および JavaMap インターフェースで invalidateUsingKeyword メソッドを使用して、対応する BackingMap でキーワードを関連付けているすべてのエントリーを無効にします。この方法で、1 回の操作で関連するエントリーを効率的に無効化できます。

```
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();
```

上記の例では、″Billy″ のエントリーは無効になり、BackingMap に挿入されません。 invalidateUsingKeyword メソッド呼び出し後に挿入されるので、″Bob″ のエントリーは無効化されません。 invalidateUsingKeyword メソッドはメソッドが呼び出されたとき、キーワードの関連付けに基づくエントリーを無効にします。

キーワードのグループ化

キーワードは、親子関係にまとめてグループ化することもできます。親キーワードは、複数の子を持つことができ、子キーワードは複数の親を持つことができます。例えば、アプリケーションがキーワード ″Dublin″、″Paris″、″New York″、および ″Los Angeles″ を使用する場合、そのアプリケーションは以下のようなキーワードのグループ化を追加できます。

- ″USA″ は、″New York″ および ″Los Angeles″ をグループ化します。
- ″Europe″ は ″Dublin″ および ″Paris″ をグループ化します。
- ″World″ は ″USA″ および ″Europe″ をグループ化します。

キーワード ″USA″ を無効化すると、″New York″ および ″Los Angeles″ キーワードに関連付けられているすべてのエントリーが無効化されます。キーワード ″World″ を無効化すると、″USA″ と ″Europe″ のグループ化に関連したすべてのエントリーが無効になります。キーワードの関連付けは、ObjectGrid インターフェースの associateKeyword メソッドを使用して定義されます。 invalidateUsingKeyword メソッド呼び出し後の親キーワードへの子キーワードの追加によって、子キーワード

に関連したエントリーが無効化されることはありません。次のコード・サンプルは、記述されている一連のキーワードの関連付けを定義します。

```
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 および LogSequence オブジェクト

アプリケーションがトランザクション中にマップに変更を加えた場合、LogSequence オブジェクトはこれらの変更を追跡します。アプリケーションがマップ内のエントリーを変更する場合、対応する LogElement が存在し、変更の詳細を提供します。アプリケーションがフラッシュを必要とするか、トランザクションにコミットすると必ず、特定のマップのための LogSequence オブジェクトにローダーが提供されます。ローダーは、LogSequence 内の LogElements で繰り返され、各 LogElement をバックエンドへ適用します。

ObjectGrid に登録された ObjectGridEventListeners も LogSequence オブジェクトを使用します。これらのリスナーには、コミット済みトランザクションの各マップに LogSequence オブジェクトが提供されます。アプリケーションはこれらのリスナーを使用して、従来のデータベースでのトリガーのような、変更に対する特定のエントリーを待機できます。

このトピックでは、 4 つのログ関連のインターフェースまたは ObjectGrid フレームワークによって提供されるクラスについて説明します。

- `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 インターフェース

LogElement は、トランザクション中のエントリーに関する操作を示します。LogElement オブジェクトには、以下の属性があります。最も一般的に使用される属性は、以下に示した *type* および *current value* 属性です。

type **属性**
: ログ・エレメントである *type* は、このログのエレメントが表している操作の種類を示します。 *type* は、`LogElement` インターフェースで定義される定数 (`INSERT`、`UPDATE`、 `DELETE`、`EVICT`、`FETCH`、または `TOUCH`) のいずれかになります。

undo type **属性**
: トランザクションがマップ・エントリーに対して行った以前の変更の「取り消し」を、どの操作で実行する必要があるのかを戻します。

current value **属性**
: *current value* は、`INSERT`、 `UPDATE` または `FETCH` 操作の新規の値を表しま

す。操作が TOUCH、DELETE、または EVICT の場合、 current value はヌルになります。 ValueInterface が使用中である場合、この値を ValueProxyInfo へキャストできます。

CacheEntry **属性**
LogElement から CacheEntry オブジェクトを参照することができ、CacheEntry オブジェクトで定義されたメソッドを使用して必要な情報を検索できます。

pending state **属性**
pending state が true である場合、このログ・エレメントによって表される変更は、まだローダーに適用されていません。 false である場合は、多くの場合フラッシュ操作によって、変更はローダーに適用されています。

versioned value **属性**
versioned value は、バージョン管理に使用できる値です。

new keywords **属性**
new keyword のコレクションは、このエントリーに関連付けられている新規キーワードをすべて含んでいます。

last access time **属性**
エントリーの最終アクセス時間を表します。

before image / after image **属性**
マップに変更を加える前と加えた後の値オブジェクトのイメージを取得するためには、 getter メソッドを使用できます。

LogSequence インターフェース

ほとんどのトランザクションで、マップ内の複数エントリーに対する操作が行われるため、複数の LogElement オブジェクトが作成されます。複数の LogElement オブジェクトのコンポジットとして動作するオブジェクトを持つことには意味があります。 LogSequence インターフェースは、LogElement オブジェクトのリストを含むことによってこの目的に対応します。 LogSequence インターフェースには以下のメソッドがあります。

size メソッド
指定された順序で LogElement オブジェクトの数を戻します。

getAllChanges メソッド
指定されたログ・シーケンスのすべての変更のイテレーターを戻します。

getPendingChanges メソッド
すべての保留中の変更のイテレーターを戻します。これは主に、ローダーが、保留中の変更のみを永続ストアへ適用するために使用します。

getChangesByKeys メソッド
入力パラメーターに基づいて、ターゲット・キーを持つ LogElement オブジェクトのイテレーターを戻します。

getChangesByTypes メソッド
指定された LogElement タイプである LogElement オブジェクトのイテレーターを戻します。

getMapName メソッド
変更が適用されるバッキング・マップの名前を戻します。呼び出し元は、`Session.getMap(string)` メソッドへの入力としてこの名前を使用できます。

isDirty メソッド
この LogSequence に、マップをダーティー にする LogElements があるかどうかにかかわらず戻します。すなわち、LogSequence が Fetch または Get 以外のタイプの LogElement オブジェクトのいずれかを含んでいる場合、LogSequence は「ダーティー」であると見なされます。

isRollback メソッド
この LogSequence が生成されたのは、トランザクションをロールバックすることが目的であったのかどうかを戻します。

getObjectGridName メソッド
これらの変更が適用されるマップを収容する ObjectGrid の名前を戻します。

LogElement と LogSequence は、ObjectGrid や、操作が 1 つのコンポーネントまたはサーバーから別のコンポーネントまたはサーバーに伝搬されるときにユーザーによって作成された ObjectGrid プラグインによって、幅広く使用されています。例えば、LogSequence オブジェクトは、分散 ObjectGrid トランザクション伝搬機能によって変更を他のサーバーに伝えるために使用できます。あるいは、ローダーによってパーシスタンス・ストアに適用することもできます。 LogSequence は主に以下のインターフェースによって使用されます。

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

これらのインターフェースに関する詳細については、API 資料を参照してください。

ローダーの例

このセクションでは、LogSequence および LogElement オブジェクトがローダーで使用される方法について説明します。*ローダー* は、永続ストアからデータをロードし、また永続ストアにデータを保管するために使用します。ローダー・インターフェースの `batchUpdate` メソッドは、以下のように LogSequence を使用します。

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

ObjectGrid が現在のすべての変更をローダーに適用する場合、必ず `batchUpdate` メソッドが呼び出されます。ローダーには、マップのための LogElement オブジェクトのリストが、カプセル化されて LogSequence オブジェクトに与えられています。`batchUpdate` メソッドの実装は、変更を繰り返さなければならず、それらの変更をバックエンドに適用します。以下のコードの断片は、ローダーが LogSequence オブジェクトを使用する方法を示しています。このコードの断片は、一連の変更を繰り

返し、3 つのバッチ Java Database Connectivity (JDBC) ステートメントをビルドします。 1 つは INSERT ステートメント、もう 1 つは UPDATE ステートメント、3 つ目は DELETE ステートメントです。

```
public void batchUpdate(TxID tx, LogSequence sequence)
throws LoaderException
{
// Get a SQL connection to use.
Connection conn = getConnection(tx);
try
{
// Process the list of changes and build a set of prepared
// statements for executing a batch update, insert, or delete
// SQL operations. The statements are cached in stmtCache.
Iterator iter = sequence.getPendingChanges();
while ( iter.hasNext() )
{
LogElement logElement = (LogElement)iter.next();
Object key = logElement.getCacheEntry().getKey();
Object value = logElement.getCurrentValue();
switch ( logElement.getType().getCode() )
{
case LogElement.CODE_INSERT:
buildBatchSQLInsert( key, value, conn );
break;
case LogElement.CODE_UPDATE:
buildBatchSQLUpdate( key, value, conn );
break;
case LogElement.CODE_DELETE:
buildBatchSQLDelete( key, conn );
break;
}
}
// Run the batch statements that were built by above loop.
Collection statements = getPreparedStatementCollection( tx, conn );
iter = statements.iterator();
while ( iter.hasNext() )
{
PreparedStatement pstmt = (PreparedStatement) iter.next();
pstmt.executeBatch();
}
}
catch (SQLException e)
{
LoaderException ex = new LoaderException(e);
throw ex;
}
}
```

前のサンプルは、LogSequence 引数の処理の高水準ロジックを示していますが、SQL の INSERT、UPDATE、または DELETE ステートメントがビルドされる方法の詳細については示されていません。この例では、getPendingChanges メソッドが LogSequence 引数に呼び出され、ローダーが処理する必要のある LogElement オブジェクトのイテレーターを取得します。また、LogElement.getType().getCode() メソッドを使用して、LogElement が SQL の INSERT、UPDATE、 または DELETE 操作用であるかどうかを判断します。

Evictor の例

この例では、LogSequence および LogElement が Evictor で使用される方法について検討します。 Evictor は、特定の基準に基づいてバッキング・マップからマッ

プ・エントリーを除去するために使用します。 Evictor インターフェースの apply メソッドは、以下のように LogSequence を使用します。

```
/**
* This is called during cache commit to allow the evictor to track object usage
* in a backing map. This will also report any entries that have been successfully
* evicted.
*
* @param sequence LogSequence of changes to the map
*/
void apply(LogSequence sequence);
```

apply メソッドが LogSequence を使用する方法の詳細については、 199 ページの『Evictor』のトピックにあるサンプル・コードを参照してください。

LogSequenceFilter および LogSequenceTransformer インターフェース

場合によっては、特定の基準の LogElement オブジェクトのみを受け入れ、その他のオブジェクトを拒否するように、 LogElement オブジェクトをフィルターに掛ける必要があります。例えば、何らかの基準に基づいて、特定の LogElement を直列化したい場合があります。 LogSequenceFilter は、以下のメソッドでこの問題を解決します。

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

このメソッドは、操作で特定の LogElement を使用する必要がある場合は true を、その必要がない場合は false を返します。

LogSequenceTransformer は、上述した LogSequenceFilter 機能を使用するクラスです。 LogSequenceFilter を使用して一部の LogElement オブジェクトにフィルターを掛け、次に、その受け入れた LogElement オブジェクトを直列化します。このクラスには、2 つのメソッドがあります。最初のメソッドは以下のとおりです。

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

このメソッドにより、呼び出し元は、直列化プロセスに組み込む LogElements を判定するためのフィルターを提供できます。呼び出し元は、**DistributionMode** パラメーターを使用して直列化プロセスを制御します。例えば、分散モードが無効化のみである場合、値を直列化する必要はありません。このクラスの 2 番目のメソッドは以下のとおりです。

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

このメソッドは、 `serialize` メソッドによって作成されたログ・シーケンスの直列化済みフォームを、提供されたオブジェクト入力ストリームから読み取ります。

ロック

このトピックでは、ObjectGrid BackingMap のサポートするロック・ストラテジーについて説明します。

各 BackingMap は、以下のロックのストラテジーの 1 つを使用するよう構成できます。

- ペシミスティック・ロック
- オプティミスティック・ロック
- None

以下は、map1、map2、および map3 の BackingMap 上にロック・ストラテジーを設定する方法の例です。ここで、各マップは異なるロック・ストラテジーを使用しています。

```
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 );
```

java.lang.IllegalStateException 例外を避けるため、 ObjectGrid インスタンスで initialize メソッドまたは getSession メソッドを呼び出す前に setLockStrategy メソッドを呼び出す必要があります。

ペシミスティックまたはオプティミスティック・ロックのストラテジーのどちらかが使用される場合、BackingMap にロック・マネージャーが作成されます。ロック・マネージャーは、ハッシュ・マップを使用して、1 つ以上のトランザクションによってロックされるエントリーを追跡します。ハッシュ・マップに多くのマップ・エントリーが存在する場合、ロック・バケットが多いほど、パフォーマンスが良好であることを意味します。バケットの数が増えるに従って、Java の同期の衝突のリスクは下がります。またロック・バケットを増やすことが、並行性の増大につながります。以下の例は、特定の BackingMap に使用するロック・バケットの数をアプリケーションで設定できる方法を示しています。

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

ここでも、java.lang.IllegalStateException 例外を避けるため、ObjectGrid インスタンスの initialize メソッドまたは getSession メソッドを呼び出す前に setNumberOfLockBuckets メソッドを呼び出す必要があります。setNumberOfLockBuckets メソッド・パラメーターは、使用するロック・バケットの数を指定する Java のプリミティブ整数です。素数を使用すると、ロック・バケット上のマップ・エントリーの一様分布が確保されます。最良のパフォーマンスを得るために適した開始点は、BackingMap の予想される数のおおよそ 10 パーセントにロック・バケットの数を設定することです。

ペシミスティック・ロック

ほかのロック・ストラテジーが可能でない場合は、マップの読み書きにペシミスティック・ロック・ストラテジーを使用します。

ObjectGrid マップがペシミスティック・ロック・ストラテジーを使用するように構成されている場合、トランザクションが最初に BackingMap からのエントリーを取得すると、マップ・エントリーのペシミスティック・トランザクション・ロックが

取得されます。ペシミスティック・ロックは、アプリケーションがトランザクションを完了するまでは保留されます。通常の場合、ペシミスティック・ロック・ストラテジーは、以下の状態で使用されます。

- BackingMap がローダー付きまたはなしで構成され、バージョン管理情報が使用できません。
- BackingMap が、並行処理制御用の ObjectGrid からの支援を必要とするアプリケーションによって直接使用されています。
- バージョン管理情報は使用できるが、更新トランザクションがバッキング・エントリー上で頻繁に衝突し、その結果、オプティミスティック更新が失敗します。

ペシミスティック・ロック・ストラテジーは、パフォーマンスとスケーラビリティーに最大のインパクトを与えるので、このストラテジーはほかのロック・ストラテジーが見えないときのマップの読み取りと書き込みにのみ使用してください。例えば、オプティミスティック更新の失敗が頻繁に発生したり、オプティミスティック障害からのリカバリーがアプリケーションが処理するには難しい場合です。

ObjectMap メソッドとロック・モード

アプリケーションが ObjectMap インターフェースのメソッドを使用すると、ObjectGrid はアクセスするマップ・エントリーに対してペシミスティック・ロックを自動的に試行します。ObjectGrid は、アプリケーションが ObjectMap インターフェース内で呼び出すメソッドを基にした以下のロック・モードを使用します。

- `get` および `getAll` メソッドは、マップ・エントリーのキーに対する *S ロック*、つまり共用ロック・モードを獲得します。 S ロックは、トランザクションが完了するまで保留になります。 S ロック・モードでは、同一キーに対して S ロック・モードまたはアップグレード可能ロック (U ロック) モードを獲得しようとするトランザクション間での並行処理が許されますが、同一キーに対して排他的ロック (X ロック) モードを取得しようとする他のトランザクションはブロックされます。
- `getForUpdate` および `getAllForUpdate` メソッドは、マップ・エントリーのキーに対する *U ロック*、つまりアップグレード可能ロック・モードを獲得します。U ロックは、トランザクションが完了するまで保留になります。U ロック・モードでは、同一キーに対して S ロック・モードを獲得するトランザクション間での並行処理が許されますが、同一キーに対して U ロック・モードまたは X ロック・モードを獲得しようとする他のトランザクションはブロックされます。
- `put`、`putAll`、`remove`、`removeAll`、 `insert`、`update`、および `touch` は、マップ・エントリーのキーに対する *X ロック*、つまり排他的ロック・モードを獲得します。 X ロックは、トランザクションが完了するまで保留になります。 X ロック・モードでは、1 つのトランザクションのみが所定のキー値のマップ・エントリーを挿入、更新、または除去することになります。 X ロックは、同一キーに対する S、U、または X ロック・モードを獲得しようとする他のすべてのトランザクションをブロックします。
- `global invalidate` および `global invalidateAll` メソッドは、無効化されているそれぞれのマップ・エントリーごとに X ロックを獲得します。 X ロックは、トランザクションが完了するまで保留になります。 `local invalidate` および

`local invalidateAll` メソッドには、ロックは獲得されません。BackingMap エントリーのどれもが、 local invalidate メソッドの呼び出しによって無効化されることがないためです。

前の定義から、S ロック・モードが U ロック・モードよりも弱体であることは明白です。それは、同一マップ・エントリーにアクセスするときに、より多くのトランザクションが並行して実行されることを許すためです。 U ロック・モードは、S ロック・モードよりも少し強力です。それは、U ロック・モードまたは X ロック・モードのどちらかを要求している他のトランザクションをブロックするためです。S ロック・モードは、X ロック・モードを要求しているその他のトランザクションのみをブロックします。この小さな差が、一部のデッドロックの発生を防止するには重要です。 X ロック・モードは、最強のロック・モードです。これは、同一のマップ・エントリーに対して S、U、または X ロックのモードを取得しようとしているその他すべてのトランザクションをブロックするためにです。 X ロック・モードの最終的な作用は、1 つのトランザクションのみがマップ・エントリーを挿入、更新、または除去できるようにすることと、複数のトランザクションが同一のマップ・エントリーを更新しようとしているときに、更新が失われないようにすることです。

次表は、ロック・モードの互換性マトリックスです。説明済みのロック・モードをまとめたもので、互いに互換性のあるロック・モードはどれかを調べるのに使用できます。このマトリックスを読み取る場合、マトリックスの行は既に認可されているロック・モードを表します。列は、別のトランザクションによって要求されたロック・モードを表します。 **Yes** が列に表示されている場合は、他のトランザクションによって要求されたロック・モードは認可されています。それは、既に認可されているロック・モードと互換性があるためです。 **No** は、ロック・モードの互換性がないことを表し、その他のトランザクションは最初のトランザクションが保持しているロックを解放するのを待たなければなりません。

表 8. ロック・モード互換性と強度

ロック	互換性のあるロック			強度
	S (共用)	U (アップグレード可能)	X (排他的)	
S (共用)	はい	はい	いいえ	最弱
U (アップグレード可能)	はい	いいえ	いいえ	通常
X (排他的)	いいえ	いいえ	いいえ	最強

ロック待ちタイムアウト

各 ObjectGrid BackingMap には、デフォルトのロック待ちタイムアウト値があります。タイムアウト値は、アプリケーション・エラーによりデッドロック条件が発生したために、アプリケーションがロック・モードを認可されるのをいつまでも待つことがないように使用します。アプリケーションは、BackingMap インターフェースを使用して、デフォルトのロック待ちタイムアウト値をオーバーライドすることができます。以下の例は、map1 バッキング・マップのロック待ちタイムアウト値を 60 秒に設定する方法を示しています。

```
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 );
```

java.lang.IllegalStateException 例外を回避するには、 setLockStrategy メソッドと setLockTimeout メソッドの両方を呼び出してから、ObjectGrid インスタンスの initialize メソッドまたは getSession メソッドのいずれかを呼び出します。 setLockTimeout メソッドのパラメーターは、 Java プリミティブの整数で、ObjectGrid がロック・モードを認可されるのを待たなければならない秒数を指定します。 BackingMap に構成されているロック待ちタイムアウト値よりも長い時間、トランザクションが待つ場合、 com.ibm.websphere.objectgrid.LockTimeoutException 例外が発生します。

LockTimeoutException が発生したら、アプリケーションは、アプリケーションの実行が予想よりも遅くなっているためにタイムアウトが発生しているのか、それともデッドロック条件のためにタイムアウトが発生したのかを判別しなければなりません。実際にデッドロック条件が発生した場合は、ロック待ちタイムアウト値を増やしても例外は除去されません。タイムアウト値を増やすと、例外の発生期間が長くなります。しかし、ロック待ちタイムアウト値を増やして例外を除去している場合は、アプリケーションが予想よりも低速で実行されるために問題が発生しました。このケースのアプリケーションでは、パフォーマンスの低下原因を判別しなければなりません。詳しくは、 365 ページの『第 14 章 トラブルシューティング』および 345 ページの『第 11 章 ObjectGrid パフォーマンスのベスト・プラクティス』を参照してください。

デッドロック

ロック・モード要求の以下のシーケンスについて検討します。

```
X ロックは、トランザクション 1 の key1 に対して認可されています。
X ロックは、トランザクション 2 の key2 に対して認可されています。
トランザクション 1 によって要求された、key2 に対する X ロック
(トランザクション 1 は、
トランザクション 2 によって所有されたロックを待機するのをブロックします。)
トランザクション 2 によって要求された、key1 に対する X ロック
(トランザクション 2 は、
トランザクション 1 によって所有されたロックを待機するのをブロックします。)
```

上記のシーケンスは、2 つのトランザクションからなる古典的なデッドロックの例です。 2 つのトランザクションが単一ロックを超えて獲得しようとし、各トランザクションは異なる順序でロック獲得します。このデッドロックを防止するには、各トランザクションが複数ロックを同じ順序で獲得しなければなりません。オプティミスティック・ロック・ストラテジーが使用され、 ObjectMap インターフェースの flush メソッドがアプリケーションによって絶対に使用されない場合は、ロック・モードがトランザクションによって要求されるのはコミット・サイクル中のみです。コミット・サイクル中、ObjectGrid は、ロックする必要があるマップ・エントリーのキーを決定し、キー・シーケンスのロック・モードを要求します。この方法で、ObjectGrid は古典的デッドロックの大多数を防止します。しかし、ObjectGrid がす

べてのデッドロック・シナリオを防止するわけでも、防止できるわけでもありません。アプリケーションが考慮する必要があるシナリオがいくつか存在します。以下は、アプリケーションが注意し、予防アクションを取らなければならないシナリオです。

1 つのシナリオは、ロック待ちタイムアウトが発生するのを待たなくとも ObjectGrid がデッドロックを検出できる場合です。このシナリオが起こる場合、com.ibm.websphere.objectgrid.LockDeadlockException 例外が発生します。以下のコードの断片について検討します。

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
sess.begin();
Person p = (IPerson)person.get("Billy");
// Billy had a birthday, so we make him 1 year older.
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
sess.commit();
```

この場合、Billy の妻が彼を実際よりも年上にしようとし、Billy と妻の両方がこのトランザクションを並行して実行するとします。この状態では、person.get(“Billy”) メソッド呼び出しの結果として両方のトランザクションが PERSON マップの **Billy** エントリーに対して S ロック・モードを所有します。 person.put (“Billy”, p) メソッド呼び出しの結果として、両方のトランザクションは S ロック・モードを X ロック・モードに格上げしようとします。両方のトランザクションは、他方のトランザクションが所有している S ロック・モードを解放するのを待つことをブロックします。結果として、デッドロックが発生します。 2 つのトランザクション間に循環待ち条件が存在するためです。循環待ち条件は、複数のトランザクションが同一のマップ・エントリーに対して弱いモードから強いモードへロックを格上げするときに発生します。このシナリオでは、ObjectGrid は、 LockTimeoutException 例外ではなく、 LockDeadlockException 例外をスローします。詳しくは、 369 ページの『LockDeadlockException』を参照してください。

アプリケーションは、前例の LockDeadlockException 例外を防止するために、ペシミスティック・ロック・ストラテジーの代わりに、オプティミスティック・ロック・ストラテジーを使用することができます。オプティミスティック・ロック・ストラテジーの使用は、マップが主として読み取りで、マップの更新がまれにしか行われない場合、推奨される解決策です。オプティミスティック・ストラテジーについての詳細は、 142 ページの『オプティミスティック・ロック』を参照してください。ペシミスティック・ロック・ストラテジーを使用する必要がある場合は、上記の例の get メソッドの代わりに、 getForUpdate メソッドを使用することができます。そうすることによって、getForUpdate メソッドを呼び出す最初のトランザクションは、S ロック・モードではなく U ロック・モードを獲得します。このロック・モードにより、 2 番目のトランザクションは、getForUpdate メソッドを呼び出したときにブロックされます。 U ロック・モードで認可されるトランザクションは 1 つだからです。 2 番目のトランザクションはブロックされるので、Billy マップ・エントリーに対するロック・モードを何も所有しません。最初のトランザクションは、最初のトランザクションからの put メソッド呼び出しの結果として、 U ロック・モードから X ロック・モードへの格上げをしようとしたときに、ブロックしません。この働きは、U ロック・モードが「アップグレード可能」ロック・モードと呼ばれる理由を説明しています。最初のトランザクションが完了すると、2 番目のトランザクションがブロックを解除し、U ロック・モードを認可されます。アプリ

ケーションは、ペシミスティック・ロック・ストラテジーが使用されている場合、get メソッドの代わりに getForUpdate メソッドを使用することにで、ロック格上げによるデッドロック・シナリオを回避できます。

重要: この解決策は、読み取り専用トランザクションがマップ・エントリーを読み取るのを妨げません。読み取り専用トランザクションは、get メソッドを呼び出しますが、 put、insert、update、または remove メソッドを呼び出すことはありません。並行性は、通常の get メソッドが使用されているときと同様に高く維持されます。唯一、並行性が低減するのは、複数のトランザクションによって同一のマップ・エントリーに対して getForUpdate メソッドが呼び出されるときです。

あるトランザクションが複数のマップ・エントリーに対して getForUpdate メソッドを呼び出す場合、各トランザクションによって U ロックが同じ順序で獲得されるように注意しなければなりません。例えば、最初のトランザクションがキー 1 に対する getForUpdate メソッドと、キー 2 に対する getForUpdate メソッドを呼び出すとします。別の並行トランザクションが 2 つの同一キーに対する getForUpdate メソッドを呼び出しますが、逆順で呼び出します。このシーケンスにより、古典的なデッドロックが発生します。複数ロックが異なるトランザクションによって異なる順序で獲得されるためです。アプリケーションでは引き続き、複数のマップ・エントリーにアクセスするどのトランザクションもキー・シーケンスに従い、デッドロックを発生しないようにする必要があります。 U ロックはコミット時ではなく、getForUpdate メソッドが呼び出される時に獲得されるので、ObjectGrid は、コミット・サイクル中に行われるようにロック要求を順序付けることはできません。アプリケーションは、このケースではロックの順序付けを制御する必要があります。

コミットの前に ObjectMap インターフェースの flush メソッドを使用すれば、ロックの順序付けの考慮を加えることができます。 flush メソッドは、通常、Loader プラグインにより、マップに行われた変更をバックエンドに強制する目的に使用されます。この状態では、バックエンドは独自のロック・マネージャーを使用して並行処理を制御するので、ロック待ち条件とデッドロックは、 ObjectGrid ロック・マネージャー内よりもむしろバックエンド内で発生します。次のトランザクションについて検討します。

```
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();
}
```

何かほかのトランザクションが Tom も更新し、flush メソッドを呼び出し、次に Billy を更新したとします。この状態が発生した場合、2 つのトランザクションの以下のインターリービングの結果、データベースはデッドロック状態になります。

```
フラッシュが実行されるときに、X ロックはトランザクション 1 の「Billy」に
対して認可されています。
フラッシュが実行されるときに、X ロックはトランザクション 2 の「Tom」に
対して認可されています。
コミット処理中に、トランザクション 1 によって要求された、
「Tom」に対する X ロック
(トランザクション 1 は、
トランザクション 2 によって所有されたロックを待機するのをブロックします。)
コミット処理中に、トランザクション 2 によって要求された、
「Billy」に対する X ロック
(トランザクション 2 は、
トランザクション 1 によって所有されたロックを待機するのをブロックします。)
```

この例は、flush メソッドの使用により、 ObjectGrid 内よりもむしろデータベース内でデッドロックが発生することを示しています。このデッドロック例は、どのロック・ストラテジーを使用しても発生する可能性があります。アプリケーションは、flush メソッドを使用しているときと、 Loader が BackingMap にプラグインされているときは、この種のデッドロックの発生を防止することに留意する必要があります。上記の例は、ObjectGrid がロック待ちタイムアウト機構を備えているもう 1 つの理由を示しています。データベース・ロックを待機するトランザクションは、 ObjectGrid マップ・エントリーのロックを所有している間中、待機し続けることになります。その結果、データベース・レベルの問題により、ObjectGrid ロック・モードの待機時間が過大になり、LockTimeoutException 例外が発生する可能性があります。

例外処理

このトピックの例には、例外処理が含まれていません。 LockTimeoutException 例外または LockDeadlockException が発生したときに、ロックが過度に長い時間保留されないようにするために、アプリケーションは、予期しない例外をキャッチし、予期しないことが発生したときに rollback メソッドを呼び出す必要があります。以下の例に示すように、前述のコードの断片を変更してください。

```
Session sess = ...;
ObjectMap person = sess.getMap("PERSON");
boolean activeTran = false;
try
{
sess.begin();
activeTran = true;
Person p = (IPerson)person.get("Billy");
// Billy had a birthday, so we make him 1 year older.
p.setAge( p.getAge() + 1 );
person.put( "Billy", p );
sess.commit();
activeTran = false;
}
finally
{
if ( activeTran ) sess.rollback();
}
```

コードの断片の finally ブロックは、予期しない例外が発生したときにトランザクションがロールバックされるようにしています。 LockDeadlockException 例外のみでなく、発生するその他の予期しない例外もすべて処理します。 finally ブロック

は、 commit メソッドの呼び出し時に例外が発生するケースも処理します。この例は、予期しない例外を処理する唯一の方法ではありません。アプリケーションが、発生する予期しない例外のいくつかをキャッチし、そのアプリケーション例外の 1 つを表示するケースも存在するかもしれません。適宜 catch ブロックを追加できますが、アプリケーションは、コードの断片がトランザクションを完了せずに終了しないようにする必要があります。

オプティミスティック・ロック

オプティミスティック・ロック・ストラテジーでは、並行して実行中に、2 つのトランザクションが同じマップ・エントリーを更新することはありません。このことから、トランザクションの存続期間中、ロック・モードを保留する必要はありません。これは、複数のトランザクションがマップ・エントリーを並行して更新するとは考えられないためです。

オプティミスティック・ロック・ストラテジーは通常、以下の場合に使用されます。

- BackingMap がローダー付き、またはローダーなしで構成され、バージョン管理情報が使用可能である。
- BackingMap がほとんど既読である。つまり、トランザクションが頻繁にマップ・エントリーを読み取り、時折マップ・エントリーの挿入、更新、除去を行うだけです。
- BackingMap は、読み取りと比べてより頻繁に挿入、更新、または除去されるが、トランザクションは同じマップ・エントリー上でほとんど衝突しない。

ペシミスティック・ロック・ストラテジーと同様に、 ObjectMap インターフェース上のメソッドは、ObjectGrid がアクセス中のマップ・エントリーのロック・モードを自動的に獲得する方法を決定します。ただし、ペシミスティック・ストラテジーとオプティミスティック・ストラテジーの間には、以下のような重要な違いがいくつかあります。

- ペシミスティック・ロック・ストラテジーと同様に、メソッドの呼び出しの際、`get` メソッドおよび `getAll` メソッドによって *S ロック・モード* が獲得されます。しかし、オプティミスティック・ロックを使用すると、 S ロック・モードはトランザクションが完了するまで保留されません。代わりに、S ロック・モードはメソッドがアプリケーションに戻す前に保留解除されます。ロック・モードの獲得の目的は、ObjectGrid によって、その他のトランザクションからのコミット済みデータのみが現行トランザクションで可視であることを確認できるようにすることです。 ObjectGrid がそのデータがコミット済みであることを確認した後で、 S ロック・モードは保留解除されます。コミット時刻に、オプティミスティック・バージョン管理チェックが実行され、現行トランザクションがその S ロック・モードを保留解除した後で、マップ・エントリーを変更したトランザクションが他にないことが確認されます。更新、無効化、または削除される前にマップからエントリーがフェッチされない場合、ObjectGrid ランタイムによって、暗黙的にマップからエントリーがフェッチされます。この暗黙的な get 操作は、エントリーの変更が要求された時点における現行値を取得するために実行されます。
- ペシミスティック・ロック・ストラテジーとは異なり、 `getForUpdate` メソッドと `getAllForUpdate` メソッドは、オプティミスティック・ロック・ストラテジーが使用された場合には、 `get` メソッドと `getAll` メソッドと同様に処理されま

す。つまり、S ロック・モードはメソッドの開始時に獲得され、 S ロック・モードはアプリケーションに戻る前に保留解除されます。

- その他の ObjectMap メソッドは、すべてペシミスティック・ロック・ストラテジーの場合と同様にハンドルされます。つまり、commit メソッドが呼び出されると、挿入、更新、除去、タッチ、または無効化されたマップ・エントリー用に X ロック・モードが獲得され、トランザクションがコミット処理を完了するまで X ロック・モードが保留されます。

このロック・ストラテジーがオプティミスティックと呼ばれるのは、想定がオプティミスティックであるためです。オプティミスティック・ロック・ストラテジーは、並行して実行中に、2 つのトランザクションが同じマップ・エントリーを更新しないストラテジーです。このことから、トランザクションの存続期間中、ロック・モードを保留する必要はありません。これは、複数のトランザクションがマップ・エントリーを並行して更新するとは考えられないためです。しかし、ロック・モードが保留されなかったため、並行トランザクションがその S ロック・モードを保留解除した後で、別の並行トランザクションがマップ・エントリーを更新する可能性があります。この可能性をハンドルするには、ObjectGrid はコミット時刻に X ロックを取得し、オプティミスティック・バージョン管理チェックを行って、並行トランザクションが BackingMap からマップ・エントリーを読み取って以降、他にマップ・エントリーを変更したトランザクションがないことを確認します。別のトランザクションがマップ・エントリーを変更した場合、バージョン・チェックは失敗し、`OptimisticCollisionException` 例外が発生します。この例外により、現行トランザクションは強制的にロールバックされ、トランザクション全体がアプリケーションによって再試行されます。オプティミスティック・ロック・ストラテジーは、マップがほとんど既読で、同じマップ・エントリーに対する更新が起こる可能性が低い場合に非常に便利です。

BackingMap の NONE ロック・ストラテジー

BackingMap が NONE のロック・ストラテジーを使用するよう構成されている場合、マップ・エントリーのトランザクション・ロックは獲得されません。このストラテジーが有効であるシナリオは、アプリケーションが Java 2 Platform, Enterprise Edition (J2EE) Enterprise JavaBeans (EJB) コンテナーのようなパーシスタンス・マネージャーであるか、ハイバネートを使用して永続データを入手している場合です。このシナリオでは、BackingMap はローダーを使用しないで構成され、パーシスタンス・マネージャーによってデータ・キャッシュとして使用されます。このシナリオ内のパーシスタンス・マネージャーは、同じ ObjectGrid Map エントリーにアクセスするトランザクション間の並行性制御を提供します。ObjectGrid は、並行性制御のためにトランザクション・ロックを入手する必要はありません。これは、パーシスタンス・マネージャーが、コミットされた変更で ObjectGrid マップを更新する前にそのトランザクション・ロックをリリースしないことを前提としています。これが該当しない場合は、ペシミスティックまたはオプティミスティック・ロック・ストラテジーを使用しなければなりません。例えば、EJB コンテナーのパーシスタンス・マネージャーが、EJB コンテナー管理のトランザクション内でコミットされたデータで ObjectGrid Map を更新していると仮定します。 ObjectGrid Map の更新が、パーシスタンス・マネージャーのトランザクション・ロックがリリースされる前に発生する場合、NONE ロック・ストラテジーを使用することができます。パーシスタンス・マネージャーのトランザクション・ロックがリリースされた後で

ObjectGrid Map 更新が発生する場合は、オプティミスティックまたはペシミスティック・ロック・ストラテジーのどちらかが要求されます。

NONE ロック・ストラテジーの使用が可能なもう 1 つのシナリオは、アプリケーションが BackingMap を直接使用し、ローダーがマップに対して構成されているときです。このシナリオでは、ローダーは、Java Database Connectivity (JDBC) またはハイバネートのどちらかを使用して、リレーショナル・データベース内のデータにアクセスすることによって、リレーショナル・データベース管理システム (RDBMS) によって提供される並行性制御サポートを使用します。ローダーの実装は、オプティミスティックまたはペシミスティックのいずれかの方法を使用できます。

オプティミスティック・ロックまたはバージョン管理方法を使用するローダーは、大量の並行性およびパフォーマンスの達成を支援します。オプティミスティック・ロック・アプローチの実装について詳しくは、216 ページの『ローダーの考慮事項』トピックの 219 ページの『OptimisticCallback』セクションを参照してください。

基礎となるバックエンドのペシミスティック・ロック・サポートを利用するローダーで、Loader インターフェースの get メソッドに渡される **forUpdate** パラメーターを利用したい場合があります。アプリケーションがデータを取得するために、ObjectMap インターフェースの `getForUpdate` メソッドを使用した場合、このパラメーターは *true* に設定されます。ローダーはこのパラメーターを使用して、既読の行のアップグレード可能なロックを要求するかどうかを、判別することができます。例えば、DB2 は、SQL の SELECT ステートメントに `for update` 文節が含まれている場合、アップグレード可能なロックを獲得します。このアプローチは、『ペシミスティック・ロック』のトピックで説明されているのと同じデッドロック防止を提供します。

ObjectGrid セキュリティー

構成またはプログラミングによってマップ・データ・アクセスおよび管理タスクを保護するには、ObjectGrid セキュリティー・メカニズムを使用します。

ObjectGrid は、マップ・データおよび管理タスクへのアクセスを保護するためにセキュリティー・メカニズムを提供します。 ObjectGrid セキュリティーは、Java 認証・承認サービス (JAAS) 機構にビルドされています。 JAAS は、Java 2 セキュリティーの整数部分です。

このセクションでは、ObjectGrid セキュリティー・メカニズムおよび ObjectGrid セキュリティー API の使用方法について説明します。

- 145 ページの『ObjectGrid セキュリティー概要』。ObjectGrid セキュリティーの概要について説明します。
- 150 ページの『クライアント・サーバー・セキュリティー』。分散 ObjectGrid プログラミング・モデルのクライアント・サーバー・セキュリティーについて説明します。
- 171 ページの『ローカル ObjectGrid のセキュリティー』。ローカル ObjectGrid セキュリティーについて説明します。
- 177 ページの『許可』。分散およびローカル両方の ObjectGrid プログラミング・モデルに適用される許可メカニズムと関連プラグインについて説明します。

- 187 ページの『ObjectGrid クラスター・セキュリティー』。ObjectGrid クラスター・セキュリティー・メカニズムと関連プラグインについて説明します。
- 191 ページの『ゲートウェイ・セキュリティー』。ゲートウェイ・セキュリティーについて説明します。
- 193 ページの『WebSphere Application Server とのセキュリティーの統合』。WebSphere Application Server との統合について説明します。

このセクションで示されているほとんどのサンプル・コードは、ObjectGrid に同梱されているサンプルです。セキュリティー・サンプルの概要については、 63 ページの『第 5 章 ObjectGrid のサンプル』を参照してください。

ObjectGrid セキュリティー概要

ObjectGrid は、分散キャッシング・システムです。キャッシュ・データへのアクセスを保護できます。一般的に、セキュリティーは、次の 3 つの主要な概念に基づいています。

- *信頼できる認証*: 要求側の ID を確実に判別します。
- *許可*: 許可により要求側にアクセス権限を付与します。
- セキュアなトランスポート: ネットワーク上でデータを安全に伝送します。

ObjectGrid は、次の局面でセキュリティーを提供します。

- 『クライアント・サーバー・セキュリティー』は、Secure Sockets Layer (SSL) を使用して、認証およびクライアント・サーバーの通信セキュリティーをアドレッシングします。
- 146 ページの『許可』メカニズムは、権限があるクライアントのみが ObjectGrid マップ・データおよび管理タスクにアクセスできることを保証します。
- 146 ページの『ObjectGrid クラスター・セキュリティー』は、権限があるサーバーのみが ObjectGrid クラスターに結合できることを検証します。
- 147 ページの『ゲートウェイ・セキュリティー』は、ゲートウェイ・クライアント認証をアドレッシングします。
- 147 ページの『ローカル ObjectGrid のセキュリティー』は、アプリケーションが直接 ObjectGrid インスタンスをインスタンス化する場合に、セキュリティー・メカニズムを提供します。

ObjectGrid セキュリティーは、オープン・アーキテクチャー上に構築され、カスタマイズ用にさまざまなプラグイン・ポイントを提供します。プラグイン・メカニズムは、重要な役割を担当します。また、ObjectGrid は、これらのプラグイン用にいくつかの標準装備のインプリメンテーションを提供します。いくつかのインプリメンテーションは、すぐに実動環境で使用でき、その他はテストまたはサンプルを目的としています。プラグインおよび標準装備のインプリメンテーションについての要約は、 147 ページの『セキュリティー・プラグイン』を参照してください。

クライアント・サーバー・セキュリティー

ObjectGrid は、分散クライアント・サーバー・フレームワークをサポートします。クライアント・サーバー・セキュリティー・インフラストラクチャーは、ObjectGrid サーバーへのアクセスを保護するために配置されています。

ObjectGrid クライアントは、ObjectGrid サーバーに対して認証する信用証明情報を使用できます。契約は、クライアントおよびサーバー間で、この信用証明情報をサーバー認証メカニズムが認識できるように確立される必要があります。Secure Sockets Layer (SSL) を使用する場合、クライアントは、ObjectGrid サーバーに対して認証する SSL 証明書を使用することもできます。

クライアント・サーバー通信を保護するため、ObjectGrid は SSL をサポートします。SSL プロトコルは、 ObjectGrid クライアントとサーバー間のセキュア接続のための、トランスポート層セキュリティー (認証性、保全性、および機密性) を提供します。SSL が提供するセキュリティー機能の一部は、次のとおりです。データ・フロー中の機密情報の漏えいを防ぐデータの暗号化。データ・フロー中のデータの認証されていない変更を防ぐデータ署名。適切な人またはマシンとやり取りすることを確実にするためのクライアントおよびサーバーの認証。SSL は、エンタープライズ環境での保護において効果的です。

詳しくは、 150 ページの『クライアント・サーバー・セキュリティー』を参照してください。

許可

ObjectGrid の許可は、サブジェクトおよび許可に基づいています。ObjectGrid では、次の 2 つの許可のカテゴリーがあります。データ・アクセスの許可、管理タスクの許可。Java 認証・承認サービス (JAAS) を使用してアクセスを許可することや、独自のメカニズムに接続して許可を処理することができます。

詳しくは、 177 ページの『許可』を参照してください。

ObjectGrid クラスター・セキュリティー

セキュア環境では、サーバーは他のサーバーの認証性を確認できる必要があります。ObjectGrid は、この目的のために共有秘密鍵ストリング・メカニズムを使用します。この秘密鍵のメカニズムは、共有パスワードと同様です。すべての ObjectGrid サーバーは、共有秘密鍵に一致しています。サーバーがクラスターに結合する場合、秘密ストリングの表示を求められます。結合するサーバーの秘密ストリングが、マスター・サーバー内のストリングに一致する場合、結合するサーバーはクラスターに結合できます。そうでない場合、結合要求は拒否されます。

平文の機密事項の送信は保護されません。ObjectGrid セキュリティー・インフラストラクチャーは、機密事項の送信前に SecureTokenManager プラグインを提供し、サーバーがこの機密事項を「保護」できるようにします。「保護」操作のインプリメント方法は、公開されています。ObjectGrid は、すぐに使用できるインプリメンテーションを提供します。これにより、「保護」操作がインプリメントされ、機密事項が暗号化されて署名が行われます。

詳しくは、 187 ページの『ObjectGrid クラスター・セキュリティー』を参照してください。

ゲートウェイ・セキュリティー

ObjectGrid ゲートウェイは、ObjectGrid サーバーに対するクライアント管理要求を代行するポイントとして動作します。管理ゲートウェイは、一連の mbean を収納しています。ゲートウェイ・クライアントは、これらの mbean を起動して ObjectGrid サーバーを管理またはモニターします。

管理ゲートウェイおよびサーバー通信は、ObjectGrid クライアント・サーバー通信メカニズムを使用します。これにより、ゲートウェイは ObjectGrid クライアントとして取り扱われます。ゲートウェイ・クライアントおよびゲートウェイ (MBean サーバー) 通信は、SSL により保護できます。この機能は、オープン・ソース・プロジェクト mx4j でインプリメントされた JMX コネクター層により提供されます。ObjectGrid では、ゲートウェイを動作させるために、指定された場所に mx4j を配置する必要があります。

認証用に、ゲートウェイは、ゲートウェイ・クライアントが示した信用証明情報を ObjectGrid サーバーに伝搬します。認証および許可は、ObjectGrid サーバー上で実行されます。

詳しくは、 191 ページの『ゲートウェイ・セキュリティー』を参照してください。

ローカル ObjectGrid のセキュリティー

WebSphere Extended Deployment Server リリース 6.0 では、ローカル ObjectGrid プログラミング・モデルが導入されました。このモデルでは、アプリケーションは、ObjectGrid インスタンスを直接インスタンス化して使用します。アプリケーションおよび ObjectGrid インスタンスは、同じ Java 仮想マシン (JVM) 内にあります。このモデルには、クライアントまたはサーバーの概念はありません。

ローカル ObjectGrid プログラミング・モデルでは認証はサポートされていません。アプリケーションは、独自の認証を管理し、認証済みサブジェクト・オブジェクトを ObjectGrid に渡す必要があります。

同様の許可メカニズムは、クライアント・サーバー・モデルに使用されたように、ローカル ObjectGrid プログラミング・モデルに使用されます。

詳しくは、 171 ページの『ローカル ObjectGrid のセキュリティー』を参照してください。

セキュリティー・プラグイン

ObjectGrid セキュリティー・フレームワークは、セキュリティー・プラグインで補足されます。これらのプラグインをインプリメントして、セキュリティー・フレームワークを拡張またはカスタマイズすることができます。

一例は、CredentialGenerator プラグインです。これは、com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator インターフェースによって表されます。ObjectGrid クライアントが ObjectGrid サーバーに接続すると、このプラグインの getCredential() メソッドが呼び出され、信用証明情報オブジェクトが生成されます。この信用証明情報オブジェクトは、次にサーバーに送信されます。サーバーは、この信用証明情報オブジェクトを使用し、Authenticator プラグイ

ンを使用して認証します。このプラグインは、thecom.ibm.websphere.objectgrid.security.plugins.CredentialGenerator.Authenticator インターフェースにより表されます。

プラグインは、ObjectGrid セキュリティー・フレームワークで重要な役割を果たします。CredentialGenerator をインプリメントして、ユーザー ID とパスワードのペア、Kerberos チケット、セキュリティー・トークンなどの特定の信用証明情報を生成できます。Authenticator プラグインをインプリメントして、クライアントを認証できます。必要に応じて、 Authenticator プラグインをインプリメントして、ユーザー・パスワードまたはセキュリティー・トークンの両方をサポートできます。

すべての使用可能なセキュリティー用プラグインを次の表に示します。

表 9. セキュリティー・プラグイン

カテゴリー	プラグイン・クラス名	インスタンス
認証	com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator	クライアント
	com.ibm.websphere.objectgrid.security.plugins.Credential	クライアント
	com.ibm.websphere.objectgrid.security.plugins.オーセンティケーター	サーバー
許可	com.ibm.websphere.objectgrid.security.plugins.MapAuthorization	ObjectGrid
	com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization	クラスター
ObjectGrid クラスター・セキュリティー	com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager	サーバー
その他	com.ibm.websphere.objectgrid.security.plugins.SubjectSource	ローカル ObjectGrid
	com.ibm.websphere.objectgrid.security.plugins.SubjectValidation	ローカル ObjectGrid

次の図は、プラグインおよび適用されるインスタンスを示したものです。例えば、MapAuthorization プラグインは ObjectGrid インスタンスで適用されますが、AdminAuthorization はサーバー・インスタンスで適用されます。

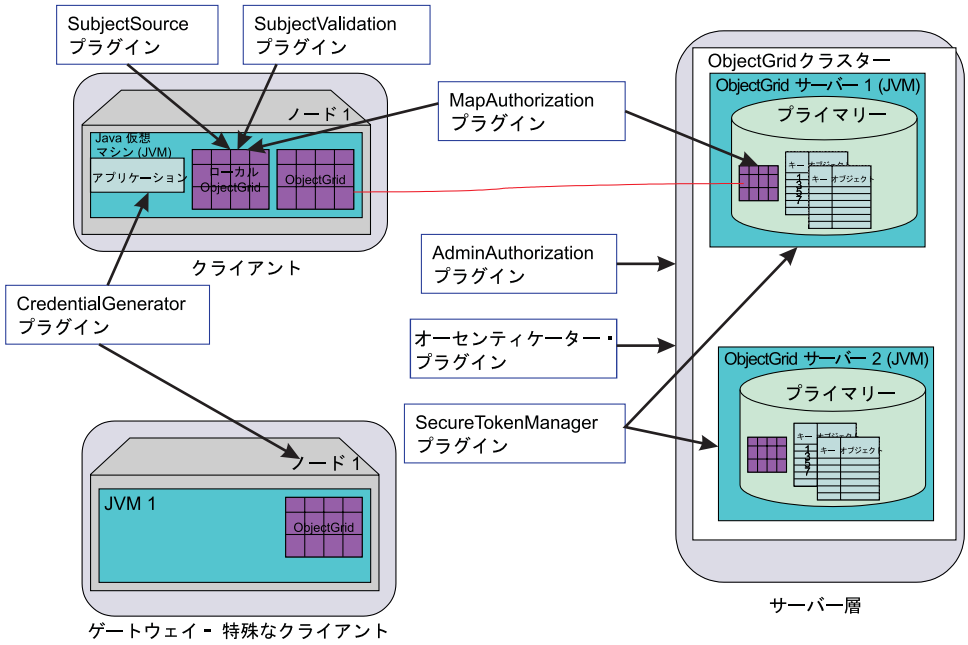


図 17. セキュリティー・プラグイン

次の表は、標準装備のインプリメンテーションを示しています。目的列には、目的を表示しています。目的は、すぐに使用できる実動用、またはテスト用になります。

表 10. セキュリティーの標準装備インプリメンテーション

プラグイン	標準装備のクラス名	目的
信用証明情報生成プログラムの信用証明情報	com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredentialGenerator	実動
	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator	実動
	com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential	実動
	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential	実動
	com.ibm.websphere.objectgrid.security.plugins.builtins.ClientCertificateCredential	実動
オーセンティケーター	com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator	実動
	com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator	テスト
	com.ibm.websphere.objectgrid.security.plugins.builtins.CertificateMappingAuthenticator	テスト
	com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator	テスト
マップ許可	com.ibm.websphere.objectgrid.security.plugins.builtins.JAASMapAuthorizationImpl	実動
	com.ibm.websphere.objectgrid.security.plugins.builtins.TAMMapAuthorizationImpl	テスト

表 10. セキュリティーの標準装備インプリメンテーション (続き)

プラグイン	標準装備のクラス名	目的
SubjectSource	com.ibm.websphere.objectgrid.security.plugins.builtins. WSSubjectSourceImpl	実動
サブジェクト妥当性検査	com.ibm.websphere.objectgrid.security.plugins.builtins. WSSubjectValidationImpl	実動

クライアント・サーバー・セキュリティー

このトピックでは、認証メカニズムおよびクライアント・サーバー通信の保護方法について説明します。

クライアント・サーバー・セキュリティーには、以下の重要な性質があります。

- 『クライアント・サーバー・セキュリティーの使用可能化』の方法
- 151 ページの『クリデンシャルおよびクリデンシャル生成プログラム』を使用して、クライアントを表すクリデンシャルを取得する方法
- 169 ページの『セキュア通信』を使用する SSL 構成で使用されるパラメーターの構成方法
- 158 ページの『オーセンティケーター』を使用して、サーバー・サイドのクライアントを認証する方法

クライアント・サーバー・セキュリティーの使用可能化

クライアント・セキュリティーの使用可能化

クライアント・セキュリティーのセキュリティーを使用可能にするには、security.ogclient.props ファイルの securityEnabled プロパティーを **true** に設定します。ObjectGrid には、クライアント・セキュリティー・プロパティー・テンプレート・ファイルが添付されています。このファイルは、WebSphere インストールの [WAS_HOME]/optionalLibraries/ObjectGrid/properties ディレクトリー、または混合サーバー・インストールの /ObjectGrid/properties ディレクトリーにある security.ogclient.props ファイルです。このテンプレート・ファイルを、適切な値で変更することができます。

次に、securityEnabled プロパティーについて説明します。

securityEnabled (true, false+)

このプロパティーは、セキュリティーが使用可能かどうかを示します。クライアントがサーバーに接続されている場合、クライアント・サイドとサーバー・サイドの securityEnabled 値は両方とも、true または false である必要があります。たとえば、接続されているサーバー・セキュリティーが使用可能な場合、クライアントではこのプロパティーを true に設定して、サーバーに接続する必要があります。

com.ibm.websphere.objectgrid.security.config.ClientSecurityConfiguration インターフェースは、security.ogclient.props ファイルを表しています。com.ibm.websphere.objectgrid.security.config. ClientSecurityConfigurationFactory public API を使用して、このインターフェースのインスタンスをデフォルト値で作成することができます。または ObjectGrid クライアント・セキュリ

ティー・プロパティー・ファイルを渡して、インスタンスを作成することもできます。 security.ogclient.props ファイルには、その他のプロパティーが含まれています。

サーバー・セキュリティーの使用可能化

サーバー・サイドのセキュリティーを使用可能にするには、クラスター XML の securityEnabled プロパティーを true に設定します。以下に例を示します。

```
<cluster>
<objectGrid name="cluster" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300">
```

クリデンシャルおよびクリデンシャル生成プログラム

サーバーに接続する際、クライアントは自身のクリデンシャルを示す必要があります。クライアントのクリデンシャルは、com.ibm.websphere.objectgrid.security.plugins.Credential インターフェースによって表されます。クリデンシャルには、ユーザー・パスワードのペア、Kerberos チケットなどを含むことができます。

クリデンシャル・インターフェースは、次のとおりです。

```
package com.ibm.websphere.objectgrid.security.plugins;
import java.io.Serializable;
/**
* This interface represents a credential used by an ObjectGrid client. It
* represents one client identity. This credential is sent to the
* ObjectGrid server for authentication. It must be serializable.
*
* A credential has to implement the equals(Object)and
* hashCode() methods. Two Credential objects are considered equal
* if and only if they represent the same identity and security information. For
* example, if the credential contains a user ID and password. Two credentials
* are equal if and only if both their user IDs and passwords are equal.
*
* ObjectGrid provides three built-in implementations for this interface:
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* ClientCertificateCredential:
* A credential containing an SSL certificate chain.
* com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential:
* A credential containing a user ID and password pair.
* com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential:
* A credential containing WebSphere Application Server specific authentication
* and authorization tokens.
*
* Refer to the respective API documentation for more details.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see CredentialGenerator
*/
public interface Credential extends Serializable {
/**
* Checks two Credential objects for equality.
*
* Two Credential objects are considered equal if and only if
* they represent the same identity and security information.
*
* @param o the object we are testing for equality with this object.
*
* @return true if both Credential objects are equivalent.
```

```
*/
boolean equals(Object o);
/**
* Returns the hashcode of the Credential object
*
* @return the hash code of the Credential object
*/
int hashCode();
}
```

このインターフェースでは、equals(Object) メソッドおよび hashCode() メソッドを明示的に定義します。これらのメソッドは、動作を保証するために、重要となります。認証済み Subject オブジェクトは、サーバー・サイドの Credential オブジェクトに基づいてキャッシュされます。

ObjectGrid は、クリデンシャル・インターフェース用として次の 3 つのデフォルトのインプリメンテーションを提供しています。

1. com.ibm.websphere.objectgrid.security.plugins.builtins.UserPasswordCredential インプリメンテーション。このクリデンシャルには、ユーザー ID とパスワードのペアが含まれます。
2. com.ibm.websphere.objectgrid.security.plugins.builtins.ClientCertificateCredential インプリメンテーション。このクリデンシャルには、クライアント証明書チェーンが含まれます。このクリデンシャルは、ObjectGrid クライアント証明書の認証に使用できます。このクリデンシャルをクライアント・サイドで作成することはできません。このクリデンシャルは、SSL ハンドシェークの一部としてサーバーで生成されます。
3. com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential インプリメンテーション。このクリデンシャルには、WebSphere Application Server 固有の認証トークンおよび許可トークンが含まれています。これらのトークンを使用すると、同じセキュリティー・ドメイン内のアプリケーション・サーバーにセキュリティー属性を伝搬することができます。

詳しくは、API 資料を参照してください。

ObjectGrid には、クリデンシャルを生成するためのプラグインも用意されています。このプラグインは、
com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator インターフェースによって表されます。以下に、CredentialGenerator インターフェースを示します。

```
/**
* This plug-in is used to get a Credential representing this client. It is a
* factory for the Credential object.
* One example implementation is to return a Credential object
* containing a user ID and password pair. The implementation of the Credential
* generated by an implementation of this class must to be understood by the
* server's Authenticator plug-in.
*
* An implementation class of this interface must have a default constructor.
* When launching the client in a secure environment, set the
* implementation class name (credentialGeneratorClass) in the client security
* configuration property file. The client runtime constructs an object of
* this implementation class and calls getCredential() to get the Credential
* to connect to an ObjectGrid cluster.
*
*Users can also specify the additional properties for this factory using the
* credentialGeneratorProps property in the client security configuration
* property file.
```

```
* These properties are be passed to this
* factory by using the setProperties(String) method. This way, you can
* customize your factory.
*
* You can also set CredentialGenerator programmatically by calling
* ClientSecurityCinfiguration.setCredentialGenerator
* (CredentialGenerator) method.
*
*
* For example, you can have the following settings in the client security
* configuration property file:
* credentialGeneratorClass=com.myco.CredGenFactory
*
* credentialGeneratorProps=user1 password1
*
*
* , a String "user1 password1" is passed to the setProperties(String)
* method, with the "user1" indicating the user name, and "password1"
* indicating the password.
*
* ObjectGrid provides two built-in implementations for this interface:
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* UserPasswordCredentialGenerator:
* A credential generator generating a UserPasswordCredential
* containing a user ID and password pair.
* com.ibm.websphere.objectgrid.security.plugins.builtins.
* WSTokenCredentialGenerator:
* A credential generator generating a WSTokenCredential containing WebSphere
* Application Server
* specific authentication and authorization tokens.
*
*
* The relationship between CredentialGenerator and Credential can be
* one to one relationship or one to many relationship. For example, the
* UserPasswordCredentialGenerator
* has a one to one relationship with UserPasswordCredential, but the
* WSTokenCredentialGenerator
* has a one to many relationship with WSTokenCredential because it could generate
* different WSTokenCredential based on what Subject is associated with the current
* thread.
*
* Refer to the respective API documentation for more details.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see Authenticator
* @see ClientSecurityConfiguration#setCredentialGenerator(CredentialGenerator)
* @see Credential
* @see CredentialGeneratorFactory#getCredentialGenerator()
*/
public interface CredentialGenerator {
/**
* Gets a Credential which represents the client.
*
* @return the Credential representing the client
*
* @throws CannotGenerateCredentialException if a failure occurs when
* generating the Credential for the client.
*
* @see Credential
*/
Credential getCredential() throws CannotGenerateCredentialException;
/**
* Set the user defined properties to the factory
*
* This method is used to add addtional CredentialGenerator properties
```

```
* to the object. These properties can be set using the credentialGeneratorProps
* property in the client security configuration property file.
* This way, you can customize your factory.
*
* @param properties user defined properties
*/
void setProperties(String properties);
}
```

ObjectGrid には、デフォルトの組み込みインプリメンテーションとして次の 2 つがあります。

1. com.ibm.websphere.objectgrid.security.plugins.builtins。UserPasswordCredentialGenerator コンストラクターは、ユーザー ID およびパスワードを取ります。getCredential() メソッドは、呼び出されると、ユーザー ID およびパスワードが含まれている UserPasswordCredential オブジェクトを戻します。
2. com.ibm.websphere.objectgrid.security.plugins.builtins。WSTokenCredentialGenerator は、WebSphere Application Server での実行中にクリデンシャル (セキュリティー・トークン) 生成プログラムを表します。getCredential() メソッドが呼び出されると、現在のスレッドに関連した Subject が検索されます。その後、この Subject オブジェクトのセキュリティー情報が WSTokenCredential オブジェクトに変換されます。定数 WSTokenCredentialGenerator.RUN_AS_SUBJECT または WSTokenCredentialGenerator.CALLER_SUBJECT を使用して、スレッドから runAs サブジェクトか呼び出し元サブジェクトのいずれを検索するかを指定できます。

詳しくは、API 資料を参照してください。

接続

ObjectGrid クライアントがサーバーに安全に接続したい場合、ObjectGridManager インターフェースでなんらかの接続メソッドを使用できます。例として、以下の接続メソッドを示します。

```
/**
* This allows client to connect to a Remote ObjectGrid
* The RemoteObject Grid is hosted as specified by the paramaters
* @param clusterName: The name of the cluster to which this client
* will attach iteself
* @param host: The host on which to connect to
* @param port: The clientAceess port which is listening.
* @param ClientSecurityConfiguration: Security configuration. It can be null
* if security is not configured
* @param overRideObjectGrid xml. This parameter can be null. If it is not
* null, the client side configuration of objectgrid plug-in is overridden.
* Not all plug-ins can be overridden. For details see the ObjectGrid documents
* @throws ConnectException
* @ibm-api
*/
public ClientClusterContext connect(String clusterName, String host, String port,
ClientSecurityConfiguration securityProps, URL overRideObjectGrid) throws
ConnectException ;
```

このメソッドは、特に型 ClientSecurityConfiguration のパラメーターを取ります。このインターフェースは、クライアント・セキュリティー構成を表します。com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory public API を使用して、このインターフェースのインスタンスをデフォルト値で作成する

か、または ObjectGrid クライアント・セキュリティー・プロパティー・ファイルを渡して、インスタンスを作成します。 security.ogclient.props ファイルには、認証に関連した以下のプロパティーが含まれています。+ でマークされた値は、デフォルト値です。

- securityEnabled (true, false+): このプロパティーは、セキュリティーが使用可能かどうかを示します。クライアントがサーバーに接続されている場合、クライアント・サイドとサーバー・サイドの securityEnabled 値は、両方とも true または false である必要があります。たとえば、接続されているサーバー・セキュリティーが使用可能な場合、クライアントではこのプロパティーを true に設定して、サーバーに接続する必要があります。
- credentialAuthentication (Never, Supported+, Required): このプロパティーは、クライアントでクリデンシャル認証がサポートされているかどうかを示します。
  - プロパティー値が **Never** の場合、このクライアントではクリデンシャル認証がサポートされていません。
  - プロパティー値が **Supported** の場合、クリデンシャル認証がサポートされているときか、またはクリデンシャル認証必要とするサーバーと通信するときに、クライアントの認証が実行されます。クライアント・クリデンシャル認証では、クリデンシャルまたはシングル・サインオン (SSO) トークンが送信されます。
  - プロパティー値が **Required** の場合、クライアントは、認証のため、サーバーにクリデンシャルを送信する必要があります。
- authenticationRetryCount (an integer value, 0+): このプロパティーでは、クリデンシャルの期限が切れている場合のログインの再試行回数を決定します。この値が 0 の場合、再試行は行われません。認証再試行は、クリデンシャルの期限が切れている場合にのみ適用されます。クリデンシャルが無効な場合、再試行は行われません。ご使用のアプリケーションで操作の再試行が行われます。
- clientCertificateAuthentication (Never+, Supported, Required): このプロパティーは、クライアントでクライアント証明書認証がサポートされているかどうかを示します。
  - プロパティー値が **Never** の場合は、クライアント・サイドでクライアント証明書認証がサポートされていません。
  - プロパティー値が **Supported** の場合は、トランスポート層のクライアント認証を実行でき、認証の段階にクライアントはサーバーにデジタル証明書を送信します。
  - プロパティー値が **Required** の場合、クライアントは、トランスポート層のクライアント認証がサポートされているサーバーとの認証のみを行います。
- transportType (TCP/IP、SSL-Supported+、SSL-Required): このプロパティーは、クライアントがサーバーに接続するときに使用するトランスポート・プロトコルを示します。クライアントがサーバーに接続するときに使用するプロトコルは、サーバー・サイドの transportType 設定によって異なります。詳しくは、 169 ページの『セキュア通信』を参照してください。.
  - 値が **TCP/IP** の場合、クライアントは TCP/IP を使用してサーバーに接続する必要があります。

  - 値が **SSL-Supported** の場合、クライアントは、TCP/IP または SSL を使用してサーバーに接続します。クライアントは、最初に SSL を使用してサーバーへの接続を試みます。SSL 接続が失敗すると、クライアントは TCP/IP の使用を試みます。
  - 値が **SSL-Required** の場合、クライアントは SSL を使用してサーバーに接続する必要があります。
- SSOEnabled: クライアントがサーバーにシングル・サインオン・トークンを渡すことができるかどうかを指定します。すべてのサーバーに対してクライアントが認証される場合は、このプロパティーを false に設定します。 1 つのサーバーに対してのみクライアントが認証される場合は、このプロパティーを true に設定します。クライアントで SSOEnabled を true に設定した場合は、クラスター XML 構成のシングル・サインオン対応プロパティーも true に設定されているかどうかを検証します。

これらのプロパティーは、ClientSecurityConfiguration インターフェースの setter を使用して設定することもできます。

ClientSecurityConfiguration 型のオブジェクトを作成した後に、以下のメソッドを使用して、このオブジェクトに credentialGenerator を設定します。

```
/**
* Set the {@link CredentialGenerator} object for this client.
* @param generator the CredentialGenerator object associated with this client
*/
void setCredentialGenerator(CredentialGenerator generator);
```

ObjectGrid クライアント・セキュリティー・プロパティー・ファイルにも CredentialGenerator を設定できます。プロパティーは、以下のとおりです。

- **credentialGeneratorClass**: CredentialGenerator のクラス・インプリメンテーション名。デフォルトのコンストラクターを指定する必要があります。
- **credentialGeneratorProps**: CredentialGenerator クラスのプロパティー。この値がヌル以外の場合、このプロパティーは、setProperties(String) メソッドを使用して、構成済みの CredentialGenerator オブジェクトに設定されます。

以下に、ClientSecurityConfiguration のインスタンスを生成し、このインスタンスを使用してサーバーに接続する例を示します。

```
/**
* Get a secure ClientClusterContext
* @return a secure ClientClusterContext object
*/
protected ClientClusterContext connect() throws ConnectException {
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration("/properties/security.ogclient.props");
UserPasswordCredentialGenerator gen= new
UserPasswordCredentialGenerator("manager", "manager1");
csConfig.setCredentialGenerator(gen);
return objectGridManager.connect(csConfig, null);
}
```

接続が呼び出されると、ObjectGrid クライアントは、CredentialGenerator.getCredential() メソッドを使用してクライアント・クリデンシャルを取得します。このクリデンシャルは、接続要求とともにサーバーに送信されて、認証されます。

セッションごとに異なる CredentialGenerator を使用する

ObjectGrid クライアントは、1 つのクライアント ID を表す場合もあれば、複数の ID を表す場合もあります。以下に、複数の ID を表す場合の例を示します。この例では、ObjectGrid クライアントは、Web サーバーで作成され、共用されます。この Web サーバーのすべてのサーブレットで、この 1 つの ObjectGrid クライアントが使用されます。各サーブレットが異なる Web クライアントを表すため、ObjectGrid サーバーへ要求を送信するときは、異なるクリデンシャルを使用します。

ObjectGrid では、セッション・レベルでのクリデンシャルの変更が提供されています。つまり、すべてのセッションで別々の CredentialGenerator を使用できます。そのため、前のシナリオは、サーブレットが別の CredentialGenerator を使用してセッションを取得できるようにすることで、実現できます。以下に、ObjectGridManager インターフェースでのこのメソッドを示します。

```
/**
* Get a session with a CredentialGenerator. This method can only be called
* by the ObjectGrid client in a client server environment.
*
* If ObjectGrid is used in a core model, that is, within the same JVM with
* no client or server existing, getSession(Subject) should be used to secure
* the ObjectGrid.
*
* @since WAS XD 6.0.1
*/
Session getSession(CredentialGenerator credGen) throws
ObjectGridException, TransactionCallbackException;
```

以下に例を示します。

```
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
CredentialGenerator credGenManager = new UserPasswordCredentialGenerator
("manager", "xxxxxx");
CredentialGenerator credGenEmployee = new UserPasswordCredentialGenerator
("employee", "xxxxxx");
ObjectGrid og = ogManager.getObjectGrid(ctx, "accounting");
// Get a session with CredentialGenerator;
Session session = og.getSession(credGenManager );
// Get the employee map
ObjectMap om = session.getMap("employee");
// start a transaction.
session.begin();
Object rec1 = map.get("xxxxxx");
session.commit();
// Get another session with a different CredentialGenerator;
session = og.getSession(credGenEmployee );
// Get the employee map
om = session.getMap("employee");
// start a transaction.
session.begin();
Object rec2 = map.get("xxxxx");
session.commit();
```

ObjectGird.getSession() メソッドを使用して Session オブジェクトを取得する場合、このセッションでは、ClientConfigurationSecurity オブジェクトに設定されている CredentialGenerator を使用します。したがって、ClientConfigurationSecurity オブジェクトに設定されている CredentialGenerator を、ObjectGird.getSession(CredentialGenerator) メソッドに渡された CredentialGenerator がオーバーライドするように処理することができます。

Session オブジェクトを再使用できる場合は、パフォーマンスが向上します。ただし、ObjectGrid.getSession(CredentialGenerator) メソッドの呼び出しは、さほど高価ではないため、主なオーバーヘッドは増加したオブジェクト・ガーベッジ・コレクション時間となります。Session オブジェクトの完了後に、必ず、参照を解放してください。つまり、Session オブジェクトで ID を共用できる場合は、Session オブジェクトを再使用するようにしてください。 Session オブジェクトで ID を共用できない場合は、ObjectGrid.getSession(CredentialGenerator) メソッドを使用します。

オーセンティケーター

ObjectGrid クライアントが CredentialGenerator オブジェクトを使用して Credential オブジェクトを検索した後、Credential オブジェクトは、クライアント要求とともに ObjectGrid サーバーに送信されます。 ObjectGrid サーバーは、要求の処理前に Credential オブジェクトの認証を行います。 Credential オブジェクトが正常に認証されると、この Credential オブジェクトを表す Subject オブジェクトが戻されます。その後、この Subject オブジェクトは要求の認証に使用されます。

この Subject オブジェクトもキャッシュされます。このオブジェクトは、存続時間がセッション・タイムアウト値に達すると、期限切れとなります。ログイン・セッション・タイムアウト値は、クラスター XML ファイルの loginSessionExpirationTime プロパティーを使用して設定できます。たとえば、loginSessionExpirationTime="300" に設定すると、Subject オブジェクトの有効期限は 300 秒で切れます。

ObjectGrid サーバーでは、オーセンティケーター・プラグインを使用して Credential オブジェクトを認証します。以下に、オーセンティケーター・インターフェースを示します。

```
/**
* This plug-in can be used to authenticate an ObjectGrid client to an ObjectGrid
* server based on the credential provided by the client. A Subject
* object is returned as a result of authentication.
*
* This plug-in is used in an ObjectGrid server. It can be configured in the
* ObjectGrid cluster XML file.
*
* The Credential passed in the authenticate(Credential)
* method can contain any credential information users desire. For example, it could be a
* Credential object containing a user password pair.
*
* ObjectGrid provides several built-in implementations for this interface:
* * com.ibm.websphere.objectgrid.security.plugins.builtins.
* CertificateMappingAuthenticator:
* An authenticator that simply maps a SSL certficate to a Subject.
* com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator:
* An authenticator that authenticates a user ID and password to a key file.
* com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator:
* An authenticator that authenticates a user ID and password to a LDAP server.
* com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator:
* An authenticator that authenticates a WebSphere Application Server securty token.
*
* Refer to the respective API documentation for more details.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see Credential
*/
public interface Authenticator {
```

```
/**
* Authenticates a user represented by the credential object.
*
* @param credential the user Credential
*
* @return a Subject object representing the user
*
* @throws InvalidCredentialException if the credential is invalid
* @throws ExpiredCredentialException if the credential is expired
*
* @see Credential
*/
Subject authenticate(Credential credential)
throws InvalidCredentialException, ExpiredCredentialException;
}
```

ここで、インプリメンテーションが Credential オブジェクトを取得し、Lightweight Directory Access Protocol (LDAP) サーバーなどのユーザー・レジストリーに対してこのオブジェクトを認証します。 ObjectGrid には、すぐに使用できるユーザー・レジストリー構成がありません。ユーザー・レジストリーへの接続およびユーザー・レジストリーに対する認証は、このプラグインで実装する必要があります。

たとえば、1 つのオーセンティケーター・インプリメンテーションでは、クリデンシャルからユーザー ID とパスワードが抽出され、このユーザー ID とパスワードを使用して、LDAP サーバーに対する接続と検証が行われます。認証の結果として、Subject オブジェクトが作成されます。このインプリメンテーションで、JAAS ログイン・モジュールを使用する可能性があります。認証の結果として、Subject オブジェクトが戻されます。

このメソッドでは、2 つの例外 `InvalidCredentialException` および `ExpiredCredentialException` がスローされることに、注意してください。`InvalidCredentialException` 例外は、クリデンシャルが無効であることを示します。`ExpiredCredentialException` 例外は、クリデンシャルの期限が切れていることを示します。認証メソッドの結果としてこの 2 つの例外のいずれかが発生した場合、例外はクライアントに送り返されます。ただし、クライアント・ランタイムによって、2 つの例外は別々に処理されます。

- 例外が InvalidCredentialException の場合は、クライアント・ランタイムによってこの例外が表示されます。例外の処理は、ご使用のアプリケーションによることが期待されます。CredentialGenerator を修正するなどして、操作を再試行します。
- 例外が ExpiredCredentialException で、再試行数が 0 以外の場合は、クライアント・ランタイムによって再度 CredentialGenerator.getCredential() メソッドが呼び出され、サーバーに新しい Credential オブジェクトが送信されます。新しいクリデンシャル認証が成功すると、サーバーは要求を処理します。新しいクリデンシャル認証が失敗すると、クライアントに例外が送り返されます。認証の再試行回数が許可値に達しても、クライアントがまだ ExpiredCredentialException を取得している場合は、ExpiredCredentialException となります。ご使用のアプリケーションで例外を処理する必要があります。

オーセンティケーター・インターフェースには、かなりの柔軟性があります。オーセンティケーター・インターフェースは、任意の方法で実装することができます。たとえば、クリデンシャル認証とクライアント証明書認証の両方をサポートするた

め、これら両方の認証を行うようにこのインターフェースを実装できます。また、2 種類のユーザー・レジストリーをサポートするように、このインターフェースを実装することもできます。

ObjectGrid では、2 種類の認証 (クリデンシャル認証およびクライアント証明書認証) がサポートされています。使用されるメカニズムは、クライアント・サイドおよびサーバー・サイドのセキュリティー・プロパティーの設定によって異なります。プロパティーは、以下のとおりです。

- `security.ogclient.props` ファイルの credentialAuthentication
- `security.ogserver.props` ファイルの credentialAuthentication
- `security.ogclient.props` ファイルの clientCertificateAuthentication
- `security.ogserver.props` ファイルの clientCertificateAuthentication

これらのプロパティーはプログラミング API を使用して設定することもできます。

以下の 2 つの表に、様々な設定でどの認証メカニズムが使用されるかを示します。

表 11. クライアントおよびサーバーの設定におけるクリデンシャル認証

クライアント credentialAuthentication	サーバー credentialAuthentication	結果
No	Never	disabled
	Supported	disabled
	Required	Error case
Supported	Never	disabled
	Supported	enabled
	Required	enabled
Required	Never	Error case
	Supported	enabled
	Required	enabled

クリデンシャル認証がない場合は (結果が disabled)、クライアント証明書認証が行われます。

以下の表に、様々な設定でのクライアント証明書認証の使用について示します。クライアント証明書認証は、通信プロトコルとして SSL が使用され、クリデンシャル認証が使用されていない場合にのみ可能となることに、注意してください。

表 12. クライアントおよびサーバーの設定におけるクライアント証明書認証

クライアント clientCertificate Authentication	サーバー clientCertificate Authentication	結果
No	Never	disabled
	Supported	disabled
	Required	Error case

表 12. クライアントおよびサーバーの設定におけるクライアント証明書認証 (続き)

クライアント clientCertificate Authentication	サーバー clientCertificate Authentication	結果
Supported	Never	disabled
	Supported	enabled*
	Required	enabled*
Required	Never	Error case
	Supported	enabled*
	Required	enabled*

*ClientCertificateAuthentication は、プロトコルとして SSL が使用され、CredentialAuthentication が使用されていない場合にのみ行われます。

次のことに注意してください。クリデンシャル認証とクライアント証明書認証の両方が使用されているが、クライアントから送信されたクリデンシャルがヌルの場合、クライアント証明書認証が使用されます。

オーセンティケーターは、クラスター XML ファイルで構成されます。以下に例を示します。

```
<cluster name="cluster1" securityEnabled="true" singleSignOnEnabled="true"
loginSessionExpirationTime="300" statisticsEnabled="true"
statisticsSpec="map.all=enabled">
<serverDefinition name="server1" host="localhost" clientAccessPort="12503"
peerAccessPort="12500" workingDirectory="" traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="localhost" clientAccessPort="12504"
peerAccessPort="12501" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<authenticator
className="com.ibm.websphere.objectgrid.security.plugins.builtins.
WSTokenAuthenticator">
</authenticator>
</cluster>
```

ObjectGrid には、鍵ファイル・ユーザー・レジストリーに対するユーザー ID およびパスワード認証、LDAP サーバーに対するユーザー ID およびパスワード認証、SSL クライアント証明書単純マッピング認証、および WebSphere Application Server セキュリティー・メカニズムに関するデフォルトの認証組み込みインプリメンテーションが 4 つあります。WebSphere Application Server セキュリティー・メカニズムに対するオーセンティケーター・インプリメンテーションを除き、これらの組み込みインプリメンテーションはテスト専用です。この 2 つの組み込みの主な目的は、コードをなにも書き込まずに単純なテストができるようにすることです。WebSphere Application Server オーセンティケーター・インプリメンテーションはすぐに使用できるインプリメンテーションであり、ObjectGrid のサーバーとクライアントの両方とも同じセキュリティー・ドメインにあるときに接続することができます。

WebSphere Application Server ユーザー・レジストリーを使用する ObjectGrid サーバーの場合は、WebSphere Application Server API を使用して、アプリケーション・サーバーで構成されたユーザー・レジストリーを取得し、オーセンティケーター・

インプリメンテーションでそのレジストリーを使用します。ただし、このプログラミング・ガイドでは、このインプリメンテーションについては説明しません。

鍵ファイル・レジストリー・オーセンティケーター・インプリメンテーション

ユーザー ID とパスワードは、鍵ストア・ファイルを呼ばれるファイルに保存できます。 keytool ツールを使用すると、鍵ストア・ファイルおよびエントリーを作成できます。たとえば、以下のコマンドでは、別名 user1 を持つエントリーが作成されます。

```
keytool -genkey -v -keystore ./keys.jks -storepass password -alias user1
-keypass password -dname CN=user1,O=MyCompany,L=MyCity,ST=MyState
```

テスト目的で、ObjectGrid には、このプラグインがユーザー名およびパスワード認証を処理するための com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator デフォルト・インプリメンテーションが用意されています。このインプリメンテーションでは、ログイン名 KeyStoreLogin を使用して、ユーザーを鍵ストア・ファイルにログインさせます。

以下は、KeyStoreLoginAuthenticator クラスでの authenticate(Credential) メソッドの実装を示すコードの断片です。

```
public Subject authenticate(Credential credential)throws
InvalidCredentialException,
ExpiredCredentialException {
UserPasswordCredential cred = (UserPasswordCredential) credential;
LoginContext lc = null;
lc = new LoginContext("KeyStoreLogin",
new UserPasswordCallbackHandlerImpl(cred.getUserName(),
cred.getPassword().toCharArray()));
lc.login();
Subject subject = lc.getSubject();
```

この断片では、最初に、Credential インターフェースのインプリメンテーションである UserPasswordCredential へ Credential がキャストされます。これは、クライアントとの契約で、クライアントが UserPasswordCredential 型のオブジェクトのみ渡すことができることになっているためです。その後、ログインする KeyStoreLogin ログイン・モジュールが呼び出されます。

この目的のため、ObjectGrid には、com.ibm.websphere.objectgrid.security.plugins.builtins.KeystoreLoginModule ログイン・モジュールが同梱されています。各ユーザーのユーザー名とパスワードのペアが含まれている鍵ストア・ファイルを指定する必要があります。鍵ストア・ファイルは、ログイン・モジュールに対するオプションとして構成されます。

以下は、ログイン・モデルの鍵ファイルへのログイン方法を示すコードの断片です。

```
/**
* Authenticates a user based on the keystore file.
*
* @see javax.security.auth.spi.LoginModule#login()
*/
public boolean login() throws LoginException {
if (debug) {
System.out.println("[KeyStoreLoginModule] login: entry");
}
```

```
String name = null;
char pwd[] = null;
if (keyStore == null || subject == null || handler == null) {
throw new LoginException("Module initialization failed");
}
NameCallback nameCallback = new NameCallback("Username:");
PasswordCallback pwdCallback = new PasswordCallback("Password:", false);
try {
handler.handle(new Callback[] { nameCallback, pwdCallback });
}
catch (Exception e) {
throw new LoginException("Callback failed: " + e);
}
name = nameCallback.getName();
char[] tempPwd = pwdCallback.getPassword();
if (tempPwd == null) {
// treat a NULL password as an empty password
tempPwd = new char[0];
}
pwd = new char[tempPwd.length];
System.arraycopy(tempPwd, 0, pwd, 0, tempPwd.length);
pwdCallback.clearPassword();
if (debug) {
System.out.println("[KeyStoreLoginModule] login: "
+ "user entered user name: " + name);
}
if (ObjectGridManagerImpl.isTraceEnabled && TC.isDebugEnabled())
Tr.debug(TC, "login", "userName="+name);
// Validate the user name and password
try {
validate(name, pwd);
}
catch (SecurityException se) {
principals.clear();
publicCreds.clear();
privateCreds.clear();
LoginException le = new LoginException(
"Exception encountered during login");
le.initCause(se);
throw le;
}
if (debug) {
System.out.println("[KeyStoreLoginModule] login: exit");
}
return true;
}
/**
* Validate the user name and password based on the keystore.
*
* @param userName user name
* @param password password
* @throws SecurityException if any exceptions encountered
*/
protected void validate(String userName, char password[])
throws SecurityException {
PrivateKey privateKey = null;
// Get the private key from the keystore
try {
privateKey = (PrivateKey) keyStore.getKey(userName, password);
}
catch (NoSuchAlgorithmException nsae) {
SecurityException se = new SecurityException();
se.initCause(nsae);
throw se;
}
catch (KeyStoreException kse) {
SecurityException se = new SecurityException();
```

```
se.initCause(kse);
throw se;
}
catch (UnrecoverableKeyException uke) {
SecurityException se = new SecurityException();
se.initCause(uke);
throw se;
}
if (privateKey == null) {
throw new SecurityException("Invalid name: " + userName);
}
// Check the certificats
Certificate certs[] = null;
try {
certs = keyStore.getCertificateChain(userName);
}
catch (KeyStoreException kse) {
SecurityException se = new SecurityException();
se.initCause(kse);
throw se;
}
if (certs != null && certs.length > 0) {
// If the first certificate is an X509Certificate
if (certs[0] instanceof X509Certificate) {
try {
// Get the first certificate which represents the user
X509Certificate certX509 = (X509Certificate) certs[0];
// Create a principal
X500Principal principal = new X500Principal(certX509
.getIssuerDN()
.getName());
principals.add(principal);
if (debug) {
System.out.println(" Principal added: " + principal);
}
}
catch (CertificateException ce) {
SecurityException se = new SecurityException();
se.initCause(ce);
throw se;
}
}
}
}
```

JAAS 認証構成ファイルでログイン名「KeyStoreLogin」を作成する必要があります。 JAAS 認証構成ファイルについての知識が十分でない場合は、「JAAS Authentication Tutorial」を参照してください。

```
KeyStoreLogin {
com.ibm.websphere.objectgrid.jaas.KeystoreLoginModule required
keyStoreFile="${user.dir}${/}security${/}.keystore";
};
```

このインプリメンテーションは、テスト目的のみです。

LDAP オーセンティケーター・インプリメンテーション

ObjectGrid には、このプラグインが LDAP サーバーに対するユーザー名およびパスワード認証を処理するための com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthenticator デフォルト・インプリメンテーションが用意されています。このインプリメンテーションでは、LDAPLogin ログイン・モジュールを使用して、ユーザーを LDAP サーバーにログインさせます。

以下の断片では、認証メソッドの実装方法を示します。

```
/**
* @see com.ibm.ws.objectgrid.security.plugins.Authenticator#
* authenticate(LDAPLogin)
*/
public Subject authenticate(Credential credential) throws
InvalidCredentialException, ExpiredCredentialException {
UserPasswordCredential cred = (UserPasswordCredential) credential;
LoginContext lc = null;
try {
lc = new LoginContext("LDAPLogin",
new UserPasswordCallbackHandlerImpl(cred.getUserName(),
cred.getPassword().toCharArray()));
lc.login();
Subject subject = lc.getSubject();
return subject;
}
catch (LoginException le) {
throw new InvalidCredentialException(le);
}
catch (IllegalArgumentException ile) {
throw new InvalidCredentialException(ile);
}
}
```

この目的のため、ObjectGrid には、ログイン・モジュール com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPLoginModule が同梱されています。JAAS ログイン構成ファイルに以下の 2 つのオプションを指定する必要があります。

- **providerURL**: LDAP サーバーのプロバイダー URL
- **factoryClass**: LDAP コンテキスト・ファクトリー・インプリメンテーション・クラス

LDAPLoginModule は、com.ibm.websphere.objectgrid.security.plugins.builtins.LDAPAuthentcationHelper.authenticate メソッドを呼び出します。以下のコードの断片は、LDAPAuthentcationHelper の認証メソッドが実装される方法を示しています。

```
/**
* Authenticate the user to the LDAP directory.
* @param user the user ID, e.g., uid=xxxxxx,c=us,ou=bluepages,o=ibm.com
* @param pwd the password
*
* @throws NamingException
*/
public String[] authenticate(String user, String pwd)
throws NamingException {
Hashtable env = new Hashtable();
env.put(Context.INITIAL_CONTEXT_FACTORY, factoryClass);
env.put(Context.PROVIDER_URL, providerURL);
env.put(Context.SECURITY_PRINCIPAL, user);
env.put(Context.SECURITY_CREDENTIALS, pwd);
env.put(Context.SECURITY_AUTHENTICATION, "simple");
InitialContext initialContext = new InitialContext(env);
// Look up for the user
DirContext dirCtx = (DirContext) initialContext.lookup(user);
String uid = null;
int iComma = user.indexOf(",");
int iEqual = user.indexOf("=");
if (iComma > 0 && iComma > 0) {
uid = user.substring(iEqual + 1, iComma);
}
```

```
else {
uid = user;
}
Attributes attributes = dirCtx.getAttributes("");
// Check the UID
String thisUID = (String) (attributes.get(UID).get());
String thisDept = (String) (attributes.get(HR_DEPT).get());
if (thisUID.equals(uid)) {
return new String[] { thisUID, thisDept };
}
else {
return null;
}
}
```

認証が成功した場合、ID とパスワードは有効であるとみなされます。その後、ログイン・モジュールは、この認証メソッドから UID 情報および部門情報を取得します。ログイン・モジュールでは、2 つのプリンシパル SimpleUserPrincipal および SimpleDeptPrincipal が作成されます。認証済みサブジェクトを使用して、グループ許可 (ここでは、部門がグループです) および個々の許可を行うことができます。

以下に、LDAP サーバーへのログインに使用されるログイン・モジュール構成例を示します。

```
LDAPLogin { com.ibm.websphere.objectgrid.security.plugins.builtins.
LDAPLoginModule required
providerURL="ldap://directory.acme.com:389/"
factoryClass="com.sun.jndi.ldap.LdapCtxFactory";
};
```

前の構成では、LDAP サーバーが `ldap://directory.acme.com:389/` サーバーを指しています。この設定をご使用の LDAP サーバーに変更します。このログイン・モジュールでは、指定されたユーザー ID およびパスワードを使用して、LDAP サーバーに接続します。このインプリメンテーションは、テスト目的のみです。

WebSphere Application Server オーセンティケーター・インプリメンテーション

ObjectGrid には、WebSphere Application Server セキュリティー・インフラストラクチャーを使用するための com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator 組み込みインプリメンテーションもあります。この組み込みインプリメンテーションは、以下の状態のときに使用されます。

- WebSphere Application Server グローバル・セキュリティーがオンになっている。
- ObjectGrid クライアントおよび ObjectGrid サーバーの両方とも、WebSphere Application Server Java 仮想マシンで起動している。
- これらのアプリケーション・サーバーが、同じセキュリティー・ドメインにある。
- ObjectGrid クライアントは、すでに WebSphere Application Server で認証済みである。

ObjectGrid クライアントでは、com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator クラスを使用して、クリデンシャルを生成でき

ます。また、ObjectGrid サーバーでは、このオーセンティケーター・インプリメンテーション・クラスを使用して、クリデンシャルを認証できます。トークンが正常に認証されると、Subject オブジェクトが戻されます。

このシナリオの利点は、ObjectGrid クライアントがすでに認証済みであることです。 ObjectGrid サーバーがあるアプリケーション・サーバーが、ObjectGrid クライアントを格納するアプリケーション・サーバーと同じセキュリティー・ドメインにあるため、ObjectGrid クライアントから ObjectGrid サーバーにセキュリティー・トークンを伝搬することができます。これにより、同じユーザー・レジストリーを再認証する必要がなくなります。

単純証明書マッピング・オーセンティケーター・インプリメンテーション

ObjectGrid には、Subject オブジェクトに証明書をマップする組み込みインプリメンテーションである com.ibm.websphere.objectgrid.security.plugins.builtins.CertificateMappingAuthenticator もあります。このインプリメンテーションでは、チェーンの最初の証明書の識別名 (DN) が抽出され、この名前を持つプリンシパルが作成されます。このインプリメンテーションは、テスト目的のみです。

Tivoli Access Manager オーセンティケーター・インプリメンテーション

Tivoli Access Manager は、セキュリティー・サーバーとして幅広く使用されています。Tivoli Access Manager 提供のログイン・モジュールを使用して、オーセンティケーターを実装することができます。

Tivoli Access Manager を使用してユーザーを認証するには、Tivoli 提供の LoginModule、com.tivoli.mts.PDLoginModule で呼び出し側アプリケーションによって以下が指定されている必要があります。

- 短縮名または X.500 名 (DN) として指定されたプリンシパル名
- パスワード

LoginModule はプリンシパルを認証し、Tivoli Access Manager クリデンシャルを戻します。この LoginModule は、呼び出し側アプリケーションによって以下の情報が提供されると想定しています。

- javax.security.auth.callback.NameCallback を介して、ユーザー名
- javax.security.auth.callback.PasswordCallback を介して、パスワード

Tivoli Access Manager クリデンシャルが正常に取得されると、JAAS LoginModule によって Subject および PDPrincipal が作成されます。TAM 認証用の組み込みは、 PDLoginModule には重要ではないため、用意されていません。詳しくは、「IBM Tivoli Access Manager Authorization Java Classes Developer Reference」を参照してください。

シングル・サインオン

サーバーに対して ObjectGrid クライアントが正常に認証されると、ObjectGrid サーバーによって Subject オブジェクトが作成されます。クライアントおよびサーバーでシングル・サインオン (SSO) がサポートされている場合、Subject オブジェクトは SSO トークンに変換されます。このトークンはクライアント・サイドに戻されて、ソケットに関連付けられます。この SSO トークンは、別のサーバーで再度認証する必要がないよう認証の新しいサーバーに渡すことができます。

SSO トークンは、ObjectGrid セキュア・トークン・マネージャー・メカニズムを使用して実装されます。セキュア・トークン・マネージャーについて詳しくは、 187 ページの『ObjectGrid クラスター・セキュリティー』を参照してください。基本的に、セキュア・トークン・メカニズムでは、暗号鍵 (秘密鍵) を使用して、サーバー間でやり取りするデータを暗号化したり、暗号化解除したりし、公開/秘密鍵を使用してデータに署名します。

また、SSO トークンには、有効期限の時間も含まれています。保護ドメインに参加しているすべての製品サーバーには、同期化された独自の時間、日付、および時間帯がある必要があります。ない場合は、SSO トークンの期限切れが早すぎるため、認証または妥当性検査が失敗します (世界時が使用される場合は、これは必要ありません)。

ObjectGrid クライアントが別のサーバーに接続される際、この SSO トークンを新しいサーバーに渡すことができます。このサーバーでは SSO トークンを検証して、未署名や暗号化解除によって改ざんされていないことを確認します。また、このサーバーでは、タイム・スタンプも検査して、有効期限が切れていないことを確認します。トークンが有効な場合、クライアントは、このサーバーに対して認証する必要はありません。

SSO トークンの有効期限が切れている場合、サーバーは再度クライアントを認証する必要があります。サーバーは、クライアントに再度クリデンシャルを提供するよう要求します。

クライアントのシングル・サインオンの使用可能化

以下の 2 とおりの方法でクライアントのシングル・サインオンを使用可能にすることができます。

- **構成。** `security.ogclient.props` ファイルの SSOEnabled プロパティーを使用して、クライアント・サイドでのシングル・サインオンを使用可能にします。
- **プログラミング。** ClientSecurityConfiguration を使用して、以下のメソッドによるシングル・サインオンを使用可能にします。

```
/**
* Set whether single sign on is enabled.
* @param enabled whether single sign on is enabled for this
* client or not.
*/
void setSingleSignOnEnabled(boolean enabled);
```

サーバーのシングル・サインオンの使用可能化

サーバー・サイドでのシングル・サインオンを使用可能にするには、クラスター XML ファイルで singleSignOnEnabled 属性を true に設定します。以下に例を示します。

```
<cluster>
<objectGrid name="cluster" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300">
```

シングル・サインオンは、セキュリティーが使用可能な場合にのみ使用可能になります。

セキュア通信

ObjectGrid では、セキュア通信用に TCP/IP および SSL がサポートされています。SSL は、クライアントとサーバーの間のセキュア通信を提供します。使用される通信メカニズムは、以下のプロパティーの設定によって異なります。

- security.ogclient.props ファイルの transportType プロパティー
- security.ogserver.props ファイルの transportType プロパティー

表 13. クライアント・トランスポートおよびサーバー・トランスポートの設定で使用されるトランスポート・プロトコル

クライアント transportType	サーバー transportType	結果のプロトコル
TCP/IP	TCP/IP	TCP/IP
	SSL サポート	TCP/IP
	SSL 必須	エラー
SSL サポート	TCP/IP	TCP/IP
	SSL サポート	SSL (SSL が失敗した場合は TCP/IP)
	SSL 必須	SSL
SSL 必須	TCP/IP	エラー
	SSL サポート	SSL
	SSL 必須	SSL

SSL が使用される場合、クライアント・サイドとサーバー・サイドの両方で SSL 構成を指定する必要があります。

ObjectGrid クライアントの SSL パラメーターの構成

クライアント・サイドの SSL パラメーターは、以下の方法で構成できます。

- ファクトリー・クラス com.ibm.websphere.objectgrid.security.config.ClientSecurityConfigurationFactory を使用して、com.ibm.websphere.objectgrid.security.config.SSLConfiguration オブジェクトを作成します。詳しくは、API 資料を参照してください。
- `security.ogclient.props` ファイルでパラメーターを構成し、ClientSecurityConfigurationFactory.getClientSecurityConfiguration(String) メソッドを使用して、オブジェクト・インスタンスを取り込みます。

以下のプロパティーは、`security.ogclient.props` ファイルでの SSL 構成に使用します。

- **provider**: SSL JSSE プロバイダーを指定します。可能な値は、IBMJSSE+、IBMJSSE2、SunJSSE などです。この値は、使用している Java Development Kit (JDK) に基づいて設定します。
- **protocol**: SSL プロトコルを指定します。可能な値は、SSL+、SSLV2、SSLV3、TLS、TLSv1 などです。このプロトコル値は、使用している Java Secure Socket Extension (JSSE) プロバイダーに基づいて設定します。

- **alias**: このストリングは鍵ストアでの別名を表します。デフォルト値はありません。鍵ストアに複数の鍵ペア証明書があり、いずれか 1 つの証明書を選択したい場合は、このプロパティーを使用します。
- **keyStoreType**: SSL 鍵ストア・タイプを指定します。可能な値は、JKS+、JCEK、PKCS12 などです。この値は、使用している Java Secure Socket Extension (JSSE) プロバイダーに基づいて設定します。
- **keyStore**: クライアント公開証明書および秘密鍵を持つ鍵ストア・パス・ファイル名を指定します。たとえば、`[OBJECTGRID_HOME]/properties/DummyClientKeyFile.jks` とします。このリリースでは、ハードウェア・サポートはサポートされていません。
- **keyStorePassword**: 鍵ストア・パスを保護するパスワードを指定します。パスワードは、ObjectGrid で「xor」アルゴリズムを使用してエンコードされます。 PropFilePasswordEncoder ツールを使用して、このプロパティー・ファイルをエンコードします。エンコードされたパスワードの例: `{x0r}CDo9Hgw¥¥`
- **trustStoreType**: トラスト・ストア・タイプを指定します。可能な値は、JKS+、JCEK、PKCS12 などです。この値は、使用される JSSE プロバイダーに基づいて設定します。
- **trustStore**: サーバー公開証明書を持つトラスト・ストア・パス・ファイル名を指定します。たとえば、`[OBJECTGRID_HOME]/properties/DummyClientTrustFile.jks` とします。
- **trustStorePassword**: トラスト・ストア・パスを保護するパスワードを指定します。パスワードは、ObjectGrid で xor アルゴリズムを使用してエンコードされます。ツール PropFilePasswordEncoder を使用して、このプロパティー・ファイルをエンコードします。エンコードされたパスワードの例: `{x0r}CDo9Hgw¥`
- **certReqSubjectDN**: サーバーからの証明書所有者の識別名 (DN) に必要なストリングです。クライアントは、サーバー証明書 DN にこのストリングが含まれている場合にのみ、サーバーに接続できます。値がヌルの場合、クライアントはサーバー証明書の特定の所有者 DN を必要としません。たとえば、証明書所有者 DN が ″CN=Server1, OU=Your Organizational Unit, O=Your Organization, S=Your State,C=Your Country″ の場合、″CN=server1″, ″O=Your Organization″, ″OU=Your Organizational Unit, O=Your Organization, S=Your State,C=Your Country″ は一致しますが、″CN=server2″ および ″OU=Your Organizational Unit, L=smething, O=Your Organization, S=Your State, C=Your Country″ は一致しません。ワイルドカード・マッチングはサポートされていません。

オブジェクト・サーバーの SSL パラメーターの構成

クライアント・サイドの SSL パラメーターは、`security.ogserver.props` ファイルで構成できます。このプロパティー・ファイルは、ObjectGrid サーバーの起動時にパラメーターとして渡すことができます。

前の SSL プロパティーを除き、サーバー・サイド SSL 構成には追加プロパティーがあります。

- **clientAuthentication** (true+, false)。このプロパティーを true に設定する場合は、SSL クライアントを認証する必要があります。これは、クライアント証明書の認証とは異なります。クライアント証明書の認証では証明

書チェーンに基づいてユーザー・レジストリーに対してクライアントを認証しますが、このプロパティーでは、サーバーが適切なクライアントに接続されていることが保証されます。

ローカル ObjectGrid のセキュリティー

このトピックでは、ローカル ObjectGrid プログラミング・モデルのセキュリティーについて説明します。ローカル ObjectGrid プログラミング・モデルにおける主なセキュリティー機能は許可です。ローカル ObjectGrid プログラミング・モデルでは、どのような種類の認証もサポートされていません。ObjectGrid の外側で認証を行う必要があります。ただし、ObjectGrid には、Subject オブジェクトを取得および検証するプラグインが備えられています。

以下の 2 とおりの方法で ObjectGrid セキュリティーを使用可能にすることができます。

- **構成。** ObjectGrid XML ファイルを使用して ObjectGrid を定義し、その ObjectGrid に対するセキュリティーを使用可能にできます。以下は、`secure-objectgrid-definition.xml` ファイルで、 ObjectGridSample エンタープライズ・アプリケーションの例に使用されます。この XML ファイルでは、セキュリティーは、securityEnabled 属性を `true` に設定することによって使用可能にできます。

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

- **プログラミング。** API を使用して ObjectGrid を作成する場合は、 ObjectGrid インターフェースで次のメソッドを呼び出して、セキュリティーを使用可能にします。

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

  ローカル ObjectGrid プログラミング・モデルには、認証が存在しません。この方法にセキュリティーを設定するには、許可を構成します。この状態は、クライアント/サーバー・モデルと一貫しています。クライアント/サーバー・モデルで ObjectGrid のセキュリティーを使用可能にした場合、その ObjectGrid インスタンス上でのみ許可が使用可能になります。

認証

ローカル ObjectGrid プログラミング・モデルでは、ObjectGrid によって認証メカニズムが提供されません。 ObjectGrid は、認証に関して、アプリケーション・サーバーまたはアプリケーションのいずれかの環境に依存しています。 ObjectGrid が WebSphere Application Server または WebSphere Extended Deployment で使用される場合、アプリケーションは WebSphere Application Server セキュリティー認証メカニズムを使用できます。 ObjectGrid が Java 2 Platform, Standard Edition (J2SE) 環境で稼働している場合、アプリケーションが Java 認証・承認サービス (JAAS)

認証またはその他の認証メカニズムを使用して認証を管理する必要があります。JAAS 認証の使用方法については、「JAAS リファレンス・ガイド」を参照してください。

アプリケーションと ObjectGrid インスタンスの契約には、javax.security.auth.Subject オブジェクトを使用します。クライアントがアプリケーション・サーバーまたはアプリケーションによって認証されると、アプリケーションは認証された javax.security.auth.Subject オブジェクトを検索し、この Subject オブジェクトを使用して ObjectGrid.getSession(Subject) メソッドを呼び出すことによって、ObjectGrid インスタンスからセッションを取得できます。この Subject オブジェクトを使用して、マップ・データへのアクセスを許可します。この契約は、サブジェクト引き渡し機構と呼ばれます。 ObjectGrid.getSession(Subject) API は、以下のとおりです。

```
/**
* This API allows the cache to use a specific subject rather than the one
* configured on the ObjectGrid to get a session.
* @param subject
* @return An instance of Session
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws InvalidSubjectException the subject passed in is invalid based
* on the SubjectValidation mechanism.
*/
public Session getSession(Subject subject)
throws ObjectGridException, TransactionCallbackException, InvalidSubjectException;
```

以下のように、 ObjectGrid インターフェースの getSession メソッドを使用して Session オブジェクトを取得することもできます。

```
/**
* This returns a Session object that can be used by a single thread at a time.
* It's not allowed to share this Session object between threads without placing a
* critical section around it. While the core framework allows the object to move
* between threads, the TransactionCallback and Loader may prevent this usage,
* especially in J2EE environments. When security is enabled, this will use the
* SubjectSource to get a Subject object.
*
* If the initialize() method has not been invoked prior to the first
* getSession invocation, then an implicit initialization will occur. This ensures
* that all of the configuration is complete before any runtime usage is required.
*
* @see #initialize()
* @return An instance of Session
* @throws ObjectGridException
* @throws TransactionCallbackException
* @throws IllegalStateException if this method is called after the
* destroy() method is called.
*/
public Session getSession()
throws ObjectGridException, TransactionCallbackException;
```

API 資料で指定されているように、セキュリティーが使用可能になると、このメソッドは SubjectSource プラグインを使用して Subject オブジェクトを取得します。SubjectSource プラグインは、Subject オブジェクトの伝搬をサポートするために ObjectGrid で定義されるセキュリティー・プラグインの 1 つです。詳しくは、173 ページの『セキュリティー関連プラグイン』を参照してください。

getSession(Subject) メソッドは、ローカル ObjectGrid インスタンスでのみ呼び出すことができます。分散 ObjectGrid 構成のクライアント・サイドで getSession(Subject) メソッドを呼び出すと、例外が発生します。

セキュリティー関連プラグイン

ObjectGrid は、サブジェクト引き渡し機構に関連する 2 つのセキュリティー・プラグイン、SubjectSource プラグインと SubjectValidation プラグインを提供します。

SubjectSource プラグイン

SubjectSource プラグインは、com.ibm.websphere.objectgrid.security.plugins.SubjectSource インターフェースによって表され、 ObjectGrid を実行している環境から Subject オブジェクトを取得するために使用されます。この ObjectGrid 環境は、ObjectGrid を使用するアプリケーションや、アプリケーションをホストするアプリケーション・サーバーなどです。インターフェースは、次のとおりです。

```
/**
* This plug-in can be used to get a Subject object which represents the
* ObjectGrid client.
* This subject is then used for ObjectGrid authorization. The method
* getSubject is called by the ObjectGrid runtime when the
* ObjectGrid.getSession() method is used to get a session and the
* security is enabled.
*
* This plug-in is useful for an already authenticated client: it
* can retrieve the authenticated Subject object and then pass to the
* ObjectGrid instance. Therefore, there is no need for another
* authentication.
*
* For example, use
* Subject.getSubject(AccessControlContext)
* to get the subject associated with the AccessControlContext and
* then return it in the getSubject implementation.
*
* This plug-in can only be used in a secure domain, such as in a
* ObjectGrid server.
*
* @ibm-api
* @since WAS XD 6.0
*/
public interface SubjectSource {
/**
* Get a Subject object which can represent the ObjectGrid client.
*
* @return a Subject object
* @throws ObjectGridSecurityException any exception during the subject
* retrieving
*/
Subject getSubject() throws ObjectGridSecurityException;
}
```

サブジェクト引き渡し機構の代わりとなる SubjectSource プラグインについて考えます。サブジェクト引き渡し機構を使用すると、アプリケーションは Subject オブジェクトを検索し、それを使用して ObjectGrid セッション・オブジェクトを取得します。 SubjectSource プラグインを使用すると、ObjectGrid ランタイムは Subject オブジェクトを検索し、それを使用してセッション・オブジェクトを取得します。サブジェクト引き渡し機構は、アプリケーションに Subject オブジェクトの制御を与え、 SubjectSource プラグイン機構は Subject オブジェクトの検索からアプリケーションを解放します。

この SubjectSource プラグインを使用すると、ObjectGrid 許可に使用できる ObjectGrid クライアントを表す Subject オブジェクトを取得できます。

ObjectGrid.getSession() メソッドが呼び出されると、Subject getSubject() は、セキュリティーが使用可能な場合に ObjectGrid ランタイムに呼び出される ObjectGridSecurityException () メソッドをスローします。

ObjectGrid は、このプラグインのデフォルトのインプリメンテーション、com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl を提供します。このインプリメンテーションを使用すると、アプリケーションが WebSphere Application Server で稼働している場合、スレッドから呼び出し元サブジェクトまたは RunAs サブジェクトを検索することができます。WebSphere Application Server で ObjectGrid を使用している場合は、このクラスを SubjectSource インプリメンテーション・クラスとして構成することができます。以下に、WSSubjectSourceImpl.getSubject() の主なフローを示すコードの断片を示します。

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

その他の詳細については、SubjectSource プラグインおよび WSSubjectSourceImpl インプリメンテーションに関する API 文書を参照してください。

SubjectValidation プラグイン

もう 1 つのセキュリティー・プラグイン、SubjectValidation プラグインは、com.ibm.websphere.objectgrid.security.plugins.SubjectValidation インターフェースによって表されます。 SubjectValidation プラグインを使用すると、ObjectGrid に渡される、または SubjectSource プラグインによって検索される javax.security.auth.Subject が、改ざんされていない有効な Subject であることを検証できます。以下にインターフェースを示します。

```
/**
* This plug-in can be used to validate a javax.security.auth.Subject
* passed to the ObjectGrid is a valid subject which has not been
* tampered with.
*
* An implementation of this plug-in needs support from the Subject
* object creator, because only the creator knows whether the Subject object
* has been tampered with. However, a subject creator may
* not know whether the Subject has been tampered with. In this
* case, this plug-in should not be used.
*
* This plug-in can only be used in a secure domain, such as in a
* application server. Do not put this plug-in on the client side, it is
* ignored.
*
* @ibm-api
*
* @since WAS XD 6.0
```

```
*/
public interface SubjectValidation {
/**
* Validate the Subject has not been tampered with.
* @param subject a subject to be validated
* @return the validated Subject object
* @throws InvalidSubjectException
*/
Subject validateSubject(Subject subject) throws
InvalidSubjectException;
}
```

Subject validateSubject(Subject subject) は、 InvalidSubjectException をスローします。 SubjectValidation インターフェースのメソッドは Subject オブジェクトを取り、 Subject オブジェクトを戻します。 Subject オブジェクトが有効であると考えられるかどうか、およびどの Subject オブジェクトが戻されるかは、すべてインプリメンテーションによって決定されます。 Subject オブジェクトが有効ではない場合は、InvalidSubjectException になります。

このプラグインは、このメソッドに渡される Subject オブジェクトを信頼できない場合に使用できます。 Subject オブジェクトを検索するコードを作成するアプリケーション開発者は信頼できると考えられるため、これは稀なケースです。

Subject オブジェクトが改ざんされたかどうかは作成者のみが知っているため、このプラグインのインプリメンテーションは、Subject オブジェクト作成者からのサポートが必要です。ただし、サブジェクトの作成者が、Subject が改ざんされたかどうかを関知していない場合もあります。その場合、このプラグインの使用はお勧めできません。

ObjectGrid は、SubjectValidation のデフォルトのインプリメンテーション、com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl を提供します。このインプリメンテーションを使用すると、WebSphere 認証サブジェクトを検証できます。 WebSphere Application Server で ObjectGrid を使用している場合は、このクラスを SubjectValidation インプリメンテーション・クラスとして構成することができます。 WSSubjectValidationImpl インプリメンテーションは、この Subject オブジェクトに関連するクリデンシャル・トークンが改ざんされていない場合にのみ、この Subject オブジェクトを有効であると見なします。つまり、その Subject オブジェクトの他のパーツを変更できます。 WSSubjectValidationImpl インプリメンテーションは、WebSphere Application Server にクリデンシャル・トークンに一致するオリジナルの Subject を依頼し、そのオリジナルの Subject オブジェクトを検証済みの Subject オブジェクトとして戻します。このため、クリデンシャル・トークン以外の Subject コンテンツに加えられた変更は、無効になります。以下のコードの断片は、WSSubjectValidationImpl.validateSubject(Subject) の基本フローを示します。

```
// Create a LoginContext with scheme WSLogin and
// pass a Callback handler.
LoginContext lc = new LoginContext("WSLogin",
new WSCredTokenCallbackHandlerImpl(subject));
// When this method is called, the callback handler methods
```

```
// will be called to log the user in.
lc.login();
// Get the subject from the LoginContext
return lc.getSubject();
```

このコードの断片では、クリデンシャル・トークンのコールバック・ハンドラー・オブジェクトである WSCredTOkenCallbackHandlerImpl は、検証対象である Subject オブジェクトと共に作成されます。次に、LoginContext がログイン・スキーム「WSLogin」で作成されます。 lc.login() メソッドが呼び出されると、 WebSphere Application Server セキュリティーは Subject オブジェクトからクリデンシャル・トークンを検索し、その後、検証済みの Subject オブジェクトとして対応する Subject を戻します。

その他、詳細については、SubjectValidation および WSSubjectValidationImpl に関する API 資料を参照してください。

プラグイン構成

SubjectValidation プラグインおよび SubjectSource プラグインは、以下の 2 とおりの方法で構成できます。

- **構成。** ObjectGrid XML ファイルを使用して ObjectGrid を定義し、これら 2 つのプラグインを設定します。以下に例を示します。この例では、WSSubjectSourceImpl クラスが SubjectSource プラグインとして構成され、WSSubjectValidation クラスが SubjectValidation プラグインとして構成されます。

```
<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>
```

- **プログラミング。** ObjectGrid through API を作成する場合、次のメソッドを呼び出して、SubjectSource または SubjectValidation プラグインを設定できます。

```
/**
* Set the SubjectValidation plug-in for this ObjectGrid instance. A
* SubjectValidation plug-in can be used to validate the Subject object
* passed in is a valid Subject. Refer to {@link SubjectValidation}
* for more details.
* @param subjectValidation the SubjectValidation plug-in
*/
void setSubjectValidation(SubjectValidation subjectValidation);
/**
* Set the SubjectSource plug-in. A SubjectSource plug-in can be used
* to get a Subject object from the environment to represent the
* ObjectGrid client.
*
* @param source the SubjectSource plug-in
*/
void setSubjectSource(SubjectSource source);
```

独自の JAAS 認証コードを作成

独自の JAAS 認証コードを作成して、認証を処理できます。独自のログイン・モジュールを作成し、認証モジュール用のログイン・モジュールを構成する必要があります。

ログイン・モジュールは、ユーザーに関する情報を受け取り、ユーザーを認証します。この情報は、ユーザーの識別に使用可能な何らかのものです。例えば、ユーザー ID およびパスワード、クライアント証明書などにすることができます。情報を受け取ったら、ログイン・モジュールは有効なサブジェクトが表示されていることを検証し、次に Subject オブジェクトを作成します。現在、ログイン・モジュールのいくつかのインプリメンテーションが公開されています。

ログイン・モジュールを作成したら、そのモジュールをランタイムが使用できるように構成します。 JAAS ログイン・モジュール構成ファイルが構成されている必要があります。このログイン・モジュールには、ログイン・モジュールおよびその認証スキームが含まれています。以下に例を示します。

```
FileLogin
{
com.acme.auth.FileLoginModule required
};
```

認証スキームは ″FileLogin″ であり、ログイン・モジュールは com.acme.auth.FileLoginModule です。必須のトークンは、FileLoginModule モジュールがこのログインを検証する必要があることを示すか、スキーム全体が失敗したことを示します。

JAAS ログイン・モジュール構成ファイルの設定は、以下のいずれか 1 つの方法で行えます。

- JAAS ログイン・モジュール構成ファイルを `java.security` ファイルの **login.config.url** に設定します。例えば、`login.config.url.1=file:${java.home}/lib/security/file.login` にします。
- JAAS ログイン・モジュール構成ファイルをコマンド行から JVM 引数 **-Djava.security.auth.login.config** を使用して設定します。例えば、`-Djava.security.auth.login.config ==$JAVA_HOME/lib/security/file.login` のようにします。

ログイン・モジュールの作成および構成方法については、「JAAS Authentication Tutorial」を参照してください。

コードが WebSphere Application Server 上で実行されている場合、管理コンソールで JAAS ログインを構成し、このログイン構成をアプリケーション・サーバー構成に格納する必要があります。詳細については、「Java 認証・承認サービスのログイン構成」を参照してください。

許可

クライアントが認証された後、ObjectGrid 許可メカニズムを使用して、ObjectGrid マップ・データおよび管理タスクへのアクセスを許可することができます。

ObjectGrid の許可は、Subject オブジェクトに基づいています。 ObjectGrid は、2 種類の許可メカニズム、Java 認証・承認サービス (JAAS) 許可とカスタム許可をサポートしています。

許可クラス

マップ内のデータへの許可、および管理タスクへの許可という 2 種類の異なる ObjectGrid 許可が存在します。それぞれの許可では、許可クラスが使用されます。マップにアクセスする許可は MapPermission クラスによって表され、管理タスクを実行する許可は AdminPermission クラスによって表されます。

MapPermission クラス

ObjectGrid では、com.ibm.websphere.objectgrid.security.MapPermission パブリック・クラスは ObjectGrid リソース、特に ObjectMap または JavaMap インターフェースのメソッドへの許可を表します。 ObjectGrid は、以下の許可ストリングを定義し、ObjectMap および JavaMap のメソッドにアクセスします。

- **read**: マップからデータを読み取る許可を与えます。整数定数は MapPermission.READ として定義されます。
- **write**: マップのデータを更新する許可を与えます。整数定数は MapPermission.WRITE として定義されます。
- **insert**: マップへデータを挿入する許可を与えます。整数定数は MapPermission.INSERT として定義されます。
- **remove**: マップからデータを除去する許可を与えます。整数定数は MapPermission.REMOVE として定義されます。
- **invalidate**: マップからのデータを無効にする許可を与えます。整数定数は MapPermission.INVALIDATE として定義されます。
- **all**: すべての許可 (read、write、insert、remote、invalidate) を与えます。整数定数は MapPermission.ALL として定義されます。

([ObjectGrid_name].[ObjectMap_name]) というフォーマットの完全修飾 ObjectGrid マップ名、および許可ストリングまたは整数値を渡すことによって、MapPermission オブジェクトを構成できます。許可ストリングは「read, insert」のように、上記の許可ストリングをコンマで区切ったストリングか、すべての許可が与えられることを意味する「all」になります。許可整数値は、上記の許可整数定数のいずれか、あるいは DGMapPermission.GET|DGMapPermission.PUT などのいくつかの整数許可定数の数学的な「or」値になります。

クライアントが ObjectMap または JavaMap のメソッドを呼び出すと許可が行われます。 ObjectGrid ランタイムが、さまざまなメソッドの異なる許可を確認します。必要な許可がクライアントに与えられていない場合は、AccessControlException が発生します。

表 14. メソッドおよび必要な許可のリスト

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

許可は、メソッドによって実際に行われることに基づいてではなく、使用されるメソッドのみに基づいて行われます。例えば、put メソッドでは、レコードが存在するかどうかに基づいてレコードを挿入または更新できます。ただし、この時点では、挿入と更新のいずれのケースであるのかは識別されません。

また、ある種類の操作は、その他の種類の操作を組み合わせて達成することもできます。例えば、更新は、除去と挿入によって達成することができます。許可ポリシーを設計する場合は、このことを考慮してください。

AdminPermission

管理許可は、com.ibm.websphere.objectgrid.security.AdminPermission クラスによって表されます。 ObjectGrid では、管理許可に対して以下の 2 つの許可アクションが定義されます。

- **管理**: 任意の管理タスクを行う許可を与えます。
- **モニター**: 読み取りアクセス専用の管理タスクのみであるアクションに対する許可を与えます。

以下の表に、さまざまな許可でユーザーに与えられる操作の詳細を示します。これらの操作は、ManagementMBean インターフェースのメソッドに対応しています。

表 15. 管理タスクと管理許可の関係

操作	管理	モニター
startServer	Y	N
stopServer	Y	N
forceStopServer	Y	N
setServerTrace	Y	N
retrieveServerStatus	Y	Y
getMapStats	Y	Y
getOGStats	Y	Y
getReplicationStats	Y	Y

クライアントに管理許可がある場合は、startServer タスクを実行できます。クライアントにモニター許可がある場合、startServer タスクを実行することはできません。

許可メカニズム

ObjectGrid は、2 種類の許可メカニズム、JAAS 許可とカスタム許可をサポートしています。これは、マップ・データ・アクセス許可と管理許可の両方に適用されます。 JAAS 許可は、ユーザー中心のアクセス制御により Java セキュリティー・ポリシーを拡張します。許可は、実行されているコードだけではなく、そのコードの実行者 (プリンシパル) にも基づいても与えられます。これは、JDK 1.4 の一部です。

また、ObjectGrid は、com.ibm.websphere.objectgrid.security.plugins.MapAuthorization プラグインおよび com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization プラグインを使用したカスタム許可をサポートしています。 JAAS 許可を使用したくない場合は、独自の許可メカニズムを実装できます。カスタム許可メカニズムを使用すると、ポリシー・データベース、ポリシー・サーバー、または Tivoli Access Manager を使用して、ObjectGrid 許可を管理できます。

ObjectGrid 許可メカニズムは、以下の 2 とおりの方法で構成できます。

- **構成。** ObjectGrid XML ファイルを使用して ObjectGrid を定義し、許可メカニズムを AUTHORIZATION_MECHANISM_JAAS または AUTHORIZATION_MECHANISM_CUSTOM のいずれかに設定します。以下は、`secure-objectgrid-definition.xml` ファイルで、ObjectGridSample エンタープライズ・アプリケーションの例に使用されます。

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

- **プログラミング。** API を使用して ObjectGrid を作成する場合、以下のメソッドを呼び出して、許可メカニズムを設定できます。これは、ObjectGrid インスタンスを直接インスタンス化する場合、ローカル ObjectGrid プログラミング・モデルにのみ適用されます。

```
/**
* Set the authorization Mechanism. The default is
* com.ibm.websphere.objectgrid.security.SecurityConstants.
* AUTHORIZATION_MECHANISM_JAAS.
* @param authMechanism the map authorization mechanism
*/
void setAuthorizationMechanism(int authMechanism);
```

JAAS 許可

javax.security.auth.Subject オブジェクトは、認証済みユーザーを表します。Subject は、プリンシパルのセットから構成され、各プリンシパルはそのユーザーの ID を表します。例えば、Subject は名前のプリンシパル (「Joe Smith」) とグループのプリンシパル (「マネージャー」) を持つことができます。

JAAS 許可ポリシーを使用すると、許可は特定のプリンシパルに付与されます。 ObjectGrid は、Subject と現行のアクセス制御コンテキストを関連付けます。 ObjectMap または JavaMap に対する各メソッド呼び出しごとに、Java ランタイムによって、ポリシーで特定のプリンシパルにのみ必要な許可が与えられるかどうかが決定されます。与えられる場合、操作は、アクセス制御コンテキストに関連する Subject に指定されたプリンシパルが含まれているときにのみ許可されます。

ポリシー・ファイルのポリシー構文について理解している必要があります。JAAS 許可の詳細については、「JAAS Authorization Tutorial」を参照してください。

ObjectGrid には、ObjectMap および JavaMap メソッド呼び出しに対する JAAS 許可の確認に使用される特別なコードベースがあります。この特別なコードベースは、
`http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction`
にあります。プリンシパルに ObjectMap または JavaMap 許可を与える場合は、このコードベースを使用します。この特別なコードは、ObjectGrid Java アーカイブ (JAR)・ファイルにすべての許可が与えられるために作成されました。

MapPermission を与えるポリシーのテンプレートは、次のようになります。

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

Principal フィールドは、以下のようになります。

```
Principal Principal_class "principal_name"
```

つまり、「Principal」の後には、Principal クラスの完全修飾名とプリンシパル名が続きます。 map_name は、[ObjectGrid Name].[Map Name] というフォーマットの完全修飾マップ名です。例えば、「secureClusterObjectGrid.employees」のようになります。アクションは、「read, insert」のように、MapPermission クラスに定義される許可ストリングのコンマで区切られたストリングか、「all」です。

制限付きのワイルドカード機能がサポートされています。 ObjectGrid 名またはマップ名を「任意」のものを示す「*」と置き換えることができます。ただし、ObjectGrid は、ObjectGrid 名またはマップ名の一部の「*」による置換をサポートしていません。このため、「*.employees」、「clusterObjectGrid.*」、および「*.*」はすべて有効な名前ですが、「cluster*.employees」は無効です。

例えば、ObjectGridSample.ear サンプル・アプリケーションで、2 つの許可ポリシー・ファイル fullAccessAuth.policy と readInsertAccessAuth.policy が定義されています。readInsertAccessAuth.policy の内容は、以下のとおりです。

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

このポリシーでは、「insert」および「read」許可のみが特定のプリンシパルに対する 4 つのマップに与えられます。他のポリシー・ファイル fullAccessAuth.policy では、プリンシパルに対するこれらのマップに「all」許可が与えられます。アプリケーションを実行する前に、principal_name とプリンシパル・クラスを適切な値に変更してください。principal_name の値は、ユーザー・レジストリーに応じて異なります。例えば、ローカル OS をユーザー・レジストリーとして使用する場合は、マシン名は MACH1、ユーザー ID は user1、principal_name は「MACH1/user1」になります。

JAAS 許可ポリシーは、Java ポリシー・ファイルに直接配置するか、別の JAAS 許可ファイルに配置してから、java.security ファイルで -Djava.security.auth.policy=file:[JAAS_AUTH_POLICY_FILE] JVM 引数または auth.poliyc.url.x= file:[JAAS_AUTH_POLICY_FILE] を使用して設定します。

JAAS 許可のこの説明は、管理タスクへのアクセスを許可するポリシーを作成および構成する場合にも該当します。ただし、「com.ibm.websphere.objectgrid.security.MapPermission マップ名、アクション ;」フォーマットを使用する代わりに、

「com.ibm.websphere.objectgrid.security.AdminPermission アクション;」を使用する点のみが異なっています。アクションは、「管理」または「モニター」のいずれかです。

カスタム・マップ許可

ObjectGrid は、MapAuthorization プラグインにより、カスタム・マップ許可もサポートしています。インターフェースは、次のとおりです。

```
/**This plugin can be used to authorize ObjectMap/JavaMap accesses to the
* principals represented by the Subject object.
*
* A typical implementation of this plug-in is to retrieve the
* principals from the Subject object, and then check whether
* the specified permissions are granted to the principals.
*
*
* @ibm-api
* @since WAS XD 6.0
*/
public interface MapAuthorization {
/**
* Check whether the principals represented by the Subject object
* in the subject has the specified MapPermission. If
* the permissions are granted, true is returned; otherwise a false
* is returned.
*
* @param subject the subject
* @param permission the permission to access ObjectMap
*
* @return true if the permission is granted; false otherwise.
*/
boolean checkPermission(Subject subject, MapPermission permission);
}
```

このプラグインを使用すると、Subject オブジェクトに含まれているプリンシパルに対する ObjectMap または JavaMap アクセスを許可することができます。以下のメソッドを使用します。

```
boolean checkPermission(Subject subject, MapPermission permission)
```

MapAuthorization インターフェースは、ObjectGrid ランタイムによって呼び出され、渡されたサブジェクト・オブジェクトに、渡された許可があるかどうかを確認します。ある場合は、MapAuthorization インターフェースのインプリメンテーションによって true が返され、ない場合は false が返されます。

このプラグインの通常のインプリメンテーションは、Subject オブジェクトからプリンシパルを検索し、指定された許可が特定のポリシーを参照してプリンシパルに与えられているかどうかを確認することです。これらのポリシーは、ユーザーが定義します。例えば、ポリシーはデータベース、プレーン・ファイル、または Tivoli Access Manager で定義できます。

ObjectGrid は、このプラグインの 2 つのデフォルトのインプリメンテーションを提供します。 com.ibm.websphere.objectgrid.security.plugins.builtins.JAASMapAuthorizationImpl クラスは、許可に JAAS メカニズムを使用する MapAuthorization のインプリメンテーションです。もう 1 つのインプリメンテーション・クラスは、
com.ibm.websphere.objectgrid.security.plugins.builtins.TAMMapAuthorizationImpl

クラスです。このインプリメンテーション・クラスは、Tivoli Access Manager を使用して ObjectGrid 許可を管理する方法を示します。以下に、JAASMapAuthorizationImpl.checkPermission(Subject, MapPermission) の基本フローを示すコードの断片を示します。

```
// Creates a PrivilegedExceptionAction to check the permssions.
PrivilegedExceptionAction action =
MapPermissionCheckAction.getInstance(permission);
Subject.doAsPrivileged(subject, action, null);
```

詳しくは、「IBM Tivoli Access Manager Authorization Java Classes Developer Reference」を参照してください。

この TAMMapAuthorizationImpl プラグインは、独創的なシナリオでは使用しないでください。このプラグインは、テスト目的のみに使用するようにしてください。このプラグインには、次のような特定の制限付き前提条件が必要となります。

- Subject オブジェクトには、com.tivoli.mts.PDPrincipal プリンシパルが含まれます。
- TAM ポリシー・サーバーには、ObjectMap または JavaMap 名オブジェクトの以下の許可を定義しました。ポリシー・サーバーに定義されるオブジェクトの名前は、[ObjectGrid_name].[ObjectMap_name] フォーマットの ObjectMap または JavaMap 名と同じである必要があります。許可は、MapPermission で定義される許可ストリングの先頭文字です。例えば、ポリシー・サーバーで定義される許可「r」は、ObjectMap に対する「read」許可を表します。

以下の断片は、checkPermission メソッドをインプリメントする方法を示します。

```
/**
* @see com.ibm.websphere.objectgrid.security.plugins.
* MapAuthorization#checkPermission
* (javax.security.auth.Subject, com.ibm.websphere.objectgrid.security.
* MapPermission)
*/
public boolean checkPermission(final Subject subject,
MapPermission permission) {
String[] str = permission.getParsedNames();
StringBuffer pdPermissionStr = new StringBuffer(5);
for (int i=0; i<str.length; i++) {
pdPermissionStr.append(str[i].substring(0,1));
}
PDPermission pdPerm = new PDPermission(permission.getName(),
pdPermissionStr.toString());
Set principals = subject.getPrincipals();
Iterator iter= principals.iterator();
while(iter.hasNext()) {
try {
PDPrincipal principal = (PDPrincipal) iter.next();
if (principal.implies(pdPerm)) {
return true;
}
}
catch (ClassCastException cce) {
// Handle exception
```

```
}
}
return false;
}
```

MapAuthorization プラグインは、以下の方法で構成することができます。

- **構成。** ObjectGrid XML ファイルを使用して、MapAuthorization プラグインを定義できます。以下に例を示します。

```
<objectGrids>
<objectGrid name="secureClusterObjectGrid" securityEnabled="true"
authorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
...
<bean id="MapAuthorization"
className="com.ibm.websphere.objectgrid.security.plugins.builtins.
JAASMapAuthorizationImpl" />
</objectGrids>
```

- **プログラミング。** API を使用して ObjectGrid を作成する場合、以下のメソッドを呼び出して、許可プラグインを設定できます。これは、ObjectGrid インスタンスを直接インスタンス化する場合、ローカル ObjectGrid プログラミング・モデルのみに適用されます。

```
/**
* Sets the MapAuthorization plug-in for this ObjectGrid instance.
*
* A {@link MapAuthorization} plug-in can be used to authorize
* access to the maps. Refer to {@link MapAuthorization}
* for more details.
* @param mapAuthorization the MapAuthorization plug-in
*/
void setMapAuthorization(MapAuthorization mapAuthorization);
```

カスタム管理許可

カスタム・マップ・データ・アクセス許可サポートと同様、ObjectGrid ではカスタム管理許可をサポートしています。プラグインは、com.ibm.websphere.objectgrid.security.plugins です。

```
/**
* This plug-in can be used to authorize management operations to the
* principals contained in the Subject object. The permissions for the
* management operations are represented by AdminPermission
* objects.
*
* This plug-in is used in an ObjectGrid server. It can be configured in the
* ObjectGrid cluster XML file.
*
* A typical implementation of this plug-in is to retrieve the
* Principal set from the Subject object, and then
* check whether the specified permissions are granted to these principals.
*
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see AdminPermission
*/
public interface AdminAuthorization {
/**
* Checks whether the user represented by the Subject object has the
* specified AdminPermission or not.
*
* If the permissions are granted, true is returned; otherwise
* false is returned.
*
```

```
* @param subject the Subject object representing the user
* @param permission the administration permission to check
*
* @return true if the permission is granted; false otherwise.
*
* @see AdminPermission
*/
boolean checkPermission(Subject subject, AdminPermission permission);
}
```

このプラグインを使用すると、Subject オブジェクトに含まれているプリンシパルに対する管理アクセスを許可することができます。
AdminAuthorization インターフェースの

```
boolean checkPermission(Subject subject, AdminPermission permission)
```

メソッドは、ObjectGrid ランタイムによって呼び出され、渡されたサブジェクト・オブジェクトに、渡された管理許可があるかどうかを確認します。該当する場合は、AdminAuthorization インターフェースのインプリメンテーションによって true が返され、該当しない場合は false が返されます。

このインターフェースは、セキュリティー要件に基づいてインプリメントすることができます。ObjectGrid には、このインターフェースに対するインプリメンテーション・クラスが同梱されていません。

AdminAuthorization プラグインは、クラスター XML のクラスター・レベルで設定できます。以下に例を示します。

```
<cluster name="cluster1" securityEnabled="true"
singleSignOnEnabled="true" loginSessionExpirationTime="300"
statisticsEnabled="true"
statisticsSpec="map.all=enabled">
<serverDefinition name="server1" host="localhost"
clientAccessPort="12503" peerAccessPort="12500" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<serverDefinition name="server2" host="localhost"
clientAccessPort="12504" peerAccessPort="12501" workingDirectory=""
traceSpec="ObjectGrid=all=disabled"
systemStreamToFileEnabled="true" />
<authenticator className ="com.ibm.websphere.objectgrid.security.plugins.
builtins.WSTokenAuthenticator"></authenticator>
<adminAuthorization className= "com.ibm.ws.objectgrid.test.security.util.
TestAdminAuthorization"></adminAuthorization>
</cluster>
```

許可確認期間

ObjectGrid は、パフォーマンス上の理由で、マップ許可確認結果のキャッシングをサポートしています。このメカニズムがないと、 179 ページの表 14 にリストされているメソッドが呼び出された際に、ObjectGrid ランタイムが、構成済みの許可メカニズムを呼び出してアクセスを許可します。この許可検査期間が設定されていると、許可メカニズムは許可検査期間に基づいて定期的に呼び出されます。

許可情報は、Subject オブジェクトに基づいてキャッシュします。クライアントがメソッドにアクセスしようとすると、ObjectGrid ランタイムによって、Subject オブジェクトに基づいてキャッシュが検索されます。キャッシュ内に見つからない場合は、ランタイムによって、この Subject オブジェクトに対して与えられている許可が確認され、許可がキャッシュに格納されます。

許可検査期間は、ObjectGrid が初期化される前に定義しておく必要があります。許可検査期間は、以下の 2 とおりの方法で構成できます。

- **構成。** ObjectGrid XML ファイルを使用して ObjectGrid を定義し、許可検査期間を設定できます。以下に、許可検査期間を 45 秒に設定する例を示します。

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

- **プログラミング。** API を使用して ObjectGrid を作成する場合、以下のメソッドを呼び出して、許可検査期間を設定します。このメソッドは、ObjectGrid インスタンスを初期化する前にのみ呼び出すことができます。このメソッドは、ObjectGrid インスタンスを直接インスタンス化する場合、ローカル ObjectGrid プログラミング・モデルにのみ適用されます。

```
/**
* This method takes a single parameter indicating how often the customer
* wants to check the permission used to allow a client access. If the
* parameter is 0 then every single get/put/update/remove/evict call will
* ask the authorization mechanism, either JAAS authorization or custom
* authorization to check if the current subject has permission. This may be
* prohibitively expensive from a performance point of view depending on
* the authorization implementation but if this is required then you can do it.
* Alternatively, if the parameter is > 0 then it indicates the number
* of seconds to cache a set of permissions before returning to
* the authorization mechanism to refresh them. This provides much
* better performance but you run the risk that if the backend
* permissions are changed during this time then the ObjectGrid will
* possibly allow or prevent access even though the backend security
* provider has been modified.
*
* @param period the permission check period in seconds.
*/
void setPermissionCheckPeriod(int period);
```

ObjectGrid クラスター・セキュリティー

ObjectGrid クラスター・セキュリティーによって、結合サーバーのクリデンシャルが適切であることが保証されるため、悪意のあるサーバーはクラスターを結合することができません。このために、ObjectGrid では共有秘密ストリング・メカニズムが使用されています。

すべての ObjectGrid サーバーは共有秘密ストリングと一致します。サーバーがクラスターに結合する場合、秘密ストリングの表示を求められます。結合サーバーの秘密ストリングがプレジデント・サーバーのいずれかの秘密ストリングと一致する場合、結合サーバーはクラスターと結合することができます。一致しない場合、結合要求は拒否されます。

平文の機密事項の送信は保護されません。ObjectGrid セキュリティー・インフラストラクチャーには、セキュア・トークン・マネージャー・プラグインがあり、サーバーは機密事項を送信する前に "セキュア" できます。 "セキュア" 操作のインプリメント方法を決定する必要があります。ObjectGrid にはすぐに使用できるインプリメンテーションがあり、"セキュア" 操作がインプリメントされて秘密を暗号化し、署名します。

秘密ストリング (authenticationSecret) は、security.ogserver.props ファイルに設定されます。

- authenticationSecret: サーバーに示される秘密ストリング。サーバーは、始動すると、このストリングをプレジデント・サーバーに示す必要があります。秘密ストリングがプレジデント・サーバーの秘密ストリングと一致した場合、このサーバーはクラスターを結合できます。

SecureTokenManager プラグイン

セキュア・トークン・マネージャー・プラグインは、com.ibm.websphere.objectgrid.security.plugins.SecureTokenManager インターフェースによって表されます。インターフェースは、次のとおりです。

```
package com.ibm.websphere.objectgrid.security.plugins;
import com.ibm.websphere.objectgrid.security.ObjectGridSecurityException;
import com.ibm.websphere.objectgrid.security.SecurityConstants;
/**
* This interface is used by ObjectGrid servers to transform an object to a
* secure token and vice versa. A secure token is a byte array.
* Here is one example of a possible usage: When a server joins the cluster,
* the joining server needs to present a password to the president server in the
* cluster. Before sending the password out, the joining server calls the
* generateToken(Object) method to generate a token for this
* password. The token should be hard to break so the password can be protected
* securely. The token will then be sent across the wire. Usually the token is
* associated with a time stamp so the malicious replay attack will be difficult.
* On the receving side, the server calls the verifyToken(byte[])
* method to verfiy the token and reconstruct the corresponding object from the
* token.
*
*ObjectGrid utilizes JCE to provide a default implementation of this
* interface. In this implementation, when generating the token, the object is
* encrypted with a time stamp and then signed. To verify a token, the token's
* signature is verified and then decrypted. This implementation will need a key
* store configured in the ObjectGrid servers to support the data
* encrypting and decrypting and signature signing and verifying. Please use
* security.ogserver.props for the secure token key settings.
*
* An implementation class should have a default constructor. Users can set the
* CustomSecureTokenManagerProps property in the server security configuration
* property file. This property will be set on the object using the
* setProperties(String) method.
*
* @ibm-api
* @since WAS XD 6.0.1
*
* @see SecurityConstants#SECURE_TOKEN_MANAGER_CUSTOM_STRING
* @see SecurityConstants#SECURE_TOKEN_MANAGER_DEFAULT_STRING
*/
public interface SecureTokenManager {
/**
*
Set the user defined properties to the factory
*
*
This method is used to set addtional SecureTokenManager properties
* to the object. These properties can be set using the SecureTokenManagerProps
* property in the server security configuration property file.
* This way, you can customize your factory.
*
* @param properties user defined properties
*/
void setProperties(String properties);
/**
```

```
* Generates the token for the specified object.
*
* The generated token should be hard to break.
*
* @param o the object to be protected
*
* @return a token representing the object to be protected
*
* @throws ObjectGridSecurityException if any exception occurs during
* generation of the token byte array
*/
byte[] generateToken(Object o) throws ObjectGridSecurityException;
/**
* Verifies the token and reconstruct the object.
*
* @param bytes the token byte array representing the protected object.
*
* @return the protected object
*
* @throws ObjectGridSecurityException if any exception occurs during
* verification of the token byte array
*/
Object verifyToken(byte[] bytes) throws ObjectGridSecurityException;
}
```

generateToken(Object) メソッドは保護されるオブジェクトを取り、外部に識別されないトークンを生成します。verifyTokens(byte[]) メソッドは逆に、トークンを元のオブジェクトに変換して戻します。

単純な SecureTokenManager インプリメンテーションは単純なエンコード・アルゴリズム (XOR アルゴリズムなど) を使用して、オブジェクトを直列化済みフォームでエンコードし、対応するデコード・アルゴリズムを使用してトークンをデコードします。このインプリメンテーションは保護されていないため、簡単に中断されます。

ObjectGrid には、このインターフェースに対してすぐに使用できるインプリメンテーションがあります。このインプリメンテーションは公開 API ではなく、ユーザーに対して透過的です。

デフォルトのインプリメンテーションでは鍵ペアを使用して、署名し、シグニチャーを検査します。また、秘密鍵を使用して、コンテンツを暗号化します。これを行うため、すべてのサーバーには JCKES タイプの鍵ストアがあり、鍵ペア (秘密鍵と公開鍵) および秘密鍵が保存されています。鍵ストアは、秘密鍵を保存する JCKES タイプである必要があります。

これらの鍵は、送信側で秘密ストリングを暗号化し、署名または検証する場合に使用されます。また、トークンは、有効期限の時間に関連しているため、特定の時間が経過すると期限が切れます。受信側で、データの検証、暗号化解除、および受信側の秘密ストリングとの比較が行われます。サーバーのペアの間での認証には、SSL のような通信プロトコルは必要ありません。これは、秘密鍵と公開鍵の目的が同じであるためです。ただし、サーバー通信が暗号化されていない場合は、通信時に侵入者にデータを盗まれる可能性があります。トークンの有効期限が近いため、リプレイ・アタックの危険性は少なくなっています。この可能性は、すべてのサーバーをファイアウォールの後ろにデプロイすると、非常に小さくなります。

この方法の欠点は、ObjectGrid 管理者が鍵を生成し、生成した鍵をすべてのサーバーに送信する必要があるため、セキュリティー上の問題が発生する可能性があることです。

構成

セキュア・トークン・マネージャーを使用するには、security.ogserver.props ファイルに以下のプロパティーが構成されている必要があります。

- **secureTokenManagerType** プロパティー: このプロパティーは、使用するセキュア・トークン・マネージャーを示します。
  - 値が **none** の場合、セキュア・トークン・マネージャーは使用されません。
  - 値が **default** の場合、デフォルトのすぐに使用できるセキュア・トークン・マネージャーが使用されます。
  - 値が **custom** の場合、ユーザー提供のセキュア・トークン・マネージャーが使用されます。
- **customSecureTokenManagerClass** プロパティー: このプロパティーは、SecureTokenManager インプリメンテーション・クラスを指定します。このプロパティーは、secureTokenManagerType 値が ″custom″ の場合にのみ使用されます。このインプリメンテーション・クラスでは、デフォルトのコンストラクターがインスタンス化される必要があります。
- **customSecureTokenManagerProps** プロパティー: このプロパティーは、カスタム SecureTokenManager プロパティーを指定します。このプロパティーは、secureTokenManagerType 値が ″custom″ の場合にのみ使用されます。この値は、setProperties(String) メソッドにより SecureTokenManager オブジェクトに設定されます。
- secureTokenManagerType 値がデフォルトに設定されている場合は、以下の署名鍵および暗号鍵の構成が必要となります。
  - secureTokenKeyStore: 公開鍵と秘密鍵のペアおよび秘密鍵が保存されている鍵ストアのファイル・パス名を指定します。
  - secureTokenKeyStoreType: JCKES などの鍵ストア・タイプを指定します。この値は、使用している Java Secure Socket Extension (JSSE) プロバイダーに基づいて設定します。ただし、この鍵ストアは秘密鍵をサポートできる必要があります。
  - secureTokenKeyStorePassword: 鍵ストアを保護するパスワードを指定します。
  - secureTokenKeyPairAlias: 署名および検証に使用される公開鍵と秘密鍵のペアの別名を指定します。
  - secureTokenKeyPairPassword: 署名および検証に使用される鍵ペアの別名を保護するパスワードを指定します。
  - secureTokenSecretKeyAlias: 暗号化に使用される秘密鍵の別名を指定します。
  - secureTokenSecretKeyPassword: 秘密鍵を保護するパスワードを指定します。
  - secureTokenCipherAlgorithm: 暗号化に使用されるアルゴリズムを指定します。この値は、使用される JSSE プロバイダーに基づいて設定します。
  - secureTokenSignAlgorithm: オブジェクトの署名に使用されるアルゴリズムを指定します。この値は、使用される JSSE プロバイダーに基づいて設定します。

ゲートウェイ・セキュリティー

ObjectGrid 管理ゲートウェイは、ObjectGrid サーバーにクライアント管理要求を委任するポイントとして機能します。このトピックでは、ゲートウェイ・アクセスの保護方法について説明します。

以下の図に、例を示します。ObjectGrid クライアントは、クラスターからの統計を取得する必要がある場合、最初にゲートウェイに要求を送信します。ゲートウェイは、この要求を両方のサーバーに送信して、複数の統計を取得し、これらの統計を結合します。結合された統計は、クライアントに送り返されます。

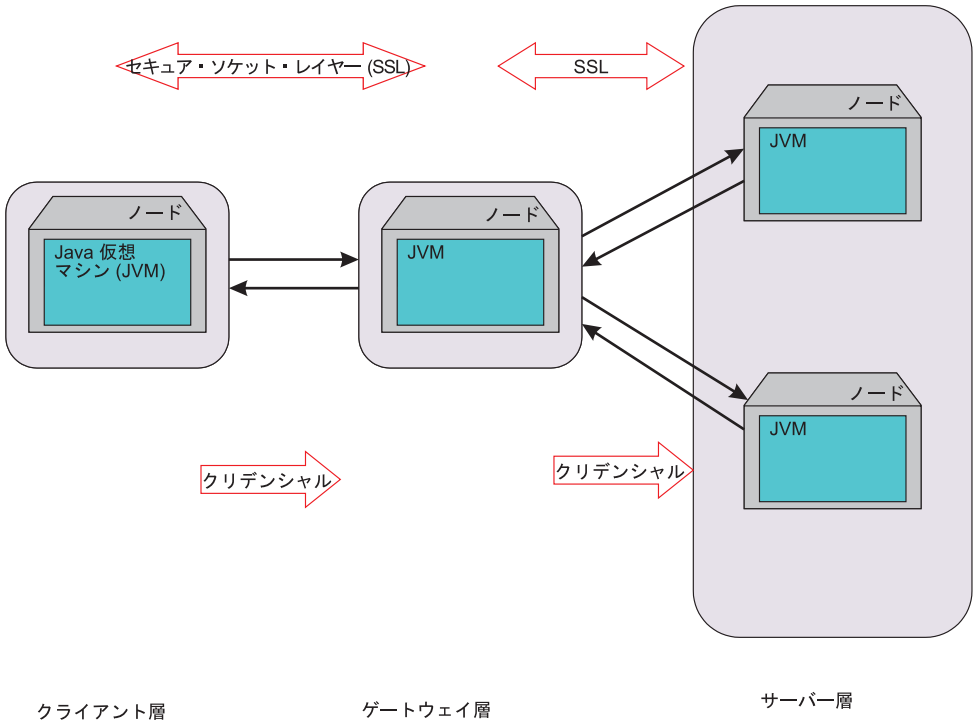


図 18. ゲートウェイ・セキュリティー

ゲートウェイとサーバーの通信では、ObjectGrid クライアント・サーバー通信メカニズムが使用されます。ゲートウェイは、ObjectGrid クライアントとして処理されます。クライアントとゲートウェイの通信は、SSL によって保護されます。この機能は、オープン・ソース・プロジェクト mx4j である JMX コネクター・レイヤーによって提供されます。 ObjectGrid では、ゲートウェイを動作させるために、指定された場所に mx4j を配置する必要があります。

認証では、ゲートウェイは、ユーザー ID とパスワードなどのクリデンシャルを伝搬します。クリデンシャルは、クライアントによってサーバーに示されます。認証および許可は、ObjectGrid サーバー上で実行されます。

ゲートウェイ・クライアントのクライアント証明書認証は、サポートされていません。

ゲートウェイ・サーバー・セキュリティー

ゲートウェイ・サーバーは、ObjectGrid クライアントです。セキュリティーに関しては、すべて ObjectGrid クライアントと同じです。コマンド行からゲートウェイ・サーバーを開始する方法については、90 ページの『管理ゲートウェイ・サーバーの始動』を参照してください。

以下のコード・スニペットは、セキュア・ゲートウェイをプログラマチックに開始する方法を示しています。

```
// Get the ClientSecurityConfiguration from the client security property file
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration("etc/test/security/security.client.props");
CredentialGenerator creGen = new UserPasswordCredentialGenerator("admin",
"xxxxxx");
csConfig.setCredentialGenerator(credGen);
// Initialize the gateway
ManagementGateway gateway = ManagementGatewayFactory.getManagementGateway();
gateway.setConnectorPort(namingPort);
gateway.setClusterName("cluster1");
gateway.setHost("localhost");
gateway.setPort("12503");
gateway.setTraceEnabled(true);
gateway.setTraceSpec("ObjectGrid=all=enabled");
gateway.setTraceFile("logs/GatewayTrace.log");
// Set the ClientSecurityConfiguration object
gateway.setCsConfig(csConfig);
// Start the gateway
gateway.startConnector();
```

前出のコードでは、ClientSecurityConfiguration オブジェクトが作成され、ManagementGateway インスタンスに設定されます。

ゲートウェイ・クライアント・セキュリティー

ゲートウェイ・クライアントは、接続時間にクリデンシャルをゲートウェイ・サーバーに渡す必要があります。以下のコード・スニペットは、クリデンシャルを渡す方法を示しています。

```
/**
* retrieve the server status from the gateway
*/
public boolean retrieveServerStatus()
throws Exception {
String serverProtocol = "rmi";
String serverHost = "host";
String namingHost = "localhost";
String jndiPath = "/jmxconnector";
JMXServiceURL url = new JMXServiceURL("service:jmx:" + serverProtocol + "://"
+ serverHost + "/jndi/rmi://" + namingHost + ":" + namingPort + jndiPath);
// Create the JMXCconnectorServer
JMXConnector cntor = JMXConnectorFactory.newJMXConnector(url, null);
// The connection environment map
Map environment = new HashMap();
// create a credential
UserPasswordCredential gatewayClientCred =
new UserPasswordCredential("admin", "admin1");
environment.put(JMXConnector.CREDENTIALS, gatewayClientCred);
// Connect and invoke an operation on the remote MBeanServer
try {
cntor.connect(environment);
}
catch (SecurityException x) {
```

```
// Uh-oh ! Bad credentials !
throw x;
}
// Obtain a stub for the remote MBeanServer
mbsc = cntor.getMBeanServerConnection();
Iterator it = mbsc.queryMBeans(
new ObjectName("ManagementServer:type=ObjectGrid,S=server1"),
null).iterator();
ObjectInstance oi = (ObjectInstance) it.next();
server1MBean = oi.getObjectName();
boolean status = ((Boolean) mbsc.invoke(
server1MBean,
"retrieveServerStatus",
new Object[] {},
new String[] {})).booleanValue();
return status;
}
```

このコード・スニペットでは、gatewayClientCred object が作成され、環境に配置されます。この環境を使用して、ゲートウェイ・サーバーに接続します。

SSL を使用して、ゲートウェイ・クライアントからゲートウェイ・サーバーに接続する場合は、システム・プロパティーを使用してトラスト・ストアおよびトラスト・ストア・パスワードを保存する必要があります。例えば、ゲートウェイ・クライアントの開始時に以下のプロパティーで渡すことができます。

- `-Djavax.net.ssl.trustStore=etc/test/security/client.public`
- `-Djavax.net.ssl.trustStorePassword=public`

詳しくは、MX4J - Open Source Java Management Extensions Web サイトを参照してください。

WebSphere Application Server とのセキュリティーの統合

ObjectGrid には、WebSphere Application Server セキュリティー・インフラストラクチャーと統合するためのセキュリティー・フィーチャーが複数あります。

WebSphere Application Server との分散 ObjectGrid セキュリティーの統合

分散 ObjectGrid モデルの場合、セキュリティー統合は以下のクラスを使用して行われます。

- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredentialGenerator.`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenCredential`

  .

これらのクラスについては、150 ページの『クライアント・サーバー・セキュリティー』を参照してください。以下に、WSTokenCredentialGenerator クラスの使用例を示します。

```
/**
* connect to the ObjectGrid Server.
*/
```

```
protected ClientClusterContext connect() throws ConnectException {
ClientSecurityConfiguration csConfig = ClientSecurityConfigurationFactory
.getClientSecurityConfiguration(proFile);
CredentialGenerator gen = getWSCredGen();
csConfig.setCredentialGenerator(gen);
return objectGridManager.connect(csConfig, null);
}
/**
* Get a WSTokenCredentialGenerator
*
private CredentialGenerator getWSCredGen() {
WSTokenCredentialGenerator gen = new WSTokenCredentialGenerator(
WSTokenCredentialGenerator.RUN_AS_SUBJECT);
return gen;
}
```

サーバー・サイドでは、WSTokenAuthentication がオーセンティケーターとして使用されて、WSTokenCredential オブジェクトが認証されます。

WebSphere Application Server とのローカル ObjectGrid セキュリティーの統合

ローカル ObjectGrid モデルの場合、セキュリティー統合は以下の 2 つのクラスを使用して行われます。

- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectSourceImpl`
- `com.ibm.websphere.objectgrid.security.plugins.builtins.WSSubjectValidationImpl`

これらのクラスについての詳細は、171 ページの『ローカル ObjectGrid のセキュリティー』を参照してください。 WSSubjectSourceImpl クラスを SubjectSource プラグインとして構成し、WSSubjectValidationImpl クラスを SubjectValidation プラグインとして構成することができます。

リスナー

ObjectGrid は、拡張可能な 2 つのリスナー・タイプのインターフェースを提供します。この拡張により、拡張インターフェースを通してユーザーに助言を与えることができ、 ObjectGrid インスタンスやマップ・インスタンスで実行される操作について解説できます。

ObjectGridEventListener インターフェース

ObjectGridEventListener インターフェースを使用して、ObjectGrid に重大なイベントが発生したときに通知を受け取ります。イベントの例として、ObjectGrid の初期化、トランザクションの開始、トランザクションの終了、および ObjectGrid の破棄などがあります。これらのイベントを listen するには、 ObjectGridEventListener インターフェースを実装するクラスを作成して、ObjectGrid に追加します。

ObjectGridEventListener インターフェース

ObjectGridEventListener インターフェースには以下のメソッドがあります。これらのメソッドは、特定の重大なイベントが ObjectGrid に発生したときに呼び出されます。

```
/**
* This method is invoked when the ObjectGrid itself is initialized.
* A usable Session instance is passed into this Listener to allow the
```

```
* optional replaying of a received LogSequence into a Map.
*
* @param session The Session instance that this Listener is associated with.
*/
void initialize(Session session);
/**
* This event signals the beginning of a transaction (session).
* A stringified version of the TxID is provided for
* correlating with the end of the transaction
* (session), if you want to use this version. The type of
* transaction (session) is
* also provided via the isWriteThroughEnabled boolean parameter.
*
* @param txid Stringified version of the TxID
* @param isWriteThroughEnabled Boolean flag indicating whether the
* Session was started via beginNoWriteThrough
*/
void transactionBegin(String txid, boolean isWriteThroughEnabled);
/**
* This signals the ending of a transaction (session). A stringified
* version of the TxID is provided for correlating with the
* begin of the transaction
* (session), if so desired. Changes are also reported. Typical uses of
* this event are for custom peer invalidation
* or peer commit push. This event
* listener outputs the changes. Calls to this method are made
* after commit and are sequenced so that they are delivered one by one,
* not in parallel. The event order is the commit order.
*
* @param txid Stringified version of the TxID
* @param isWriteThroughEnabled a boolean flag indicating
* whether the Sesison was
* started via beginNoWriteThrough
* @param committed a boolean flag indicating whether the Session
* was committed
* (true) or rolled back (false)
* @param changes A Collection of LogSequences that have been
* processed for the current Session.*/
void transactionEnd(String txid, boolean isWriteThroughEnabled,
boolean committed, Collection /*/* <LogSequence> */*/ changes);
/**
* This method will be invoked when the ObjectGrid is destroyed. It's the
* opposite of initialize. When this method is called, the
* ObjectGridEventListener can free up any resource it uses.
*/
void destroy();
```

ObjectGridEventListeners オブジェクトの追加と除去

ObjectGrid は、複数の ObjectGridEventListener を持つことが可能です。ObjectGridEventListeners の追加を許可する ObjectGrid には、2 つのメソッドがあります。追加された ObjectGridEventListeners を ObjectGrid から除去することもできます。

addEventListener メソッドを使用して、ObjectGrid に ObjectGridEventListener を追加することも可能です。

```
/**
* Add an EventListener to the Session. Significant events
* will be communicated to interested listeners via this callback.
* Multiple event listeners are allowed to be registered, with no
* implied ordering of event notifications.
*
* Note, this method is allowed to be invoked before and after the
* {@link ObjectGrid#initialize()} method.
*
```

```
* @param cb An instance of ObjectGridEventListener
*/
void addEventListener(ObjectGridEventListener cb);
```

ObjectGridEventListeners のリストに追加するには、setEventListeners メソッドを使用します。

```
/**
* This overwrites the current list of callbacks and replaces it with the
* supplied list of callbacks.
*
* Note, this method is allowed to be invoked before and after the
* {@link ObjectGrid#initialize()} method.
* @param callbacks
*/
void setEventListeners(List callbacks);
```

ObjectGridEventListener を ObjectGrid から除去するには、removeEventListener メソッドを使用します。

```
/**
* Removes an EventListener from the Session. If the desired EventListener
*is not found on the Session, no error will be returned.
*
* Note, this method is allowed to be invoked before and after the
* {@link ObjectGrid#initialize()} method.
* @param cb An instance of ObjectGridEventListener
*/
void removeEventListener(ObjectGridEventListener cb);
```

カスタム ObjectGrid イベント・リスナーの作成

カスタム ObjectGrid イベント・リスナーを使用するには、最初に com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener インターフェースを実装するクラスを作成します。カスタム・リスナーを ObjectGrid に追加して、重大なイベントの通知を受け取ります。 ObjectGridEventListener は、プログラマチックに構成することも、XML で構成することもできます。

- **プログラマチックな構成。**ObjectGrid イベント・リスナーのクラス名が、com.company.org.MyObjectGridEventListener クラスであると仮定します。このクラスは ObjectGridEventListener インターフェースを実装します。以下のコードの断片は、カスタム ObjectGridEventListener を作成し、それを ObjectGrid に追加します。

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

- **XML を使用する。** ObjectGridEventListner を、XML を使用して構成することも可能です。以下の XML は、上述のプログラムで作成された ObjectGrid イベント・リスナーと同等の構成を作成します。以下のテキストは、myGrid.xml ファイルに存在しなければなりません。

  ```
  <?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>
```

このファイルを ObjectGridManager に提供することで、簡単にこの構成を作成できます。以下のコードの断片は、この XML ファイルを使用した ObjectGrid の作成方法を示すものです。作成した ObjectGrid の myGrid ObjectGrid には、ObjectGridEventListener セットがあります。

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

MAP での変更の監視

ObjectGridEventListener インターフェースの transactionEnd メソッドは、ローカル Map 内のエントリーを監視するアプリケーションには非常に有用です。アプリケーションは、これらのリスナーの 1 つを追加して、transactionEnd メソッドを使用してエントリーが変更されたときに参照することができます。例えば、ObjectGrid が分散モードで作動している場合、アプリケーションは受信した変更を監視できます。複製されたエントリーが最新の株価に関するものであったと仮定します。このリスナーは、これらの着信する変更を監視でき、ポートフォリオの位置の値を保持する 2 次 Map を更新できます。このリスナーは、ObjectGridEventListener インターフェースの initialize メソッドのリスナーに提供されるセッションを使用して、すべての変更を行う必要があります。リスナーは、通常、トランザクションがライトスルーであるかどうかを検査することで、ローカル側の変更と着信するリモート側の変更とを区別できます。対等の ObjectGrid から着信する変更は、常にライトスルーです。

MapEventListener インターフェース

MapEventListener インターフェースを使用して、マップに関する重大なイベントを受け取ります。エントリーがマップから除去されたとき、およびマップのプリロードが完了したときに、イベントは MapEventListener に送られます。

MapEventListener インターフェース

MapEventListener インターフェースには、以下のメソッドがあります。com.ibm.websphere.objectgrid.plugins.MapEventListener インターフェースを実装して、カスタム MapEventListener を作成します。

```
/**
* This method is invoked when the specified entry is evicted from
* the map. The eviction could have occurred either by Evictor
* processing or by invoking one of the invalidate methods on the
* ObjectMap.
*
* @param key The key for the map entry that was evicted.
* @param value The value that was in the map entry evicted. The value
* object should not be modified.
*
*/
void entryEvicted(Object key, Object value);
/**
* This method is invoked when preload of this map has completed.
*
```

```
* @param t A Throwable object that indicates if preload completed without
* any Throwable occurring during the preload of the map. A null reference
* indicates preload completed without any Throwable objects occurring
* during the preload of the map.
*/
void preloadCompleted( Throwable t );
```

MapEventListeners の追加と除去

以下の BackingMap メソッドによって、MapEventListeners をマップに追加したり、マップから除去したりできます。

```
/**
* Adds a MapEventListener to this BackingMap.
*
* Note, this method is allowed to be invoked before and after the
* ObjectGrid.initialize() method.
* @param eventListener A non-null reference to a MapEventListener to add
* to the list.
*
* @throws IllegalArgumentException if eventListener is null.
*
* @see MapEventListener
*/
public void addMapEventListener(MapEventListener
eventListener );
/**
* Sets the list of MapEventListener objects.
*
* If this BackingMap already has a List of
* MapEventListeners, that list is replaced by the
* List passed as an argument to the current invocation
* of this method. This method can be called before and
* after the ObjectGrid.initialize() method.
*
* @param eventListenerList A non-null reference to a List of
* MapEventListener objects.
*
* @throws IllegalArgumentException is thrown if
* eventListenerList is null
* or the eventListenerList contains either a null
* reference or an object that is not an instance of
* MapEventListener.
*
* @see MapEventListener
*/
public void setMapEventListeners( List /*MapEventListener*/
eventListenerList );
/**
* Removes a MapEventListener from this BackingMap.
*
* Note, this method is allowed to be invoked before and after the
* ObjectGrid.initialize() method.
*
* @param eventListener A non-null reference to an event listener
* that was previously added by invoking either the
* addMapEventListener(MapEventListener) or
* setMapEventListeners(List) method of this interface.
*
* @throws IllegalArgumentException if eventListener is null.
*
* @see MapEventListener
*/
public void removeMapEventListener(MapEventListener eventListener );
```

MapEventListener の作成

カスタム MapEventListener を作成するには、

com.ibm.websphere.objectgrid.plugins.MapEventListener インターフェースを実装します。使用する MapEventListener を、BackingMap に追加します。MapEventListener は、プログラマチックに作成および構成することも、XML を使用して作成および構成することもできます。

- **プログラマチックな作成。** カスタム MapEventListener のクラス名は、com.company.org.MyMapEventListener クラスです。このクラスは MapEventListener インターフェースを実装します。以下のコードの断片は、カスタム MapEventListener を作成し、それを BackingMap に追加します。

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

- **XML を作成する**。MapEventListner を、XML を使用して構成することも可能です。以下の XML は上述のプログラマチックな作成と同等の構成を実現します。以下の XML は、myGrid.xml ファイルに存在しなければなりません。

```
<?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>
```

このファイルを ObjectGridManager に提供することで、簡単にこの構成を作成できます。以下のコードの断片は、この XML ファイルを使用した ObjectGrid の作成方法を示すものです。新規に作成された ObjectGrid の myMap BackingMap 上には、MapEventListener セットがあります。

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

Evictor

ObjectGrid は、デフォルトの Evictor 機構を提供します。プラグ可能 Evictor 機構を提供することもできます。

Evictor は、各 BackingMap 内のエントリーのメンバーシップを制御します。デフォルトの Evictor では、BackingMap ごとに、*存続時間* 除去ポリシーを使用します。

プラグ可能 Evictor 機構を提供すると、この機構では通常、時刻ではなく、エントリーの数に基づいた除去ポリシーが使用されます。このトピックでは、両方のタイプの Evictor について説明します。

デフォルトの存続時間 Evictor

ObjectGrid は、各 BackingMap に対して存続時間 (TTL) Evictor を提供します。TTL Evictor は、作成される各エントリーの有効期限の時間を保守します。あるエントリーに有効期限の時間が来ると、Evictor はそのエントリーを BackingMap から除去します。パフォーマンスへの影響を最小化するために、エントリーの除去を TTL Evictor は、有効期限切れになるまで待機しますが、エントリーが有効期限になる前に除去することはありません。

BackingMap には、存続時間 Evictor が各エントリーの有効期限の時間を計算する方法を制御する際に使用する属性があります。アプリケーションは、ttlType 属性を設定して、 TTL Evictor が有効期限の時間を計算する方法を指定します。 ttlType 属性は、以下の値のいずれかに設定できます。

- 「**None**」は、BackingMap 内のエントリーの有効期限が切れないことを示します。 TTL Evictor は、これらのエントリーを除去しません。
- 「**Creation time**」は、有効期限の時間計算にエントリーの作成時刻が使用されることを示します。
- 「**Last access time**」は、有効期限の時間計算に、エントリーが最後にアクセスされた時刻が使用されることを示します。

BackingMap に ttlType 属性が設定されていない場合は、TTL Evictor がエントリーを除去しないように、デフォルトのタイプである「**None**」が使用されます。ttlType 属性が「**creation time**」または「**last access time**」のいずれかに設定されている場合は、有効期限の時間を計算する際に、BackingMap の存続時間属性の値が、作成時刻または最終アクセス時刻のいずれかに追加されます。存続時間マップ属性の時刻の精度は、秒単位です。存続時間属性の値 `0` は、マップ・エントリーが永続的であることを示す場合に使用する特殊値です。つまり、アプリケーションによってマップ・エントリーが明示的に除去または無効化されるまで、そのエントリーがマップ内に存在し続けることを示します。

TTL Evictor の属性を指定する

TTL Evictor は、BackingMap インスタンスと関連しています。次のコードの断片は、各エントリーの作成時に、そのエントリーの有効期限の時間が、作成から 10 分後に設定されるようにするために、BackingMap インターフェースを使用して必要な属性を設定する方法を示しています。

```
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 );
```

setTimeToLive メソッドの引数は、存続時間の値が秒単位であることを指示するため、600 に設定されます。前掲のコードは、ObjectGrid インスタンスで initialize メソッドを呼び出す前に実行する必要があります。これらの BackingMap 属性は、ObjectGrid の初期化後に変更することはできません。コードが実行された後、myMap BackingMap 内に挿入されたエントリーには有効期限の時間が設定されます。有効期限の時間に到達した後、TTL Evictor はそのエントリーを消去します。

アプリケーションで、有効期限の時間を最終アクセス時刻プラス 10 分に設定する必要がある場合は、前のコードを 1 行変更する必要があります。 setTtlEvictorType メソッドに渡される引数は、TTLType.CREATION_TIME から TTLType.LAST_ACCESS_TIME に変更されます。この値を使用すると、有効期限の時間は最終アクセス時刻プラス 10 分として計算されます。最初にエントリーが作成されたときの最終アクセス時刻は、作成時刻です。

TTLType.LAST_ACCESS_TIME を使用すると、ObjectMap インターフェースと JavaMap インターフェースを使用して BackingMap の存続時間の値をオーバーライドすることができます。この機構により、アプリケーションが、作成される各エントリーに対して異なる存続時間の値を使用できるようになります。 ttlType 属性を LAST_ACCESS_TIME に設定するために、前掲のコードの断片が使用され、BackingMap に対して存続時間の値が 10 分に設定されたと想定します。アプリケーションは、エントリーを作成または変更する前に次のコードを実行して、各エントリーの存続時間をオーバーライドします。

```
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" );
```

前掲のコードの断片では、key1 キーを持つエントリーの有効期限の時間は、ObjectMap の setTimeToLive(1800) メソッドを呼び出した結果、挿入時刻プラス 30 分になります。 oldTimeToLive1 変数は 600 に設定されます。それは、以前に ObjectMap の setTimeToLive メソッドが呼び出されていなかった場合は、デフォルト値として BackingMap の存続時間の値が使用されるからです。

key2 キーを持つエントリーの有効期限の時間は、ObjectMap の setTimeToLive(1200) メソッドを呼び出した結果、挿入時刻プラス 20 分になります。 oldTimeToLive2 変数は 1800 に設定されます。それは、前の ObjectMap.setTimeToLive メソッド呼び出しで存続時間の値が 1800 に設定されたからです。

前の例では、キー値が key1 と key2 の 2 つのマップ・エントリーが、myMap マップに挿入されています。新規スレッドのアプリケーションは、後で新規のマップ値を用いてこれらのマップ・エントリーを更新することができます。しかし、このアプリケーションでは、挿入時にマップ・エントリーごとに使用された存続時間の値を保持しておく必要があります。以下の例では、まさにこの目的のために、ObjectMap インターフェースで定義されている定数を使用して、存続時間値を保持する方法を示しています。

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

ObjectMap.USE_DEFAULT 特殊値は setTimeToLive メソッド呼び出しで使用されるので、key1 はその存続時間値として 1800 秒を保持し、key2 はその存続時間値として 1200 秒を保持します。その理由は、それらの値は、これらのマップ・エントリーが前のトランザクションによって挿入されたときに使用されたからです。

前の例では、key3 の新規マップ・エントリーも挿入されています。この場合、USE_DEFAULT 特殊値は、このマップには存続時間値のデフォルト設定を使用することを示しています。デフォルト値は、BackingMap の存続時間属性により定義されます。BackingMap に存続時間属性を定義する方法については、115 ページの『BackingMap 属性』を参照してください。

ObjectMap インターフェースおよび JavaMap インターフェースの setTimeToLive メソッドについては、API の資料を参照してください。
BackingMap.getTtlEvictorType() メソッドから TTLType.LAST_ACCESS_TIME 以外の値が戻された場合は、IllegalStateException 例外が生じることに注意してください。 ObjectMap および JavaMap は、LAST_ACCESS_TIME TTL Evictor タイプを使用しているときに、存続時間値をオーバーライドする場合にのみ使用できます。このメソッドは、CREATION_TIME TTL Evictor タイプまたは NONE TTL Evictor タイプを使用しているときに、存続時間の値をオーバーライドする場合には使用できません。

XML ファイルを使用して TTL Evictor の属性を指定する

BackingMap インターフェースを使用して、 TTL Evictor が使用する BackingMap 属性をプログラマチックに設定する代わりに、 XML ファイルを使用して各 BackingMap を構成することができます。次のコードは、3 つの異なる BackingMaps に対するこれらの属性を設定する方法を示しています。

```
<?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>
```

前掲の例は、map1 BackingMap が NONE TTL Evictor タイプを使用することを示しています。 map2 BackingMap は、LAST_ACCESS_TIME TTL Evictor タイプを使用し、存続時間の値は 1800 秒、すなわち 30 分です。 map3 BackingMap は、CREATION_TIME TTL Evictor タイプを使用するように定義されており、その存続時間の値は 1200 秒、すなわち 20 分です。

オプションのプラグ可能 Evictor

デフォルトの TTL Evictor は、時刻ベースの除去ポリシーを使用し、 BackingMap 内のエントリーの数は、エントリーの有効期限の時間には影響を及ぼしません。オプションのプラグ可能 Evictor を使用して、時刻ではなく、存在するエントリー数をベースにエントリーを除去することができます。以下のオプションのプラグ可能 Evictor は、 BackingMap が一定のサイズの限界を超えたときに除去するエントリーを決定するために、一般に使用されるアルゴリズムをいくつか提供します。

- **LRUEvictor** は、BackingMap が最大エントリー数を超えたときに除去するエントリーを決定する際に*最長未使用時間* アルゴリズムを使用する Evictor です。
- **LFUEvictor** は、BackingMap が最大エントリー数を超えたときに除去するエントリーを決定するのに、*最少使用頻度* アルゴリズムを使用する Evictor です。

BackingMap は、Evictor を、トランザクション内で作成、変更、または除去済みのエントリーとして通知します。 BackingMap は、これらのエントリーを継続的に追跡し、 BackingMap から 1 つ以上のエントリーをいつ除去するかを選択します。

BackingMap には、最大サイズについての構成情報はありません。代わりに、Evictor の振る舞いを制御する Evictor プロパティーが設定されます。 LRUEvictor と LFUEvictor の両方の最大サイズ・プロパティーを使用して、最大サイズを超えた後、Evictor がエントリーを除去開始するようにします。 TTL Evictor と同様に、LRU Evictor と LFU Evictor では、最大エントリー数に達した場合、パフォーマンスへの影響を最小化するためにエントリーを直ちに除去することはありません。

特定のアプリケーションに LRU または LFU 除去アルゴリズムが適していない場合、独自の Evictor を作成して、必要な除去ストラテジーを実行できます。

プラグ可能 Evictor を指定する

Evictor は BackingMaps に関連しているため、 BackingMap インターフェースを使用して、使用するプラグ可能 Evictor を指定します。次のコードの断片は、map1 BackingMap 用の LRUEvictor Evictor、または map2 BackingMap 用の LFUEvictor Evictor を指定する例です。

```
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);
```

前のコードの断片は、最大エントリー数が 1000 である map1 BackingMap に使用される LRUEvictor Evictor を示しています。 LFUEvictor Evictor は、最大エントリー数が 2000 である map2 BackingMap に使用されます。 LRU Evictor と LFU Evictor はいずれもスリープ時間プロパティーを持ちます。このプロパティーは、ウェイクアップしてエントリーを除去する必要があるかどうかを検査するまで Evictor がスリープする時間を示します。スリープ時間は、秒単位で指定されます。値 15 秒は、パフォーマンスの影響と BackingMap が大きくなりすぎないようにするための妥当な値です。目標は、BackingMap のサイズが超過することなく、できる限り長いスリープ時間を使用することです。

setNumberOfLRUQueues メソッドは、LRUEvictor プロパティーを設定します。このプロパティーは、LRU 情報を管理するために Evictor が使用する LRU キューの数を示します。各エントリーが同じキュー内の LRU 情報を保持しないように、キューのコレクションを使用します。この方法により、同じキュー・オブジェクトで同期化する必要があるマップ・エントリーの数が最小化され、パフォーマンスが向上します。キューの数を増やすのは、LRU Evictor がパフォーマンスに与えるインパクトを最小化するための良い方法です。開始点としては、キューの数として最大エントリー数の 10 % を使用することが適切です。一般に、基本数以外を使用するよりも、基本数を使用するほうが適しています。

setNumberOfHeaps メソッドは、LFUEvictor プロパティーを設定します。このプロパティーは、LFU 情報を管理するために LFUEvictor が使用するバイナリー・ヒープ・オブジェクトの数を設定します。この場合も、コレクションを使用するとパフォーマンスが向上します。開始点としては、最大エントリー数の 10% を使用するのことが適切であり、一般に、基本数以外を使用するよりも、基本を使用するほうが適しています。

XML を使用してプラグ可能 Evictor を指定する

さまざまな API を使用して、Evictor をプログラマチックにプラグインし、そのプロパティーを設定する代わりに、次の例に示すように、XML ファイルを使用して各 BackingMap を構成することができます。

```
<?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>
```

カスタム Evictor の作成

ObjectGrid を拡張して、除去アルゴリズムを使用することができます。com.ibm.websphere.objectgrid.plugins.Evictor インターフェースを実装するカスタム Evictor を作成する必要があります。インターフェースは、次のとおりです。

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

- initialize メソッドは、BackingMap オブジェクトの初期化中に呼び出されます。このメソッドは、BackingMap への参照、および com.ibm.websphere.objectgrid.plugins.EvictionEventCallback インターフェースを実装するオブジェクトへの参照を用いて、Evictor プラグインを初期化します。
- apply メソッドは、BackingMap の 1 つ以上のエントリーにアクセスするトランザクションがコミットされたときに呼び出されます。 apply メソッドは、com.ibm.websphere.objectgrid.plugins.LogSequence インターフェースを実装するオブジェクトへの参照に渡されます。 LogSequence インターフェースを使用すると、Evictor プラグインは、トランザクションによって作成、変更、または除去された BackingMap エントリーを判別することができます。 Evictor は、この情報を使用して、いつ、どのエントリーを除去するかを決定します。
- destroy メソッドは、BackingMap を破棄するときに呼び出されます。このメソッドを使用すると、Evictor は作成した任意のスレッドを終了できます。

EvictionEventCallback インターフェースには、以下のメソッドがあります。

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

EvictionEventCallback メソッドは Evictor プラグインによって使用され、次のように、ObjectGrid フレームワークにコールバックします。

- setEvictorData メソッドは Evictor によって使用され、使用されている ObjectGrid フレームワークに、その Evictor が作成した Evictor オブジェクトを保管し、引数 key で示されるエントリーにその Evictor オブジェクトを関連付けるよう要求します。このデータは、Evictor 固有であり、使用するアルゴリズムを実装するために Evictor が必要とする情報によって判別されます。例えば、最少使用頻度アルゴリズムでは、Evictor は、指定されたキーのエントリーを参照する

LogElement で apply メソッドが呼び出された回数を追跡するために、そのカウント数を Evictor データ・オブジェクトに保持します。

- getEvictorData メソッドは Evictor によって使用され、前の apply メソッドの呼び出し中に setEvictorData メソッドに渡されたデータを取り出します。指定された引数 key に対応する Evictor データが見つからない場合は、 EvictorCallback インターフェースで定義されている特別な KEY_NOT_FOUND オブジェクトが戻されます。
- evictMapEntries メソッドは Evictor によって使用され、1 つ以上のマップ・エントリーの除去を要求します。evictorDataList パラメーター内の各オブジェクトは、com.ibm.websphere.objectgrid.plugins.EvictorData インターフェースを実装する必要があります。また、setEvictorData メソッドに渡されるのと同じ EvictorData インスタンスが、このメソッドの Evictor データ・リスト・パラメーターに存在している必要があります。除去するマップ・エントリーを決定する場合は、EvictorData インターフェースの getKey メソッドが使用されます。キャッシュ・エントリーの Evictor データ・リストにあるのとまったく同一の EvictorData インスタンスが現在このキャッシュ・エントリーに含まれている場合、このマップ・エントリーは除去されます。
- evictEntries メソッドは Evictor によって使用され、1 つ以上のマップ・エントリーの除去を要求します。このメソッドが使用されるのは、setEvictorData メソッドに渡されるオブジェクトが、 com.ibm.websphere.objectgrid.plugins.EvictorData インターフェースを実装していない場合のみです。

ObjectGrid は、トランザクションの完了後 に、Evictor インターフェースの apply メソッドを呼び出します。完了しているトランザクションによって獲得されたすべてのトランザクション・ロックは、保持されなくなります。そのため、複数のスレッドが同時に apply メソッドを呼び出し、各スレッドが別々のトランザクションを完了することも起こり得ます。トランザクション・ロックは、完了しているトランザクションによってすでに解放されているので、apply メソッドはそれ自体で同期化を行って、それがスレッド・セーフであることを保証する必要があります。

EvictorData インターフェースを実装して、evictEntries メソッドの代わりに evictMapEntries メソッドを使用する理由は、そのような時間帯をなくすことにあります。次のイベント・シーケンスを考えてみましょう。

1. トランザクション 1 が完了し、LogSequence で apply メソッドを呼び出して、キー 1 のマップ・エントリーを削除する。
2. トランザクション 2 が完了し、LogSequence で apply メソッドを呼び出して、キー 1 の新規マップ・エントリーを挿入する。つまり、トランザクション 2 は、トランザクション 1 が削除したマップ・エントリーを再作成します。

Evictor は、トランザクションを実行するスレッドとは非同期で実行するので、その Evictor でキー 1 を除去することにした場合、Evictor は、トランザクション 1 が完了する前に存在していたマップ・エントリーを除去するか、トランザクション 2 が再作成したマップ・エントリーを除去する可能性があります。時間帯をなくすため、どのバージョンのキー 1 のマップ・エントリーを除去するのかを明確にするために、setEvictorData メソッドに渡されるオブジェクトによって EvictorData インターフェースを実装します。マップ・エントリーの存続期間中は、同じ EvictorData インスタンスを使用します。そのマップ・エントリーが削除され、次に別のトランザクションによって再作成されるときは、Evictor は、EvictorData 実装の新規インス

タンスを使用する必要があります。Evictor は、EvictorData 実装および evictMapEntries メソッドを使用することにより、マップ・エントリーに関連付けられているキャッシュ・エントリーに正しい EvictorData インスタンスが含まれている場合に限り、そのマップ・エントリーが除去されることを保証できます。

Evictor インターフェースと EvictonEventCallback インターフェースにより、アプリケーションは、ユーザー定義の除去アルゴリズムを実装する Evictor を接続することができます。次のコードの断片は、Evictor インターフェースの initialize メソッドを実装する方法を示しています。

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
import java.util.LinkedList;
// Instance variables
private BackingMap bm;
private EvictionEventCallback evictorCallback;
private LinkedList queue;
private Thread evictorThread;
public void initialize(BackingMap map, EvictionEventCallback callback)
{
bm = map;
evictorCallback = callback;
queue = new LinkedList();
// spawn evictor thread
evictorThread = new Thread( this );
String threadName = "MyEvictorForMap-" + bm.getName();
evictorThread.setName( threadName );
evictorThread.start();
}
```

前掲のコードでは、マップ・オブジェクトとコールバック・オブジェクトへの参照をインスタンス変数内に保管します。これにより、これらのオブジェクトを apply メソッドと destroy メソッドで使用することができます。この例では、最長未使用時間 (LRU) アルゴリズムを実装するための *先入れ、先出し* キューとして使用されるリンク・リストが作成されます。スレッドが作成され、そのスレッドへの参照は、インスタンス変数として保持されます。この参照を保持することにより、destroy メソッドは作成されたスレッドに割り込んで終了させることができます。

次のコードの断片は、コードをスレッド・セーフにするための同期要件を無視して、Evictor インターフェースの apply メソッドを実装する方法を示しています。

```
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.EvictionEventCallback;
import com.ibm.websphere.objectgrid.plugins.Evictor;
import com.ibm.websphere.objectgrid.plugins.EvictorData;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
public void apply(LogSequence sequence)
{
Iterator iter = sequence.getAllChanges();
while ( iter.hasNext() )
{
LogElement elem = (LogElement)iter.next();
Object key = elem.getCacheEntry().getKey();
LogElement.Type type = elem.getType();
if ( type == LogElement.INSERT )
{
// do insert processing here by adding to front of LRU queue.
```

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

apply メソッド内への挿入処理では、通常、EvictionEventCallback インターフェースの setEvictorData メソッドに渡される Evictor データ・オブジェクトが作成されます。この Evictor は、LRU 実装を示すためのものなので、initialize メソッドで作成されたキューの前に EvictorData も追加されます。 apply メソッド内の更新処理は通常、apply メソッドの前の呼び出し (例えば、apply メソッドの挿入処理による) によって作成された Evictor データ・オブジェクトを更新します。この Evictor は LRU 実装であるため、 EvictorData オブジェクトをその現在のキュー位置からキューの前方に移動させる必要があります。作成された Evictor スレッドは、最後のキュー・エレメントが LRU エントリーを表しているため、キュー内の最後の EvictorData オブジェクトを除去します。その前提として、EvictorData オブジェクトは getKey メソッドを持っています。これによって、Evictor スレッドは、除去する必要があるエントリーのキーを認識します。この例では、コードをスレッド・セー

フにするための同期要件を無視していることに留意してください。実際のカスタム Evictor は、同期と同期点の結果として発生するパフォーマンスの障害を取り扱っているため、より複雑です。

以下のコードの断片は、initialize メソッドが作成した実行可能スレッドの destroy メソッドと run メソッドを示しています。

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

オプションの RollBackEvictor インターフェース

com.ibm.websphere.objectgrid.plugins.RollbackEvictor インターフェースは、オプションで、Evictor プラグインによって実装することができます。このインターフェースを実装することにより、トランザクションがコミットされたときだけでなく、トランザクションがロールバックされたときにも、Evictor を呼び出すことができます。

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

apply メソッドは、トランザクションがコミットされたときにのみ呼び出されます。トランザクションがロールバックされたとき、Evictor が RollbackEvictor インターフェースを実装している場合は、rollingBack メソッドが呼び出されます。RollbackEvictor インターフェースが実装されていない場合は、トランザクションがロールバックされても、apply メソッドおよび rollingBack メソッドは呼び出されません。

ローダー

ObjectGrid ローダーはプラグ可能なコンポーネントであり、通常は同一システムまたはそれ以外のシステムのいずれかの永続ストアに保持されるデータのメモリー・キャッシュとして、ObjectGrid マップが動作できるようにします。

通常、データベースまたはファイル・システムは永続ストアとして使用されます。リモート Java 仮想マシン (JVM) は、データ・ソースとしても使用できます。このデータ・ソースは、ObjectGrid を使用してハブ・ベースのキャッシュを作成できるようにします。ローダーには、永続ストアとの間でデータの読み取りおよび書き込みを行うロジックがあります。

ローダーは、ObjectGrid のバッキング・マップ用のプラグインです。指定されたバッキング・マップに関連付けることのできるローダーは 1 つだけです。各バッキング・マップはそれ自身のローダー・インスタンスを持っています。バッキング・マップは、そのローダーから組み込まないすべてのデータを要求します。マップに対するすべての変更は、ローダーにプッシュ・アウトされます。ローダー・プラグインにより、バッキング・マップはデータをマップと永続ストアとの間で移動できるようになります。

ローダーのプラグイン

以下のコードの断片は、アプリケーション提供のローダーを、ObjectGrid API を使用して map1 のバッキング・マップにプラグインする方法を示しています。

```
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 );
```

MyLoader が、com.ibm.websphere.objectgrid.plugins.Loader インターフェースを実装するアプリケーション提供のクラスであることが前提になります。 ObjectGrid の初期化後は、ローダーとバッキング・マップとの関連付けを変更できないので、呼び出されている ObjectGrid インターフェースの initialize メソッドを起動する前にコードを実行する必要があります。 setLoader メソッド呼び出しが、初期化後に呼び出された場合、 IllegalStateException 例外が発生します。

アプリケーションが提供する Loader には、set プロパティーがあります。例では、MyLoader ローダーを使用して、リレーショナル・データベースの表からデータを読み書きします。ローダーには、データベースの名前と使用する SQL 分離レベルが必要です。 MyLoader ローダーには、setDataBaseName メソッドと setIsolationLevel メソッドがあり、アプリケーションはこれらのメソッドを使用してこれら 2 つの Loader プロパティーを設定できます。

アプリケーションが提供する Loader は、XML ファイルを使用してプラグインすることも可能です。以下の例は、MyLoader ローダーが、同じデータベース名および設定されている分離レベル・ローダー・プロパティーで map1 バッキング・マップにプラグインされる方法を示しています。

```
<?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>
```

Loader インターフェースの実装

アプリケーションが提供する Loader は、*com.ibm.websphere.objectgrid.plugins.Loader* インターフェースを実装する必要があります。この Loader インターフェースには、以下の定義があります。

```
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;
}
```

Loader インターフェースで各メソッドを実装する場合の説明および考慮事項は、以下のセクションで提供します。

get メソッド

バッキング・マップは Loader の get メソッドを呼び出し、 **keyList** 引数として渡されるキー・リストに関連付けられた値を取得します。 get メソッドは、キー・リストにある各キーのうちの 1 つの値の java.lang.util.List リストを戻す必要があります。値リストに戻される最初の値はキー・リストの最初のキーに対応し、値リストに戻される 2 番目の値はキー・リストの 2 番目のキーに対応し、以降同様になります。キー・リスト内でキーの値を検出できなかったローダーは、ローダー・インターフェースで定義された特別な KEY_NOT_FOUND 値オブジェクトを戻す必要があります。バッキング・マップは、null を有効な値として許可するよう構成できるので、キーを検出できないローダーが特別な KEY_NOT_FOUND オブジェクトを戻すことが重要になります。この値により、バッキング・マップは null 値とキーが検出できなかったため存在しない値とを区別できます。バッキング・マップが null 値をサポートしない場合、存在しないキーについて KEY_NOT_FOUND オブジェクトではなくヌル値を戻すローダーは、例外を発生します。

forUpdate 引数は、アプリケーションがマップ上で get メソッドまたは getForUpdate メソッドのいずれを呼び出したかをローダーに通知します。詳細については、com.ibm.websphere.objectgrid.ObjectMap インターフェースを参照してください。ローダーは、永続ストアへの並行アクセスを制御する、並行性制御ポリシーの実装を担当します。例えば、多くのリレーショナル・データベース管理システムは、リレーショナル・テーブルからデータを読み取るために使用される SQL SELECT ステートメントの FOR UPDATE 構文をサポートします。ローダーは、ブール値 true が、このメソッドの **forUpdate** パラメーターに引数値として渡されるかどうかに基づいて、SQL SELECT ステートメントの FOR UPDATE 構文を使用することを選択できます。通常、ローダーはペシミスティック並行性の制御ポリシーを使用する場合にのみ FOR UPDATE 構文を使用します。オプティミスティック並行性制御の場合、ローダーは SQL SELECT ステートメントで FOR UPDATE 構文を使用することはありません。ローダーは、そのローダーが使用している並行性制御ポリシーに基づいて forUpdate 引数の使用を判別します。

txid パラメーターの説明については、 226 ページの『TransactionCallback プラグイン』トピックを参照してください。

batchUpdate メソッド

batchUpdate メソッドは、ローダー・インターフェースにおいて重要です。このメソッドは、ObjectGrid が現在のすべての変更をローダーに適用する必要がある場合に必ず呼び出します。ローダーには、このマップの変更のリストが与えられます。変更は繰り返され、バックエンドに適用されます。このメソッドは現行の TxID 値および適用する変更を受け取ります。以下のサンプルは、一連の変更を繰り返し、INSERT、UPDATE、および DELETE という 3 つの Java Database Connectivity (JDBC) ステートメントをバッチ処理します。

```
import java.util.Collection;
import java.util.Map;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import com.ibm.websphere.objectgrid.TxID;
import com.ibm.websphere.objectgrid.plugins.Loader;
import com.ibm.websphere.objectgrid.plugins.LoaderException;
import com.ibm.websphere.objectgrid.plugins.LogElement;
```

```
import com.ibm.websphere.objectgrid.plugins.LogSequence;
public void batchUpdate(TxID tx, LogSequence sequence)
throws LoaderException
{
// Get a SQL connection to use.
Connection conn = getConnection(tx);
try
{
// Process the list of changes and build a set of prepared
// statements for executing a batch update, insert, or delete
// SQL operation.
Iterator iter = sequence.getPendingChanges();
while ( iter.hasNext() )
{
LogElement logElement = (LogElement)iter.next();
Object key = logElement.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;
}
}
// Execute the batch statements that were built by above loop.
Collection statements = getPreparedStatementCollection( tx, conn );
iter = statements.iterator();
while ( iter.hasNext() )
{
PreparedStatement pstmt = (PreparedStatement) iter.next();
pstmt.executeBatch();
}
}
catch (SQLException e)
{
LoaderException ex = new LoaderException(e);
throw ex;
}
}
```

前のサンプルは、LogSequence 引数の処理の高水準ロジックを示していますが、 SQL の INSERT、UPDATE、または DELETE ステートメントがビルドされる方法の詳細については示されていません。示されているキーポイントには、以下のようなものがあります。

- getPendingChanges メソッドは、LogSequence 引数で呼び出され、ローダーが処理を必要とする LogElements のリストのイテレーターを取得します。
- LogElement.getType().getCode() メソッドを使用して、 LogElement が SQL の INSERT、UPDATE、または DELETE 操作用であるかどうかを判断します。
- SQLException 例外はキャッチされ、バッチ更新中に発生した例外を報告するために発行される LoaderException 例外にチェーンされます。
- JDBC バッチ更新サポートは、作成する必要のあるバックエンドへの照会の数を最小化するために使用されます。

preloadMap メソッド

ObjectGrid の初期化中に、定義された各バッキング・マップは初期化されます。ローダーがバッキング・マップにプラグインされると、バッキング・マップはローダー・インターフェースで preloadMap メソッドを呼び出し、ローダーがバックエンドからデータをプリフェッチし、マップにデータをロードできるようにします。以下のサンプルでは、Employee テーブルの最初の 100 行がデータベースから読み取られて、マップにロードされると仮定します。 EmployeeRecord クラスはアプリケーションが提供するクラスであり、従業員テーブルから読み取った従業員データを保持します。

```
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();
// Get a auto-commit connection to use that is set to
// a read committed isolation level.
conn = getAutoCommitConnection(tx);
// Preload the Employee Map with EmployeeRecord
// objects. Read all Employees from table, but
// limit preload to first 100 rows.
stmt = conn.createStatement();
results = stmt.executeQuery( SELECT_ALL );
int rows = 0;
while ( results.next() && rows < 100 )
{
int key = results.getInt(EMPNO_INDEX);
EmployeeRecord emp = new EmployeeRecord( key );
emp.setLastName( results.getString(LASTNAME_INDEX) );
emp.setFirstName( results.getString(FIRSTNAME_INDEX) );
emp.setDepartmentName( results.getString(DEPTNAME_INDEX) );
emp.updateSequenceNumber( results.getLong(SEQNO_INDEX) );
emp.setManagerNumber( results.getInt(MGRNO_INDEX) );
map.put( new Integer(key), emp );
++rows;
}
// Commit the transaction.
session.commit();
tranActive = false;
}
catch (Throwable t)
{
throw new LoaderException("preload failure: " + t, t);
}
finally
{
if ( tranActive )
{
try
```

```
{
session.rollback();
}
catch ( Throwable t2 )
{
// Tolerate any rollback failures and
// allow original Throwable to be thrown.
}
}
// Be sure to clean up other databases resources here
// as well such a closing statements, result sets, etc.
}
}
```

このサンプルは以下のキーポイントを示します。

- preloadMap のバッキング・マップはセッション引数として渡されるセッション・オブジェクトを使用します。
- Session.beginNoWriteThrough() メソッドを使用して、begin メソッドではなく、トランザクションを開始します。マップのロードに関してこのメソッドで発生する各 put 操作にローダーを呼び出すことはできません。
- ローダーは、従業員テーブルの列を EmployeeRecord Java オブジェクトのフィールドにマップすることができます。
- ローダーは、発生したすべてのスロー可能な例外をキャッチし、LoaderException 例外を、その例外にチェーンされているキャッチしたスロー可能な例外と一緒にスローします。
- finally ブロックにより、beginNoWriteThrough メソッドが呼び出される時点からコミット method が呼び出される時点までの間に発生するすべてのスロー可能な例外は、確実に finally ブロックにアクティブなトランザクションをロールバックします。このアクションは、preloadMap メソッドによって開始されたすべてのトランザクションが、呼び出し側に戻される前に確実に完了させるために、重要です。 finally ブロックは、JDBC 接続やその他の JDBC オブジェクトのクローズのような、必要とされる可能性のあるそれ以外のクリーンアップ・アクションを行う場所としても適切です。

preloadMap サンプルは、テーブルの行をすべて選択する SQL SELECT ステートメントを使用しています。アプリケーションが提供するローダーでは、マップにプリロードするテーブルの数を制御するために、 1 つ以上の Loader プロパティーを設定します。

preloadMap メソッドは BackingMap の初期化中に 1 回しか呼び出されないので、 1 回だけのローダー初期化コードの実行場所としても適切です。ローダーがバックエンドからデータをプリフェッチせず、データをマップにロードしないことを選択した場合であっても、それ以外に何らかの 1 回だけの初期化を実行し、さらに効率的なローダーの別のメソッドを作成する必要があると考えられます。以下は、TransactionCallback オブジェクトおよび OptimisticCallback オブジェクトをローダーのインスタンス変数としてキャッシングして、ローダーの別のメソッドがこれらのオブジェクトにアクセスするためにメソッド呼び出しを行わなくても済むようにする例です。BackingMap の初期化後に、TransactionCallback オブジェクトおよび OptimisticCallback オブジェクトを変更または置換できなくなるため、この

ObjectGrid プラグインの値のキャッシングを行うことが可能です。これらのオブジェクト参照をローダーのインスタンス変数としてキャッシュに入れることは許容されます。

```
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.OptimisticCallback;
import com.ibm.websphere.objectgrid.plugins.TransactionCallback;
// Loader instance variables.
MyTransactionCallback ivTcb; // MyTransactionCallback
// extends TransactionCallback
MyOptimisticCallback ivOcb; // MyOptimisticCallback
// implements OptimisticCallback
...
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
// Cache TransactionCallback and OptimisticCallback objects
// in instance variables of this Loader.
ivTcb = (MyTransactionCallback)
session.getObjectGrid().getTransactionCallback();
ivOcb = (MyOptimisticCallback) backingMap.getOptimisticCallback();
// The remainder of preloadMap code (such as shown in prior example).
}
```

複製フェイルオーバーに関係するプリロードおよび回復可能なプリロードについては、238 ページの『複製のプログラミング』を参照してください。

ローダーの考慮事項

ローダーを実装するときには、以下の考慮事項を参照します。

プリロードの考慮事項

それぞれのバッキング・マップにはブール preloadMode 属性があり、マップのプリロードが非同期的に終了するかどうかを示すよう設定できます。デフォルトでは、preloadMode 属性は false に設定されており、マップのプリロードが完了するまでバッキング・マップの初期化が完了しないことを示しています。例えば、preloadMap メソッドが戻るまで、バッキング・マップの初期化は完了しません。preloadMap メソッドがバックエンドから大量のデータを読み取って、マップにロードしようとしている場合、完了するまでに比較的長い時間を要する可能性があります。このような場合、preloadMode 属性を true に設定して、マップの非同期プリロードを使用するようにバッキング・マップを構成できます。この設定により、バッキング・マップ初期化コードが preloadMap メソッドを呼び出すスレッドを作成し、マップのプリロードの進行中に、バッキング・マップの初期化を完了できるようになります。

以下のコードの断片は、非同期プリロードが有効になるよう preloadMode 属性を設定する方法を表しています。

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

preloadMode 属性は、以下の例に示すように、 XML ファイルを使用して設定することもできます。

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

TxID と TransactionCallback インターフェースの使用

Loader インターフェースの get メソッドと batchUpdate メソッドの両方に、 get 操作または batchUpdate 操作の実行を必要とするセッション・トランザクションを表す TxID オブジェクトが渡されます。 get メソッドおよび batchUpdate メソッドは、トランザクションごとに複数回呼び出すことが可能です。したがって、ローダーが必要とするトランザクション・スコープのオブジェクトは通常 TxID オブジェクトのスロットに保持されます。 Java Database Connectivity (JDBC) のローダーは、ローダーが TxID および TransactionCallback インターフェースを使用する方法を示すために使用します。

いくつかの ObjectGrid マップを、同じデータベースに格納することも可能です。各マップは独自のローダーを持ち、各ローダーは同一のデータベースに接続する必要があります。同一のデータベースに接続するとき、各テーブルへの変更が同じデータベース・トランザクションのパーツとしてコミットされるようにするため、各ローダーは同じ JDBC 接続を使用しようとします。通常、ローダー実装を作成する同じ担当者が TransactionCallback 実装を作成します。一番良い方法は、TransactionCallback インターフェースが拡張されて、ローダーがデータベースの接続を得て、準備済みステートメントのキャッシングを必要とするメソッドを追加する場合です。この方法論の根拠は、ローダーが TransactionCallback インターフェースおよび TxID インターフェースを使用する方法を調べると明らかになります。

例として、ローダーが、以下のように拡張される TransactionCallback インターフェースを必要とする場合を示します。

```
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 );
}
```

これらの新しいメソッドを使用すると、Loader の get メソッドおよび batchUpdate メソッドは、以下のようにして接続を取得することができます。

```
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;
}
```

前の例および以下の例で、*ivTcb* および *ivOcb* は、 216 ページの『プリロードの考慮事項』セクションに説明する方法で初期化されたローダーのインスタンス変数です。 *ivTcb* 変数は MyTransactionCallback インスタンスへの参照であり、 *ivOcb* は

MyOptimisticCallback インスタンスへの参照です。 *databaseName* 変数は、ローダーのインスタンス変数であり、バッキング・マップの初期化中に Loader プロパティーとして設定されています。 isolationLevel 引数は、JDBC がサポートするさまざまな分離レベルに関して定義されている JDBC 接続定数の 1 つです。ローダーがオプティミスティック・インプリメンテーションを使用している場合、 get メソッドは通常 JDBC 自動コミット接続を使用してデータベースからデータを取り出します。この場合、ローダーは以下のように実装される getAutoCommitConnection メソッドを持つことがあります。

```
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;
}
```

batchUpdate メソッドに以下の switch ステートメントがあれば、再呼び出しします。

```
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;
}
```

各 buildBatchSQL メソッドは、MyTransactionCallback インターフェースを使用して、準備済みステートメントを取得します。以下は、EmployeeRecord エントリーを更新する SQL UPDATE ステートメントをビルドし、それをバッチ更新用に追加する buildBatchSQLUpdate メソッドを示すコードの断片です。

```
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();
}
```

batchUpdate ループは、準備済みステートメントをすべてビルドした後で、getPreparedStatementCollection メソッドを呼び出します。このメソッドは、以下のように実装できます。

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

アプリケーションがセッションの commit メソッドを呼び出すと、セッション・コードは、トランザクションによって変更された各マップのローダーに、トランザクションによって変更されたすべての変更がプッシュされた後で、TransactionCallback メソッドの commit メソッドを呼び出します。すべてのローダーが、必要な接続と準備済みステートメントを取得するために MyTransactionCallback メソッドを使用したため、TransactionCallback メソッドは、バックエンドが変更をコミットすることを要求するために使用する接続を認識しています。したがって、各ローダーが必要とするメソッドを持つ TransactionCallback インターフェースを拡張することによって、以下の利点が得られます。

- TransactionCallback オブジェクトは、トランザクション・スコープ・データの TxID スロットの使用をカプセル化するので、ローダーは TxID スロットに関する情報を必要としません。ローダーは、ローダーが必要とする機能をサポートするための、 MyTransactionCallback インターフェースを使用する TransactionCallback に追加されるメソッドに関してのみ認識する必要があります。
- TransactionCallback オブジェクトは、2 フェーズ・コミット・プロトコルを回避できるようにするため、同じバックエンドに接続する各ローダー間で、接続の共用が確実に起こるようにすることができます。
- TransactionCallback オブジェクトは、バックエンドへの接続が適切な場合接続に呼び出されたコミットまたはロールバックを通して確実に完了できるようにします。
- TransactionCallback は、トランザクションの完了時にデータベース・リソースのクリーンアップが確実に行われるようにできます。
- WebSphere Application Server や、その他の Java 2 Platform、Enterprise Edition (J2EE) 準拠のアプリケーション・サーバーのような管理された環境から管理対象の接続を取得している場合、 TransactionCallback を不可視にすることができます。この利点により、環境が管理されている、いないにかかわらず、同じローダーのコードを使用できます。 TransactionCallback プラグインのみを変更する必要があります。

TransactionCallback の実装がトランザクション・スコープのデータの TxID スロットを使用する方法の詳細については、 226 ページの『TransactionCallback プラグイン』を参照してください。

OptimisticCallback

これまでに述べたように、ローダーは並行性制御のオプティミスティック・アプローチの使用を決定する場合があります。その場合、オプティミスティック・アプローチを実装するために、この buildBatchSQLUpdate メソッドの例に若干の変更を加える必要があります。オプティミスティック・アプローチを使用する方法は、いくつかあります。行の各更新をバージョン管理するために、タイム・スタンプの列かシーケンス番号のカウンターの列のいずれかを設ける方法が一般的です。従業員のテーブルには、行が更新されるたびに増分するシーケンス番号の列があります。

次に、buildBatchSQLUpdate メソッドのシグニチャーを変更して、キーと値のペアの代わりに LogElement オブジェクトが渡されるようにします。初期バージョンのオブジェクトを取得し、そのバージョンのオブジェクトを更新するには、バッキング・マップにプラグインされた OptimisticCallback オブジェクトも使用する必要があります。以下は、preloadMap のセクションで説明されている、初期化された *ivOcb* インスタンス変数を使用する変更済み buildBatchSQLUpdate メソッドの例です。

```
private void buildBatchSQLUpdate( TxID tx, LogElement le,
Connection conn )throws SQLException, LoaderException
{
// Get the initial version object when this map entry was last read
// or updated in the database.
Employee emp = (Employee) le.getCurrentValue();
long initialVersion = ((Long) le.getVersionedValue()).longValue();
// Get the version object from the updated Employee for the SQL update
//operation.
Long currentVersion = (Long)ivOcb.getVersionedObjectForValue( emp );
long nextVersion = currentVersion.longValue();
// Now build SQL update that includes the version object in where clause
// for optimistic checking.
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();
}
```

この例は、初期バージョンの値を取得するために LogElement が使用されることを示しています。トランザクションがマップ・エントリーに最初にアクセスするとき、マップから取得した初期の従業員のオブジェクトに関して LogElement が作成されます。この初期 Employee オブジェクトは、OptimisticCallback インターフェースの getVersionedObjectForValue メソッドにも渡され、その結果は LogElement に保管されます。この処理が行われるのは、初期 Employee オブジェクトへの参照がアプリケーションに与えられ、そのアプリケーションがある機会を捉えて、初期 Employee オブジェクトの状態を変更するなんらかのメソッドを呼び出す前です。

この例は、Loader が getVersiondObjectForValue メソッドを使用して、現行の更新済み Employee オブジェクトのバージョン・オブジェクトを取得しているところを示しています。ObjectGrid は、Loader インターフェースの batchUpdate メソッドを呼び出す前に、OptimisticCallback インターフェースの updateVersionedObjectForValue メソッドを呼び出して、更新済みの Employee オブジェクトに対して新規バージョン・オブジェクトを生成させます。batchUpdate メソッドが ObjectGrid に戻った後に、LogElement は、それが新規の初期バージョン・オブジェクトになるよう、現行バージョン・オブジェクトで更新されます。アプリケーションは Session の commit メソッドの代わりに、マップ上の flush メソッドを呼び出す可能性があるので、このステップが必要になります。同一のキー用の単一のトランザクションによって、ローダーを複数回呼び出すことは可能です。そのような理由から、ObjectGrid は、従業員テーブルの行が更新されるたびに、LogElement が新規バージョン・オブジェクトで更新されることを保証しています。

ローダーには、初期バージョンのオブジェクトと次期バージョンのオブジェクトがあるので、次期バージョンのオブジェクト値に SEQNO 列を設定し、where 文節で初期バージョンのオブジェクトを使用する SQL UPDATE ステートメントを実行できます。この方法は、過剰 update ステートメントと呼ばれることがあります。この過剰 update ステートメントを使用することで、リレーショナル・データベースは、このトランザクションがデータベースからデータを読み取ってからデータベースを更新するまでの間に、別のトランザクションが行を変更していないかどうかを検証できます。別のトランザクションが行を変更していた場合、バッチ更新によって戻されるカウント配列は、このキーに関してゼロ行が更新されたことを示します。ローダーは、SQL update 操作が実際に行を更新したことを検証します。更新されていない場合、ローダーは com.ibm.websphere.objectgrid.plugins.OptimisticCollisionException 例外を表示して、複数の並行トランザクションがデータベース表の同一行に対して更新を試みたため、 batchUpdate メソッドが失敗したことをセッションに通知します。この例外はセッションにロールバックを行わせるので、アプリケーションはトランザクション全体を再試行する必要があります。この方法は、再試行が成功することを予測して行われるため、オプティミスティックと呼ばれます。データがほとんど変更されないか、並行トランザクションによる同一行の更新がほとんど試みられない場合は、このオプティミスティック・アプローチは実際にうまく機能します。

Loader が OptimisticCollisionException コンストラクターの キー ・パラメーターを使用して、どのキーまたはキーのセットがオプティミスティック batchUpdate メソッドの失敗の原因になったかを識別することが重要です。キー・パラメーターには、キー・オブジェクトそのものを使用することもできますし、複数のキーが原因でオプティミスティック更新が失敗した場合は、キー・オブジェクトの配列とすることもできます。 ObjectGrid は、OptimisticCollisionException コンストラクターの getKey メソッドを使用して、どのマップ・エントリーに失効したデータが含まれているか、そしてどのマップ・エントリーが例外の発生原因であるかを判別します。ロールバック処理の一環として、失効した各マップ・エントリーをマップから除去します。失効したエントリーを除去する必要があるのは、同じキーまたは複数のキーにアクセスする後続のいずれかのトランザクションで、Loader インターフェースの get メソッドが呼び出されて、データベースの現在のデータによってマップ・エントリーが更新されるようにするためです。

ローダーがオプティミスティック・アプローチを実施するそれ以外の方法として、以下のようなものがあります。

- タイム・スタンプまたはシーケンス番号の列を無くします。この場合、OptimisticCallback インターフェースの getVersionObjectForValue メソッドは、単に、値オブジェクト自身をバージョンとして戻します。この方法で、ローダーは初期バージョン・オブジェクトの各フィールドを組み込む where 文節をビルドする必要があります。この方法はあまり効率的ではなく、列タイプのすべてが過剰 SQL UPDATE ステートメントの where 文節での使用に適しているわけではありません。この方法は通常使用しません。
- タイム・スタンプまたはシーケンス番号の列を無くします。ただし、前の方法とは異なり、where 文節はトランザクションによって変更された値フィールドのみを含んでいます。変更されたフィールドを検出する方法の 1 つは、バッキング・マップのコピー・モードを CopyMode.COPY_ON_WRITE モードに設定することです。このコピー・モードは、BackingMap インターフェースの setCopyMode メソ

ッドに渡される値インターフェースを必要とします。 BackingMap は、提供される値インターフェースを実装する動的プロキシー・オブジェクトを作成します。このコピー・モードにより、ローダーは com.ibm.websphere.objectgrid.plugins.ValueProxyInfo オブジェクトに各値をキャストできます。 ValueProxyInfo インターフェースには、トランザクションによって変更された属性名のリストをローダーが取得できるメソッドがあります。このメソッドにより、ローダーは値インターフェースで get メソッドを呼び出して、属性名に変更されたデータを取得し、変更された属性のみを設定する SQL UPDATE ステートメントをビルドすることができます。 where 文節は、基本キーの列と変更された各属性の列を持つようにビルドできるようになりました。この方法は前の方法よりも効果的ですが、ローダーにさらに多くのコードを書き込む必要があり、さまざまな置換を処理するために、さらに多くの準備済みステートメントのキャッシュが必要になる可能性があります。ただし、トランザクションが、通常ごく一部の属性しか変更しない場合、この制限は問題になりません。

- 一部のリレーショナル・データベースには API があるため、オプティミスティックなバージョン管理に役立つ列データを自動的に保守します。ご使用のデータベースの資料を参照して、この可能性が該当するかどうかを判断してください。

ObjectTransformer プラグイン

ハイパフォーマンスを必要とするときは、 ObjectTransformer プラグインを使用します。 CPU の使用に関するパフォーマンス上の問題がある場合は、各マップに ObjectTransformer プラグインを追加します。ObjectTransformer プラグインを使用しない場合、合計 CPU 時間の 60% から 70% まではエントリーの直列化とコピーに費やされます。

目的

ObjectTransformer プラグインの目的は、アプリケーションが以下の操作のカスタム・メソッドを提供出来るようにすることです。

- エントリーに対するキーの直列化または非直列化
- エントリーに対する値の直列化または非直列化
- エントリーに対するキーまたは値のコピー

ObjectTransformer プラグインが提供されない場合、 ObjectGrid は直列化および非直列化のシーケンスを使用してオブジェクトをコピーするので、ユーザーがキーと値の直列化を行う必要があります。この方法には費用がかかるので、パフォーマンスが重大である場合には ObjectTransformer プラグインを使用してください。アプリケーションが、トランザクションのオブジェクトを最初に検索する際に、コピーが行われます。このコピーは、マップのコピー・モード を NO_COPY に設定すると行われません。あるいは、コピー・モードを COPY_ON_READ に設定すると、コピー数を軽減できます。アプリケーションの必要に応じて、このプラグインにカスタム・コピー・メソッドを提供することによって、コピー操作を最適化します。このようなプラグインにより、コピー・オーバーヘッドを合計 CPU 時間の 65-70% から 2/3% に軽減できます。

デフォルトの copyKey および copyValue メソッド実装では、最初に clone() メソッドの使用を試行します (そのメソッドが提供されている場合)。 clone() メソッド実装が提供されていない場合は、実装のデフォルトは直列化になります。

ObjectGrid が分散モードで実行されているときは、オブジェクト直列化も直接使用されます。 LogSequence は、変更内容を ObjectGrid のピアに送信する前に、ObjectTransformer プラグインを使用して、キーおよび値の直列化を支援します。組み込み JDK シリアライゼーションを使用するのではなく、直列化のカスタム・メソッドを提供するときは、注意が必要です。オブジェクトのバージョン管理は複雑な問題であり、カスタム・メソッドがバージョン管理用に設計されていることが確認できない場合、バージョンの互換性に問題が発生することがあります。

以下のリストでは、ObjectGrid がキーと値の両方の直列化を試みる方法を詳細に述べています。

- カスタム ObjectTransformer プラグインが作成され、プラグインされている場合、ObjectGrid は ObjectTransformer メソッド内のメソッドを呼び出して、キーと値を直列化し、オブジェクトのキーおよび値のコピーを取得します。
- カスタム ObjectTransformer プラグインが使用されていない場合、 ObjectGrid はデフォルトに従って直列化と非直列化を行います。デフォルトが使用されている場合、各オブジェクトは、外部化可能または直列化可能として実装されます。e
  - オブジェクトが Externalizable インターフェースをサポートする場合、 `writeExternal` メソッドが呼び出されます。外部化可能として実装されたオブジェクトは、パフォーマンスを向上させます。
  - Externalizable インターフェースをサポートせず、直列化可能を実装しないオブジェクトは、`ObjectOutputStream` メソッドを使用して保管されます。

ObjectTransformer インターフェース

ObjectTransformer インターフェースの詳細については、 API 資料を参照してください。ObjectTransformer インターフェースには、キーまたは値の直列化と非直列化を行い、キーまたは値をコピーする以下のメソッドが含まれています。

```
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);
}
```

ObjectTransformer インターフェースの使用法

ObjectTransformer インターフェースは、以下の状態で使用します。

- 直列化不能オブジェクト
- 直列化可能であるが、直列化パフォーマンスを改善したオブジェクト。
- キーまたは値のコピー

以下の例で、ObjectGrid は Stock クラスのストアに使用されます。

```
/**
* Stock object for ObjectGrid demo
*
*
*/
public class Stock implements Cloneable {
String ticket;
double price;
String company;
String description;
int serialNumber;
long lastTransactionTime;
/**
* @return Returns the description.
*/
public String getDescription() {
return description;
}
/**
* @param description The description to set.
*/
public void setDescription(String description) {
this.description = description;
}
/**
* @return Returns the lastTransactionTime.
*/
public long getLastTransactionTime() {
return lastTransactionTime;
}
/**
* @param lastTransactionTime The lastTransactionTime to set.
*/
public void setLastTransactionTime(long lastTransactionTime) {
this.lastTransactionTime = lastTransactionTime;
}
/**
* @return Returns the price.
*/
public double getPrice() {
return price;
}
/**
* @param price The price to set.
*/
public void setPrice(double price) {
this.price = price;
}
/**
* @return Returns the serialNumber.
*/
public int getSerialNumber() {
return serialNumber;
}
/**
* @param serialNumber The serialNumber to set.
*/
public void setSerialNumber(int serialNumber) {
this.serialNumber = serialNumber;
}
/**
* @return Returns the ticket.
*/
public String getTicket() {
return ticket;
```

```
}
/**
* @param ticket The ticket to set.
*/
public void setTicket(String ticket) {
this.ticket = ticket;
}
/**
* @return Returns the company.
*/
public String getCompany() {
return company;
}
/**
* @param company The company to set.
*/
public void setCompany(String company) {
this.company = company;
}
//clone
public Object clone() throws CloneNotSupportedException
{
return super.clone();
}
}
```

Stock クラス用に、カスタム・オブジェクト変換プログラム・クラスを作成できます。

```
/**
* Custom implementation of ObjectGrid ObjectTransformer for stock object
*
*/
public class MyStockObjectTransformer implements ObjectTransformer {
/* (non-Javadoc)
* @see
* com.ibm.websphere.objectgrid.plugins.ObjectTransformer#serializeKey
* (java.lang.Object,
* java.io.ObjectOutputStream)
*/
public void serializeKey(Object key, ObjectOutputStream stream)
throws IOException {
String ticket= (String) key;
stream.writeUTF(ticket);
}
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#serializeValue(java.lang.Object,
java.io.ObjectOutputStream)
*/
public void serializeValue(Object value, ObjectOutputStream stream)
throws IOException {
Stock stock= (Stock) value;
stream.writeUTF(stock.getTicket());
stream.writeUTF(stock.getCompany());
stream.writeUTF(stock.getDescription());
stream.writeDouble(stock.getPrice());
stream.writeLong(stock.getLastTransactionTime());
stream.writeInt(stock.getSerialNumber());
}
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateKey(java.io.ObjectInputStream)
*/
public Object inflateKey(ObjectInputStream stream) throws IOException,
ClassNotFoundException {
String ticket=stream.readUTF();
```

```
return ticket;
}
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#inflateValue(java.io.ObjectInputStream)
*/
public Object inflateValue(ObjectInputStream stream) throws IOException,
ClassNotFoundException {
Stock stock=new Stock();
stock.setTicket(stream.readUTF());
stock.setCompany(stream.readUTF());
stock.setDescription(stream.readUTF());
stock.setPrice(stream.readDouble());
stock.setLastTransactionTime(stream.readLong());
stock.setSerialNumber(stream.readInt());
return stock;
}
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyValue(java.lang.Object)
*/
public Object copyValue(Object value) {
Stock stock = (Stock) value;
try{
return stock.clone();
}
catch (CloneNotSupportedException e)
{
//streamize one
}
}
/* (non-Javadoc)
* @see com.ibm.websphere.objectgrid.plugins.
ObjectTransformer#copyKey(java.lang.Object)
*/
public Object copyKey(Object key) {
String ticket=(String) key;
String ticketCopy= new String (ticket);
return ticketCopy;
}
}
```

次に、このカスタム MyStockObjectTransformer クラスを BackingMap にプラグインします。

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

TransactionCallback プラグイン

アプリケーションは、通常、 TransactionCallback プラグインとローダーの両方を対としてプラグインします。ローダーは、変更内容をバックエンドに適用するばかりでなく、バックエンドからデータをフェッチする役割もあります。このフェッチとフラッシュは、通常、 ObjectGrid トランザクションのコンテキスト内で行われます。

TransactionCallback プラグインには、以下の役割があります。

- トランザクションとローダーに必要なトランザクション固有の状態用にスロットを予約します。

- ObjectGrid トランザクションをプラットフォーム・トランザクションに変換またはマップします。
- ObjectGrid がトランザクションを開始するときのトランザクション別状態をセットアップします。
- ObjectGrid トランザクションがコミットするときにトランザクションをコミットします。
- ObjectGrid トランザクションがロールバックするときにトランザクションをロールバックします。

ObjectGrid は、XA トランザクション・コーディネーターではありません。ObjectGrid は、その機能を提供するプラットフォームに依存します。セッションで与えられる ObjectGrid の `begin`、`commit`、および `rollback` メソッドは、ライフ・サイクル呼び出しです。 TransactionCallback プラグインは、これらのイベントを受信し、ローダーが使用するリソースに対するトランザクション機能をプラットフォームに提供させます。このトピックでは、さまざまなシナリオを調査し、これらのシナリオで機能するように TransactionCallback プラグインを作成する方法について検討します。

TransactionCallback プラグイン概要

TransactionCallback プラグインは、 TransactionCallback インターフェースをインプリメントする POJO です。 TransactionCallback インターフェースは、以下の例のようになります。

```
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);
}
```

`initialize` メソッド

`initialize` メソッドは、 ObjectGrid が初期化されるときに呼び出されます。コールバックは、必要な TxID オブジェクトに対するスロットを予約します。通常の場合、トランザクションが開始するときに、 `begin` メソッド内に作成する、それぞれの状態またはオブジェクト用にスロットを予約します。例えば、ローダーとしてパーシスタンス・マネージャーを ObjectGrid と使用する場合です。このパーシスタンス・マネージャーにセッションおよびトランザクションの状態オブジェクトが存在するとすれば、TransactionCallback はセッションおよびトランザクションを取得し、 TxID のスロットにこれら 2 つのオブジェクトに対する参照を保持します。このケースでは、`initialize` メソッドは以下の例のようになります。

```
/**
* This is called when the grid first initializes. We'll just
* reserve our slots in the TxID.
*/
public void initialize(ObjectGrid objectGrid) throws
TransactionCallbackException
{
// reserve a slot for the persistence manager transaction
```

```
TXslot = objectGrid.reserveSlot(TxID.SLOT_NAME);
// reserve a slot for the persistence manager session
SessionSlot = objectGrid.reserveSlot(TxID.SLOT_NAME);
}
```

TxID にはスロットがあります。スロットは ArrayList アレイ上のエントリーです。プラグインは、ObjectGrid.reserveSlot メソッドを呼び出し、TxID オブジェクトのスロットを必要とすることを示すことによって、ArrayList アレイのエントリーを予約します。そして、このメソッドは、次のエントリー・インデックスをアプリケーションに戻します。アプリケーションは、このスロットに情報をストアできます。次のメソッドは、この手法を示しています。

begin メソッド

ObjectGrid は、新規トランザクションを開始するときにこのメソッドを呼び出します。このプラグインは、実トランザクションにこのイベントをマップします。実トランザクションは、commit メソッドが呼び出される前に到着する get および update メソッド呼び出し用にローダーが使用できます。以下の例は、begin メソッド で、ObjectGrid begin をパーシスタンス・マネージャーのトランザクション begin にマップしています。

```
/**
* This is called when the grid starts a new transaction. We just create a
* persistence manager transaction and call begin on it. We then store
* the transaction in the TxID slot so we can get it again later
* without needing ThreadLocal etc.
*/
public void begin(TxID id) throws TransactionCallbackException
{
Session PMsession = getPMcurrentSession();
Transaction tx = PMsession.beginTransaction();
id.putSlot(TXslot, tx);
id.putSlot(SessionSlot, PMsession);
}
```

この例は、initialize メソッドが TxID オブジェクトに 2 つのスロットを予約したことに依存しています。 1 つのスロットはパーシスタンス・マネージャーのセッション用で、もう 1 つはパーシスタンス・マネージャーの Transaction 用です。 begin メソッドは、パーシスタンス・マネージャーを呼び出して、セッションを取得し、それを索引付き SessionSlot スロットにストアし、セッションに Transaction を作成し、索引付き TXSlot スロットを使用して、このトランザクションへの参照をストアします。

commit メソッド

commit メソッドは、ObjectGrid トランザクションがコミットしているときに呼び出されます。すべてのローダーは既にフラッシュ済みです。プラグインの役割は、このコミット・イベントをプラットフォームに通知することです。

```
/**
* This is called when the grid wants to commit a transaction.
* We just pass it on to persistence manager.
*/
public void commit(TxID id) throws TransactionCallbackException
{
Transaction tx = (Transaction)id.getSlot(TXslot);
tx.commit();
}
```

このメソッドは、スロットにストアされているパーシスタンス・マネージャーのトランザクションを検索してから、commit メソッドを呼び出します。

rollback メソッド

このメソッドは、ObjectGrid トランザクションがトランザクションをロールバックするときに呼び出されます。プラグインは、これをプラットフォームのトランザクション・マネージャーに転送します。以下は、コードの断片です。

```
/**
* This is called when the grid wants to rollback a transaction.
* We just pass it on to persistence manager.
*/
public void rollback(TxID id) throws TransactionCallbackException
{
Transaction tx = (Transaction)id.getSlot(TXslot);
tx.rollback();
}
```

このメソッドは、commit メソッドに非常に似ています。スロットからパーシスタンス・マネージャーのトランザクションへの参照を取得してから、rollback メソッドを呼び出します。

isExternalTransactionActive メソッド

ObjectGrid セッションは通常、自動コミット・モードまたはトランザクション・モードで機能します。自動コミット・モードは、暗黙のトランザクションがセッションの ObjectMap インスタンスに対するメソッド呼び出しの頃に毎回作成されることを意味します。アクティブなトランザクションが存在しない場合に、アプリケーションが ObjectMap メソッドの呼び出しをすると、フレームワークは TransactionCallback プラグインのこのメソッドを呼び出して、アクティブ状態の適切なトランザクションが存在するかどうかをチェックします。このメソッドが真を戻す場合、フレームワークは自動的 begin を行い、そうでない場合は自動コミットを行います。このメソッドにより、ObjectGrid API の代わりにプラットフォーム API の begin、commit、または rollback のメソッドをアプリケーションが呼び出す環境に ObjectGrid を統合できます。

シナリオ: シンプルな Java Database Connectivity (JDBC) ベースの Java 2 Platform, Standard Edition (J2SE) 環境

この例では、アプリケーションが JDBC ベースのローダーを備える J2SE 環境を使用します。2 つのマップが存在し、それぞれには、データベースの異なるテーブルによって各マップを戻すローダーがあります。 TransactionCallback プラグインは、JDBC 接続を取得してから、その接続の begin、commit、および rollback のメソッドを呼び出します。以下は、TransactionCallback インプリメンテーションの例です。

```
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;
}
}
```

この例は、ObjectGrid トランザクション・イベントを JDBC 接続に変換する TransactionCallback プラグインを示しています。プラグインが初期化されると、JDBC 接続の参照を保持するために単一スロットを予約します。 begin メソッドは、新規トランザクションの JDBC 接続を取得し、自動コミットをオフにしてから、この接続への参照を TxID スロットにストアします。 commit と rollback のメソッドは、 TxID スロットから接続を取得して、その接続の適切なメソッドを呼び出します。 isExternalTransaction メソッドは、常時、偽を戻します。これは、アプリケーションが ObjectGrid トランザクション API を明示的に使用してトランザクションを制御しなければならないことを示しています。このプラグインと対のローダーは、TxID から JDBC 接続を取得します。ローダーは、以下の例のようになります。

```
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());
// implement get here
return null;
}
public void batchUpdate(TxID txid, LogSequence sequence)
throws LoaderException, OptimisticCollisionException
{
Connection conn = (Connection)txid.getSlot(tcb.getConnectionSlot());
// TODO implement batch update here
}
}
```

ローダーは、initialize メソッドが呼び出されたときに、 JDBCTCB インスタンスへの参照を取得します。次に、get および batchUpdate メソッドで必要とされると、 JDBCTCB によって取得された接続を取得します。 TransactionCallback インプリメンテーションとローダーは、対で通常、作成され、互いに協調します。TransactionCallback インプリメンテーションは、トランザクションを処理し、ローダーが必要とするオブジェクトを TxID 内のスロットにストアします。ローダーは、通常 TCB によって取得されたリソースを使用して、 get および batchUpdate メソッドを TransactionCallback によって管理されるトランザクションのコンテキストにインプリメントします。

シナリオ: サーブレット・エンジン環境

このシナリオでは、 ObjectGrid は JDBC ベースのローダーを使用していますが、管理下のサーブレット・エンジン内です。コンテナーは UserTransaction メソッドを使用してトランザクションを開始およびコミットすることを期待しています。これは、J2SE のケースとは少し異なります。それは、JDBC 接続への参照を TxID のスロットにストアする必要がないためです。コンテナーは JDBC 接続を管理しま

す。コンテナー・トランザクションがアクティブなときは、データ・ソースを使用して検索される接続によって毎回同じ接続がもたらされます。それは、コンテナーがこのトランザクションによって使用されている接続を記憶し、DataSource.getConnection メソッドが呼び出される度に、同一の接続を戻すためです。以下の例で、データ・ソース参照が Shareable として構成されているとします。

```
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;
}
}
```

この例は initialize メソッド内の UserTransaction メソッドへの参照を取得してから、適切な UserTransaction メソッドへ begin、commit、および rollback をマップします。スロットは不要です。それは、正確な接続情報がこのトランザクション用に取得されていることをコンテナーが検査するためです。以下は、この TransactionCallback インプリメンテーションと協力する JDBC ローダーです。

```
public class ManagedJDBCLoader implements Loader
{
DataSource myDataSource;
ManagedJDBCLoader(DataSource ds)
{
myDataSource = ds;
}
public void preloadMap(Session session, BackingMap backingMap)
throws LoaderException
{
}
public List get(TxID txid, List keyList, boolean forUpdate)
throws LoaderException
{
try
{
Connection conn = myDataSource.getConnection();
// TODO implement get here with this connection
return null;
}
catch(SQLException e)
{
throw new LoaderException("Cannot get objects", e);
}
}
public void batchUpdate(TxID txid, LogSequence sequence)
throws LoaderException, OptimisticCollisionException
{
try
{
Connection conn = myDataSource.getConnection();
// TODO implement update here using this connection
}
catch(SQLException e)
{
throw new LoaderException("Cannot update objects", e);
}
}
}
```

この例は、基本 JDBC バージョンよりも簡単になります。それは、コンテナーが接続を管理し、同一トランザクション内で、アクティブな同一トランザクションを使用して呼び出される度に DataSource.getConnection メソッドが必ず同一接続を戻すことをコンテナーが検査するためです。 アプリケーションは、選択すれば接続をキャッシュに入れることができますが、結果として、スロット内の接続をキャッシュに入れようとしてはなりません。

OptimisticCallback インターフェース

com.ibm.websphere.objectgrid.plugins.OptimisticCallback インターフェースを実装するプラグ可能オプティミスティック・コールバック・オブジェクトを提供できます。

目的

OptimisticCallback インターフェースを使用して、マップの値としてオプティミスティック比較演算を提供します。 OptimisticCallback は、 142 ページの『オプティミスティック・ロック』で説明されるように、オプティミスティック・ロック・ストラテジーを使用する場合に必要です。 ObjectGrid は、デフォルトの OptimisticCallback 実装を提供します。ただし、通常、アプリケーションは、その独自の OptimisticCallback インターフェースの実装をプラグインする必要があります。

アプリケーション提供の OptimisticCallback オブジェクトのプラグイン

次の例は、grid1 ObjectGrid インターフェース内の従業員のバックアップ・マップ用に、アプリケーションが OptimisticCallback オブジェクトをプラグインする方法を示しています。

```
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 );
```

前の例の EmployeeOptimisticCallbackImpl オブジェクトは、 OptimisticCallback インターフェースを実装する必要があります。次の例に示すように、アプリケーションは、XML ファイルを使用して、その OptimisticCallback オブジェクトをプラグインすることもできます。

```
<?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>
```

デフォルト・インプリメンテーション

ObjectGrid フレームワークは、 OptimisticCallback インターフェースのデフォルト・インプリメンテーションを提供します。このインプリメンテーションは、前のセクションで説明したように、アプリケーション提供の OptimisticCallback オブジェクト

をアプリケーションがプラグインしない場合に使用します。デフォルト・インプリメンテーションは、値のバージョン・オブジェクトとして、常に特殊値 NULL_OPTIMISTIC_VERSION を戻し、バージョン・オブジェクトの更新は行いません。このアクションにより、オプティミスティック比較は「ノーオペレーション」関数になります。オプティミスティック・ロック・ストラテジーを使用しているとき、たいていの場合、「ノーオペレーション」関数が発生することは望まないと思われます。ご使用のアプリケーションが OptimisticCallback インターフェースを実装し、独自の OptimisticCallback インプリメンテーションをプラグインする必要がある場合、デフォルト・インプリメンテーションは使用しません。ただし、デフォルト提供の OptimisticCallback インプリメンテーションが有用なシナリオが、少なくとも 1 つ存在します。次のような状態について考えてみます。

- ローダーがバックアップ・マップ用にプラグインされている。
- ローダーが、OptimisticCallback プラグインからの支援なしに、オプティミスティック比較を実行する方法を認識している。

ローダーが、OptimisticCallback オブジェクトからの支援なしで、オプティミスティック・バージョン管理を認識できる方法について考えてみます。ローダーは、値クラス・オブジェクトについての知識を持ち、オプティミスティック・バージョン管理の値としてどの値オブジェクトのフィールドを使用するかを認識しています。例えば、従業員マップの値オブジェクトに対して次のインターフェースを使用するとします。

```
public interface Employee
{
// Sequential sequence number used for optimistic versioning.
public long getSequenceNumber();
public void setSequenceNumber(long newSequenceNumber);
// Other get/set methods for other fields of Employee object.
}
```

この場合、ローダーは、getSequenceNumber メソッドを使用して、 Employee 値オブジェクトの現行バージョン情報を取得できることを認識しています。ローダーは、戻り値を増分して、新規 Employee 値で永続ストレージを更新する前に、新規バージョン番号を生成します。 Java Database Connectivity (JDBC) ローダーの場合、過剰 SQL 更新ステートメントの where 文節内の現行シーケンス番号が使用され、新規生成シーケンス番号が使用されて、シーケンス番号列を新規シーケンス番号の値に設定されます。このほかにも、オプティミスティック・バージョン管理に使用できる非表示の列を自動的に更新するなんらかのバックエンド提供の関数をローダーが利用する可能性があります。場合によっては、ストアード・プロシージャーまたはトリガーを使用して、バージョン情報が入っている列を保守できるようにすることもあります。ローダーが、バージョン譲歩を保守するためにこれらの技法のいずれかを使用している場合は、アプリケーションが OptimisticCallback インプリメンテーションを提供する必要はありません。デフォルトの OptimisticCallback は、ローダーが OptimisticCallback オブジェクトからの支援なしにオプティミスティック・バージョン管理を処理できるため、このような場合に便利です。

OptimisticCallback インターフェースの実装

OptimisticCallback インターフェースには、以下のメソッドと特殊値があります。

```
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;
}
```

次のリストには、OptimisticCallback インターフェース内の各メソッドについての説明または考慮事項があります。

NULL_OPTIMISTIC_VERSION

この特殊値は、アプリケーション提供の OptimisticCallback インプリメンテーションの代わりにデフォルトの OptimisticCallback インプリメンテーションが使用される場合に、 getVersionedObjectForValue メソッドによって戻されます。

getVersionedObjectForValue メソッド

このメソッドは、値のコピーを戻します。あるいはバージョン管理のために使用できる値の属性を戻すことがあります。このメソッドは、オブジェクトがトランザクションに関連付けられるたびに呼び出されます。ローダーがバックアップ・マップ内にプラグインしていない場合、バックアップ・マップは、コミット時刻にこの値を使用してオプティミスティック・バージョン管理比較を行います。オプティミスティック・バージョン管理比較は、このトランザクションが、このトランザクションによって変更されたマップ・エントリーに最初にアクセスしてから、バージョンが変更されていないことを確認するために、バックアップ・マップによって使用されます。別のトランザクションがすでにこのマップ・エントリーのバージョンを変更している場合、バージョン比較は失敗し、バックアップ・マップは OptimisticCollisionException 例外を表示して、トランザクションを強制的にロールバックします。ローダーがプラグインされている場合、バックアップ・マップはオプティミスティック・バージョン管理情報を使用しません。代わりに、ローダーは、オプティミスティック・バージョン管理比較を行い、必要に応じてバージョン管理情報を更新する責任があります。ローダーは通常、ローダーの batchUpdate に渡される LogElement から、初期バージョン管理オブジェクトを取得します。このオブジェクトは、フラッシュ操作が発生するか、トランザクションがコミットされたときに呼び出されます。

次のコードは、EmployeeOptimisticCallbackImpl オブジェクトによって使用されるインプリメンテーションを示しています。

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

前の例に示すように、sequenceNumber 属性は、ローダーが予期するように、java.lang.Long オブジェクト内に戻されます。これは、ローダーの作成

者と同一人物が EmployeeOptimisticCallbackImpl を作成したか、EmployeeOptimisticCallbackImpl を実装した人物と協力して作業を行ったか (例えば、 getVersionedObjectForValue メソッドによって戻された値に合意した) のいずれかであることを示しています。

前に示したように、デフォルトの OptimisticCallback は、バージョン・オブジェクトとして特殊値 NULL_OPTIMISTIC_VERSION を戻します。

updateVersionedObjectForValue メソッド

このメソッドは、トランザクションが値を更新し、新バージョンのオブジェクトが必要になるたびに呼び出されます。 getVersionedObjectForValue がこの値の属性を戻した場合、このメソッドは通常、属性値を新バージョンのオブジェクトに更新します。 getVersionedObjectForValue がこの値のコピーを戻した場合、このメソッドは通常、何もしません。デフォルトの OptimisticCallback は、 getVersionedObjectForValue のデフォルト・インプリメンテーションがバージョン・オブジェクトとして常に特殊値 NULL_OPTIMISTIC_VERSION を戻すため、何も行いません。

次のコードは、 OptimisticCallback セクションで使用される EmployeeOptimisticCallbackImpl オブジェクトによって使用されるインプリメンテーションを示しています。

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

前の例で示すように、sequenceNumber 属性は、次に getVersionedObjectForValue メソッドが呼び出されたときに、戻される java.lang.Long 値が長整数値を持つように、1 ずつ増分されます。この長整数値は、元のシーケンス番号の値に 1 を加えたもの (例えば、この従業員インスタンスの次のバージョン値) です。この場合も、この例は、ローダーを作成者が EmployeeOptimisticCallbackImpl の作成者と同一人物であるか、EmployeeOptimisticCallbackImpl を実装した人物と協力して作業を行ったかのいずれかであることを示しています。

serializeVersionedValue メソッド

このメソッドは、指定されたストリームにバージョン値を書き込みます。インプリメンテーションによっては、バージョン値を使用して、オプティミスティック更新の衝突を識別することができます。一部のインプリメンテーションでは、バージョン値は元の値のコピーです。それ以外のインプリメンテーションでは、値のバージョンを示すシーケンス番号またはその他のいくつかのオブジェクトがあります。実際のインプリメンテーションが不明であるため、このメソッドは適切な直列化を実行するために提供されます。デフォルト・インプリメンテーションは、writeObject 呼び出しを行います。

inflateVersionedValue メソッド

このメソッドは、バージョン値の直列化バージョンを取り、実際のバージョン値オブジェクトを戻します。インプリメンテーションによっては、バージ

ョン値を使用して、オプティミスティック更新の衝突を識別することができます。一部のインプリメンテーションでは、バージョン値は元の値のコピーです。それ以外のインプリメンテーションでは、値のバージョンを示すシーケンス番号またはその他のいくつかのオブジェクトがあります。実際のインプリメンテーションが不明であるため、このメソッドは、適切な非逐次化を行うために提供されます。デフォルト・インプリメンテーションは readObject を実行します。

複製のプログラミング

複製を構成するには、MapSet を ReplicationGroup および複製ポリシー属性に関連付けます。ReplicationGroup は、プライマリーおよび関連するレプリカと待機に使用されるサーバー・メンバーを定義します。また、この構成に必要なレプリカの最小数と最大数も定義します。複製ポリシー属性では、同期複製または非同期複製が必要かどうか、レプリカへの読み取りアクセスを許可するかどうか、およびレプリカに複製データを送信するときに圧縮を使用するかどうかを指定します。複製は、プログラミング・モデルに対してわずかな影響しか与えません。主な影響は、Map にデータをプリロードするアプリケーションに対してです。

マップのプリロード

各 Map に Loader を関連付けることができます。 Loader は、Map でオブジェクトが見つからないときにオブジェクトをフェッチするときに使用され、また、トランザクションがコミットしたときに変更内容をバックエンドに書き込む場合にも使用されます。 Loader は、マップへのデータのプリロードに使用することもできます。 Loader インターフェースのプリロード・メソッドは、Java 仮想マシン (JVM) が複製グループのプライマリーになった場合に呼び出されます。プリロード・メソッドは、レプリカまたは待機では呼び出されません。プリロード・メソッドは、提供されている Session を使用して、対象となる参照データすべてをバックエンドから Map にロードします。使用される Map は、プリロード・メソッドに渡される BackingMap 引数によって識別されます。

```
void preloadMap(Session session, BackingMap backingMap) throws LoaderException;
```

区画に分割された MapSet でのプリロード

マップは、*N* 個の区画に分割することができます。マップは、複数のサーバーにわたって格納することができます。この場合、各エントリーは、これらのサーバーのうちの 1 つにのみ格納されているキーによって識別されます。アプリケーションが、Map のすべてのエントリーを保持する場合に単一 JVM のヒープ・サイズによる制限を受けなくなったため、非常に大きい Map を ObjectGrid に保持することができます。 Loader インターフェースのプリロード・メソッドを使用してプリロードを行うアプリケーションでは、プリロードする必要のあるデータのサブセットを識別する必要があります。常に、固定数の区画が存在します。このことは、以下のコードの断片を使用して判別できます。

```
int numPartitions = backingMap.getPartitionManager().getNumOfPartitions();
int myPartition = backingMap.getPartitionId();
```

このコードの断片は、アプリケーションがデータベースからプリロードするデータのサブセットを識別する方法を示しています。アプリケーションは、マップが最初に区画に分割されていない場合でも、これらのメソッドを常に使用しなければなり

ません。これらのメソッドによって柔軟性が実現されます。管理者が後で Map を区画に分割した場合でも、ローダーは正常に機能し続けます。

アプリケーションは、バックエンドからサブセット myPartition を検索する照会を発行する必要があります。テーブル内のデータを簡単に区画に分割できるなんらかの自然な照会がある場合を除き、データベースが使用されているのであれば、所定レコードの区画 ID を列に持たせる方が処理が容易である可能性があります。

パフォーマンス

プリロードの実装では、単一トランザクションの Map に複数のオブジェクトを格納して、バックエンドから Map にデータをコピーする必要があります。トランザクションごとに格納できるレコード数は、状況に応じて異なります。トランザクションに 100 エントリーを超えるブロックが格納されると、パフォーマンス上の利点は少なくなります。最適な数は、オブジェクトの複雑さやサイズなどの要因によって異なります。 100 エントリーから開始して、パフォーマンス向上が見られなくなるまで数字を増やします。トランザクションが大きいほど、複製パフォーマンスが向上します。ただし、プライマリーのみがプリロード・コードを実行することに注意してください。プリロードされたデータは、プライマリーから、オンラインになっているすべてのレプリカに複製されます。

MapSets のプリロード

アプリケーションが複数の Map を持つ MapSet を使用する場合、各 Map はそれぞれ独自の Loader を持ちます。各 Loader に、プリロード・メソッドがあります。各 Map は、ObjectGrid によって順次ロードされます。単一 Map をプリロード Map として指定することによってすべての Map をプリロードすると、より効率的になる可能性があります。これは単なるアプリケーションの規則にすぎません。たとえば、部門と従業員という 2 つの Map が、部門 Map と従業員 Map の両方をプリロードするために、部門 Loader を使用する可能性があります。これにより、トランザクション上、アプリケーションで部門が必要な場合、その部門の従業員がキャッシュされるようになります。つまり、部門 Loader がバックエンドから部門をプリロードするときに、その部門の従業員もフェッチすることを意味します。このようにするには、単一トランザクションを使用して、部門オブジェクトとその関連従業員オブジェクトを Map に追加する必要があります。

リカバリー可能なプリロード

お客様には、非常に大きいデータ・セットをキャッシュすることが必要な場合があります。このデータのプリロードは、非常に時間がかかる可能性があります。アプリケーションがオンラインになる前に、プリロードを完了しなければならない場合もあります。このような場合、プリロードをリカバリー可能にすることがあります。100 万個のレコードをプリロードする必要があるとします。プライマリーがこれらのレコードをプリロードし、 800,000 件目のレコードの時点でプライマリーが失敗するとします。通常、新規プライマリーになるよう選択されているレプリカは、複製状態をクリアして、最初からプリロードを開始します。 ObjectGrid では、ReplicaPreloadController を使用して、これよりも効率よく処理できます。アプリケーションの Loader で、ReplicaPreloadController インターフェースを実装する必要が生ずることもあります。これにより、Loader に単一のメソッドが追加されます。

```
Status checkPreloadStatus(Session session, BackingMap bmap);
```

Loader インターフェースのプリロード・メソッドが正常に呼び出される前に、このメソッドが ObjectGrid ランタイムによって呼び出されます。 ObjectGrid は、このメソッド (Status) の結果をテストして、レプリカがプライマリーに格上げされるかどうかの振る舞いを判別します。

返される状況値	対応する ObjectGrid の振る舞い
Status.PRELOADED_ALREADY	この状況値は Map が完全にプリロードされていることを示しているため、ObjectGrid はプリロード・メソッドを呼び出しません。
Status.FULL_PRELOAD_NEEDED	ObjectGrid は Map をクリアし、正常にプリロード・メソッドを呼び出します。
Status.PARTIAL_PRELOAD_NEEDED	ObjectGrid は Map を現状のままにして、プリロードを呼び出します。このストラテジーによって、アプリケーションの Loader は、この時点以降プリロードを継続することができます。

プライマリーは、Map のプリロード中、返す必要のある状況をレプリカ側で判別できるように、複製中の MapSet 内の Map になんらかの状態を残す必要があります。 RecoveryMap などと呼ばれる追加の Map を使用することができます。この RecoveryMap は、プリロード中の同じ MapSet の一部である必要があります。これにより、プリロード中のデータと整合性を保ちながら複製が行われるようになります。

推奨のインプリメンテーションは、以下のとおりです。プリロードがレコードの各ブロックをコミットするとき、そのトランザクションの一部として RecoveryMap のカウンター/値も更新する必要があります。つまり、プリロードされたデータと RecoveryMap データは、レプリカにアトミックに複製されます。レプリカがプライマリーに格上げされると、RecoveryMap をチェックして何が起こったかを確認できるようになります。RecoveryMap は、単にキー「状態」を持つ単一のエントリーを保持するだけです。このキーに対するオブジェクトが存在しない場合は、完全なプリロードが必要となります (checkPreloadStatus は FULL_PRELOAD_NEEDED を返します)。この「状態」キーに対するオブジェクトが存在し、値が「COMPLETE」の場合は、プリロードが完了し、checkPreloadStatus が PRELOADED_ALREADY を返します。これ以外の場合、値オブジェクトは、どこからプリロードを開始する必要があるのかを示し、checkPreloadStatus メソッドは PARTIAL_PRELOAD_NEEDED を返すはずです。Loader は、プリロードが呼び出されたときに開始点がわかるように、Loader のインスタンス変数にリカバリー・ポイントを格納できます。また、各 Map が個別にプリロードされる場合、RecoveryMap も Map ごとのエントリーを保持できます。

Loader での同期複製モードにおけるリカバリーの処理

ObjectGrid ランタイムは、プライマリーが失敗したときにコミット済みデータを損失しないよう設計されています。以下のセクションでは、このことを実現するために使用されるアルゴリズムについて説明します。これらのアルゴリズムは、複製グループが同期複製を使用する場合にのみ適用されます。Loader はオプションです。

ObjectGrid ランタイムは、すべての変更がプライマリーからレプリカに同期複製されるように構成することができます。 JVM がレプリカに格上げされた場合、プライマリーは最初に Map のスナップショットをレプリカに送信します。レプリカがこのスナップショットを処理した後、プライマリーはスナップショットの生成以降のすべての変更 (完了したトランザクション) の送信を開始します。最終的に、レプリカはプライマリーに追いつきます。この初期の複製処理は、非同期で行われます。レプリカがプライマリーに追いつくと、ペアはピア・モードに入り、最後に同期複製が開始されます。この時点以降、プライマリーでコミットされる各トランザクションは、ピア・モードのすべてのレプリカに送信され、プライマリーは確認メッセージを待機します。これにより、確認メッセージの受信にかかわる待ち時間があるため、非同期複製シナリオの場合と比較してプライマリーがスローダウンします。プライマリーでの同期コミット・シーケンスは、以下のようになります。

Loader を使用する場合のステップ	**Loader を使用しない場合のステップ**
エントリーのロックを取得します。	同じ
Loader に変更をフラッシュします。	NOOP
キャッシュに変更を保存します。	同じ
レプリカに変更を送信し、確認通知を待機します。	同じ
TransactionCallback プラグインでローダーをコミットします。	TransactionCallBack プラグイン・コミットが呼び出されますが、通常、何も行われません。
エントリーのロックを解除します。	同じ

レプリカに変更が送信された後、Loader にコミットされることに注意してください。レプリカで変更がコミットされるタイミングについては、このシーケンスを確認してください。

初期化時に、プライマリーの tx リストを初期化します。

- CommitedTx = {}、RolledBackTx = {} を設定します。

同期コミット処理中:

Loader を使用する場合のステップ	**Loader を使用しない場合のステップ**
エントリーのロックを取得します。	同じ
Loader に変更をフラッシュします。	NOOP
キャッシュに変更を保存します。	同じ
コミット済みトランザクションおよびロールバック済みトランザクションでの変更をレプリカに送信し、確認通知を待機します。	同じ
コミット済みトランザクションおよびロールバック済みトランザクションのリストをクリアします。	同じ
TransactionCallBack プラグインで Loader をコミットします。	TransactionCallBack プラグイン・コミットが呼び出されますが、通常、何も行われません。

Loader を使用する場合のステップ	Loader を使用しない場合のステップ
コミットが成功した場合、トランザクションがコミット済みトランザクションに追加され、成功しなかった場合はロールバック済みトランザクションに追加されます。	NOOP
エントリーのロックを解除します。	同じ

レプリカ処理

- レプリカが変更されます。
- コミット済みトランザクション・リスト内のすべての受信済みトランザクションをコミットします。
- ロールバック済みトランザクション・リスト内のすべての受信済みトランザクションをロールバックします。
- トランザクションまたはセッションを開始します。
- トランザクションまたはセッションに変更を適用します。
- 保留リストにトランザクションまたはセッションを保存します。
- 応答を返信します。

レプリカ・モード中、レプリカでは Loader による相互作用がないことに注意してください。プライマリーは、すべての変更を Loader を介してプッシュする必要があります。レプリカは何も行いません。

このアルゴリズムの副次作用は、レプリカに常にトランザクションがあるが、次のプライマリー・トランザクションによってこれらのトランザクションのコミット状況が送信されるまで、コミットされないことです。その場合、トランザクションはレプリカでコミットまたはロールバックされます。ただし、このようになるまでは、トランザクションはコミットされません。短い時間 (数秒) 後にトランザクションの結果が送信されるようなタイマーをプライマリーに追加することができます。このようにすると、失効性はその時間ウィンドウに限定されますが、完全にはなくなりません。こうした失効性は、レプリカ読み取りモードを使用する場合のみの問題です。それ以外の場合、失効性は不可視であり、アプリケーションに影響を与えません。

プライマリーが失敗した場合、プライマリーでコミット/ロールバックされたトランザクションがいくつかある可能性がありますが、これらの結果が含まれるメッセージがレプリカに到達しませんでした。レプリカが新規プライマリーに格上げされると、このプライマリーが行う最初のアクションの 1 つは、この状態に対処することです。各保留中のトランザクションは、新規プライマリーの Map のセットに対して再処理されます。Loader がある場合は、その Loader に各トランザクションが送られます。これらのトランザクションは、厳密な FIFO 順序で送られます。失敗したトランザクションは無視されます。3 つのトランザクション A、B、および C が保留中の場合、A がコミットし、B がロールバックし、C がコミットすることがあります。1 つのトランザクションが他のトランザクションに影響を与えることはありません。これらのトランザクションは独立したものと見なされます。

Loader で使用されるロジックは、「フェイルオーバー・リカバリー」モードと「通常」モードの場合では若干異なることがあります。 Loader は

ReplicaPreloadController インターフェースを実装することによって、フェイルオーバー・リカバリー・モードにあることを容易に認識できます。 checkPreloadStatus メソッドは、フェイルオーバー・リカバリーが完了した場合にのみ呼び出されます。このため、Loader インターフェースの apply メソッドが checkPreloadStatus より前に呼び出される場合は、リカバリー・トランザクションになります。checkPreloadStatus メソッドが呼びされると、フェイルオーバー・リカバリーが完了します。

複製を使用したステートフル singleton

WebSphere Extended Deployment では、区画化機能を備えた最初のリリースに、singleton に対するサポートが追加されました。これにより、アプリケーションは、クラスター内に singleton を作成できるようになりました。ObjectGrid ランタイムでは、複製された MapSet を使用して類似のフィーチャーが実現されています。ObjectGrid singleton パターンには多くの利点がある一方で、いくつかの欠点もあります。ローカルで singleton/区画が活動状態であるときに、区画機能によってアプリケーションへのイベントが提供されると、区画機能の partitionLoad メソッドを使用して通信が行われます。複製された MapSet にも singleton (プライマリー) があります。singleton がプライマリーになると、Loader の ReplicaPreloadController#checkPreloadStatus メソッドによって、アプリケーションにこのことが通知されます。このことは、同様の方法で区画化機能にも使用することができますが、さまざまなバージョンの WebSphere Application Server または競合アプリケーション・サーバー間で移植可能であるという利点があります。

区画機能には非活動化イベントがありますが、ObjectGrid ランタイムにはこの機能がありません。ObjectGrid のプライマリーは、通常、失敗するまで稼働します。プライマリーを移動させることはできません。この点は、区画化機能が ObjectGrid よりも優れている点です。以下の表に機能を示します。

表 16.

機能	区画化機能	ObjectGrid singleton
singleton 開始イベント	はい	はい
singleton 停止イベント	はい	いいえ
singleton 状態の複製	いいえ	はい
複製の可変のサービスの品質 (QoS)	いいえ	はい
柔軟な singleton の配置	はい	いいえ
実行時の singleton の移動の可否	はい	いいえ
singleton への作業の IIOP ルーティング	はい	いいえ
Java 2 Platform, Enterprise Environment (J2EE) サーバーの必要性	はい	いいえ
WebSphere Extended Deployment のフル・バージョンの必要性	はい	いいえ

表 16. (続き)

機能	区画化機能	ObjectGrid singleton
Enterprise JavaBeans (EJB) の必要性	はい	いいえ
アプリケーションの他のアプリケーション・サーバーへの移植可能性	いいえ	はい

singleton 状態

区画化機能には、状態管理のための組み込みサポートがありませんでした。singleton に状態が必要となった場合、アプリケーションは、独自のデバイスに残されたままになっていました。通常、このことは、状態がデータベースにプッシュされたことを意味します。区画が失敗すると、区画をホストしてリカバリーするよう選択されているサーバーが、そのデータベースからこの状態を取得する必要がありました。これに代わってアプリケーションで ObjectGrid を使用した場合、singleton は、singleton を管理する ReplicaPreloadController に関連付けられている Map に、その状態を保持することができます。プライマリーまたは singleton が失敗した場合、新規プライマリーになるよう選択されているレプリカは、複製の結果としてその状態を既にローカルで持っています。アプリケーションでデータ損失が許容される場合を除いて、同期複製を使用する必要があります。

柔軟な singleton の配置

区画化機能では、HA マネージャー・ポリシー・メカニズムを使用して、区画をホストする場所、およびこれらのポリシーを実行時に変更し、その変更が即時に有効になる場所を判別します。 ObjectGrid 複製グループ・ポリシーは、HA マネージャーのポリシーほど柔軟ではないため、すべてのサーバーを再始動せずに変更することができません。 ObjectGrid を使用する場合、実行時に singleton を移動させることはできなくなります。

複製の可変の QoS

区画機能には、状態管理が備えられていません。一方 ObjectGrid には、さまざまな複製アプローチが備えられています。

- 複製なし
- 非同期複製
- 同期複製

ポリシーは、状態に関してユーザーが使用している Map に関連付けられている MapSet の複製ポリシーによって決定されます。同期複製では、データは失われませんが、低速になります。非同期複製は高速ですが、プライマリーが失敗すると、プライマリーでコミットされた 1 つ以上のトランザクションが失われる可能性があります。

レプリカ間のロード・バランシング

特に構成されていない限り、ObjectGrid は、すべての読み取り要求と書き込み要求を指定された複製グループのプライマリー・サーバーに送信します。つまり、プラ

イマリーだけが、クライアントからのすべての要求にサービスを提供する必要があります。読み取り要求をプライマリーのレプリカに送信できるようにしたい場合があります。これにより、読み取り要求のロードを複数の JVM で共有することができますが、読み取り要求をレプリカに送信すると整合性が低下することになります。

このことは、通常、クライアントが常時変更されるデータをキャッシュしているか、またはクライアントがペシミスティック・ロックを使用している場合にのみ使用されます。

データが絶えず変更され、そのためクライアント・ニア・キャッシュで無効化された場合は、クライアントからプライマリーへの get 要求率が比較的高くなります。同様に、ペシミスティック・ロック・モードでは、ローカル・キャッシュがないためすべての要求がプライマリーに送信されます。

データが比較的静的であるか、またはペシミスティック・モードが使用されていない場合、ウォーム・キャッシュを持つクライアントからの get 要求の頻度は高くならないため、レプリカ読み取りはパフォーマンスにそれほど大きな影響を与えません。

ただし、クライアントが最初に始動された場合、そのニア・キャッシュは空であり、空のキャッシュに対するキャッシュ要求はプライマリーに転送されます。時間が経過してクライアント・キャッシュにデータが入れられると、この要求ロードは除去されます。数多くのクライアントがあり、これらのクライアントの多くが同時に始動される場合、このロードは大きくなる可能性があるため、パフォーマンス上、レプリカ読み取りを選択するほうが適切であることがあります。

レプリカおよび非同期複製からの読み取り

複製グループのデータがそれほど頻繁に変更されない場合、このことは通常、有効なトレードオフとなります。これにより、クライアントからの get 要求を、オンラインになっている任意のレプリカ上のデータに送信できます。get 要求は、コピーを持たないレプリカに送信され、この時点ではレプリカにキー/値が複製されていないことがあります。データがレプリカにない場合は、get 要求はプライマリーに転送されます。

データが変更されると、レプリカの get によって失効データが返される可能性が高くなります。このことは、アプリケーションによって許容される場合も、許容されない場合もあります。アプリケーションによって許容されない場合は、レプリカからの読み取りを使用可能にしないでください。

同期複製モードでのレプリカからの読み取り

同期複製では、レプリカを、プライマリーとまったく同じ状態に保持しようとします。プライマリーが失敗した場合、プライマリーでコミットされたすべてのデータは、失敗が発生した時点でピア・モードにあったすべてのレプリカ上に確実に存在します。失敗が発生したときにはこのようになりますが、レプリカからの読み取りを許可すると、使用されているアルゴリズムのいくつかの副次作用が明らかになります。

プライマリーがトランザクションをコミットしようとしているときに、変更のコピーがレプリカに送信されると、以下の 2 つの場合にレプリカはこのトランザクションをコミットします。

- プライマリーが失敗した場合
- プライマリーの次のトランザクションが送信された場合

プライマリーが失敗すると、レプリカの保留中のトランザクションがすべてコミットされます。

保留中のトランザクションは、プライマリーの後続のトランザクションがコミットされたときにのみコミットされます。プライマリーは、このレプリカ・メッセージでコミットの結果を通知します。レプリカは、これらのメッセージのうちの 1 つを受け取ると、このメッセージで指定されている結果を持つ保留中のトランザクションをすべてコミットまたはロールバックします。

保留中のトランザクションは、コミットされると単にレプリカでの読み取りが可能になるだけです。もちろん、プライマリーにロードがあり、通常の変更が行われた場合、これらの保留中のトランザクションはすぐにコミットされます。プライマリーの変更ロードが低い場合、次のプライマリーの変更が行われるまで、保留トランザクションがコミットされない期間があります。

もちろん、変更を受けるプライマリーのレプリカは、通常、読み取り時点を基準にして少なくとも 1 トランザクション分プライマリーから遅れています。これらの保留中のトランザクションの結果がプライマリーから送信されるまで、データは失われず、これらのトランザクションは物理的にレプリカにあってコミットされない状態でいます。次の読み書きトランザクションが実行されると、コミットが行われます。

要約

レプリカからの読み取りが使用可能になっている場合、一部の get によって失効データが返されることを許容できるようにアプリケーションを準備しておく必要があります。このことは、同期複製か非同期複製が使用されているかどうかにかかわらず、該当します。

区画化

単一 Java 仮想マシン (JVM) で使用できる以上のメモリーが MapSet 内のオブジェクトで必要とされたり、更新に必要なスループットが JVM で提供できない場合は、区画化を使用します。

どこにエントリーが保持されるか

ハッシュ・アルゴリズムによって、どのサーバーで各エントリーが保持されるか判別されます。管理者は、PartitionSet 定義を使用して、使用する区画数を指定します。この構成は、JVM が開始した後は変更することができません。エントリーのキーから単純なハッシュ値が取得され、この値を区画数で除算した結果のモジュロ (%) によって、そのエントリーを「所有する」サーバーが示されます。

通常、キー・オブジェクトの Java hashCode メソッドが使用されます。この値をオーバーライドするには、hashCode インプリメンテーションをオーバーライドします。

アプリケーションでは場合により、通常のハッシュの値の変更は望まないが、エントリー配布には別のハッシュ・アルゴリズムを使用したい場合があります。このような状況には、com.ibm.websphere.objectgrid.plugins.PartitionableKey インターフェースが対応することができます。このインターフェースには、次の 1 つのメソッドがあります。

```
Object ibmGetPartition();
```

キーがこのインターフェースを実装すると、ObjectGrid ランタイムは、キー・オブジェクトのハッシュではなく、このメソッドによって返されたオブジェクトのハッシュを使用します。

実行時の区画化

com.ibm.websphere.objectgrid.PartitionManager インターフェースに備えられている API を使用すると、アプリケーションは、実行時の区画化に関する情報を判別することができます。アプリケーションは、BackingMap インターフェースの getPartitionManager メソッドを使用して、このインターフェースのインスタンスへの参照を取得することができます。 Map の BackingMap への参照は、ObjectGrid インスタンスの ObjectGrid インターフェースの getMap(String) メソッドを使用して取得することができます。または、この参照は、一部のプラグイン・コールバックのパラメーターとして渡されます (ローダー・インターフェースの preload(Session, BackingMap) など)。

PartitionManager インターフェースのメソッド

PartitionManager インスタンスを使用すると、アプリケーションは、区画化に関する以下の項目を判別することができます。

メソッド名	説明
int getNumOfPartitions()	Map を分割する区画の数を返します。
int getPartition(Object key)	指定されたキーを持つエントリーに対して使用される 0 ベースの区画数を返します。
List /*Integer*/ getPartitions(List /*Object*/ keys)	このメソッドは、getPartition メソッドと同じですが、キーのリストに対して動作します。返される整数リストには、対応する入力キーごとの区画番号が含まれます。
List /*List Integer*/ getPartitionLists(List /* Object */ keys)	このメソッドは、getPartitions メソッドと同じですが、区画リストの優先順位リストを返します。たとえば、返されるリストの最初のエントリーには、区画 0 に対応する入力キーのリストが含まれ、次のエントリーには区画 1 に対応する入力キーのリストが含まれる、などのようになります。

メソッド名	説明
List /*LogSequence*/ partitionLogSequence(LogSequence ls)	このメソッドは、1 つの LogSequence を、指定された区画の LogSequence のリストに分割します。入力 LogSequence は検査され、その中の各 LogElement について適切な区画が判別されます。次に、LogElement を持つ各区画についてシーケンスが検査された後、これらの LogElement の LogSequence が返されます。

区画化に関する制限事項

トランザクションは、トランザクションあたり 1 つの区画でのみエントリーを変更することができます。トランザクションが MapSet 内の複数のエントリーを変更し、これらのエントリーが別の区画にハッシュされる場合、トランザクションのコミットが試行されるとトランザクションがロールバックします。トランザクションは、別の区画からオブジェクトを読み取ることができます。ただし、トランザクションは、単一の区画内でのみ複数のエントリーを変更することができます。

区画に対してプライマリーが選択された場合のアプリケーション・イベント

Map にローダーが提供され、ローダーによって ReplicaPreloadController もインプリメントされている場合、アプリケーションは、checkPreloadStatus コールバックを使用して、そのメソッド呼び出しを受け取っている JVM がこの区画に対するプライマリーになっていることを示すイベントを受信することができます。区画 ID は、BackingMap インターフェースの getPartitionId メソッドを使用して識別することができます。プリロードについての詳細は、210 ページの『ローダー』を参照してください。

クライアントでの区画化とサーバー上での実行の比較

区画化は、アプリケーションが、ObjectGrid インターフェースの接続メソッドを使用して取得された ObjectGrid を使用している場合にのみ機能します。プラグインのコールバックによって ObjectGrid がアプリケーションに提供されている場合、ObjectGrid はルーティングを行わないローカル ObjectGrid になります。サーバー上での実行中に区画化機能を透過的に利用する場合は、すべてのトランザクションに対して、接続メソッドを使用して取得した ObjectGrid を使用します。ただし、この場合、フレームワークによって提供されるローカル ObjectGrid 参照を使用するときと比較するとパフォーマンスが低下します。区画化機能が不要な場合は、プラグインに提供されているローカル参照を使用してください (可能な場合)。

索引付け

索引付けフィーチャーは、1 つの索引または複数の索引を BackingMap に作成する場合に使用できます。1 つの索引は、BackingMap 内の 1 つのオブジェクトの 1 つの属性から作成されます。このフィーチャーにより、アプリケーションはより迅速に特定のオブジェクトを見つけることができます。索引がない場合、アプリケーションはオブジェクトを探し出すのに、オブジェクトのキーを使用する必要がありま

す。索引付けフィーチャーを使用すると、アプリケーションは特定の値を持つオブジェクトや、ある値の範囲内にあるオブジェクトを見つけることができます。これは、Enterprise JavaBeans (EJB) Query が、指定された基準を使用して照会を行うことにより、EJB オブジェクトを見つけるのに似ています。索引付けを使用すると、アプリケーションは、オブジェクト検索プロセスでより簡単にオブジェクトを見つけ出すことができ、そのときのパフォーマンスも向上します。

索引付けには、静的および動的という 2 つのタイプがあります。*静的* 索引付けの場合は、ObjectGrid インスタンスを初期化する前に、BackingMap に索引プラグインを構成する必要があります。この構成を行うには、BackingMap を XML で構成するか、またはプログラマチックに構成します。静的索引付けでは、まず最初に、ObjectGrid の初期化中に索引を作成します。索引は常に BackingMap に同期しており、いつでも使用できる準備ができています。静的索引付けプロセスがすでに開始している場合、索引は、ObjectGrid トランザクション管理プロセスの一環として保守されます。トランザクションが変更をコミットした場合、これらの変更により、静的索引も更新されます。トランザクションがロールバックされると、索引の変更もロールバックされます。

動的索引付けの場合は、索引を含む ObjectGrid インスタンスの初期化の前または後に、BackingMap に索引を作成することができます。アプリケーションは、動的索引付けプロセスに対して、ライフ・サイクル制御を行います。動的索引は、不要になると除去できます。アプリケーションが動的索引を作成する場合は、索引作成プロセスを完了するまでに時間がかかるために、その索引をすぐに使用できないことがあります。この時間は索引付けされるデータの量に依存するので、特定の索引付けイベントが発生したときにそのことを通知してもらいたいアプリケーションのために、DynamicIndexCallback インターフェースが提供されています。これらのイベントには、準備完了、エラー、および破棄があります。アプリケーションは、このコールバック・インターフェースを実装し、動的索引付けプロセスに登録できます。

索引付けフィーチャーは、MapIndexPlugin プラグインと表されるか、または略して Index で表されます。MapIndexPlugin は BackingMap プラグインです。BackingMap は、索引プラグインが索引構成規則に従っている限り、複数の索引プラグインを構成できます。

BackingMap に索引プラグインが構成されている場合、対応する ObjectMap から索引プロキシー・オブジェクトを取り出すことができます。 ObjectMap の getIndex メソッドを呼び出し、索引プラグインの名前を渡すと、索引プロキシー・オブジェクトが戻されます。この索引プロキシー・オブジェクトは、MapIndex、MapRangeIndex、またはカスタマイズされた索引インターフェースなど、適切なアプリケーション索引インターフェースにキャストする必要があります。

現在のところ、索引付けフィーチャーはローカル・キャッシュにおいてのみサポートされており、分散キャッシュではサポートされていません。索引付け操作を分散キャッシュに対して行うと、`UnsupportedOperationException` 例外が発生します。

索引プラグインの実装

com.ibm.websphere.objectgrid.plugins.index パッケージ内の HashIndex クラスは、組み込みアプリケーション索引インターフェースである MapIndex と MapRangeIndex の両方をサポートすることのできる、組み込み索引プラグイン実装です。

アプリケーションは、より複雑な索引をプログラムできるように、それぞれ独自の索引プラグイン実装を提供することができます。索引実装クラスは、com.ibm.websphere.objectgrid.plugins.index.MapIndexPlugin インターフェースを実装する必要があります。MapIndexPlugin の定義は、以下のとおりです。

```
/**
* An index implementation must implement this interface so that modifications
* to the Map are propagated to it so that it can maintain the index as
* transactions are committed. Only attributes that implement the
* {@link java.lang.Comparable} interface are eligible to be indexed.
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapIndex
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex
*/
public interface MapIndexPlugin
{
/**
* This should be the name of the attribute to be indexed. If the object
* has an attribute called EmployeeName then the index will call the
* "getEmployeeName" method. The attribute name must be the name
* as that in the get method and the attribute must implement the
* {@link java.lang.Comparable} interface.
*
* @param attributeName
* The name of the attribute to set.
*/
public void setAttributeName(String attributeName);
/**
* This index name.
*
* @return The name of the index.
*
* @see com.ibm.websphere.objectgrid.ObjectMap#getIndex
*/
String getName();
/**
* Gets an index proxy object for performing index lookup operations. The
* caller must cast the object returned to either a MapIndex or MapRangeIndex
* object to perform the lookup operations.
*
* @param map The MapIndexInfo object required for maintaining the index.
* .
* @return a proxy to either an object that implements MapIndex or MapRangeIndex.
*/
Object getIndexProxy(MapIndexInfo map);
/**
* This is called by the core to allow the index to be updated as the result
* of changes applied to map during the commit cycle of a transaction.
* Use the {@link LogElement#getType()} method to determine what operation is
* required to for updating the index. Use the {@link LogElement#getBeforeImage()}
* to get the value object that existed prior to committing transaction applying
* a change to the map and the {@link LogElement#getAfterImage()} to get the value
* object after the committing transaction applied the change to the map entry.
*
* Note, the {@link #undoBatchUpdate(TxID, LogSequence)} method may be called
* later to undo these changes if an exception occurs that causes committing
* transactions to be rolled back instead.
*
* @param txid The transaction for the changes.
* @param sequence The log sequence that contains changes from transaction.
*
* @throws ObjectGridRuntimeException is a failure occurs that requires transaction
* to be rolled back.
*/
void doBatchUpdate(TxID txid, LogSequence sequence) throws
ObjectGridRuntimeException;
```

```
/**
* This is called by the core to undo any changes made to the index as a result of
* a prior call to the {@link #doBatchUpdate(TxID, LogSequence)} method. This
* method is called when an exception or error condition that requires all
* changes made by transaction to be rolled back. For this reason, the
* implementation of this method should catch all Throwable and continue with
* next LogElement in the LogSequence until all LogElements are processed so that
* as many changes to the index is undone as possible. An ObjectGridException
* should only be thrown after processing the entire LogSequence and this method
* was unable to successfully undo 1 or more changes in the LogSequence.
*
* Use the {@link LogElement#getUndoType()} method to determine what operation is
* required to undo any change made to the index. Use the
* {@link LogElement#getBeforeImage()} to get the value object that existed prior
* to committing transaction applying a change to the map and the {@link
* LogElement#getAfterImage()} to get the value object after the committing
* transaction applied the change to the map entry.
*
* @param txid The transaction for the changes.
* @param sequence The log sequence that contains changes from transaction.
*
*/
void undoBatchUpdate( TxID txid, LogSequence sequence) throws ObjectGridException;
}
```

setAttributeName および getName メソッドの動作は、その名前が表しているとおりです。その他のメソッドにはより注意が必要です。

getIndexProxy メソッド

getIndexProxy メソッドは、MapIndex インターフェース、MapRangeIndex インターフェース、またはカスタムの Index インターフェースを実装する索引プロキシー・オブジェクトを戻す必要があります。索引プロキシー・オブジェクトの実装は、索引プラグインの中核をなす部分です。

MapIndexInfo オブジェクトは、このメソッドに渡されて、トランザクション変更情報を提供します。この情報は、getIndexProxy メソッドを呼び出す現行トランザクションからのみ可視になっているデータです。索引プロキシー・オブジェクトは、この MapIndexInfo オブジェクトを使用して、このトランザクション・データを検索できます。

MapIndexInfo インターフェースの定義を以下に示します。

```
/**
* This interface is used to provide an index with detailed change information
* for a specific Map in a transaction.
*/
public interface MapIndexInfo
{
/**
* An index contains the key values of a set of map entries that have a
* a specific attribute value. This method returns the ObjectMap the
* index is referring to ObjectMap that the index is associated with.
*
* @return ObjectMap this index is associated with.
*/
ObjectMap getMap();
/**
* Returns the set of all changes made by the current transaction to the
* ObjectMap that is returned by the {@link #getMap()} method.
*
* @param includeRemoved must be set to true to include LogElement.DELETE types
```

```
* in the list returned by this method.
*
* @return a List of LogElement created for each ObjectMap entry that was
* either inserted, updated, or removed by current transaction.
*
* @throws ObjectGridRuntimeException
*/
List getTransactionChanges(boolean includeRemoved) throws
ObjectGridRuntimeException;
/**
* This returns the set of changes as they apply to a particular set of keys
* in the current transaction for the ObjectMap that is returned by the
* {@link #getMap()} method. If a key has not been referenced
* in the transaction then null is returned.
*
* @param keys The list of keys for which the data is required.
* @return a List of LogElement corresponding to the keys or null if the keys
* was not referenced.
*
* @throws ObjectGridRuntimeException
*
* @see com.ibm.websphere.objectgrid.plugins.LogElement
* @see com.ibm.websphere.objectgrid.ObjectMap
*/
List getTransactionChanges(List keys) throws ObjectGridRuntimeException;
}
```

getIndexProxy メソッドは、ObjectMap インターフェースの getIndex(String name) メソッドをサポートするよう設計されています。戻される索引プロキシー・オブジェクトは、ObjectMap の getIndex メソッドによって戻される索引プロキシー・オブジェクトです。例えば、アプリケーションが ObjectMap の getIndex メソッドを呼び出すと、次にこのメソッドが、この getIndexProxy メソッドを呼び出して、この getIndexProxy メソッドが戻したオブジェクトを戻します。このアプリケーションは、戻された索引プロキシー・オブジェクトをアプリケーション索引インターフェース (MapIndex、MapRangeIndex、または別のカスタマイズされた索引インターフェースなど) にキャストする必要があります。

以下のコード例では、getIndexProxy メソッドが戻すことのできる索引プロキシー・オブジェクトのいくつかの実装を示しています。

```
/**
* A class used to return a proxy to this map index
* so that applications can perform query operations
* using MapIndex interface.
*/
class Proxy implements MapIndex
{
/**
* The MapIndexInfo object associated with this index proxy object.
*/
protected MapIndexInfo ivMap;
/**
* Maximum number of retries when concurrent transactions
* modify index during a query operation.
*/
protected static final int RETRY_LIMIT = 10;
/**
* EQUAL comparator to use.
*/
final protected ProxyEQComparator ivEQ = new ProxyEQComparator();
final protected ProxyGTComparator ivGT = new ProxyGTComparator();
final protected ProxyRangeComparator ivRange = new ProxyRangeComparator();
/**
```

```
* Construct a proxy object for a given ObjectMap.
*
* @param map
* is the MapIndexInfo object.
*/
Proxy(MapIndexInfo map)
{
ivMap = map;
}
/**
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapIndex#findAll
*/
public Iterator findAll(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivEQ.ivAttribute = (Comparable) attributeValue;
ArrayList resultList = null;
int retryCount = 0;
boolean retry;
do
{
// Variables that need to be re-initialize each time thru loop.
retry = false;
resultList = new ArrayList();
// Use index to obtains the Set of keys for map entries that
// contain the specified attribute value.
Set s = (Set) index.get( attributeValue );
Set keySet = processSet( s, ivEQ );
if ( keySet != null )
{
resultList.addAll( keySet );
}
else
{
// Whoops, another transaction modified Set obtained from index
// while the above was iterating over the Set to perform the
// addAll operation. Therefore, we need to retry by starting
// over beginning with getting Set from index to pickup changes
// from the transaction that just modified the Set.
++retryCount;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
retry = true;
}
} while ( retry );
// Return iterator for result list created by above loop.
Iterator result = resultList.iterator();
return result;
}
/**
* Process a Set obtained from index to determine if which of the keys
* are for map entries that meet the query select criteria.
*
* @param s
* is the Set of key values for entries in BackingMap
* this index is built over. A null reference indicates only
* changes from current transaction needs to be processed.
*
* @param comparator
```

```
* is the comparator to use for making range check.
*
* @return Set of keys that met the select criteria or a null reference
* if a Exception occurs while iterating over the Set.
*
* @throws FinderException
* if an error condition prevents processing of Set
* from being performed.
*/
protected Set processSet(Set s, ProxyComparator comparator)
throws FinderException {
HashSet resultSet = new HashSet();
//...
//process the s Set, use comparator and prepare the resultSet.
//...
return resultSet;
}
} // end class Proxy
/**
* A class used to return a proxy to this map index so that applications can
* perform query operations using MapRangeIndex interface.
*/
class RangeProxy extends Proxy implements MapRangeIndex
{
/**
* Various comparators needed by proxy to perform the
* the range check of attribute value.
*/
final private ProxyLTComparator ivLT = new ProxyLTComparator();
final private ProxyLEComparator ivLE = new ProxyLEComparator();
final private ProxyGEComparator ivGE = new ProxyGEComparator();
/**
* Index is a synchronized SortedMap.
*/
final SortedMap ivIndexSortedMap;
/**
* Construct a MapRangeIndex proxy.
*/
RangeProxy(MapIndexInfo map)
{
super( map );
ivIndexSortedMap = (SortedMap) index;
}
/**
* Execute query operation on a specified Map and ProxyComparator object.
*
* @param map
* is a subset of the index to perform finder operation on.
* @param proxyComparator
* is a comparator used to perform range check on attribute value.
*
* @return Set of keys required to be returned by finder method.
*
* @throws FinderException
* is any failure occurs during execution of the query.
*/
private Set executeQuery(Map map, ProxyComparator proxyComparator)
throws FinderException {
HashSet resultList = null;
int retryCount = 0;
boolean retry;
do
{
// Variables that need to be re-initialize each time thru loop.
retry = false;
resultList = new HashSet();
// Use index to obtains the Set of keys for map entries that
```

```
// contain the specified attribute value.
SortedMap treeMap = (SortedMap) index;
Collection values = map.values();
if ( values.isEmpty() )
{
// Nothing in range currently in index, so we only
// need to check changes from current transaction.
Set keySet = processSet( null, proxyComparator );
if ( keySet != null )
{
resultList.addAll( keySet );
}
}
else
{
// Index does contains some keys in range, so we need to query
// both index entries as well as current transaction changes.
Iterator iter = values.iterator();
while ( iter.hasNext() )
{
Set keySet;
try
{
Set s = (Set) iter.next();
keySet = processSet( s, proxyComparator );
}
catch (ConcurrentModificationException e)
{
// Indicate unable to get keySet.
keySet = null;
}
if ( keySet != null )
{
resultList.addAll( keySet );
}
else
{
++retryCount;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
retry = true;
}
}
}
} while ( retry );
return resultList;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findGreater
*/
public Iterator findGreater(Object attributeValue)
throws FinderException {
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivGT.ivAttribute = (Comparable) attributeValue;
SortedMap tailMap = ivIndexSortedMap.tailMap( attributeValue );
Set resultSet = executeQuery( tailMap, ivGT );
Iterator result = resultSet.iterator();
return result;
```

```
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findGreaterEqual
*/
public Iterator findGreaterEqual(Object attributeValue)
throws FinderException {
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivGE.ivAttribute = (Comparable) attributeValue;
SortedMap tailMap = ivIndexSortedMap.tailMap( attributeValue );
Set resultSet = executeQuery( tailMap, ivGE );
Iterator result = resultSet.iterator();
return result;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findLess
*/
public Iterator findLess(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivLT.ivAttribute = (Comparable) attributeValue;
SortedMap headMap = ivIndexSortedMap.headMap( attributeValue );
Set resultSet = executeQuery( headMap, ivLT );
Iterator result = resultSet.iterator();
return result;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findLessEqual
*/
public Iterator findLessEqual(Object attributeValue) throws FinderException
{
if ( attributeValue == null )
{
throw new IllegalArgumentException(
"the attributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivLE.ivAttribute = (Comparable) attributeValue;
Set resultSet;
int retryCount = 0;
boolean retry;
do
{
// re-initialize for each retry that occurs.
retry = false;
SortedMap headMap = ivIndexSortedMap.headMap( attributeValue );
resultSet = executeQuery( headMap, ivLE );
Set s = (Set) ivIndexSortedMap.get( attributeValue );
ivEQ.ivAttribute = (Comparable) attributeValue;
Set equalSet = processSet( s, ivEQ );
if ( equalSet != null )
{
```

```
if ( ! equalSet.isEmpty() )
{
resultSet.addAll( equalSet );
}
}
else
{
// Whoops, another transaction modified index while processSet
// was executing. Therefore, we need to retry the entire query.
++retryCount;
retry = true;
if ( retryCount >= RETRY_LIMIT )
{
throw new FinderException( "query retry limit exceeded" );
}
}
} while ( retry );
// Return iterator for result list created by above loop.
Iterator result = resultSet.iterator();
return result;
}
/*
* (non-Javadoc)
*
* @see com.ibm.websphere.objectgrid.plugins.index.MapRangeIndex#findRange
*/
public Iterator findRange(Object lowAttributeValue, Object highAttributeValue)
throws FinderException {
if ( lowAttributeValue == null )
{
throw new IllegalArgumentException(
"the lowAttributeValue must be a non null reference" );
}
if ( highAttributeValue == null )
{
throw new IllegalArgumentException(
"the highAttributeValue must be a non null reference" );
}
// Use the greater than comparator for range check.
ivRange.ivLowAttribute = (Comparable) lowAttributeValue;
ivRange.ivHighAttribute = (Comparable) highAttributeValue;
SortedMap subMap = ivIndexSortedMap.
subMap( lowAttributeValue, highAttributeValue );
Set resultSet = executeQuery( subMap, ivRange );
Iterator result = resultSet.iterator();
return result;
}
}
/**
* Abstract base class used for determining if attribute value is in range.
*/
abstract class ProxyComparator
{
abstract boolean inRange(Object attribute);
}
/**
* Performs less than range check.
*/
class ProxyLTComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
return false;
}
```

```
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) < 0 );
}
}
}
/**
* Performs less than or equal range check.
*/
class ProxyLEComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) <= 0 );
}
}
}
/**
* Performs equal range check.
*/
class ProxyEQComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
return false;
}
else
{
return ( ivAttribute.compareTo( attribute ) == 0 );
}
}
}
/**
* Performs greater than range check.
*/
class ProxyGTComparator extends ProxyComparator
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) > 0 );
}
}
}
/**
* Performs greater than or equal range check.
*/
class ProxyGEComparator extends ProxyComparator
```

```
{
Comparable ivAttribute;
boolean inRange(Object attribute)
{
if ( attribute == null )
{
return false;
}
else
{
Comparable attr = (Comparable) attribute;
return ( attr.compareTo( ivAttribute ) >= 0 );
}
}
}
/**
* Performs lowAttribute <= attribute < highAttribute range check.
*/
class ProxyRangeComparator extends ProxyComparator
{
Comparable ivLowAttribute;
Comparable ivHighAttribute;
boolean inRange(Object o)
{
if ( o == null )
{
return false;
}
Comparable attribute = (Comparable) o;
if ( attribute.compareTo( ivLowAttribute ) < 0 )
{
return false; // attribute < ivLowAttribute
}
else
{
// ivLowAttribute <= attribute
if ( attribute.compareTo( ivHighAttribute ) < 0 )
{
return true; // ivLowAttribute <= attribute < ivHighAttribute
}
else
{
return false; // attribute >= ivHighAttribute
}
}
}
}
```

doBatchUpdate および undoBatchUpdate メソッド

doBatchUpdate および undoBatchUpdate メソッドは、MapIndexPlugin インターフェースの重要なメソッドです。 doBatchUpdate メソッドは、トランザクションのコミット・サイクル中にマップに適用された変更の結果として呼び出されます。
undoBatchUpdate メソッドは、doBatchUpdate メソッドへの前の呼び出しの結果として索引に対して行われた変更を取り消す場合に使用されます。このメソッドは、トランザクションで行われた変更をすべてロールバックすることが必要になるような例外やエラー状態が発生したときに呼び出されます。これら両方のメソッドに、この Map の現在の TxID および変更リストが与えられます。すべての変更についてこれらのメソッドをそれぞれ反復し、これらのメソッドで変更を処理する必要があります。

これら 2 つのメソッドおよびサポート・メソッドを実装する方法を、以下のコード例で示します。

```
/**
* The synchronized Map used as the index implementation where
* the attribute value object is the key and a Java Set is the value.
* A Set member is the key of a BackingMap entry that matches attribute value.
*/
Map index; //<Object attribute, Set keys>
public void doBatchUpdate(TxID txid, LogSequence sequence)
throws ObjectGridRuntimeException
{
Iterator iter = sequence.getAllChanges();
while ( iter.hasNext() )
{
LogElement elem = (LogElement) iter.next();
Object key = elem.getCacheEntry().getKey();
LogElement.Type doType = elem.getType();
if ( doType == LogElement.INSERT )
{
Object newAttribute = getAttribute( elem.getAfterImage() );
insertIntoIndex( key, newAttribute );
}
else if ( doType == LogElement.UPDATE )
{
Object newAttribute = getAttribute( elem.getAfterImage() );
Object oldAttribute = getAttribute( elem.getBeforeImage() );
updateIndex( key, oldAttribute, newAttribute );
}
else if ( doType == LogElement.DELETE )
{
Object oldAttribute = getAttribute( elem.getBeforeImage() );
removeFromIndex( key, oldAttribute );
}
else if ( doType == LogElement.EVICT )
{
Object beforeImage = elem.getBeforeImage();
if ( beforeImage != null )
{
Object oldAttribute = getAttribute( beforeImage );
removeFromIndex( key, oldAttribute );
}
}
}
}
public void undoBatchUpdate(TxID txid, LogSequence sequence)
throws ObjectGridException
{
int errors = 0;
Iterator iter = sequence.getAllChanges();
while ( iter.hasNext() )
{
try
{
LogElement elem = (LogElement) iter.next();
Object key = elem.getCacheEntry().getKey();
LogElement.Type undoType = elem.getUndoType();
if ( undoType == LogElement.INSERT )
{
Object newAttribute = getAttribute( elem.getBeforeImage() );
insertIntoIndex( key, newAttribute );
}
else if ( undoType == LogElement.UPDATE )
{
Object oldAttribute = getAttribute( elem.getAfterImage() );
Object newAttribute = getAttribute( elem.getBeforeImage() );
updateIndex( key, oldAttribute, newAttribute );
```

```
}
else if ( undoType == LogElement.DELETE )
{
Object oldAttribute = getAttribute( elem.getAfterImage() );
removeFromIndex( key, oldAttribute );
}
}
catch ( Throwable t )
{
++errors;
}
}
if ( errors > 0 )
{
throw new ObjectGridException( errors
+ " exceptions occurred during rollback of index changes.");
}
}
/**
* Extracts the attribute from a specified value Object.
*
* @param value The value Object.
*
* @return attribute from the value Object, which may be a null reference.
*
* @throws ObjectGridRuntimeException is thrown if any exception occurs
* attempting to extract the attribute value from the value Object.
*/
private Object getAttribute(Object value) throws ObjectGridRuntimeException
{
try
{
Object attribute = null;
if ( value != null )
{
Method m = getAttributeMethod( value );
attribute = getAttributeMethod.invoke( value, emptyArray );
}
return attribute;
}
catch ( InvocationTargetException e )
{
Throwable t = e.getTargetException();
throw new ObjectGridRuntimeException( "Caught unexpected Throwable", t );
}
catch ( Throwable t )
{
throw new ObjectGridRuntimeException( "Caught unexpected Throwable", t );
}
}
private void updateIndex(Object key, Object oldAttribute, Object newAttribute)
{
// Was attributed changed by the update?
if ( newAttribute != null && oldAttribute != null &&
oldAttribute.equals( newAttribute ) )
{
// Nope, then nothing needs to be changed in index.
return;
}
// Unless we restrict Loader to only access tables with non-nullable columns,
// we have to handle the possibility that the attribute is null.
Set oldKeys = null;
if ( oldAttribute != null )
{
// Remove oldAttribute from index entry.
oldKeys = (Set) index.get( oldAttribute );
if ( oldKeys != null )
```

```
{
oldKeys.remove( key );
if ( oldKeys.isEmpty() )
{
index.remove( oldAttribute );
}
}
}
// Unless we restrict Loader to only access tables with non-nullable columns,
// we have to handle the possibility that the attribute is null.
Set keys = null;
if ( newAttribute != null )
{
keys = (Set) index.get( newAttribute );
// Add newAttribute to index.
if ( keys == null )
{
// Since different transactions can be updating different BackingMap
// entries and multiple map entries can have same attribute value,
// we need to use a synchronized Set object to ensure only
// one transaction at a time can make changes to the Set.
keys = Collections.synchronizedSet( new HashSet() );
index.put( newAttribute, keys );
}
// Add key for this map entry to the Set of keys for the new attribute value.
keys.add( key );
}
}
private void insertIntoIndex( Object key, Object newAttribute )
{
// Unless we restrict Loader to only access tables with non-nullable columns,
// we have to handle the possibility that the attribute is null.
if ( newAttribute != null )
{
Set keys = (Set) index.get( newAttribute );
if ( keys == null )
{
// Since different transactions can be updating different
// Map entries and multiple map entries can have same attribute
// value, we need to use a synchronized Set object to ensure only
// one transaction at a time can make changes to the Set.
keys = Collections.synchronizedSet( new HashSet() );
index.put( newAttribute, keys );
}
// Add key for this map entry to the Set of keys for the new attribute value.
keys.add( key );
}
}
private void removeFromIndex(Object key, Object oldAttribute )
{
// Extract the old attribute value
Set oldKeys = null;
// Unless we restrict Loader to only access tables with non-nullable columns,
// we have to handle the possibility that the attribute is null.
if ( oldAttribute != null )
{
oldKeys = (Set) index.get( oldAttribute );
if ( oldKeys != null )
{
oldKeys.remove( key );
if ( oldKeys.isEmpty() )
{
index.remove( oldAttribute );
}
}
}
}
```

アプリケーション索引インターフェース

アプリケーション索引インターフェースは、照会メソッドをサポートするよう設計されています。現在のところ、MapIndex と MapRangeIndex の 2 つのアプリケーション索引インターフェースが定義されています。

MapIndex

MapIndex は、オブジェクトを属性値で検索するための単純な索引です。この索引を使用すると、Map の任意の属性値に索引を付けることができます。これにより、アプリケーションは、特定の属性値を持つ、Map 内のすべてのオブジェクトを素早く見つけることができます。MapIndex インターフェースの定義を以下に示します。

```
/**
* This is an abstract index that can be created on an empty Map. The
* index can be used to perform efficient look ups and possibly other
* operations such as relational operations on an attribute in a Map.
* The MapIndex is provided with all update events and maintains an
* index that can be used to issue simple queries against the index
* later. The index could use an index defined callback to make an
* index on composite attributes.
*/
public interface MapIndex
{
/**
* Returns the Keys for the entries that have the specified attribute
* value.
*
* @param attributeValue
* a non-null reference to the attribute value to search for.
*
* @return A list of the keys for the entries with that attribute.
*
* @throws IllegalArgumentException if attributeValue argument is null.
* @throws FinderException is thrown if exception or retry limit is
* reached when concurrent transactions updating the index
* prevent findAll from completing.
*/
Iterator findAll(Object attributeValue) throws FinderException;
}
```

MapRangeIndex

MapRangeIndex は、特定の範囲内の属性値を持つオブジェクトを検索するための単純な索引です。この索引を使用すると、Map の任意の属性値に索引を付けることができます。MapIndex との違いは、この索引を使用すると、値範囲および値比較演算を使用して照会を行うことができる点です。これにより、特定の値よりも小さい、または大きい属性値を持つすべてのオブジェクトを照会によって見つけることができます。MapRangeIndex インターフェースの定義を以下に示します。

```
/**
* This is an index that allows comparison type searches.
*/
public interface MapRangeIndex extends MapIndex
{
/**
* This find all keys with entries with an attribute greater than the
* specified value.
*
* @param attributeValue is the low endpoint of range excluding the
* low attribute value.
*
```

```
* @return The set of keys with values greater than the attribute.
*
* @throws IllegalArgumentException if attributeValue argument is null.
* @throws FinderException is thrown if exception or retry
* limit is reached
* when concurrent transactions updating the index prevent
* findAll from completing.
*/
Iterator findGreater(Object attributeValue) throws FinderException;
/**
* This find all keys with entries with an attribute greater or equal
* to the specified value.
*
* @param attributeValue is the low endpoint of range including the
* low attribute value.
*
* @return The set of keys with attributes meeting the criteria
*
* @throws IllegalArgumentException if attributeValue argument is null.
* @throws FinderException is thrown if exception or retry
* limit is reached
* when concurrent transactions updating the index prevent findAll
* from completing.
*/
Iterator findGreaterEqual(Object attributeValue) throws FinderException;
/**
* This find all keys with entries with an attribute less than the
* specified value.
*
* @param attributeValue is the high endpoint of range excluding high
* endpoint value.
*
* @return The set of keys with attributes meeting the criteria
*
* @throws IllegalArgumentException if attributeValue argument is null.
* @throws FinderException is thrown if exception or retry limit
* is reached
* when concurrent transactions updating the index prevent
* findAll from completing.
*/
Iterator findLess(Object attributeValue) throws FinderException;
/**
* This find all keys with entries with an attribute less than or equal
* to the specified value.
*
* @param attributeValue is the high endpoint of range including high
* endpoint value.
*
* @return The set of keys with attributes meeting the criteria
*
* @throws IllegalArgumentException if attributeValue argument is null.
* @throws FinderException is thrown if exception or retry limit
* is reached
* when concurrent transactions updating the index prevent
* findAll from completing.
*/
Iterator findLessEqual(Object attributeValue) throws FinderException;
/**
* This returns all keys for the entries with the attribute inclusively
* within the specified range such that lowAttributeValue <= attribute
* < highAattributeValue.
*
* @param lowAttributeValue is the low endpoint of range including the
* low attribute value.
* @param highAttributeValue is the high endpoint of range excluding
* high attribute value.
*
```

```
* @return The list of keys with entries in that range, in ascending order.
*
* @throws IllegalArgumentException if either lowAttributeValue or
* highAttributeValue
* argument is null or lowAttributeValue > highAttributeValue.
* @throws FinderException is thrown if exception or retry limit is reached
* when concurrent transactions updating the index prevent
* findAll from completing.
*/
Iterator findRange(Object lowAttributeValue, Object highAttributeValue)
throws FinderException;
}
```

アプリケーションでは、ObjectMap インスタンスの getIndex メソッドから取得した索引オブジェクトを、望ましいアプリケーション索引インターフェースにキャストする必要があります。索引プラグインが MapRangeIndex インターフェースをサポートするように設計されている場合は、索引オブジェクトを MapRangeIndex 型にキャストできます。そのように設計されていない場合は、MapIndex 型にキャストする必要があります。

カスタマイズしたアプリケーション索引インターフェースを定義することができます。カスタム・アプリケーション索引は、MapIndexPlugin の getIndexProxy メソッドによって戻すことのできる索引プロキシー・オブジェクトとして実装します。ObjectMap インスタンスの getIndex メソッドから取得した索引オブジェクトを、このカスタマイズしたアプリケーション索引インターフェースにキャストし、それを使用してください。

静的索引プラグインの追加

静的索引プラグインを BackingMap 構成に追加するには、XML 構成およびプログラマチック構成という 2 つのアプローチがあります。以下に、XML 構成アプローチの例を示します。

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

BackingMap インターフェースには、静的索引プラグインを追加するために使用可能なメソッドとして、addMapIndexPlugin および setMapIndexPlugins という 2 つのメソッドがあります。これら 2 つのメソッドの定義を以下に示します。

```
/**
* This method adds an index plugin to this Map. We assume the index implementation
* was constructed
* with the name of the attribute to index. The name of the index is specified when
* the index is constructed.
*
* Note, to avoid an {@link IllegalStateException}, this method must be called
* prior to {@link ObjectGrid#initialize()} method. Also, keep in mind that the
* {@link ObjectGrid#getSession()} method implicitly calls the
* {@link ObjectGrid#initialize()} method if it has yet to be called by the
* application.*
```

```
* @param index The index implementation.
*
* @throws IndexAlreadyDefinedException This index already exists.
* @throws IllegalStateException if this method is called after the
* {@link ObjectGrid#initialize()} method is called.
*/
public void addMapIndexPlugin(MapIndexPlugin index)
throws IndexAlreadyDefinedException;
/**
* This method sets the list of MapIndexPlugin objects for this BackingMap.
* If the BackingMap already has a List of MapIndexPlugin objects,
* that list is replaced by the List passed as
* an argument to the current invocation of this method.
*
* Note, to avoid an {@link IllegalStateException}, this method must be called
* prior to {@link ObjectGrid#initialize()} method. Also, keep in mind that the
* {@link ObjectGrid#getSession()} method implicitly calls the
* {@link ObjectGrid#initialize()} method if it has yet to be called by the
* application.*
* @param indexList A non-null reference to a List of {@link MapIndexPlugin}
* objects.
*
* @throws IllegalArgumentException is thrown if indexList is null
* or the indexList contains an object that is not
* an instanceof {@link MapIndexPlugin}.
*/
public void setMapIndexPlugins(List indexList );
```

以下のコードの断片は、プログラマチック構成アプローチを示したものです。

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid ivObjectGrid = ogManager.createObjectGrid( "grid" );
BackingMap personBackingMap = ivObjectGrid.getMap("person");
//use the builtin HashIndex class as the index plugin class.
HashIndex mapIndexPlugin = new HashIndex();
mapIndexPlugin.setName("CODE");
mapIndexPlugin.setAttributeName("EmployeeCode");
mapIndexPlugin.setRangeIndex(true);
personBackingMap.addMapIndexPlugin(mapIndexPlugin);
```

静的索引の使用

静的索引プラグインが BackingMap 構成に追加され、索引を含む ObjectGrid インスタンスが初期化されていると、アプリケーションは、BackingMap の ObjectMap インスタンスから索引オブジェクトを名前で取得することができます。索引オブジェクトは、アプリケーション索引インターフェースにキャストします。これで、アプリケーション索引インターフェースがサポートしている索引操作を実行できます。ObjectMap インターフェースの getIndex メソッドの定義を以下に示します。

```
/**
* This returns a reference to the named index that can be used with this Map.
* This index cannot be shared between threads and works on the same rules as
* Session. The returned value should be cast to the right index interface
* such as MapIndex or MapRangeIndex or a custom index interface such as geo
* spatial index.
*
* @param name The index name
*
* @return A reference to the index, it must be cast to the appropriate index
* interface.
*
```

```
* @throws IndexUndefinedException if the index is not defined on the BackingMap
* @throws IndexNotReadyException if the index is not ready
* @throws UnsupportedOperationException if the map is a distributed map
*/
Object getIndex(String name)
throws IndexUndefinedException, IndexNotReadyException,
UnsupportedOperationException;
```

次のコードの断片に、静的索引を取得して使用する方法を示します。

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

動的索引の追加および除去

動的索引は、いつでも、プログラマチックに BackingMap インスタンスに作成し、そこから除去できます。動的索引と静的索引の違いは、動的索引は、索引を含む ObjectGrid インスタンスが初期化されたあとでも作成できる、という点です。静的索引付けとは異なり、動的索引付けは非同期プロセスであり、その目的に供するために、事前に作動可能状態になっている必要があります。動的索引の取得方法および使用方法は、静的索引と同じです。動的索引が不要になった場合は、除去できます。BackingMap インターフェースには、動的索引を作成および除去するためのメソッドがあります。以下に、これらのメソッドの定義を示します。

```
/**
* Create a dynamic index on the BackingMap
*
* @param name the name of the index. The name can not be null.
* @param isRangeIndex Indicate whether to create a MapRangeIndex or a MapIndex.
* If set to true, the index will be type of MapRangeIndex.
* @param attributeName The name of the attribute to be indexed.
* The attributeName can not be null.
* @param dynamicIndexCallback The callback that will invoke upon dynamic
* index events.
* The dynamicIndexCallback is optional and can be null.
*
* @throws IndexAlreadyDefinedException if a MapIndexPlugin with the specified
* name already exists.
* @throws UnsupportedOperationException if the map is a distributed map.
*
*/
public void createDynamicIndex(String name, boolean isRangeIndex,
String attributeName, DynamicIndexCallback cb)
throws IndexAlreadyDefinedException, UnsupportedOperationException;
/**
* Create a dynamic index on the BackingMap.
*
* @param index The index implementation. The index can not be null.
* @param dynamicIndexCallback The callback that will invoke upon dynamic
* index events.
* The dynamicIndexCallback is optional and can be null.
*
* @throws IndexAlreadyDefinedException if a MapIndexPlugin with the
* specified name already exists.
* @throws UnsupportedOperationException if the map is a distributed map.
*/
public void createDynamicIndex(MapIndexPlugin index, DynamicIndexCallback
dynamicIndexCallback)
```

```
throws IndexAlreadyDefinedException, UnsupportedOperationException;
/**
* remove a dynamic index from the BackingMap
*
* @param name the name of the index. The name can not be null.
*
* @throws IndexUndefinedException if a MapIndexPlugin with the specified name
* does not exists.
* @throws OperationNotSupportedException if the map is a distributed map.
*/
public void removeDynamicIndex(String name) throws IndexUndefinedException;
```

以下のコードの断片に、プログラマチック・アプローチにより動的索引を作成、使用、および除去する方法を示します。

```
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
ObjectGridManager ogManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.getMap("person");
og.initialize();
//create index after ObjectGrid initialization without DynamicIndexCallback.
bm.createDynamicIndex("CODE", true, "employeeCode", null);
try {
//If not using DynamicIndexCallback, need to wait for the Index to be ready.
//The waiting time depends on the current size of the map
Thread.sleep(3000);
} catch (Throwable t) {
//...
}
//Once the index is ready, applications can try to get application index
//interface instance.
//Applications have to find a way to ensure the index is ready to use,
//if not using DynamicIndexCallback interface.
//The following demonstrates the way to wait for the index to be ready
//The total waiting time should consider the size of the map
Session session = og.getSession();
ObjectMap m = session.getMap("person");
MapRangeIndex codeIndex = null;
int counter = 0;
int maxCounter = 10;
boolean ready = false;
while(!ready && counter < maxCounter){
try {
counter++;
codeIndex = (MapRangeIndex) m.getIndex("CODE");
ready = true;
} catch (IndexNotReadyException e) {
//implies index is not ready, ...
System.out.println("Index is not ready. continue to wait.");
try {
Thread.sleep(3000);
} catch (Throwable tt) {
//...
}
} catch (Throwable t) {
//unexpected exception
t.printStackTrace();
}
}
if(!ready){
System.out.println("Index is not ready. Need to handle this situation.");
}
//use the index to peform queries
```

```
//Refer to MapIndex or MapRangeIndex interface for supported operations.
//The object attribute that the index created on is the EmployeeCode.
//Assuming the EmployeeCode attribute is Integer type, which should be
//the data type of the parameter passed into index operations.
Iterator iter = codeIndex.findLessEqual(new Integer(15));
//remove the dynamic index when no longer needed
bm.removeDynamicIndex("CODE");
```

DynamicIndexCallback インターフェース

DynamicIndexCallback インターフェースは、作動可能、エラー、または破棄という索引付けイベントの発生時に、そのことを通知してもらう必要のあるアプリケーションのために設計されています。このインターフェースは、BackingMap の createDynamicIndex() メソッドのオプション・パラメーターです。アプリケーションは、索引付けイベントの通知を受け取ると、登録済みの DynamicIndexCallback インスタンスを使用して、ビジネス・ロジックを実行することができます。例えば、作動可能イベントは、索引を使用する準備が整ったことを意味します。アプリケーションは、このイベントの通知を受け取ると、アプリケーション索引インターフェースのインスタンスを取得して、それを使用してみることができます。DynamicIndexCallback インターフェースの定義を以下に示します。

```
/**
* This is the callback interface for dynamic indexing process.
* If applications wish to get notification at the event of ready, error,
* or destroy, they can implement this callback interface and register with
* the dynamic indexing process when creating dynamic index.
*
*/
public interface DynamicIndexCallback {
/**
* This callback method will be invoked when the dynamic index is ready.
*
* @param indexName
* The index name
*
*/
public void ready(String indexName);
/**
* Invoked when the dynamic indexing process encounters an unexpected error.
*
* @param indexName the index name
* @param t A Throwable object that causes the error situation in dynamic
* indexing process.
*/
public void error(String indexName, Throwable t);
/**
* This callback method will be invoked when the dynamic index is removed
*
* @param indexName
* The index name
*/
public void destroy(String indexName);
}
```

以下のコード断片は、DynamicIndexCallback インターフェースの使い方を示したものです。

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

```
public void ready(String indexName) {
System.out.println("DynamicIndexCallbackImpl.ready() -> indexName = " +
indexName);
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
Session session = og.getSession();
ObjectMap map = session.getMap("person");
MapIndex codeIndex = (MapIndex) map.getIndex("CODE");
Iterator iter = codeIndex.findAll(codeValue);
}
public void error(String indexName, Throwable t) {
System.out.println("DynamicIndexCallbackImpl.error() ->
indexName = " + indexName);
t.printStackTrace();
}
public void destroy(String indexName) {
System.out.println("DynamicIndexCallbackImpl.destroy() -> indexName = " +
indexName);
}
}
```

パフォーマンスの考慮事項

索引付けフィーチャーの主な目的の 1 つは、BackingMap の全体的なパフォーマンスを改善することですが、このフィーチャーを使用するにあたっては、あらかじめ考慮しておく必要のある要因がいくつかあります。索引付けの使い方が不適切な場合は、アプリケーションのパフォーマンスが低下する可能性があります。

- 同時書き込みトランザクションの数。索引処理は、トランザクションが BackingMap にデータを書き込むたびに発生します。アプリケーションが索引照会操作を試行しているときに、多くのトランザクションがデータをマップに書き込んでいると、パフォーマンスが低下します。
- 照会操作によって戻された結果セットのサイズ。結果セットのサイズが大きくなると、照会のパフォーマンスが低下します。ある実験では、結果セットのサイズが BackingMap の 15% 以上の場合、パフォーマンスが低下するという結果が得られています。
- 同一の BackingMap に作成された索引の数。各索引は、システム・リソースを消費します。BackingMap に作成される索引の数が増えると、パフォーマンスが低下します。

結論としては、索引付け機能は、特定の環境下では BackingMap のパフォーマンスを劇的に改善する可能性があります。索引付けの理想的な事例は、BackingMap の動作のほとんどが読み取りの場合、照会の結果セットが、BackingMap エントリー数の内のわずかな割合しか占めていない場合、そして BackingMap に作成される索引がほんのわずかしかない場合です。

ObjectGrid 構成

ObjectGrid は、分散環境で実行するように構成するか、または単一の JVM 内でのみ使用可能なローカル・キャッシュとして実行するよう構成できます。ローカル ObjectGrid は、プログラマチックに構成するか、または ObjectGrid XML ファイルを用いて構成することができます。ObjectGrid XML ファイルは、ObjectGrids、BackingMaps、およびそれぞれのプラグインを定義する場所です。

ローカル ObjectGrid は、分散環境にマイグレーションできます。分散 ObjectGrid を構成するには、ObjectGrid XML とともにクラスター XML を提供する必要があります。クラスター XML ファイルでは、ObjectGrid トポロジー内のサーバーを定義し、また ObjectGrid データを区画に分割し、サーバーにわたってそのデータを複製する方法を定義します。このセクションでは、ローカル ObjectGrid および分散 ObjectGrid の構成方法について説明します。

ローカル ObjectGrid 構成

ローカル ObjectGrid の構成については、『ローカル ObjectGrid 構成』を参照してください。

分散 ObjectGrid

分散 ObjectGrid の構成については、285 ページの『分散 ObjectGrid 構成』を参照してください。

ローカル ObjectGrid 構成

ローカル ObjectGrid は、プログラマチックに構成することも、XML で構成することもできます。 ObjectGridManager は、これら両方の構成手段のエントリー・ポイントです。

ObjectGridManager には、ローカル ObjectGrid の作成に使用できるメソッドがいくつかあります。各メソッドの詳しい説明については、95 ページの『ObjectGridManager インターフェース』を参照してください。

ローカル ObjectGrid を構成する場合は、以下のトピックを使用してください。

- 『基本の ObjectGrid 構成』では、1 つの ObjectGrid と 1 つの BackingMap を定義している非常に単純な XML ファイルを作成する方法について解説しています。
- 272 ページの『完全な ObjectGrid 構成』では、XML ファイルの各エレメントと属性を定義し、XML ファイルと同じ結果をプログラマチックに達成する方法について解説しています。
- 284 ページの『混合モードの ObjectGrid 構成』では、 XML による構成方法とプログラマチックな構成方法を組み合わせて使用する方法について解説しています。

基本の ObjectGrid 構成

このトピックでは、1 つの ObjectGrid と 1 つの BackingMap を定義した、ごく単純な ObjectGrid XML ファイルである `bookstore.xml` ファイルを作成する方法について説明します。

ファイルの最初の数行は、各 ObjectGrid XML ファイルの必須のヘッダーです。以下の XML では、*book* という BackingMap を持つ *bookstore* という ObjectGrid を定義しています。

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>
```

この XML ファイルは ObjectGridManager インターフェースに送られ、このファイルに基づいて ObjectGrid インスタンスが作成されます。以下のコードの断片では、bookstore.xml ファイルの妥当性検査を XML スキーマと照合して行い、*bookstore* ObjectGrid を作成します。新規に作成された ObjectGrid インスタンスはキャッシュされません。

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

以下のコードは XML を使用しないで同じタスクを実行します。このコードを使用して、ObjectGrid に BackingMap をプログラマチックに定義します。このコードは、bookstoreGrid ObjectGrid 上に book BackingMap を作成します。

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid
("bookstore", false);
BackingMap bookMap = bookstoreGrid.defineMap("book");
```

完全な ObjectGrid 構成

このトピックは、ObjectGrid の構成に関する完全なガイドです。 XML ファイルの各エレメントおよび属性について定義されています。 XML ファイルの例が、同じタスクをプログラマチックに実行するコードと共に提示されています。

以下の XML ファイル、bookstore.xml、は、このトピック全体を通して参照されます。このファイルのエレメントと属性については、以下の例に続いて詳しく説明されています。

bookstore.xml ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<bean id="ObjectGridEventListener"
className="com.company.organization.MyObjectGridEventListener" />
<backingMap name="books" pluginCollectionRef="collection1" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="collection1">
<bean id="Evictor"
classname="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
<property name="maxSize" type="int" value="321" />
</bean>
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

objectGridConfig エレメント

出現回数: 1 回

子エレメント: objectGrids エレメントおよび backingMapPluginCollections エレメント

objectGridConfig エレメントは、XML ファイルの最上位レベルのエレメントです。前の例で示されているように、XML 文書に書き込む必要があります。このエレメントは、ファイルのネーム・スペースとスキーマ・ロケーションをセットアップします。スキーマは objectGrid.xsd ファイルで定義されます。 ObjectGrid は、ObjectGrid Java archive (JAR) ファイルのルート・ディレクトリー内でこのファイルを探します。

objectGrids エレメント

出現回数: 1 回

子エレメント: objectGrid エレメント

objectGrids エレメントは、すべての objectGrid エレメントのコンテナーです。sample1.xml ファイル内で、objectGrids エレメントは bookstore という名前の 1 つの objectGrid を含んでいます。

objectGrid エレメント

出現回数:1 回から多数回まで

子エレメント: Bean エレメントおよび backingMap エレメント

objectGrid エレメントを使用して、XML ファイルに ObjectGrid を定義します。objectGrid エレメント上の各属性は、ObjectGrid インターフェース上のメソッドに対応します。

```
<objectGrid
(1) name="objectGridName"
(2) securityEnabled="true|false"
(3) authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS|
AUTHORIZATION_MECHANISM_CUSTOM"
(4) permissionCheckPeriod="permission check period"
(5) txTimeout="seconds"
/>
```

属性:

1. **name** 属性 (必須): ObjectGrid に割り当てられる名前を指定します。この属性が欠落している場合、XML 妥当性検査は失敗します。
2. **securityEnabled** 属性 (オプション、デフォルトでは false): この属性を true に設定すると、ObjectGrid のセキュリティーが使用可能になります。 ObjectGrid レベルでセキュリティーを使用可能にするということは、マップ内のデータへのアクセス許可を使用可能にすることです。デフォルトでは、セキュリティーは使用不可です。
3. **authorizationMechanism** 属性 (オプション、デフォルトでは AUTHORIZATION_MECAHNISM_JAAS): この ObjectGrid の許可メカニズムを設定します。この属性は、AUTHORIZATION_MECHANISM_JAAS または

AUTHORIZATION_MECHANISM_CUSTOM の 2 つのどちらかに設定できます。カスタム MapAuthorization プラグインを使用する場合、AUTHORIZATION_MECHANISM_CUSTOM に設定します。この設定は、securityEnabled 属性が true に設定されている場合に有効になります。

4. **permissionCheckPeriod** (オプション、デフォルトでは 0): クライアント・アクセスを許可するために使用されるアクセス権の検査頻度を、秒単位の整数値で指定します。この属性値が 0 であれば、すべての get、put、update、remove、または evict メソッドが asks 許可メカニズム、JAAS 権限またはカスタム権限のいずれかを呼び出し、現行サブジェクトがアクセス権を持っているかどうかを検査します。 0 より大きな値は、アクセス権のセットを、更新のために許可メカニズムへ戻す前に、キャッシュに入れる秒数を示します。この設定は、securityEnabled 属性が true に設定されている場合に有効になります。詳しくは、144 ページの『ObjectGrid セキュリティー』を参照してください。
5. **txTimeout** (オプション。デフォルトは 0): トランザクションを完了するのに許されている秒数。トランザクションがこの時間内に完了しなかった場合、そのトランザクションはロールバック対象としてマークされ、TransactionTimeoutException 例外が発生します。この値を 0 に設定した場合は、トランザクションがタイムアウトになることはありません。

以下の XML ファイルの例である bookstoreObjectGridAttr.xml ファイルは、objectGrid の属性を構成するための 1 つの方法を説明するものです。この例では、セキュリティーは使用可能に、許可メカニズムは JAAS に設定され、アクセス権検査期間は 45 秒に設定されています。

bookstoreObjectGridAttr.xml ファイル

```
<?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>
```

以下のコードは、前の例の bookstoreObjectGridAttr.xml ファイルと同じ構成を達成するプログラマチックな方法を示すものです。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory
.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore", false);
bookstoreGrid.setSecurityEnabled();
bookstoreGrid.setAuthorizationMechanism(
SecurityConstants.AUTHORIZATION_MECHANISM_JAAS);
bookstoreGrid.setPermissionCheckPeriod(45);
```

backingMap エレメント

出現回数: 0 回から多数回まで

子エレメント: なし

backingMap エレメントは、ObjectGrid 上の BackingMap を定義するために使用します。 backingMap エレメント上の各属性は、BackingMap インターフェース上のメソッドに対応します。

```
<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"
/>
```

属性:

1. **name** 属性 (必須): BackingMap に割り当てられる名前を指定します。この属性が欠落している場合、XML 妥当性検査は失敗します。
2. **readOnly** 属性 (オプション、デフォルトでは `false`): この属性を `true` に設定すると、BackingMap は読み取り専用になります。この属性を `false` に設定すると、BackingMap は読み取り/書き込みになります。値が指定されていない場合は、デフォルトである読み取り/書き込みになります。
3. **pluginCollectionRef** 属性 (オプション): backingMapPluginCollection プラグインへの参照を指定します。この属性の値は、backingMapCollection プラグインの ID 属性と一致しなければなりません。マッチング ID が存在しない場合、妥当性検査は失敗します。この参照は、BackingMap プラグインを簡単に再利用できる方法となるように設計されています。
4. **numberOfBuckets** 属性 (オプション、デフォルトでは `503`): BackingMap によって使用されるバケットの数。 BackingMap は、その実装のためにハッシュ・マップを使用します。 BackingMap 内に多くのエントリーが存在する場合は、バケットの数が増加するほど衝突のリスクが下がるので、バケットを増やすことによってパフォーマンスが良好になります。また、バケットが多いと並行性も高くなります。
5. **preloadMode** 属性 (オプション。デフォルトでは `false`): Loader プラグインがこの BackingMap に設定されている場合は、プリロード・モードを設定します。属性が `true` に設定されている場合は、Loader.preloadMap(Session, BackingMap) メソッドが非同期に起動されます。それ以外の場合は、プリロードが完了するまでキャッシュが使用不可になるように、データのロード中のメソッドの実行がブロックされます。プリロードは、ObjectGrid の初期化中に発生します。
6. **lockStrategy** 属性 (オプション、デフォルトでは `OPTIMISTIC`): BackingMap に関して使用される LockStrategy を設定します。ロック・ストラテジーは、トランザクションがマップ・エントリーにアクセスするたびに、内部 ObjectGrid ロ

ック・マネージャーを使用するかどうかを判別します。この属性は、OPTIMISTIC、PESSIMISTIC、または NONE の 3 つの値のいずれかに設定できます。

OPTIMISTIC は通常、Loader プラグインを持たないマップで、そのマップが大部分既読で、ロックが、サイド・キャッシュとして objectGrid を使用するパーシスタンス・マネージャーによっても、アプリケーションによっても提供されないマップの場合に使用します。オプティミスティック・ロック・ストラテジーの場合は、排他ロックが、コミット時間に挿入され、更新され、または除去されるマップ・エントリー上で獲得されます。ロックによって、コミットされたトランザクションがオプティミスティック・バージョン管理を実行している間に、バージョン情報が別のトランザクションによって変更できないことが保証されます。

PESSIMISTIC は通常、Loader プラグインを持たないマップで、ロックが、サイド・キャッシュとして ObjectGrid を使用するパーシスタンス・マネージャーによっても、Loader プラグインでも、アプリケーションによっても提供されないマップの場合に使用します。ペシミスティック・ロック・ストラテジーは、同じマップ・エントリー上で更新トランザクションが頻繁に衝突を起こすことが原因で、オプティミスティック方式の失敗回数が多すぎる場合に使用されます。オプティミスティック方式は、マップが大部分既読ではないか、多数のクライアントが小さなマップにアクセスする場合に失敗する可能性があります。

NONE は、並行性制御が ObjectGrid の外部で提供される (ObjectGrid をサイド・キャッシュとして使用するパーシスタンス・マネージャーによって、アプリケーションによって、またはデータベース・ロックを使用して並行性を制御する Loader プラグインによるかのいずれかで提供される) ので、内部ロック・マネージャーの使用が必要ないことを示します。

7. **numberOfLockBuckets** 属性 (オプション、デフォルトでは 383): この BackingMap に関してロック・マネージャーが使用するロック・バケットの数を設定します。 lockStrategy 属性が OPTIMISTIC または PESSIMISTIC に設定されている場合、BackingMap についてロック・マネージャーが作成されます。ロック・マネージャーは、ハッシュ・マップを使用して、1 つ以上のトランザクションによってロックされるエントリーを追跡します。多くのエントリーが存在する場合は、ロック・バケットが多い方がパフォーマンスが向上します。バケット数が増えるにつれて、衝突のリスクが低減するからです。またロック・バケットを増やすことが、並行性の増大につながります。 lockStrategy 属性が NONE に設定されると、この BackingMap はロック・マネージャーを使用しません。この場合は、numberOfLockBuckets 属性を設定する必要はありません。
8. **lockTimeout** 属性 (オプション、デフォルトでは 15): この BackingMap に関してロック・マネージャーが使用するロック・タイムアウトを設定します。lockStrategy 属性が OPTIMISTIC または PESSIMISTIC に設定されている場合、BackingMap についてロック・マネージャーが作成されます。デッドロック発生を回避するために、ロック・マネージャーには、ロックの認可を待機するためのデフォルトのタイムアウト値があります。このタイムアウト制限を超過すると、LockTimeoutException 例外が発生します。デフォルト値の 15 秒は、ほとんどのアプリケーションに対して適切ですが、負荷の過剰なシステム上では、デッドロックが存在しないときにタイムアウトが発生する可能性があります。

この場合、このメソッドを使用して、ロック・タイムアウト値をデフォルトより大きく設定し、 false タイムアウト例外の発生を回避します。ロック・ストラテジーが NONE である場合、この BackingMap はロック・マネージャーを使用しません。この場合は、lockTimeout 属性を設定する必要はありません。

9. **copyMode** 属性 (オプション、デフォルトでは COPY_ON_READ_AND_COMMIT): copyMode 属性は、BackingMap 内のエントリーの get 操作が実際の値、その値のコピー、またはその値のプロキシーを戻すかどうかを判別します。 copyMode 属性は、COPY_ON_READ_AND_COMMIT、COPY_ON_READ、COPY_ON_WRITE、または NO_COPY の 4 つの値のいずれかに設定できます。

   COPY_ON_READ_AND_COMMIT モードは、アプリケーションが BackingMap 内にある値オブジェクトへの参照を決して行わず、代わりに BackingMap 内にあるその値のコピーを使用して常時作業を行うことを保証します。

   COPY_ON_READ モードは、トランザクションがコミットされたときに発生するコピーを除去することによって、COPY_ON_READ_AND_COMMIT モード全体にわたるパフォーマンスを改善します。 BackingMap データの整合性を保持するために、トランザクションがコミットされた後で、アプリケーションはエントリーに対して持っているすべての参照を破棄します。このモードによって、ObjectMap.get メソッドは、値に対する参照の代わりにその値のコピーを戻し、トランザクションがコミットされるまで、アプリケーションによってその値に行われた変更が BackingMap 値に影響を与えないことを保証します。

   COPY_ON_WRITE モードは、指定したキーのトランザクションによって ObjectMap.get メソッドが初めて呼び出されるときに起こるコピーを排除することにより、COPY_ON_READ_AND_COMMIT モードを超えるパフォーマンスを実現します。その代わりに、ObjectMap.get メソッドは、値オブジェクトを直接参照するのではなく、その値にプロキシーを戻します。プロキシーは、アプリケーションが値インターフェース上に set メソッドを呼び出さない限りは、その値のコピーを作成しないことを保証します。

   NO_COPY モードによって、パフォーマンスの改善の代わりに、アプリケーションに ObjectMap.get メソッドを使用して獲得した値オブジェクトを決して変更させないことが可能になります。このモードを使用すると、値はコピーされません。

10. **valueInterfaceClassName** 属性 (オプション): copyMode 属性が COPY_ON_WRITE に設定されている場合、valueInterfaceClassName 属性が必要です。他のすべてのモードの場合は無視されます。書き込みのコピーは、ObjectMap.get メソッド呼び出しが行われるときに、プロキシーを使用します。プロキシーは、アプリケーションが valueInterfaceClassName 属性として指定されたクラス上に set メソッドを呼び出さない限りは、その値のコピーを作成しないことを保証します。

11. **copyKey** 属性 (オプション、デフォルトでは false): この属性は、マップ・エントリーが作成されたときに、キーをコピーする必要があるかどうかを判別します。キーのコピーによって、アプリケーションがそれぞれの ObjectMap 操作に対して同じキー・オブジェクトを使用することが可能になります。 true に設定すると、マップ・エントリーが作成されるときに、キー・オブジェクトをコピーします。

12. **nullValuesSupported** 属性 (オプション、デフォルトでは true): ヌル値をサポートする、つまりマップにヌル値を設定できます。 true に設定した場合、ヌル値は ObjectMap 内でサポートされます。それ以外の場合、ヌル値はサポートされません。ヌル値がサポートされている場合は、null を戻す get 操作は、値が NULL である、またはマップに渡されたキーが含まれていないことを意味すると考えられます。
13. **ttlEvictorType** 属性 (オプション、デフォルトは NONE): ttlEvictorType 属性は、BackingMap エントリーの有効期限の時間を算出する方法を判別します。この属性は、CREATION_TIME、LAST_ACCESS_TIME、または NONE の 3 つの値のいずれかに設定できます。

    NONE の場合、エントリーは、有効期限の時間を持たず、アプリケーションが明示的にこのエントリーを除去または無効化するまで、BackingMap 内に存続できます。

    CREATION_TIME の場合、エントリーの有効期限の時間は、このエントリーの作成時間と timeToLive 属性値の合計になります。

    LAST_ACCESS_TIME の場合、エントリーの有効期限の時間は、このエントリーの最終アクセス時間と timeToLive 属性値の合計になります。
14. **timeToLive** 属性 (オプション、デフォルトでは 0): 各マップ・エントリーの存続時間 (秒単位)。デフォルト値の 0 は、このマップ・エントリーが永久に存続するか、アプリケーションが明示的にこのエントリーを除去または無効化するまで存続することを意味します。この属性が 0 以外である場合、この値に基づいてマップ・エントリーを除去する TTL Evictor が使用されます。

以下の XML ファイル、bookstoreBackingMapAttr.xml ファイルは、backingMap 構成の例を示すものです。この例では、pluginCollectionRef 属性を除くすべてのオプション属性を使用しています。 pluginCollectionRef の使用法を表示する例については、 282 ページの『backingMapPluginCollection エレメント』を参照してください。

bookstoreBackingMapAttr.xml ファイル

```
<?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>
```

以下のサンプル・コードは、前の例の bookstoreBackingMapAttr.xml ファイルと同じ構成を達成するプログラマチックな方法を示すものです。

```
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); // set time to live to 50 minutes
```

Bean エレメント

出現回数 (objectGrid エレメント内): 0 回から多数回

出現回数 (backingMapPluginCollection エレメント内): 0 回から多数回

子エレメント: property エレメント

Bean エレメントを使用して、プラグインを定義します。プラグインは ObjectGrids と BackingMaps に接続できます。

ObjectGrid プラグインには、以下のようなものがあります。

- TransactionCallback プラグイン
- ObjectGridEventListener プラグイン
- SubjectSource プラグイン
- MapAuthorization プラグイン
- SubjectValidation プラグイン

BackingMap プラグインには、以下のようなものがあります。

- Loader プラグイン
- ObjectTransformer プラグイン
- OptimisticCallback プラグイン
- Evictor プラグイン
- MapEventListener プラグイン
- MapIndex プラグイン

Bean エレメント属性

```
<bean
(1) id="TransactionCallback|ObjectGridEventListener|SubjectSource|
MapAuthorization|SubjectValidation|Loader|ObjectTransformer|
OptimisticCallback|Evictor|MapEventListener|MapIndexPlugin"
(2) className="class name"
/>
```

1. **ID** 属性 (必須): 作成するプラグインのタイプを指定します。 objectGrid エレメントの子エレメントである Bean の場合、有効な値は `TransactionCallback`、`ObjectGridEventListener`、`SubjectSource`、`MapAuthorization`、および `SubjectValidation` プラグインです。 backingMapPluginCollection エレメントの子エレメントである Bean の場合、有効な値は

Loader、ObjectTransformer、OptimisticCallback、 Evictor、および MapEventListener プラグインです。 ID 属性のそれぞれの有効値は、インターフェースを表しています。

2. **className** 属性 (必須): インスタンスを生成するクラスの名前を指定して、プラグインを作成します。クラスはプラグイン・タイプのインターフェースを実装しなければなりません。

以下の bean.xml ファイルの例は、プラグインを構成するための Bean エレメントの使用法を説明しています。この XML ファイルでは、ObjectGridEventListener プラグインが bookstore ObjectGrid に追加されます。この Bean の className 属性は com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener クラスです。このクラスは、必要に応じて com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener インターフェースを実装します。

BackingMap プラグインも、以下の bookstoreBean.xml ファイルの例で定義されます。 evictor プラグインが book BackingMap に追加されます。 Bean ID が *Evictor* であるため、className 属性は com.ibm.websphere.objectgrid.plugins.Evictor インターフェースを実装するクラスを指定しなければなりません。 com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor クラスがこのインターフェースを実装します。 backingMap は pluginCollectionRef 属性を使用して、そのプラグインを参照します。 BackingMap にプラグインを追加する方法の詳細については、114 ページの『BackingMap インターフェース』を参照してください。

bookstoreBean.xml ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config">
<objectGrids>
<objectGrid name="bookstore">
<bean id="ObjectGridEventListener"
className="com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener" />
<backingMap name="book" pluginCollectionRef="bookPlugins" />
</objectGrid>
</objectGrids>
<backingMapPluginCollections>
<backingMapPluginCollection id="bookPlugins">
<bean id="Evictor"
classnName="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
</backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

以下のコードは、前の bookstoreBean.xml ファイルと同じ構成を達成するためのプログラマチックな方法を示すものです。

```
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);
```

property エレメント

出現回数: 0 回から多数回まで

子エレメント: なし

property エレメントは、プラグインにプロパティーを追加するために使用します。property の名前は、その property を含む Bean の className 属性上の set メソッドに対応しています。

property エレメント属性

```
<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. **name** 属性 (必須): プロパティーの名前を指定します。この属性に割り当てられる値は、Bean エレメントで className 属性として提供されるクラス上の set メソッドと対応している必要があります。例えば、Bean の className 属性が com.ibm.MyPlugin に設定され、提供されたプロパティーの名前が `size` である場合、com.ibm.MyPlugin クラスは setSize メソッドを持たなければなりません。
2. **type** 属性 (必須): プロパティーのタイプを指定します。 name 属性によって識別された set メソッドへ渡されるパラメーターのタイプです。有効な値は、Java プリミティブ、それに対応する `java.lang` プリミティブ、および `java.lang.String` です。 name 属性と type 属性は、Bean の className 属性のメソッド・シグニチャーに対応していなければなりません。例えば、名前が `size`、タイプが `int` である場合は、その Bean の className 属性として指定されたクラス上で、`setSize(int)` メソッドが既存でなければなりません。
3. **value** 属性 (必須): プロパティーの値を指定します。この値は type 属性によって指定されたタイプに変換され、次に name 属性と type 属性で識別された set メソッドへの呼び出しでパラメーターとして使用されます。この属性の値は、どんな方法でも妥当性検査されません。プラグイン・インプリメンターは、渡された値が有効であることを検証しなければなりません。インプリメンターは、パラメーターが有効でない場合、set メソッド内で `IllegalArgumentException` 例外を表示することができます。
4. **description** 属性 (オプション): この属性を使用して、プロパティーの説明を作成します。

以下の `bookstoreProperty.xml` ファイルは、Bean に property エレメントを追加する方法を示しています。この例で、名前 `maxSize` およびタイプ `int` を持つプロパティーが、Evictor に追加されます。
com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor Evictor は、 setMaxSize(int) メソッドと一致するメソッド・シグニチャーを持っています。整数値 `499` が com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor クラス上の setMaxSize(int) メソッドに渡されます。

bookstoreProperty.xml ファイル

```
<?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>
```

以下のコードは、bookstoreProperty.xml ファイルと同じ構成を達成するものです。

```
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();
// if the XML file were used instead,
// the property that was added would cause the following call to be made
lruEvictor.setMaxSize(449);
bookMap.setEvictor(lruEvictor);
```

backingMapPluginCollections エレメント

出現回数: 0 回から 1 回まで

子エレメント: backingMapPluginCollection エレメント

backingMapPluginCollections エレメントは、すべての backingMapPluginCollection エレメントのコンテナーです。 sample1.xml ファイルにおいて、backingMapPluginCollections エレメントは、ID collection1 を持つ 1 つの backingMapPluginCollection エレメントを含んでいます。

backingMapPluginCollection エレメント

出現回数: 0 回から多数回まで

子エレメント: Bean エレメント

backingMapPluginCollection エレメントは、BackingMap プラグインを定義します。各 backingMapPluginCollection エレメントは、その ID 属性によって識別されます。各 backingMap エレメントは、backingMap エレメント上で pluginCollectionRef 属性を使用してそのプラグインを参照しなければなりません。同じように構成しなければならないプラグインを持ついくつかの BackingMaps が存在する場合、そのそれぞれが同じ backingMapPluginCollection エレメントを参照することができます。

backingMapPluginCollection エレメント属性

```
<backingMapPluginCollection
(1) id="id"
/>
```

1. **ID** 属性 (必須): backingMapPluginCollection の ID。各 ID は固有でなければなりません。この ID は、backingMap エレメントの pluginCollectionRef 属性によって参照されます。 pluginCollectionRef 属性の値が、1 つの backingMapPluginCollection エレメントの ID と一致しない場合、XML 妥当性検査は失敗します。任意の数の backingMap エレメントが、単一の backingMapPluginCollection エレメントを参照できます。

以下の bookstoreCollection.xml ファイルは、backingMapPluginCollection エレメントの使用方法を示しています。このファイル内で 3 つの backingMap エレメントが定義されています。 book および customer BackingMaps はいずれも collection1 backingMapPluginCollection を使用します。これら 2 つの BackingMap は、それぞれ独自の LRUEvictor Evictor を持っています。 employee BackingMap は、collection2 backingMapPluginCollection を参照します。この BackingMap は、Evictor プラグインとしての LFUEvictor Evictor セットと、 OptimisticCallback プラグインとしての EmployeeOptimisticCallbackImpl クラス・セットを持っています。

bookstoreCollection.xml ファイル

```
<?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>
```

以下のコードは、bookstoreCollection.xml ファイルと同じ構成をプログラマチックに達成する方法を示すものです。

```
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);
```

混合モードの ObjectGrid 構成

ObjectGrid は、XML 構成と、プログラマチックな構成の組み合わせを使用して構成できます。

混合構成を実行するには、まず XML ファイルを作成し ObjectGridManager インターフェースに渡します。 ObjectGrid が XML ファイルに基づいて作成された、ObjectGrid.initialize() メソッドが呼び出されない限り、 ObjectGrid をプログラマチックに操作できます。アプリケーションが ObjectGrid.initialize() メソッドを呼び出さない場合は、 ObjectGrid.getSession() メソッドが暗黙的に呼び出します。

例

以下は、混合モードの構成を達成する方法の説明です。以下の mixedBookstore.xml ファイルが使用されます。

mixedBookstore.xml ファイル

```
<?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>
```

以下のコードの断片は、XML が ObjectGridManager に渡され、新規作成された ObjectGrid がさらに操作されることを示します。

```
ObjectGridManager objectGridManager =
ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid bookstoreGrid = objectGridManager.createObjectGrid("bookstore",
new URL("file:etc/test/document/mixedBookstore.xml"), true, false);
// at this point, we have the ObjectGrid that was defined in the XML
// now modify the BackingMap that was created and configured
BackingMap bookMap = bookstoreGrid.getMap("book");
// the XML set readOnly to true
// here the readOnly attribute is changed to false
bookMap.setReadOnly(false);
// the XML did not set nullValuesSupported, so
// it would default to true. Here the
// value is set to false
bookMap.setNullValuesSupported(false);
// get the Evictor that was set in the XML,
// and set its maxSize
```

```
LFUEvictor lfuEvictor = (LFUEvictor) bookMap.getEvictor();
lfuEvictor.setMaxSize(443);
bookstoreGrid.initialize();
// further configuration is not allowed
// to this ObjectGrid after the initialize call
```

分散 ObjectGrid 構成

分散 ObjectGrid を作成するには、ObjectGrid XML ファイルを使用してクラスター XML ファイルを作成し、ペアにしておく必要があります。

クラスター XML ファイルおよび ObjectGrid XML ファイルを使用することで、ObjectGrid サーバーを始動することができます。

クラスター XML ファイルを作成する前に、ローカル ObjectGrid と同様に ObjectGrid XML ファイルを作成します。ObjectGrid XML ファイルの構成方法について詳しくは、271 ページの『ローカル ObjectGrid 構成』を参照してください。 286 ページの『クラスター構成』の例では、以下の university.xml ファイルが ObjectGrid XML として使用されています。

university.xml ファイル

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

以下は、ObjectGrid サーバーを始動するために university.xml ファイルとともに使用される universityCluster.xml クラスター XML ファイルです。universityCluster.xml ファイルは、すべてのオプション XML 属性を外した、非常に基本的なクラスター XML ファイルです。

universityCluster.xml ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
</cluster>
<objectgridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
```

```
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

多数のオプション XML エレメントおよび属性の使用例は、『クラスター構成』セクションにあります。

クラスター構成

クラスター XML の各エレメントおよび属性については、このセクションで説明します。 ObjectGrid XML を持つクラスター XML を使用して構成を行う方法についても、例を示します。これらの例では、university.xml ファイルが ObjectGrid XML として使用されます。

clusterConfig エレメント

出現回数: 1 回

子エレメント: cluster エレメント、objectgridBinding エレメント、partitionSet エレメント、および replicationGroup エレメント

clusterConfig エレメントは、クラスター XML ファイルの最上位レベルのエレメントです。これは、universityCluster.xml ファイルで示されているように、ファイルの先頭に配置する必要があります。このエレメントは、ファイルのネーム・スペースとスキーマ・ロケーションをセットアップします。スキーマは objectGridCluster.xsd ファイルで定義されます。

cluster エレメント

出現回数: 1 回

子エレメント: serverDefinition エレメント、authenticator エレメント、および adminAuthorization エレメント

cluster エレメントは ObjectGrid クラスターの定義に使用されます。クラスター内の各サーバーは、cluster エレメントで定義されます。 cluster エレメントは、セキュリティーおよびネットワーク関連の属性の定義にも使用されます。

```
<cluster
(1) name="clusterName"
(2) securityEnabled="true|false"
(3) statisticsEnabled="true|false"
(4) statisticsSpec="statisticsSpecification"
(5) singleSignOnEnabled="true|false"
(6) loginSessionExpirationTime="seconds"
(7) adminAuthorizationEnabled="true|false"
(8) adminAuthorizationMechanism=""
(9) clientMaxRetries="numberOfRetries"
```

```
(10) clientMaxForwards="numberOfForwards"
(11) clientStartupRetries="numberOfRetries"
(12) clientRetryInterval="seconds"
(13) tcpConnectionTimeout="seconds"
(14) tcpMinConnections="numberOfConnections"
(15) tcpMaxConnections="numberOfConnections"
(16) tcpInactivityTimeout="seconds"
(17) tcpMaxWaitTime="seconds"
(18) peerHeartbeatInterval="seconds"
(19) peerTransportBufferSize="sizeInMBs"
(20) threadPoolMinSize="minThreads"
(21) threadPoolMaxSize="maxThreads"
(22) threadPoolInactivityTimeout="seconds"
(23) managementTimeout="seconds"
(24) threadsPerClientConnect="numberOfThreads"
/>
```

属性:

1. **name** 属性 (必須): クラスターに割り当てられる名前です。この属性が欠落している場合、XML 妥当性検査は失敗します。
2. **securityEnabled** 属性 (オプション、デフォルトでは **false**): true に設定すると、クラスターのセキュリティーが使用可能になります。false に設定すると、クラスター全体のセキュリティーが使用不可になります。詳しくは、 144 ページの『ObjectGrid セキュリティー』を参照してください。
3. **statisticsEnabled** 属性 (オプション、デフォルトでは **false**): true に設定すると、クラスターの統計が使用可能になります。統計が使用可能になると、statisticsSpec 属性を使用して統計の仕様が設定されます。
4. **statisticsSpec** 属性 (オプション): 統計の仕様の設定に使用されるストリングを指定します。このストリングが、収集する統計を決定します。
5. **singleSignOnEnabled** 属性 (オプション、デフォルトでは false): singleSignOnEnabled 属性を true に設定すると、クライアントは、サーバーのいずれか 1 つに対して認証された後、任意のサーバーに接続できるようになります。この属性を false に設定すると、クライアントは接続を許可される前に、それぞれのサーバーごとに認証を受けることが必要になります。
6. **loginSessionExpirationTime** 属性 (オプション): ログイン・セッションの有効時間 (秒単位)。ログイン・セッションが失効すると、クライアントは再度認証を受ける必要があります。
7. **adminAuthorizationEnabled** 属性 (オプション、デフォルトでは false): この値は管理許可を使用可能にするために使用されます。値が true の場合は、すべての管理用タスクに許可が必要です。使用される許可メカニズムは、adminAuthorizationMechanism 属性の値で指定されます。
8. **adminAuthorizationMechanism** 属性 (オプション、デフォルトでは AUTHORIZATION_MECHANISM_JAAS): この属性はどの許可メカニズムが使用されるかを示します。ObjectGrid は、Java 認証・承認サービス (JAAS) とカスタムの、2 つの許可メカニズムをサポートしています。JAAS 許可メカニズムは、標準の JAAS ポリシー・ベースのアプローチを使用します。許可メカニズムとして JAAS を指定するには、値に AUTHORIZATION_MECHANISM_JAAS を設定します。カスタム許可メカニズムは、ユーザー・プラグイン AdminAuthorization インプリメンテーションを使用します。カスタム許可メカニズムを指定するには、値に AUTHORIZATION_MECHANISM_CUSTOM を設定します。これら 2 つのメカ

ニズムがどのように使用されるかについての詳細は、 144 ページの『ObjectGrid セキュリティー』を参照してください。

9. **clientMaxRetries** 属性 (オプション、デフォルトでは 4): サービスが使用不可の場合に、サーバーに対する要求が自動的に再試行される最大回数。
10. **clientMaxForwards** 属性 (オプション、デフォルトでは 5): 失敗した要求が別のサーバーに転送される最大回数。
11. **clientStartupRetries** 属性 (オプション、デフォルトでは 8): サーバーの始動の完了を待つ間に要求が自動的に再試行される最大回数。 HA マネージャーは始動に非常に長い時間を要するため、この数値は十分に大きな値を設定してください。数値が十分に大きくないと、サーバーが完全に始動する前に実行依頼されたクライアント要求は失敗します。
12. **clientRetryInterval** 属性 (オプション、デフォルトでは 10): クライアントが再試行する時間間隔 (秒単位)。これは、clientMaxRetries 属性と clientStartupRetries 属性の両方に使用されます。
13. **tcpConnectionTimeout** 属性 (オプション、デフォルトでは 180): tcpConnectionTimeout 属性はソケット接続タイムアウトです。値は秒単位です。
14. **tcpMinConnections** 属性 (オプション、デフォルトでは 2): 接続のプールに対する接続の最小数。
15. **tcpMaxConnections** 属性 (オプション、デフォルトでは 20): 接続のプールに対する接続の最大数。
16. **tcpInactivityTimeout** 属性 (オプション、デフォルトでは無限大): パスする必要のある接続が、接続のプールから除去されるまでの非活動の接続の秒数。
17. **tcpMaxWaitTime** 属性 (オプション、デフォルトでは 120): すべての接続が使用中で、接続数が tcpMaxConnections 属性の値に達している場合に、システムが使用可能な接続を待機する最大秒数。
18. **peerHeartbeatInterval** 属性 (オプション、デフォルトでは 120): peerHeartbeatInterval 属性は、HA マネージャーによって使用されるハートビート間隔です。値は秒単位です。
19. **peerTransportBufferSize** 属性 (オプション、デフォルトでは 10): peerTransportBufferSize 属性は、HA マネージャーによって使用される移送メッセージ・バッファーのサイズを表します。この値はメガバイト単位で指定されます。
20. **threadPoolMinSize** 属性 (オプション、デフォルトでは 6): HA マネージャーのスレッド・プールの最小サイズを指定します。
21. **threadPoolMaxSize** 属性 (オプション、デフォルトでは 20): HA マネージャーのスレッド・プールの最大サイズを指定します。
22. **threadPoolInactivityTimeout** 属性 (オプション、デフォルトでは 6000): HA マネージャーのスレッド・プールのスレッド非活動タイムアウトを指定します。タイムアウト値は秒単位です。
23. **managementTimeout** 属性 (オプション、デフォルトでは 30): 一部の ObjectGrid MBean 機能は、サーバーから情報を収集するため、あるいはサーバー上で操作を実行するために、クラスター内のサーバーにメッセージを送信します。 managementTimeout 値は、クライアントがサーバーから戻されるメッセージの受け取りを試行する時間の長さを指示する値です。クライアントとサーバーの間に通信問題が存在する場合や、サーバーが使用中の場合、クライアン

トは managementTimeout 値で指定された時間だけ再試行します。 managementTimeout 値は秒単位で指定されます。

24. **threadsPerClientConnect** 属性 (オプション、デフォルトでは 5): ClientClusterContext ごとに作成されるスレッド数。 ObjectGridManager インターフェースで接続メソッドが呼び出されるごとに、結果として新しい ClientClusterContext が作成されます。

以下の universityClusterAttr.xml ファイルは、クラスター・エレメント上のさまざまなオプション属性を利用する構成例です。この例では、セキュリティーは使用不可です。さまざまなクライアント、TCP、ピア、およびスレッドに関連する属性も変更されます。 universityClusterAttr.xml では、属性に割り当てられている値は推奨ではありません。これは属性値を設定する方法の例です。

universityClusterAttr.xml ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster" securityEnabled="false" statisticsEnabled="true"
statisticsSpec="map.all=enabled" singleSignOnEnabled="false"
loginSessionExpirationTime="1800" adminAuthorizationEnabled="false"
adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_JAAS" clientMaxRetries="2"
clientMaxForwards="2" clientStartupRetries="2" clientRetryInterval="5"
tcpConnectionTimeout="160" tcpMinConnections="2" tcpMaxConnections="15"
tcpInactivityTimeout="3600" tcpMaxWaitTime="160" peerHeartbeatInterval="130"
peerTransportBufferSize="15" threadPoolMinSize="8" threadPoolMaxSize="25"
threadPoolInactivityTimeout="6050" managementTimeout="60">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectGridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectGridBinding>
<objectGridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectGridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

serverDefinition エレメント

出現回数: 1 回から多数回まで

子エレメント: なし

serverDefinition エレメントは ObjectGrid サーバーの定義に使用されます。各サーバーは、それぞれの Java 仮想マシン (JVM) で実行され、2 つのポートを必要とします。

属性:

```
<serverDefinition
(1) name="serverName"
(2) host="hostname"
(3) clientAccessPort="portNumber"
(4) peerAccessPort="portNumber"
(5) traceSpec="traceSpecification"
(6) systemStreamToFileEnabled="true|false"
(7) workingDirectory="logsDirectory"
/>
```

1. **name** 属性 (必須): これはサーバーに割り当てられる名前です。この属性が欠落している場合、XML 妥当性検査は失敗します。
2. **host** 属性 (必須): サーバー JVM が実行されるマシンのホスト名。各マシンは複数の ObjectGrid サーバーをホストすることができます。この属性が欠落している場合、XML 妥当性検査は失敗します。
3. **clientAccessPort** 属性 (必須): クライアント接続に使用されるサーバー上のポート。この属性が欠落している場合、XML 妥当性検査は失敗します。
4. **peerAccessPort** 属性 (必須): ObjectGrid サーバー間の通信に使用されるサーバー上のポート。この属性が欠落している場合、XML 妥当性検査は失敗します。
5. **traceSpec** 属性 (オプション、デフォルトでは *=all=disabled): traceSpec 属性を設定すると、指定されたストリングを使用したサーバーのトレースが使用可能になります。
6. **systemStreamToFileEnabled** 属性 (オプション、デフォルトでは **true**): この属性を true に設定すると、System.out、System.err、およびトレース出力ストリームがファイルに送られます。この属性を false に設定すると、System.out は stdout ストリームに送られ、System.err は stderr ストリームに送られます。トレースが使用可能な場合、トレース出力は systemStreamToFileEnabled 属性の値に関係なく、ファイルに送られます。
7. **workingDirectory** 属性 (オプション): workingDirectory 属性は、ログ・ファイルを書き込む場所を指定します。 workingDirectory 属性を指定しないと、ログは現行ディレクトリーに書き込まれます。

universityClusterServerAttr.xml ファイルは、 serverDefinition 属性の使用方法について例示しています。この XML ファイルでは、server1 サーバーが lion.ibm.com ホスト上で実行されるように構成されています。ポート 12501 はサーバーへのクライアント・アクセスに使用され、ポート 12502 はサーバーからサーバーへの通信に使用されます。 systemStreamToFileEnabled 属性が true に設定されているため、System.out、System.err、およびトレースが、workingDirectory 属性で指定されているディレクトリー内のファイルに出力されます。この例では、そのファイルは /objectgrid/ ディレクトリー内にあります。 traceSpec 属性は ″ObjectGrid=all=enabled″ に設定されているため、すべての ObjectGrid 関連トレースがキャプチャーされ、ファイルに出力されます。

universityClusterServerAttr.xml ファイル

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

objectgridBinding エレメント

出現回数: 1 回から多数回まで

子エレメント: mapSet エレメント

objectgridBinding エレメントは、ObjectGrid XML 内の objectGrid エレメントをクラスター XML で定義されているトポロジーにバインドするために使用されます。ref 属性に割り当てられる値は、ObjectGrid XML 内の objectGrid エレメントのいずれかの name 属性と一致している必要があります。 ObjectGrid XML にある objectGrid エレメントは、クラスター XML 内の 1 つの objectgridBinding でのみ参照できます。

属性

```
<objectgridBinding
(1) ref="objectGridReference"
(2) minThreadPoolSize="minSize"
(3) maxThreadPoolSize="maxSize"
/>
```

1. **ref** 属性 (必須): ref 属性は、ObjectGrid XML ファイルで定義されている objectGrid エレメントを参照するために使用されます。各 objectgridBinding エレメントは、ObjectGrid XML からの objectGrid エレメントの 1 つを参照する必要があります。ref 属性は、ObjectGrid XML 内の objectGrid エレメントのいずれかの name 属性と一致している必要があります。
2. **minThreadPoolSize** 属性 (オプション、デフォルトでは 3): minThreadPoolSize 属性は、各複製グループ・メンバーのスレッド・プールで許可されるスレッドの

最小数です。スレッドの数はスレッド・プール・マネージャーによって制御されますが、その数が minThreadPoolSize の値を下回ることはできません。通常、スレッドの数を大きくすると、クライアントはサーバーからの応答をより早く受け取ることができるようになります。また、スレッドを増やした結果、競合も増えます。しかし、より速いマシンは追加の並行スレッドを効果的に処理することができます。

3. **maxThreadPoolSize** 属性 (オプション、デフォルトでは 10):
   maxThreadPoolSize 属性は、各複製グループ・メンバーのスレッド・プールで許可されるスレッドの最大数です。スレッドの数はスレッド・プール・マネージャーによって制御されますが、その数が maxThreadPoolSize の値を上回ることはできません。通常、スレッドの数を大きくすると、クライアントはサーバーからの応答をより早く受け取ることができるようになります。また、スレッドを増やした結果、競合も増えます。しかし、より速いマシンは追加の並行スレッドを効果的に処理することができます。

universityClusterOGBinding.xml ファイルは、 objectgridBinding エレメントおよびその属性の使用方法を例示しています。この例では、1 つの objectgridBinding エレメントが定義されます。 objectgridBinding エレメントは university.xml ファイルで定義されている ″academics″ を参照します。 ″athletics″ objectGrid が university.xml ファイル内に存在していても、universityClusterOGBinding.xml ファイル内の athletics ObjectGrid を参照する objectgridBinding エレメントはありませんので注意してください。 ″athletics″ ObjectGrid は universityClusterOGBinding.xml ファイル内に組み込まれていないため、クラスター化されません。この場合は、university.xml ファイル内にあり universityClusterOGBinding.xml ファイルで参照される ″academics″ ObjectGrid のみ、作成およびクラスター化されます。

この例では minThreadPoolSize 属性および maxThreadPoolSize 属性も設定されます。 minThreadPoolSize 値は 2 に設定され、maxThreadPoolSize 値は 11 に設定されます。各複製グループ・メンバー上のスレッド・プール・マネージャーは、この ObjectGrid 内のすべてのマップのスレッド数を、この境界内に維持します。

universityClusterOGBinding.xml ファイル

```
<?xml version="1.0" encoding="UTF-8" ?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
</cluster>
<objectgridBinding ref="academics" minThreadPoolSize="2" maxThreadPoolSize="11">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
```

```
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

mapSet エレメント

出現回数: 1 回から多数回まで

子エレメント: map エレメント

mapSet エレメントは複数のマップをまとめてグループ化するために使用されます。mapSet 内のマップは同様の区画に分割されます。分散 ObjectGrid では、各マップは単一の mapSet のみに所属する必要があります。

属性

```
<mapSet
(1) name="mapSetName"
(2) partitionSetRef="partitionSetReference"
(3) synchronousReplication="true|false"
(4) replicaReadEnabled="true|false"
(5) replicaDeliveryRate="deliveryRate"
(6) compression="true|false"
/>
```

1. **name** 属性 (必須): mapSet に割り当てられる名前を指定します。
2. **partitionSetRef** 属性 (必須): 各 mapSet は partitionSetRef 属性を使用して partitionSet に割り当てられる必要があります。 partitionSetRef の値は、いずれか 1 つの partitionSet エレメントの name 属性の値と一致している必要があります。 partitionSetRef 属性とそれに対応する partitionSet を使用することによって、mapSet 内のマップは区画に分割されます。
3. **synchronousReplication** 属性 (オプション、デフォルトでは **false**): この属性を true に設定すると、複製グループ・メンバー間の複製は同期的に発生します。 false に設定すると、複製は非同期的に発生します。
4. **replicaReadEnabled** 属性 (オプション、デフォルトでは **false**): synchronousReplication 値が false に設定され、かつ replicaReadEnabled 値が true に設定されている場合、クライアントはレプリカのデータを読み取ることができます。プライマリーとそのレプリカの間で、読み取り要求をベスト・エフォートで分散します。 synchronousReplication 属性が true に設定されている場合、replicaReadEnabled 属性は無視されます。
5. **replicaDeliveryRate** (オプション、デフォルトでは **1000**): replicaDeliveryRate 値は、各レプリカに配信される、それぞれの LogSequence ごとのレコードの最大数を表します。
6. **compressReplicationEnabled** 属性 (オプション、デフォルトでは **true**): compressReplicationEnabled を true に設定すると、複製メッセージが圧縮されます。

universityClusterMapSet.xml ファイルは、前の XML ファイルの例より若干複雑です。このファイルでは、academics ObjectGrid は 2 つのマップ・セットに分割されます。academicsMapSet1 マップ・セットには faculty マップと course マップが含まれます。これら 2 つのマップは、partitionSet1 partitionSet に従って分割されます。これらのマップの複製設定も、同じ mapSet 内にあるため同様です。

academics objectgridBinding には academicsMapSet2 mapSet も含まれます。この mapSet には student マップのみが含まれます。 student マップは academicsMapSet1 mapSet 内のマップとは違う方法で分割されます。 student マップは studentPSet partitionSet に従って分割されます。 academicsMapSet2 には、synchronousReplication、replicaReadEnabled、 replicaDeliveryRate、および compressReplicationEnabled などの複製関連の属性に対して明示的に値が記述されていないため、デフォルト値が割り当てられます。これは、academics objectgridBinding 内の 2 つの mapSet の振る舞いに差をつける別の方法です。

athletics objectgridBinding には athleticsMapSet mapSet が含まれます。 academics objectgridBinding 内の academicsMapSet1 mapSet と同様に、これは partitionSet1 partitionSet に従って分割されます。この mapSet の複製関連の属性は、明示的に記述されていないため、デフォルト値に設定されます。

universityClusterMapSet.xml ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster">
<serverDefinition name="server1" host="lion.ibm.com" clientAccessPort="12501"
peerAccessPort="12502" />
<serverDefinition name="server2" host="tiger.ibm.com" clientAccessPort="12503"
peerAccessPort="12504" />
</cluster>
<objectgridBinding ref="academics"
<mapSet name="academicsMapSet1" partitionSetRef="partitionSet1"
synchronousReplication="true" replicaReadEnabled="true"
replicaDeliveryRate="1500" compressReplicationEnabled="true">
<map ref="faculty" />
<map ref="course" />
</mapSet>
<mapSet name="academicsMapSet2" partitionSetRef="studentPSet">
<mapRef="student" />
</mapSet>
</objectGridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetREf="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<partitionSet name="studentPSet">
<partition name="studentPartition1" replicationGroupRef="replicationGroup1" />
<partition name="studentPartition2" replicationGroupRef="replicationGroup2" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
<replicationGroup name="replicationGroup2" minReplicas="1" maxReplicas="2">
<replicationGroupMember serverRef="server1" priority="2" />
<replicationGroupMember serverRef="server2" priority="1" />
</replicationGroup>
</clusterConfig>
```

map エレメント

出現回数: 1 回から多数回まで

子エレメント: なし

mapSet 内の各マップは、ObjectGrid XML ファイルで定義されている backingMap エレメントの 1 つを参照します。分散 ObjectGrid を定義する場合、ObjectGrid XML の objectGrid エレメント内の各 backingMap は、クラスター XML 内のマップから参照される必要があります。分散 ObjectGrid 内の各マップは、単一の mapSet のみに属す必要があります。

属性

```
<map
(1) ref="backingMapReference"
/>
```

1. **ref** 属性 (必須): ObjectGrid XML 内の backingMap への参照。mapSet 内の各マップは、ObjectGrid XML ファイルの backingMap を参照する必要があります。ref に割り当てられる値は、ObjectGrid XML 内の backingMap エレメントのいずれかの name 属性と一致している必要があります。

map エレメントの使用例については、`universityClusterMapSet.xml` ファイルを参照してください。 `university.xml` 内の academics objectGrid の各 backingMap は、`universityClusterMapSet.xml` 内にある唯一の mapSet の中のマップから参照される必要があります。これは、athletics objectGrid の場合も該当します。objectgridBinding が ObjectGrid XML の objectGrid を参照するが、 mapSet 内にそのすべてのマップが含まれるわけではない場合は、結果として `ObjectGridException` 例外が発生します。

partitionSet エレメント

出現回数: 1 回から多数回まで

子エレメント: partition

partitionSet エレメントは mapSet の区画を定義するために使用されます。 mapSet 内の各マップは、partitionSet の区画全体に区分されます。 mapSet は、mapSet エレメント上に partitionSetRef 属性がある partitionSet と関連付けられます。partitionSet 内で定義されている区画が 1 つのみの場合、関連する mapSet のマップ内に含まれているデータは分割されません。

属性

```
<partition-set
(1) name="partitionSetName"
/>
```

1. **name** 属性 (必須): この属性は partitionSet に名前を割り当てるために使用されます。partitionSet の名前は mapSet の partitionSetRef 属性によって参照されます。

partitionSet の使用例については、`universityClusterMapSet.xml` ファイルを参照してください。`universityClusterMapSet.xml` では、partitionSet1 と studentPSet の 2 つの partitionSet が定義されます。 partitionSet1 partitionSet に対して定義される区

画は 1 つのみです。 1 つの区画しか定義されないため、partitionSet1 partitionSet を利用する任意の mapSet のデータは分割されます。partitionSet1 partitionSet に従って分割される universityClusterMapSet.xml ファイルには、2 つの mapSet が存在します。 mapSet エレメント上の partitionSetRef を使用して、academicsMapSet1 mapSet および athleticsMapSet mapSet が partitionSet1 partitionSet にバインドされます。

studentPSet partitionSet には 2 つの区画が含まれます。この partitionSet を使用する任意の mapSet のマップ・データは、2 つの区画に分割されます。universityClusterMapSet.xml ファイルでは、academicsMapSet2 mapSet は studentPSet partitionSet を使用しています。

partition エレメント

出現回数: 1 回から多数回まで

子エレメント: なし

partition エレメントは partitionSet 内の区画を定義するために使用されます。区画は mapSet のマップ内のデータを分割するために使用されます。

属性

```
<partition
(1) name="partitionName"
(2) replicationGroupRef="replicationGroupReference"
/>
```

1. **name** 属性 (必須): name 属性は区画に名前を割り当てるために使用されます。区画名は、その partitionSet 内で固有の名前である必要があります。
2. **replicationGroupRef** 属性 (必須): replicationGroupRef 属性は、replicationGroup を区画と関連付けるために使用されます。 replicationGroupRef は、いずれか 1 つの replicationGroup エレメントの name 属性と一致している必要があります。

partition エレメントの使用例については、universityClusterMapSet.xml ファイルを参照してください。 universityClusterMapSet.xml ファイルでは、複数の区画が定義されています。partitionSet1 partitionSet には partition1 という名前の区画が 1 つあります。studentPSet partitionSet には studentPartition1 と studentPartition2 の 2 つの区画が含まれます。

各区画は、replicationGroupRef 属性を使用して replicationGroup と関連付けられます。 universityClusterMapSet.xml ファイルでは、studentPartition1 区画が replicationGroup1 replicationGroup 全体に複製されます。 studentPartition2 区画は replicationGroup2 replicationGroup 全体に複製されます。

replicationGroup エレメント

出現回数: 1 回から多数回まで

子エレメント: replicationGroupMember エレメント

replicationGroup は、マップまたはその区画の複製方法を定義するために使用されます。マップの区画は replicationGroup 内の複製グループ・メンバー全体に複製されます。

属性

```
<replicationGroup
(1) name="replicationGroupName"
(2) minReplicas="minNumberOfReplicas"
(3) maxReplicas="maxNumberOfReplicas"
```

1. **name** 属性 (必須): name 属性は replicationGroup に名前を割り当てるために使用されます。
2. **minReplicas** 属性 (オプション、replicationGroup 内の replicationGroupMember が 1 つのみの場合のデフォルトは **0**、replicationGroup 内に複数の replicationGroupMember がある場合のデフォルトは **1**): minReplicas 属性は、書き込みアクセス権限が、この replicationGroup 内のマップ・データに許可される前に使用可能になっている必要がある replicationGroupMembers の数を示すために使用されます。使用可能なレプリカの数が minReplicas で指定された数より小さいと、マップには読み取りアクセスしか許可されません。minReplicas を 0 に設定すると、すべてのレプリカが使用不可でも、プライマリーに対して書き込みアクセス権限が許可されます。

   複製を活動化するには、2 つ以上の複製グループ・メンバーが使用可能で、かつ minReplicas 属性の値が 1 以上である必要があります。ObjectGrid クラスターの ″bringup″ ステージ中に replicationGroup がどのような振る舞いをするのかを知っておくことは重要です。単一のサーバーの始動後にマップ・データを使用可能にする場合は、replicationGroupMember を 1 つだけ持つ replicationGroup を定義します。 replicationGroupMember を 1 つしか持たない replicationGroup では、データは複製されません。

   以下は、minReplicas の値を設定する際のルールの一部です。

   ```
   minReplicas >= 0
   minReplicas <= maxReplicas
   minReplicas <= # of members in the replicationGroup -1
   ```

3. **maxReplicas** 属性 (オプション、replicationGroup 内の replicationGroupMember が 1 つのみの場合のデフォルトは **0**、replicationGroup 内に複数の replicationGroupMember がある場合のデフォルトは **1**): maxReplicas 属性は、replicationGroup 内で活動化されるレプリカの最大数を表します。replicationGroup では、多数のメンバーが使用可能な場合、maxReplicas で指定されている数だけ複製が行われます。 maxReplicas がグループ内の複製グループ・メンバーの数より小さい場合、余分なメンバーは待機となり、いずれか 1 つのレプリカが使用不可になるまで休止となります。

   以下は、maxReplicas の値を設定する際のルールの一部です。

   ```
   maxReplicas >= 0
   maxReplicas >= minReplicas
   ```

partition エレメントの使用例については、`universityClusterMapSet.xml` ファイルを参照してください。`universityClusterMapSet.xml` では、replicationGroup1 と replicationGroup2 の 2 つの replicationGroup が定義されます。 replicationGroup1 replicationGroup に含まれる replicationGroupMember は 1 つのみです。複製には複数の replicationGroupMember が必要なため、この replicationGroup にバインドされている区画は複製されません。

replicationGroup2 replicationGroup には 2 つの replicationGroupMember が含まれます。 studentPSet partitionSet の studentPartition2 区画は、この replicationGroup を使用しています。そのため、studentPartition2 区画は 2 つの replicationGroupMember に複製されます。 replicationGroup2 replicationGroup にも minReplicas 属性と maxReplicas 属性が設定されます。minReplicas が 1 に設定されているため、この replicationGroup に収容されているマップ・データは、プライマリーおよび 1 つ以上のレプリカが使用可能になるまで読み取り専用となります。 maxReplicas 値 1 は、この replicationGroup のプライマリーがそのデータのレプリカを多くても 1 つしか生成しないことを示しています。 replicationGroup2 replicationGroup のケースでは、グループに 2 つのメンバーしか含まれていないため、レプリカは 1 つまでしかありえません。一方のメンバーがプライマリーで、もう一方がレプリカです。

replicationGroupMember エレメント

出現回数: 1 回から多数回まで

子エレメント: なし

replicationGroupMember エレメントは、サーバー定義を参照するために使用されます。各 replicationGroupMember エレメントには関連する優先度も含まれます。この優先度は、どの replicationGroupMember がプライマリー・サーバーで、どのメンバーがレプリカかを判別するために使用されます。

属性

```
<replicationGroupMember
(1) serverRef="serverDefinitionReference"
(2) priority="priority"
/>
```

1. **serverRef** 属性 (必須): serverRef 属性は、サーバー定義を replicationGroupMember エレメントと関連付けるために使用されます。 serverRef 属性は、それぞれの replicationGroupMember を特定のサーバーと関連付けます。
2. **priority** エレメント (必須): priority 属性は、どの複製グループ・メンバーがプライマリーかを判別するために使用されます。 priority の値の範囲は 1 から複製グループ・メンバー数までで、1 が最高の優先度となります。ObjectGrid は、各複製グループ・メンバーの優先度をベスト・エフォートで実現できるようにします。事情によって阻止されない限り、優先度 1 の replicationGroupMember エレメントはがプライマリーとなります。すべてのサーバーとその replicationGroupMembers がほぼ同時に使用可能になった場合には、優先度の設定に従います。ただし、優先度 2 の replicationGroupMember が他の repliationGroupMember よりずっと以前に使用可能になった場合は、これがプライマリーとなります。

   プライマリーが正常に始動した後で、しばらくしてから障害が発生した場合は、新しいプライマリーが選択される必要があります。通常は、次に高い優先度を持つ replicationGroupMember エレメントが新しいプライマリーとなります。ただし、次に高い優先度を持つレプリカの複製が遅れていると判断された場合は、別のレプリカが新しいプライマリーとして選択される可能性があります。

partition エレメントの使用例については、`universityClusterMapSet.xml` ファイルを参照してください。`universityClusterMapSet.xml` では、replicationGroup1

replicationGroup には 1 つの replicationGroupMember しかありません。この定義があるため、この replicationGroup にはプライマリーしかありません。このグループにはレプリカはありません。定義済みの replicationGroupMember は、server1 サーバー上で、serverRef 値状態として活動中です。

replicationGroup2 replicationGroup には複数の replicationGroupMember があります。最初にリストされる replicationGroupMember は、server1 サーバーで活動化されます。このメンバーの優先度は 2 です。 2 番目にリストされる replicationGroupMember は、sever2 サーバーで活動化されます。 2 番目のメンバーの優先度が 1 なので、グループ・メンバーがほぼ同時に使用可能になった場合は、これがこの replicationGroup のプライマリーとなります。最初にリストされている replicationGroupMember は、優先度が 2 のためレプリカとして働きます。

minReplicas の値が replicationGroup2 replicationGroup に与える影響を知っておくことも重要です。 server1 サーバーと server2 サーバーがどちらも実行されているシナリオを例に考えてみましょう。この場合は、どちらの replicationGroupMembers も使用可能です。したがって、minReplicas の値と maxReplicas の値はどちらも満たされ、データはこのグループのプライマリーとレプリカの間で複製されます。 server1 が使用不可になると、いずれか 1 つの replicationGroupMembers が使用不可になります。この状態になると、minReplicas の値が満たされなくなるため、replicationGroup2 replicationGroup のデータは読み取り専用となります。

authenticator エレメント

出現回数: 0 回から 1 回まで

子エレメント: property エレメント

authenticator エレメントは、クラスター内の ObjectGrid サーバーに対してクライアントを認証するために使用されます。 className 属性によって指定されるクラスは、com.ibm.websphere.objectgrid.security.plugins.Authenticator インターフェースをインプリメントする必要があります。authenticator はプロパティーを使用して、className 属性によって指定されるクラス上のメソッドを呼び出すことができます。プロパティーの使用について詳しくは、301 ページの『property エレメント』を参照してください。

属性

```
<authenticator
(1) className="authenticatorClassName"
/>
```

1. **className** 属性 (必須): className 属性は、com.ibm.websphere.objectgrid.security.plugins.Authenticator インターフェースをインプリメントするクラスを指定するために使用されます。このクラスは、ObjectGrid クラスター内のサーバーに対してクライアントを認証するために使用されます。

以下の universityClusterSecurity.xml ファイルは、 authenticator エレメントの使用方法を例示しています。この例では、オーセンティケーターとして com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator クラス

が指定されています。このクラスは
com.ibm.websphere.objectgrid.security.plugins.Authenticator インターフェースをインプリメントします。

universityClusterSecurity.xml ファイル

```
<?xml version="1.0" encoding="UTF-8"?>
<clusterConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config/cluster
../objectGridCluster.xsd"
xmlns="http://ibm.com/ws/objectgrid/config/cluster">
<cluster name="universityCluster" securityEnabled="true"
singleSignOnEnabled="true"
loginSessionExpirationTime="1800" adminAuthorizationEnabled="true"
adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_CUSTOM">
<serverDefinition name="server1" host="lion.ibm.com"
clientAccessPort="12501" peerAccessPort="12502" />
<authenticator
className="com.ibm.websphere.objectgrid.security.plugins.builtins.
KeyStoreLoginAuthenticator" />
<adminAuthorization className= "com.ibm.MyAdminAuthorization">
<property name="interval" type="int" value="60" description="Set the
interval to 60 seconds" />
</adminAuthorization>
</cluster>
<objectgridBinding ref="academics">
<mapSet name="academicsMapSet" partitionSetRef="partitionSet1">
<map ref="faculty" />
<map ref="student" />
<map ref="course" />
</mapSet>
</objectgridBinding>
<objectgridBinding ref="athletics">
<mapSet name="athleticsMapSet" partitionSetRef="partitionSet1">
<map ref="athlete" />
<map ref="equipment" />
</mapSet>
</objectgridBinding>
<partitionSet name="partitionSet1">
<partition name="partition1" replicationGroupRef="replicationGroup1" />
</partitionSet>
<replicationGroup name="replicationGroup1">
<replicationGroupMember serverRef="server1" priority="1" />
</replicationGroup>
</clusterConfig>
```

adminAuthorization エレメント

出現回数: 0 回から 1 回まで

子エレメント: property エレメント

adminAuthorization エレメントは、ObjectGrid クラスターへの管理アクセスをセットアップするために使用されます。管理用タスクは管理アクセスが提供された後に実行することができます。

属性

```
<adminAuthorization
(1) className="adminAuthClassName"
/>
```

1. **className** 属性 (必須): className 属性は、com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization インターフェースをインプリメントするクラスを指定するために使用されます。

adminAuthorization エレメントの使用例については、universityClusterSecurity.xml ファイルのオーセンティケーターのセクションを参照してください。 universityClusterSecurity.xml ではカスタム adminAuthorization が使用されます。 com.ibm.MyAdminAuthorization クラスは adminAuthorization クラスとして使用されます。カスタム adminAuthorization を使用するには、securityEnabled 属性が true で、adminAuthorizationMechanism は AUTHORIZATION_MECHANISM_CUSTOM に設定され、adminAuthorization エレメントが提供される必要があります。この adminAuthorization エレメントはプロパティーも使用します。プロパティーの使用方法について詳しくは、『property エレメント』を参照してください。

property エレメント

出現回数: 0 回から多数回まで

子エレメント: なし

property エレメントは、authenticator および adminAuthorization 上で set メソッドを呼び出すために使用されます。property の名前は、その property を含む authenticator または adminAuthorization エレメントの className 上の set メソッドに対応しています。

属性

```
<property
(1) name="propertyName"
(2) type="java.lang.String|boolean|java.lang.Boolean|int|
java.lang.Integer|double|java.lang.Double|byte|
java.lang.Byte|short|java.lang.Short|long|
java.lang.Long|float|java.lang.Float|char|
java.lang.Character"
(3) value="propertyValue"
(4) description="description"
/>
```

1. **name** 属性 (必須): プロパティーの名前。この属性に割り当てられる値は、authenticator または adminAuthorization の className として提供されるクラス上の set メソッドと対応している必要があります。例えば、authenticator の className が com.ibm.MyAuthenticator に設定され、提供されたプロパティーの名前が interval である場合、com.ibm.MyAuthenticator クラスは setInterval メソッドを持たなければなりません。
2. **type** 属性 (必須): プロパティーのタイプ。 name 属性によって識別された set メソッドへ渡されるパラメーターのタイプです。有効な値は、Java プリミティブ、それに対応する java.lang プリミティブ、および java.lang.String です。name および type は、Bean の className のメソッド・シグニチャーに対応している必要があります。例えば、name が interval で type が int の場合は、その authenticator または adminAuthorization の className として指定されたクラス上に setInterval(int) メソッドが存在している必要があります。
3. **value** 属性 (必須): プロパティーの値。この値は type 属性によって指定されたタイプに変換され、次に name 属性と type 属性で識別された set メソッドへの

呼び出しでパラメーターとして使用されます。この属性の値がどのような方法でも妥当性検査されないことを理解しておくことは重要です。プラグイン・インプリメンターは、渡された値が有効であることを検証しなければなりません。インプリメンターは、パラメーターが有効でない場合、set メソッド内で `IllegalArgumentException` 例外を表示することができます。

4. **description** 属性 (オプション): この属性を使用して、プロパティーの説明を作成します。

adminAuthorization エレメントの使用例については、`universityClusterSecurity.xml` ファイルを参照してください。 property エレメントは、クラスター XML 内の authenticator エレメントまたは adminAuthorization エレメントの中で使用することができます。 `universityClusterSecurity.xml` ファイルでは、property は adminAuthorization 上の set メソッドを呼び出すために使用されます。この場合、setInterval メソッドは com.ibm.MyAdminAuthorization クラス上で呼び出されます。これには整数値 `60` が渡されます。

第 10 章 ObjectGrid と WebSphere Application Server の統合

ObjectGrid を、WebSphere Application Server で提供されているフィーチャーと共に使用し、ObjectGrid の機能でアプリケーションを強化します。

WebSphere Application Server および WebSphere Extended Deployment をインストールします。 WebSphere Extended Deployment をインストールした後、ObjectGrid 関数を Java 2 Platform, Enterprise Edition (J2EE) アプリケーションに追加することができます。

ObjectGrid API は、WebSphere Application Server をターゲットとする J2EE アプリケーションで使用することができます。WebSphere Extended Deployment をインストールすると、 `wsobjectgrid.jar` ファイルが `¥base¥lib` ディレクトリーに配置されます。 ObjectGrid API と J2EE アプリケーション・プログラミング・モデルの統合の他に、分散トランザクション伝搬サポートを利用することもできます。このサポートを使用して、 WebSphere Application Server クラスターでトランザクション・コミット結果を調整するように、 ObjectGrid インスタンスを構成することができます。

1. J2EE アプリケーションを ObjectGrid で使用可能にするための基本的なプログラミング・ステップを実行します。 詳しくは、 304 ページの『Java 2 Platform, Enterprise Edition 環境における ObjectGrid の統合』を参照してください。
2. ObjectGrid アプリケーションのパフォーマンス・データをモニターします。 詳しくは、 308 ページの『WebSphere Application Server Performance Monitoring Infrastructure (PMI) を使用した ObjectGrid パフォーマンスのモニター』を参照してください。
3. ObjectGrid が組み込まれている場合は、外部トランザクション・コーディネーターによってトランザクションが開始または終了されることがあります。 詳しくは、 315 ページの『ObjectGrid と外部トランザクションとの相互作用』を参照してください。
4. 区画化機能を ObjectGrid とともに使用します。 ObjectGrid フィーチャーは、トランザクションの方法で鍵と値のペアをキャッシュする容量を提供し、区画化機能フィーチャーは、オブジェクトの特性に応じてコンテキスト・ベースのルーティングの容量を提供します。詳しくは、 319 ページの『ObjectGrid と区画化機能の統合』を参照してください。
5. WebSphere Application Server バージョン 6.0.2 以降でコンテナー管理パーシスタンス (CMP) Bean を使用して、ObjectGrid を組み込みキャッシュではなく外部キャッシュとして利用できます。 詳しくは、 341 ページの『コンテナー管理の Bean と連動する ObjectGrid の構成』を参照してください。

ObjectGrid を JMS とともに使用して、異なる層に、または混合プラットフォームを持つ環境に変更を分散することもできます。詳しくは、 361 ページの『トランザクションの変更を配布する Java Message Service』を参照してください。

Java 2 Platform, Enterprise Edition 環境における ObjectGrid の統合

ObjectGrid は、Java 2 Platform, Enterprise Edition (J2EE) 環境において、サーブレットと Enterprise JavaBeans (EJB) の両方のプログラミング・モデルをサポートします。

このトピックでは、ObjectGrid とともに J2EE アプリケーションを使用可能にするための一般的なプログラミング・ステップについて検討します。

ローカル ObjectGrid のシナリオ

1. ObjectGrid 構成を定義します。 XML ファイルを使用するか、プログラマチック・インターフェースを介するか、または XML ファイルとプログラマチック・インターフェースを混合して使用することによって ObjectGrid 構成を定義します。詳しくは、『ObjectGrid 構成』を参照してください。
2. URL オブジェクトを作成します。 ObjectGrid 構成が XML ファイル内にある場合、その XML ファイルを指す URL オブジェクトを作成します。この URL オブジェクトを使用して、 ObjectGridManager API を使用することで ObjectGrid インスタンスを作成できます。 ObjectGrid 構成 XML ファイルが Web アーカイブ (WAR) または Enterprise JavaBeans (EJB) Java アーカイブ (JAR) ファイルに含まれている場合、 Web と EJB モジュールの両方のクラス・ローダーへのリソースとしてアクセスすることができます。例えば、 ObjectGrid 構成 XML ファイルが Web モジュール WAR ファイルの WEB-INF フォルダーにある場合、その WAR ファイル内にあるサーブレットは、以下のパターンを使用して URL オブジェクトを作成することができます。

   ```
   URL url =className.class.getClassLoader().
   getResource("META-INF/objectgrid-definition.xml");
   URL objectgridUrl = ObjectGridCreationServlet.class.getClassLoader().
   getResource("WEB-INF/objectgrid-definition.xml");
   ```

3. ObjectGrid インスタンスを作成または取得します。 ObjectGridManager API を使用して、ObjectGrid インスタンスを取得および作成します。ObjectGridManager API を使用して、 XML を持つ ObjectGrid インスタンスを作成し、ユーティリティー・メソッドを使用して単純な ObjectGrid インスタンスを即時に作成することができます。アプリケーションは ObjectGridManagerFactory API を使用して、ObjectGridManager API への参照を取得する必要があります。以下のコーディング例を参照してください。

   ```
   import com.ibm.websphere.objectgrid.ObjectGridManager;
   import com.ibm.websphere.objectgrid.ObjectGridManagerFactory ;
   ...
   ObjectGridManager objectGridManager = ObjectGridManagerFactory.
   getObjectGridManager();
   ObjectGrid ivObjectGrid = objectGridManager.
   createObjectGrid(objectGridName, objectgridUrl, true, true);
   ```

   ObjectGridManager API についての詳細は、『ObjectGridManager インターフェース』のトピックを参照してください。
4. ObjectGrid インスタンスを初期設定します。 ObjectGrid インターフェースで初期設定メソッドを使用して、ObjectGrid および Session インスタンスのブートストラッピングを開始します。getSession メソッドに対する最初の呼び出しは暗黙的な初期設定を実行するため、この初期設定メソッドはオプションとみなされます。このメソッドが呼び出された後、ObjectGrid 構成は完了とみなされ、ランタ

イム使用の準備が整います。 defineMap(String mapName) メソッドの呼び出しなど、追加の構成メソッドの呼び出しは、結果として例外になります。

5. Session および ObjectMap インスタンスを取得します。 セッションは、ObjectMap インスタンスのコンテナーです。スレッドは、独自の Session オブジェクトを取得して、ObjectGrid コアと対話する必要があります。この技法は、一度に単一のスレッドによってのみ使用できるセッションと考えることができます。このセッションが一度に 1 つのスレッドしか使用しない場合、セッションはスレッド間で共用可能です。ただし、J2EE 接続またはトランザクション・インフラストラクチャーが使用される場合、セッション・オブジェクトはスレッド間で共用ができません。このオブジェクトのよい例は、データベースに対する Java Database Connectivity (JDBC) 接続です。

   ObjectMap マップは、指定されたマップへのハンドルです。マップは、同種の鍵および値を持っていなければなりません。ObjectMap インスタンスは、この ObjectMap インスタンスを取得するために使用されたセッションに現在関連付けられているスレッドによってのみ使用可能です。複数のスレッドが、Session オブジェクトおよび ObjectMap オブジェクトを並行して共用することはできません。キーワードは、トランザクション内で適用されます。トランザクション・ロールバックは、このトランザクション中に適用されるキーワード関連をロールバックします。以下にコーディング例を示します。

   ```
   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. セッションを開始し、オブジェクトを読み取るか書き取り、セッションをコミットまたはロールバックします。 マップ操作は、トランザクション・コンテキスト内で行われる必要があります。 Session オブジェクトの開始メソッドは、明示的なトランザクション・コンテキストを開始するために使用されます。セッションが開始した後、アプリケーションはマップ操作の実行を開始することができます。最も一般的な操作には、マップに対するオブジェクトの取得、更新、挿入、および除去のメソッド呼び出しが含まれます。マップ操作の最後には、 Session オブジェクトのコミットまたはロールバック・メソッドが呼び出され、明示的なトランザクション・コンテキストをコミットするか、明示的なトランザクション・コンテキストをロールバックします。以下にプログラミング例を示します。

   ```
   ivSession.begin();
   Integer key = new Integer(1);
   if (ivCounterMap.containsKey(key) == false) {
   ivCounterMap.insert(key, new Counter(10));
   }
   ivSession.commit();
   ```

分散 ObjectGrid のシナリオ

この分散 ObjectGrid のシナリオは、ObjectGrid インスタンスの取得方法の点でのみローカル ObjectGrid のシナリオと異なります。以下のコード例は、分散 ObjectGrid インスタンスの取得方法を説明しています。

```
//Use ObjectGridManagerFactory to get the reference to the ObjectGridManager API
ObjectGridManager objectGridManager = ObjectGridManagerFactory.
getObjectGridManager();
//Obtain the ClientClusterContext represents the ObjectGrid cluster from
//ObjectGridManager
```

```
//Assuming the WebSphere server also host an ObjectGrid server
//that is a member of the ObjectGrid cluster.
ClientClusterContext context = objectGridManager.connect(null, null);
//Get the ObejctGrid instance from the ObjectGridManager a specific
//ClientClusterContext.
ObjectGrid ivObjectGrid= objectGridManager.getObjectGrid(context,
"objectgridName");
```

ObjectGrid とともに J2EE アプリケーションを使用可能にするための基本的なプログラミング・ステップを実行しました。

詳しくは、『ObjectGrid 対応 Java 2 Platform, Enterprise Edition (J2EE) アプリケーションのビルド』および『Java 2 Platform, Enterprise Edition (J2EE) アプリケーションと ObjectGrid の統合に関する考慮事項』を参照してください。

ObjectGrid 対応 Java 2 Platform, Enterprise Edition アプリケーションのビルド

このタスクを使用して、ObjectGrid 対応 Java 2 Platform, Enterprise Edition (J2EE) アプリケーションのビルド・パスまたはクラスパスを構成します。クラスパスには、 `$install_root/lib` ディレクトリーに配置された `wsobjectgrid.jar` ファイルが含まれていなければなりません。

ObjectGrid 対応 J2EE アプリケーションを開発します。詳しくは、『J2EE 環境における ObjectGrid の統合』を参照してください。

このタスクは、ビルド・パスを設定して、`wsobjectgrid.jar` ファイルを IBM Rational Software Development Platform バージョン 6.0 に組み込む方法について説明します。

1. J2EE パースペクティブの「Project Explorer」ビューで、 **WEB** または Enterprise JavaBeans (**EJB**) プロジェクトを右クリックし、「**プロパティー**」を選択します。「プロパティー」ウィンドウが表示されます。
2. 左側のパネルで「**Java ビルド・パス**」を選択し、右側のパネルで「**ライブラリー**」タブをクリックし、「**変数の追加**」をクリックします。「**New Variable Classpath Entry**」ウィンドウが表示されます。
3. 「**変数の構成**」をクリックして、「**設定**」ウィンドウを開きます。
4. 新規変数項目を追加します。
   a. 「**新規作成**」をクリックします。
   b. 「**名前**」フィールドに `OBJECTGRID_JAR` と入力します。「**ファイル**」をクリックして、「**JAR Selection**」ウィンドウを開きます。
   c. `/lib` ディレクトリーを参照し、**wsobjectgrid.jar** ファイルをクリックし、「**オープン**」をクリックして「**JAR Selection**」ウィンドウを閉じます。
   d. 「**OK**」をクリックして「**New Variable Entry**」ウィンドウを閉じます。

   `OBJECTGRID_JAR` 変数が、クラスパス変数リストに表示されます。
5. 「**OK**」をクリックして「**設定**」ウィンドウを閉じます。
6. 変数リストから `OBJECTGRID_JAR` 変数を選択し、「**OK**」をクリックして「**New Variable Classpath Entry**」ウィンドウを閉じます。 `OBJECTGRID_JAR` 変数が、「**ライブラリー**」パネルに表示されます。
7. 「**OK**」をクリックして「**プロパティー**」ウィンドウを閉じます。

ビルド・パスを設定して、wsobjectgrid.jar ファイルを IBM Rational Software Development Platform バージョン 6.0 に組み込みました。

Java 2 Platform, Enterprise Edition アプリケーションと ObjectGrid の統合に関する考慮事項

Java 2 Platform, Enterprise Edition (J2EE) アプリケーションと ObjectGrid を統合する場合は、次の考慮事項に留意してください。

開始 Bean と ObjectGrid

アプリケーションの開始 Bean を使用して、アプリケーションが起動する際に ObjectGrid インスタンスをブートストラップし、アプリケーションが停止する際に ObjectGrid インスタンスを破棄することができます。開始 Bean は、com.ibm.websphere.startupservice.AppStartUpHome リモート・ホームと com.ibm.websphere.startupservice.AppStartUp リモート・インターフェースを持つ Stateless Session Bean です。 WebSphere Application Server が Enterprise JavaBean (EJB) を見る場合、開始 Bean を認識します。リモート・インターフェースには 2 つのメソッド、start メソッドと stop メソッドがあります。start メソッドを使用してグリッドをブートストラップし、stop メソッドでグリッドを破棄するメソッドを呼び出します。ObjectGridManager.getObjectGrid メソッドを使用して、必要な場合に参照を取得することにより、アプリケーションはグリッドへの参照を維持することができます。詳しくは、『ObjectGridManager インターフェース』のトピックを参照してください。

クラス・ローダーと ObjectGrid インスタンス

1 つの ObjectGrid インスタンスを、異なるクラス・ローダーを使用するアプリケーション・モジュール間で共用する場合は注意が必要です。異なるクラス・ローダーを使用するアプリケーション・モジュールは動作せず、その結果アプリケーションでクラス・キャスト例外が発生します。 ObjectGrid は、同じクラス・ローダーを使用するアプリケーション・モジュールでのみ共用するか、アプリケーション・オブジェクト (プラグイン、キー、値など) が共通のクラス・ローダー上にある場合のみ、共用する必要があります。

サーブレット内の ObjectGrid インスタンスのライフ・サイクルを管理する

サーブレットの init メソッドおよび destroy メソッドを使用して ObjectGrid インスタンスのライフ・サイクルを管理できます。アプリケーションで必要な ObjectGrid インスタンスを作成し、初期化するには、init メソッドを使用します。ObjectGrid インスタンスが作成され、キャッシュされた後で、ObjectGridManager API を使用して、名前によってインスタンスを取得することができます。これらの ObjectGrid インスタンスを破棄し、システム・リソースを解放するには、destroy メソッドを使用します。詳しくは、『ObjectGridManager インターフェース』のトピックを参照してください。

WebSphere Application Server Performance Monitoring Infrastructure (PMI) を使用した ObjectGrid パフォーマンスのモニター

ObjectGrid は、WebSphere Application Server または WebSphere Extended Deployment アプリケーション・サーバーでの稼働中に、Performance Monitoring Infrastructure (PMI) をサポートします。 PMI は、ランタイム・アプリケーションでパフォーマンス・データを収集し、パフォーマンス・データをモニターするための外部アプリケーションをサポートするインターフェースを提供します。

ObjectGrid が提供する統計についての詳細は、『ObjectGrid 統計』を参照してください。

ObjectGrid は、WebSphere Application Server のカスタム PMI 機能を使用し、独自の PMI 媒介機能を追加します。この方法で、管理コンソール、または Java Management Extensions (JMX) インターフェースを使用して ObjectGrid PMI を使用可能および使用不可にすることができます。さらに、標準 PMI、および Tivoli Performance Viewer を含むモニター・ツールによって使用される JMX インターフェースを使用して ObjectGrid 統計にアクセスすることができます。

1. ObjectGrid PMI を使用可能にします。 PMI 統計を表示するには、PMI を使用可能にする必要があります。詳しくは、 311 ページの『ObjectGrid PMI の使用可能化』を参照してください。
2. ObjectGrid PMI 統計を取得します。 Tivoli Performance Viewer を使用して、ObjectGrid アプリケーションのパフォーマンスを表示します。詳しくは、 314 ページの『ObjectGrid PMI 統計の取得』を参照してください。

ObjectGrid 統計

ObjectGrid は、objectGridModule モジュールと mapModule モジュールの 2 つのパフォーマンス・モニター・インフラストラクチャー (PMI) モジュールを提供します。

objectGridModule モジュール

objectGridModule モジュールは、 1 つの時間統計 (トランザクション応答時間) を含みます。ObjectGrid トランザクションは、 Session.begin メソッド呼び出しと Session.commit メソッド呼び出しの間の所要時間として定義されます。この所要時間は、トランザクション応答時間として追跡されます。

objectGridModule モジュールのルート・エレメント (ObjectGrids エレメント) は、ObjectGrid 統計への入り口点として機能します。このルート・エレメントは、トランザクション・タイプが子である ObjectGrid インスタンスを子に持ちます。応答時間統計はそれぞれのトランザクション・タイプと関連しています。次の図は objectGridModule モジュールの構造です。

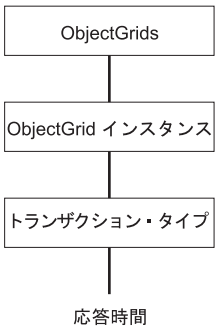


図 19. ObjectGridModule モジュールの構造

次の図は ObjectGrid PMI モジュール構造の例です。この例では、システムに 2 つの ObjectGrid インスタンスが存在します。 1 つは objectGrid1 ObjectGrid、1 つは objectGrid2 ObjectGrid です。objectGrid1 インスタンスは更新と読み込みの 2 つのタイプのトランザクションを持ちますが、objectGrid2 インスタンスは更新のトランザクション・タイプしか持ちません。

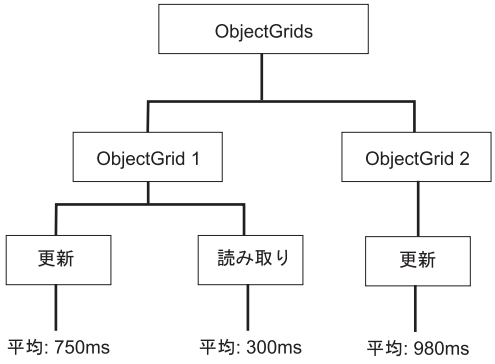


図 20. ObjectGrid PMI モジュール構造

アプリケーション開発者は、アプリケーションがどのタイプのトランザクションを使用するのかを知っているため、トランザクション・タイプはアプリケーション開発者によって定義されます。トランザクション・タイプは次の Session.setTransactionType(String) メソッドを使用して設定されます。

```
/**
* Sets the transaction type for future transactions.
*
* After this method is called, all of the future transactions have the
* same type until another transaction type is set.If  no transaction
* type is set, the default  TRANSACTION_TYPE_DEFAULT transaction type
* is used.
*
* Transaction types are used mainly for statistical data tracking purpose.
* Users can predefine types of transactions that run in an
* application.The idea is to categorize transactions with the same characteristics
* to one category (type), so one transaction response time statistic can be
* used to track each transaction type.
*
* This tracking is useful when your application has different types of
* transactions.
* Among them, some types of transactions, such as update transactions, process
* longer than other transactions, such as read-only transactions. By using the
* transaction type, different transactions are tracked by different statistics,
```

```
* so the statistics can be more useful.
*
* @param tranType the transaction type for future transactions.
*/
void setTransactionType(String tranType);
```

次の例は、 updatePrice へのトランザクション・タイプを設定します。

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

最初の行は、後続のトランザクション・タイプが updatePrice であることを示します。updatePrice 統計は、例にあるセッションに対応する ObjectGrid インスタンスに置かれています。Java Management Extensions (JMX) インターフェースを使用して、updatePrice トランザクション用のトランザクション応答時間を取得できます。指定した ObjectGrid で、トランザクションのすべてのタイプの集約統計も取得できます。

mapModule モジュール

mapModule PMI モジュールは、 ObjectGrid マップに関連した 3 つの統計を含んでいます。

- **マップ・ヒット率:** この BoundedRangeStatistic 統計はマップのヒット率を追跡します。ヒット率は 0 以上 100 以下の浮動値で、マップ取得操作に関係するマップ・ヒットの比率です。
- **エントリー数**: この CountStatistic 統計はマップのエントリー数を追跡します。
- **ローダー・バッチ更新応答時間**: この TimeStatistic 統計はローダー・バッチ更新操作に使用される応答時間を追跡します。

mapModule モジュールのルート・エレメント (ObjectGrid マップ・エレメント) は、 ObjectGrid マップ統計の入り口点として機能します。このルート・エレメントは、マップ・インスタンスを子に持つ、 ObjectGrid インスタンスを子に持ちます。すべてのマップ・インスタンスは、3 つのリスト統計を持っています。次の図は mapModule 構造です。

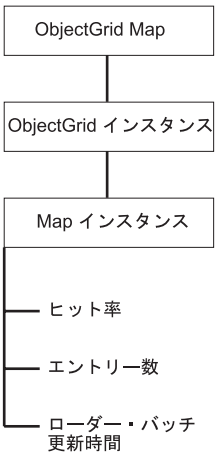


図 21. mapModule モジュール構造

次の図は mapModule 構造の例です。

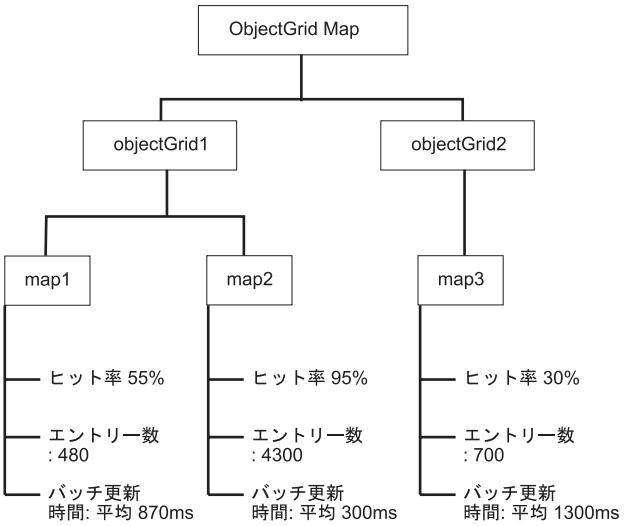


図 22. mapModule モジュール構造の例

ObjectGrid PMI の使用可能化

WebSphere Application Server Performance Monitoring Infrastructure (PMI) を使用して、任意のレベルで統計を使用可能または使用不可にすることができます。例えば、特定のマップのマップ・ヒット率統計を使用可能にするが、エントリー数統計またはローダー・バッチ更新時間統計は使用可能にしないことを選択できます。このトピックでは、管理コンソールおよび wsadmin スクリプトを使用して ObjectGrid PMI を使用可能にする方法を示します。

WebSphere Application Server PMI を使用して、任意のレベルで統計を使用可能または使用不可にできる細かいメカニズムを提供します。例えば、特定のマップのマップ・ヒット率統計を使用可能にするが、エントリー数統計またはローダー・バッチ

更新時間統計は使用可能にしないことを選択できます。このセクションでは、管理コンソールおよび wsadmin スクリプトを使用して ObjectGrid PMI を使用可能にする方法を示します。

1. 管理コンソールを開きます。例えば、http://localhost:9060/ibm/console です。
2. 「**モニターおよびチューニング**」>「**Performance Monitoring Infrastructure**」>「***server_name***」をクリックします。
3. 「**Performance Monitoring Infrastructure (PMI) を使用可能にする**」が選択されていることを確認します。この設定は、デフォルトで使用可能になっています。この設定が使用可能になっていない場合、チェック・ボックスを選択して、サーバーを再始動します。
4. 「**カスタム**」をクリックします。構成ツリーで、**ObjectGrid** および **ObjectGrid Maps** モジュールを選択します。各モジュールの統計を使用可能にします。

ObjectGrid 統計のトランザクション・タイプ・カテゴリーは、実行時に作成されます。「ランタイム」パネルには、ObjectGrid と Map 統計のサブカテゴリーのみを表示できます。

例えば、以下のステップを実行して、サンプル・アプリケーションの PMI 統計を使用可能にすることができます。

1. `http://host:port/ObjectGridSample` Web アドレスを使用してアプリケーションを立ち上げます。ここで、host および port は、サンプルをインストールするサーバーのホスト名および HTTP ポート番号です。
2. サンプル・アプリケーションでは、**ObjectGridCreationServlet** をクリックし、次に、アクション・ボタン 1、2、3、4、および 5 をクリックして、 ObjectGrid およびマップに対するアクションを生成します。この時点で、このサーブレット・ページを閉じないでください。
3. 管理コンソールに戻り、「**モニターおよびチューニング**」>「**Performance Monitoring Infrastructure**」>「***server_name***」をクリックします。「**ランタイム**」タブをクリックします。
4. 「**カスタム**」ラジオ・ボタンをクリックします。
5. ランタイム・ツリーで「**ObjectGrid Maps**」モジュールを展開し、「**clusterObjectGrid**」リンクをクリックします。「**ObjectGrid Maps**」グループの下に、 clusterObjectGrid と呼ばれる 1 つの ObjectGrid インスタンスが存在し、この clusterObjectGrid グループの下に、カウンター、従業員、オフィス、およびサイトという 4 つのマップが存在します。**ObjectGrids** インスタンスには、1 つの clusterObjectGrid インスタンスが存在し、そのインスタンスの下には、`DEFAULT` と呼ばれるトランザクション・タイプがあります。
6. 興味のある統計を使用可能にすることができます。デモンストレーションの目的で、従業員マップの**マップ・エントリー数**、および DEFAULT トランザクション・タイプの**トランザクション応答時間**を使用可能にできます。

スクリプトを使用して PMI を使用可能にするタスクを自動化することができます。詳しくは、『スクリプトを使用した ObjectGrid PMI の使用可能化』を参照してください。

スクリプトを使用した ObjectGrid PMI の使用可能化

wsadmin ツールを使用して ObjectGrid PMI を使用可能にするタスクを自動化します。

アプリケーション・サーバーを始動し、ObjectGrid 対応アプリケーションがインストールされている必要があります。また、wsadmin ツールにログインして使用できなければなりません。 wsadmin ツールについての詳細は、WebSphere Extended Deployment バージョン 6.0.x インフォメーション・センターの「スクリプトの使用 (wsadmin)」を参照してください。

このタスクを使用して、PMI の使用可能化を自動化します。管理コンソールを使用して PMI を使用可能にするには、『ObjectGrid PMI の使用可能化』を参照してください。

1. コマンド行プロンプトを開きます。*install_root*/bin ディレクトリーへナビゲートします。wsadmin と入力し、wsadmin コマンド行ツールを開始します。
2. ObjectGrid PMI ランタイム構成を変更します。 以下のコマンドを使用して、PMI がサーバーに対して使用可能になっているかどうかを確認します。

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

   PMI が使用可能になっていない場合は、以下のコマンドを実行して、PMI を使用可能にします。

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

   PMI を使用可能にする必要がある場合は、サーバーを再始動します。
3. 統計セットをカスタム・セットに変更するための変数を設定します。 以下のコマンドを実行します。

   ```
   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. 統計セットをカスタムに設定します。 以下のコマンドを実行します。

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
   ```
5. 変数を設定し、objectGridModule PMI 統計を使用可能にします。 以下のコマンドを実行します。

   ```
   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. 統計ストリングを設定します。 以下のコマンドを実行します。

   ```
   wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params $sigs
   ```
7. 変数を設定し、mapModule PMI 統計を使用可能にします。 以下のコマンドを実行します。

```
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. 統計ストリングを設定します。 以下のコマンドを実行します。

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

これらのステップにより、ObjectGrid ランタイム PMI は使用可能になりますが、PMI 構成は変更されません。アプリケーション・サーバーを再始動する場合、メイン PMI 使用可能性を除いて、PMI 設定は失われます。

PMI が使用可能になった後、管理コンソールまたはスクリプトを介して PMI 統計を表示することができます。詳しくは、『ObjectGrid PMI 統計の取得』および『スクリプトを使用した ObjectGrid PMI 統計の取得』を参照してください。

ObjectGrid PMI 統計の取得

ObjectGrid アプリケーションのパフォーマンス統計を参照してください。

ObjectGrid 統計が使用可能になると、この統計を取得することができます。ObjectGrid PMI を使用可能にするには、『ObjectGrid PMI の使用可能化』を参照してください。

このタスクを使用して、ObjectGrid アプリケーションのパフォーマンス統計を表示します。

1. 管理コンソールを開きます。 例えば、http://localhost:9060/ibm/console です。
2. 「**モニターおよびチューニング**」>「**Performance Viewer**」>「**現行アクティビティー**」をクリックします。
3. Tivoli Performance Viewer を使用してモニターするサーバーをクリックし、モニターを使用可能にします。
4. サーバーをクリックし、「Performance viewer」ページを表示します。
5. 構成ツリーを展開します。「**ObjectGrid Maps**」>「**clusterObjectGrid**」をクリックし、「**employees**」を選択します。「**ObjectGrids**」>「**clusterObjectGrid**」を展開し、「**DEFAULT**」を選択します。
6. ObjectGrid サンプル・アプリケーションで、ObjectGridCreationServlet サーブレットに戻り、ボタン 1 をクリックして、**マップを取り込み**ます。ビューアーに統計が表示されます。

Tivoli Performance Viewer に ObjectGrid Statistics を表示することができます。

Java Management Extensions (JMX) または wsadmin ツールを使用して、統計を取得するタスクを自動化することができます。『スクリプトを使用した ObjectGrid PMI 統計の取得』を参照してください。

スクリプトを使用した ObjectGrid PMI 統計の取得

このタスクを使用して、ObjectGrid アプリケーションのパフォーマンス統計を取得します。

アプリケーション・サーバー環境で、Performance Monitoring Infrastructure (PMI) を使用可能にします。詳しくは、『ObjectGrid PMI の使用可能化』または『スクリプトを使用した ObjectGrid PMI の使用可能化』を参照してください。また、wsadmin ツールにログインして使用できなければなりません。 wsadmin ツールについての詳細は、WebSphere Extended Deployment バージョン 6.0.x インフォメーション・センターの「スクリプトの使用 (wsadmin)」を参照してください。

このタスクを使用して、アプリケーション・サーバー環境のパフォーマンス統計を取得します。取得できる ObjectGrid 統計についての詳細は、 308 ページの『ObjectGrid 統計』を参照してください。

1. コマンド行プロンプトを開きます。*install_root*/bin ディレクトリーへナビゲートします。wsadmin と入力し、wsadmin コマンド行ツールを開始します。
2. 環境の変数を設定します。 以下のコマンドを実行します。

```
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. 変数を設定し、mapModule 統計を取得します。 以下のコマンドを実行します。

```
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. mapModule 統計を取得します。 以下のコマンドを実行します。

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

5. 変数を設定し、objectGridModule 統計を取得します。 以下のコマンドを実行します。

```
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. objectGridModule 統計を取得します。 以下のコマンドを実行します。

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

戻される統計についての詳細は、 308 ページの『ObjectGrid 統計』を参照してください。

ObjectGrid と外部トランザクションとの相互作用

通常、ObjectGrid トランザクションは session.begin メソッドで始まり、session.commit メソッドで終了します。ただし、ObjectGrid が組み込まれていると、トランザクションは外部トランザクション・コーディネーターによって開始および終了する場合があります。この場合は session.begin メソッドを呼び出す必要も、session.commit メソッドで終了する必要もありません。

外部トランザクションの調整

ObjectGrid TransactionCallback プラグインは、ObjectGrid セッションと外部トランザクションを関連づける isExternalTransactionActive(Session session) メソッドにより拡張されます。このメソッドのヘッダーは次のとおりです。

```
public synchronized boolean isExternalTransactionActive(Session session)
```

例えば、ObjectGrid をセットアップして WebSphere Application Server および WebSphere Extended Deployment と統合することができます。このシームレスな統合の鍵となるのが、WebSphere Application Server バージョン 5.x およびバージョン 6.x の ExtendedJTATransaction API の利用です。ただし、WebSphere Application Server Version 6.0.2 をお使いの場合は、このメソッドをサポートするために APAR PK07848 を適用する必要があります。次のサンプル・コードを使用して、ObjectGrid セッションを WebSphere Application Server トランザクション ID と関連付けます。

```
/**
* This methodis required to associate an objectGrid session with a WebSphere
* transaction ID.
*/
Map/**/ localIdToSession;
public synchronized boolean isExternalTransactionActive(Session session)
{
// remember that this localid means this session is saved for later.
localIdToSession.put(new Integer(jta.getLocalId()), session);
return true;
}
```

外部トランザクションの検索

ObjectGrid TransactionCallback プラグインを使用するために、外部トランザクション・サービス・オブジェクトを検索しなければならない場合があります。WebSphere Application Server サーバーでは、次の例に示すように、ネームスペースから ExtendedJTATransaction オブジェクトを検索します。

```
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");
}
}
```

他の製品の場合は、トランザクション・サービス・オブジェクトを検索するために同じような方法を使用することができます。

外部コールバックによりコミットを制御する

TransactionCallback プラグインは、ObjectGrid セッションをコミットまたはロールバックするために、外部信号を受信する必要があります。この外部信号を受信するには、外部トランザクション・サービスからのコールバックを使用します。外部コールバック・インターフェースを実装し、それを外部トランザクション・サービスで登録する必要があります。例えば、WebSphere Application Server の場合、次の例に示すように、SynchronizationCallback インターフェースを実装する必要があります。

```
public class J2EETransactionCallback implements
com.ibm.websphere.objectgrid.plugins.TransactionCallback,
SynchronizationCallback
{
public J2EETransactionCallback() {
super();
String lookupName="java:comp/websphere/ExtendedJTATransaction";
localIdToSession = new HashMap();
try
{
InitialContext ic = new InitialContext();
jta = (ExtendedJTATransaction)ic.lookup(lookupName);
jta.registerSynchronizationCallback(this);
}
catch(NotSupportedException e)
{
throw new RuntimeException("Cannot register jta callback", e);
}
catch(NamingException e)
{
throw new RuntimeException("Cannot get transaction object");
}
}
public synchronized void afterCompletion(int localId, byte[] arg1,
boolean didCommit)
{
Integer lid = new Integer(localId);
// find the Session for the localId
Session session = (Session)localIdToSession.get(lid);
if(session != null)
{
try
{
// if WebSphere Application Server is committed when
// hardening the transaction to backingMap.
// We already did a flush in beforeCompletion
if(didCommit)
{
session.commit();
}
else
{
// otherwise rollback
session.rollback();
}
}
catch(NoActiveTransactionException e)
{
// impossible in theory
}
catch(TransactionException e)
{
// given that we already did a flush, this should not fail
}
finally
{
```

```
// always clear the session from the mapping map.
localIdToSession.remove(lid);
}
}
}
public synchronized void beforeCompletion(int localId, byte[] arg1)
{
Session session = (Session)localIdToSession.get(new Integer(localId));
if(session != null)
{
try
{
session.flush();
}
catch(TransactionException e)
{
// WebSphere Application Server does not formally define
// a way to signal the
// transaction has failed so do this
throw new RuntimeException("Cache flush failed", e);
}
}
}
}
```

TransactionCallback プラグインで ObjectGrid API を使用する

このプラグインは、ObjectGrid 用の TransactionCallback プラグインとして使用する場合、自動コミットを使用不可にします。 ObjectGrid の通常の使用パターンは以下のとおりです。

```
Session ogSession = ...;
ObjectMap myMap = ogSession.getMap("MyMap");
ogSession.begin();
MyObject v = myMap.get("key");
v.setAttribute("newValue");
myMap.update("key", v);
ogSession.commit();
```

この TransactionCallback プラグインが使用されている場合、 ObjectGrid は、コンテナー管理対象トランザクションが存在するときにアプリケーションが ObjectGrid を使用すると想定します。前出のコードの断片は、この環境で次のコードに変わります。

```
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();
}
```

myMethod メソッドは、Web アプリケーションの場合に似ています。アプリケーションは通常の UserTransaction インターフェースを使用してトランザクションを開始、コミット、およびロールバックします。 ObjectGrid はコンテナー・トランザクションなどを自動的に開始およびコミットします。メソッドが TX_REQUIRES 属性を使用する Enterprise JavaBeans (EJB) メソッドの場合は、UserTransaction 参照および UserTransaction 呼び出しを除去してトランザクションを開始、コミットすると、

メソッドが同じように動作します。この場合、コンテナーがトランザクションの開始と終了を行います。

ObjectGrid と区画化機能の統合

ObjectGridPartitionCluster サンプル・アプリケーションを使用して、 ObjectGrid と区画化機能 (WPF) を結合した機能について学習します。

ObjectGrid と区画化機能を連携させる方法の要約については、『ObjectGrid および区画化機能』を参照してください。

ObjectGrid を区画化機能とともに使用するには、ご使用の環境に WebSphere Extended Deployment がインストールされていなければなりません。

ObjectGridPartitionCluster サンプルは、 ObjectGrid と区画化機能 (WPF) を結合した機能について説明します。ObjectGrid フィーチャーは、トランザクションによって鍵と値のペアをキャッシュする容量を提供し、区画化機能フィーチャーは、オブジェクトの特性に応じてコンテキスト・ベースのルーティングの容量を提供します。

- ObjectGridPartitionCluster サンプル・アプリケーションをインストールし、実行します。 詳しくは、 322 ページの『ObjectGridPartitionCluster サンプル・アプリケーションのインストールおよび実行』を参照してください。
- サンプル・アプリケーションのソース・コードを表示または変更する場合は、エンタープライズ・アーカイブ (EAR) ファイルを開発ツールにロードすることができます。 詳しくは、 325 ページの『統合された ObjectGrid と区画化機能アプリケーションのビルド』を参照してください。
- サンプル・アプリケーションについて学習します。 サンプル・アプリケーション内のコードに関する説明については、 330 ページの『例: ObjectGrid および区画化機能プログラミング』を参照してください。

ObjectGrid を他の WebSphere Application Server フィーチャーと統合する方法についての詳細は、 303 ページの『第 10 章 ObjectGrid と WebSphere Application Server の統合』を参照してください。 ObjectGrid プログラミング・モデルについての詳細は、 95 ページの『第 9 章 ObjectGrid アプリケーション・プログラミング・インターフェースの概要』を参照してください。

ObjectGrid および区画化機能

ObjectGrid および区画化機能 (WPF) フィーチャーを連動させて、オブジェクトの特性に基づいたキーおよび値のペアおよびコンテキスト・ベースのルーティングのキャッシングを提供することができます。

ObjectGridPartitionCluster サンプルは、ObjectGrid および区画化機能 (WPF) の結合された機能を示します。 ObjectGrid および区画化機能は、WebSphere Extended Deployment 製品にある 2 つのフィーチャーです。 ObjectGrid フィーチャーは、トランザクションの状況でキャッシング・キーおよび値のペアの機能を提供し、区画化機能フィーチャーは、オブジェクトの特性に従ってコンテキスト・ベースのルーティングの機能を提供します。

このサンプルでは、Loader および TransactionCallback プラグイン・フィーチャーに加え、 ObjectGridEventListener、ObjectTransformer、および OptimisticCallback プラ

グインの使用方法についてもデモンストレーションします。特に、サンプルでは、ローカル ObjectGrid トランザクションの伝搬方法や、オプティミスティック・バージョン・チェッカーの有無に関係なく、あるサーバーから別のサーバーに変更されたオブジェクトを無効にする方法も示しています。

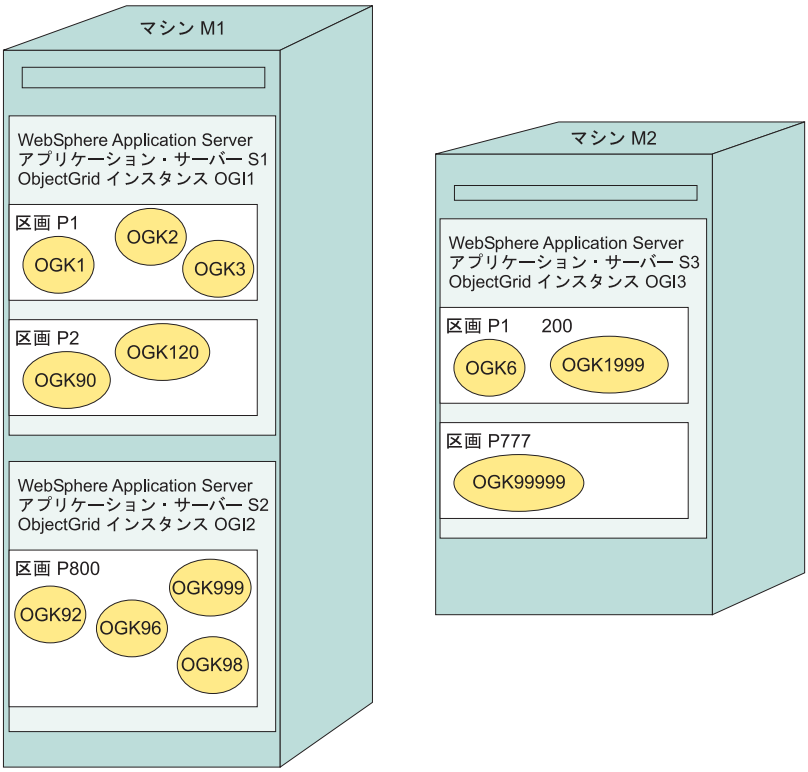
区画化機能のコンテキスト・ベース・ルーティング・フィーチャーを使用し、同じキーのオブジェクト更新、挿入、および除去要求が、同じ Java 仮想マシン (JVM) にルーティングされ、オブジェクト検索要求がワークロード管理によって ObjectGrid JVM のすべてを介して配布できることを確認する必要があります。区画化機能を使用すると、異なるクラスター・メンバーの ObjectGrid インスタンスを介してデータ保全性を維持します。

ObjectGrid の整合性および保全性を維持するには、区画化機能を使用して、大きな ObjectGrid を多くの区画化された ObjectGrid に広げることができ、区画化機能のコンテキスト・ベースのルーティングを ObjectGrid キーに従って要求を送信します。例えば、JVM ObjectGrid にフィットしない多数のオブジェクトを処理するのに ObjectGrid が必要です。区画化機能フィーチャーを使用すると、プリロードとして partitionLoadEvent メソッドで異なるサーバーにデータをロードでき、区画化機能のコンテキスト・ベース・ルーティングはお客様に合った ObjectGrid を見つけます。

サンプルでは、以下のようにハッシュ・ベースの区画および区画クラスター・ルーティング・コンテキストを作成します。

- 多対多ストラテジーで WPF 区画に ObjectGrid キーを区画化し、マップすることができます。
- WPF 区画は、多対多ストラテジーにおいて、WebSphere Application Server クラスターでホストできます。

以下の図は、ObjectGridPartitionCluster サンプルの通常の設定および構成を示します。



前の図では、M1 マシンと M2 マシンを使用して、ObjectGridPartitionCluster サンプルをデプロイしています。各物理的マシンは、1 つ以上の WebSphere Application Server をホストできます。例えば、M1 マシンは 2 つのアプリケーション・サーバー (S1 アプリケーション・サーバーと S2 アプリケーション・サーバー) をホストします。M2 マシンは 1 つのサーバー (S3 アプリケーション・サーバー) をホストします。各サーバーには、S1 アプリケーション・サーバーの OGI1ObjectGrid インスタンス、 S2 アプリケーション・サーバーの OGI2 ObjectGrid インスタンス、S3 アプリケーション・サーバーの OGI3 ObjectGrid インスタンスという、ObjectGrid インスタンスがあります。

各アプリケーション・サーバーは、多くの区画をホストできます。例えば、S1 サーバーは P1 区画および P2 区画をホストし、S3 サーバーは P1200 区画および P777 区画をホストします。

各区画は、多くの ObjectGrid キーをホストできます。例えば、P1 区画は OGK1、OGK2、および OGK3 ObjectGrid キーをホストし、P800 区画は OGK92、OGK96、OGK98、および OGK9999 区画をホストします。

ObjectGrid 更新、挿入、および除去要求のすべては、ObjectGrid キーに従ってルーティングされます。オブジェクト検索には、ワークロード管理対象ストラテジー内のサーバーから検索する、またはこのキーの特定のサーバー区画から検索するという 2 つのオプションがあります。

ObjectGridPartitionCluster サンプル・アプリケーションのインストールおよび実行

このタスクを使用して、ObjectGridPartitionCluster サンプル・アプリケーションをインストールおよび実行し、ObjectGrid と区画化機能間の機能性をテストします。

WebSphere Extended Deployment をインストールします。その手順については、WebSphere Extended Deployment Library ページを参照してください。

ObjectGridPartitionCluster サンプルを実行するために適した環境には、 WebSphere Extended Deployment の 2 つの物理マシンへのインストール、または 2 つのノードの作成とデプロイメント・マネージャーとの統合が含まれます。

1. このサンプルのフィーチャーを十分に説明するには、3 つ以上のクラスター・メンバーを持つクラスターを構成します。
2. D_ObjectGridPartitionClusterSample.ear ファイルをインストールします。 区画化機能 (WPF) のデプロイ済み D_ObjectGridPartitionClusterSample.ear ファイルは、インストールして、実行できる準備ができています。サンプル・ソース・コードを変更する場合、ビルドおよび wpf デプロイの指示に従い、エンタープライズ・アーカイブ (EAR) ファイルをビルドおよびデプロイします。

   アプリケーション EAR ファイルをインストールする一般的な方法では、管理コンソールを使用します。エンタープライズ・アプリケーションのインストール手順に従い、 D_ObjectGridPartitionClusterSample.ear ファイルをインストールします。管理コンソールのこの部分にアクセスするには、「**アプリケーション**」>「**新規アプリケーションのインストール**」をクリックします。インストール中に EAR ファイルをデプロイしないでください。インストール場所の選択を求められるステップを除き、デフォルトの設定値を使用します。このステップでは、デフォルトの server1 サーバーの代わりに、ユーザーが定義したクラスターを選択します。
3. ObjectGridPartitionClusterSample クライアントを実行します。
   a. クラスターを開始します。 管理コンソールで、「**サーバー**」>「**クラスター**」をクリックします。クラスターを選択し、「**開始**」をクリックします。
   b. **WAS_INSTALL_ROOT¥bin¥wpfadmin balance** スクリプト・コマンドを実行します。 **WAS_INSTALL_ROOT¥bin¥wpfadmin list** コマンドを使用して、区画にアクティブ状況があるかどうかを確認します。wpfadmin スクリプトおよびそのコマンドの情報については、「WebSphere Extended Deployment インフォメーション・センター」の『区画化入門』を参照してください。
   c. ObjectGridPartitionClusterSample クライアントを実行するには、以下のコマンドを実行します。

      ```
      WAS_INSTALL_ROOT/bin/launchClient.bat|sh ¥
      WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear ¥
      -CCBootstrapPort=PORT
      ```

      ここで、PORT はサーバーの RMI ポートです。これは、サーバーを始動した後の、サーバーの SystemOut.log ファイル内にあります。通常、このポートの値は、9810、9811、9812 のいずれかです。

      例えば、以下のコマンドを実行する場合があります。

```
WAS_INSTALL_ROOT/bin/launchClient.bat|sh
WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear
-CCBootstrapPort=9811
```

このスクリプトの高度な使い方については、 324 ページの『ObjectGridPartitionClusterSample アプリケーション・クライアント・オプション』を参照してください。

4. 区画の数を変更します。 ObjectGridPartitionCluster セッション・エンタープライズ Bean が作成する区画の数を変更します。PFClusterObjectGridEJB セッション Bean によって作成される区画の数は、`META-INF¥ejb-jar.xml` ファイル内にある NumberOfPartitions 環境エントリー変数によって決定されます。デフォルト値は `10` です。この環境変数の値を変更し、アプリケーションを再インストールして、異なる数の区画を作成することができます。区画数を `999999` より小さい値に設定します。
5. 分散リスナー・オプションを変更します。 以下の ObjectGrid 分散リスナー・オプションを変更することができます。

表 17. 分散リスナー・オプション

変数名	説明
enableDistribution,	EJB デプロイメント記述子内にある enableDistribution 環境エントリー変数を使用して、 ObjectGrid 分散リスナーを使用可能にすることができます。デフォルトは `true` です。つまり、使用可能です。値を `false` に設定すると、分散リスナーがオフになります。
propagationMode	EJB デプロイメント記述子内にある propagationMode 環境エントリー変数を使用して、伝搬モードを変更することができます。デフォルトは `update` です。デフォルト値を望まない場合は、値を `invalidate` に変更することができます。
propagationVersionOption,	EJB デプロイメント記述子に配置されている propagationVersionOption 環境エントリー変数を使用して、伝搬バージョンのオプションを変更することができます。デフォルトは `enable` です。この値は `disable` に設定できます。
compressionMode	EJB デプロイメント記述子内に配置されている compressionMode 環境エントリー変数を使用して、圧縮モードを変更することができます。デフォルトは `enable` です。この値は `disable` に設定できます。

デフォルトでは、バージョン・チェックで更新を伝搬します。この値を、バージョン・チェックなしで、無効化モードに設定する場合があります。

ObjectGridPartitionCluster サンプル・アプリケーションをインストールおよび実行しました。

ObjectGridPartitionClusterSample アプリケーション・クライアント・オプション

D_ObjectGridPartitionClusterSample.ear ファイルの実行で、拡張使用をするにはこのオプションを使用します。

拡張サンプル使用法

D_ObjectGridPartitionClusterSample.ear ファイルのインストールと実行についての詳細は 322 ページの『ObjectGridPartitionCluster サンプル・アプリケーションのインストールおよび実行』を参照してください。

サンプルの拡張使用については、以下の完全使用法ガイドを参照してください。

```
WAS_INSTALL_ROOT/bin/launchClient.bat|sh
WAS_INSTALL_ROOT/installableApps/D_ObjectGridPartitionClusterSample.ear
-CCproviderURL=corbaloc::HOSTNAME:SERVER_RMI_PORT [-loop LOOP] [-threads
NUMBER_OF_THREADS] [-add NUMBER_OF_STOCKS_PER_PARTITION] [-waitForPropagation
SECONDS_TO_WAIT_FOR_PROPAGATION] [-getIteration
NUMBER_OF_ITERATION_PER_OGKEY]
```

以下の変数を入力してください。

- HOSTNAME: 実行中のアプリケーション・サーバーのホスト名を指定します。
- SERVER_RMI_PORT: アプリケーション・サーバーの Bootstrap ポートを指定します。
- LOOP: クライアントが実行するループ数を指定します。このパラメーターはオプションです。デフォルト値は 1 です。
- NUMBER_OF_THREADS: クライアントが実行するスレッド数を指定します。このパラメーターはオプションです。デフォルト値は 1 です。
- NUMBER_OF_STOCKS_PER_PARTITION: 各区画に対して、追加する在庫数を指定します。このパラメーターはオプションです。デフォルトは 3 です。
- SECONDS_TO_WAIT_FOR_PROPAGATION: 他のサーバーへの伝搬のため新規に追加または更新する ObjectGrid オブジェクトを待つ時間を秒単位で指定します。デフォルトは 2 秒です。
- NUMBER_OF_ITERATION_PER_OGKEY: ワークロード管理対象形式の ObjectGrid でオブジェクト検索の反復回数を指定します。デフォルトは 6 です。より大きな反復回数を指定すると、別の WebSphere Application Server サーバーにある同一キーのオブジェクトの明確なパターンが見られます。

出力例

このコマンドの出力は以下の例のように見えます。

```
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: Processing command line arguments.
WSCL0013I: Initializing the J2EE Application Client Environment.
WSCL0035I: Initialization of the J2EE Application Client Environment has completed.
WSCL0014I: Invoking the Application
Client class com.ibm.websphere.samples.objectgrid.partitionclust
er.client.PartitionObjectGrid
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>
```

統合された ObjectGrid と区画化機能アプリケーションのビルド

ObjectGrid 区画化サンプル・アプリケーションを開き、変更し、インストールします。

次のステップを使用して、WebSphere Extended Deployment 環境で ObjectGridPartitionSample.ear ファイルを変更、エクスポート、およびインストールします。サンプル・ファイルを変更したくない場合は、デプロイ済みで、区画化機能 (WPF) 対応の D_ObjectGridPartitionClusterSample.ear ファイルを使用することができます。D_ObjectGridPartitionClusterSample.ear ファイルを使用する場合は、以下のステップを実行しないで、ファイルをインストールおよび実行することができます。どちらのエンタープライズ・アーカイブ (EAR) ファイルも WAS_INSTALL_ROOT/installableApps ディレクトリーにあります。

1. IBM Rational Application Developer バージョン 6.0.x または Application Server Toolkit バージョン 6.0.x. などのビルド環境で、ObjectGridPartitionSample.ear ファイルをセットアップします。 詳しくは、『ObjectGrid および区画化機能アプリケーションのビルドの開始』を参照してください。
2. サンプル内のソース・コードを変更します。
3. ObjectGridPartitionClusterSample アプリケーションを、EAR ファイルとして、ご使用のビルド環境からエクスポートします。 詳しくは、 328 ページの『IBM Rational Application Developer 内の ObjectGridPartitionClusterSample.ear ファイルのエクスポート』を参照してください。
4. 区画化機能と連動するように、アプリケーションをデプロイします。 詳しくは、 329 ページの『区画化機能と連動するための ObjectGridPartitionClusterSample.ear ファイルのデプロイ』を参照してください。
5. ObjectGridPartitionClusterSample.ear ファイルを WebSphere Extended Deployment にインストールします。 アプリケーション EAR ファイルをインストールする一般的な方法では、WebSphere Application Server 管理コンソールを使用します。管理コンソールのエンタープライズ・アプリケーション・インストール手順に従い、 D_ObjectGridPartitionClusterSample.ear ファイルをインストールします。インストール中は、ファイルをデプロイしないでください。代わりに、デフォルトを使用します。インストール先の選択を求められる場合を除き、各ステップでデフォルトの設定値を使用します。このステップでは、デフォルトの server1 サーバーの代わりに、ユーザーが定義したクラスターを選択します。

ObjectGridPartitionClusterSample.ear ファイルを WebSphere Extended Deployment 環境にインストールしました。

ObjectGrid、区画化機能、およびサンプル・アプリケーションを用いたプログラミングについての詳細は、 330 ページの『例: ObjectGrid および区画化機能プログラミング』を参照してください。

ObjectGrid および区画化機能アプリケーションのビルドの開始

Application Server Toolkit バージョン 6.0.x または IBM Rational Application Developer バージョン 6.0.x を使用して、サンプル・アプリケーションを再ビルドします。

WAS_INSTALL_ROOT/installableApps ディレクトリー内の ObjectGridPartitionClusterSample.ear ファイルには、すべてのソース・コードが含まれています。Application Server Toolkit バージョン 6.0.x または IBM Rational Application Developer バージョン 6.0.x を使用して、このサンプル・アプリケーションを再ビルドすることができます。このタスクでは、例として Rational

Application Developer を使用し、 ObjectGridPartitionClusterSample.ear ファイルのビルド環境を確立します。 Application Server Toolkit を使用することもできます。これは、別個の CD で、 WebSphere Application Server に同梱されるフリー・アセンブリー・ツールです。

デプロイ済みの、WPF 対応エンタープライズ・アーカイブ (EAR) ファイル D_ObjectGridPartitionClusterSample.ear (これも WAS_INSTALL_ROOT/installableApps ディレクトリーにあります) は、インストールして、実行できる準備ができています。

1. ObjectGridPartitionClusterSample.ear ファイルを Rational Application Developer にインポートします。
   a. Rational Application Developer を開始します。
   b. **オプション:** J2EE プロジェクトを操作するために、Java 2 Platform, Enterprise Edition (J2EE) パースペクティブを開きます。 「**ウィンドウ**」>「**パースペクティブのオープン**」>「**その他**」>「**J2EE**」をクリックします。
   c. **オプション:** 「Project Explorer」ビューを開きます。 「**ウィンドウ**」>「**Show View**」>「**Project Explorer**」をクリックします。この他に、「ナビゲーター」ビューが役立ちます。これを表示するには、「**ウィンドウ**」>「**Show View**」> 「**ナビゲーター**」とクリックします。
   d. ObjectGridPartitionClusterSample.ear ファイルをインポートします。「**ファイル**」>「**インポート**」>「**EAR ファイル**」をクリックし、「**次へ**」をクリックします。
   e. WAS_INSTALL_ROOT/installableApps ディレクトリーから ObjectGridPartitionClusterSample.ear ファイルを選択します。
   f. **オプション:** 「**新規作成**」をクリックして、「New Server Runtime」ウィザードを開き、指示に従います。
   g. 「ターゲット・サーバー」フィールドで、サーバー・ランタイムのタイプ「**WebSphere Application Server V6.0**」を選択します。
   h. 「**終了**」をクリックします。

   ObjectGridPartitionClusterSample、ObjectGridPartitionClusterSampleEJB、および ObjectGridPartitionClusterSampleClient プロジェクトが作成され、「Project Explorer」ビューに表示されなければなりません。
2. ObjectGridPartitionClusterSampleEJB プロジェクトをセットアップします。
   a. J2EE パースペクティブの「Project Explorer」ビューで、EJB プロジェクトの **ObjectGridPartitionClusterSampleEJB** プロジェクトを右クリックし、「**プロパティー**」を選択します。 「プロパティー」ウィンドウが表示されます。
   b. 左側のパネルで「**Java ビルド・パス**」をクリックし、右側のパネルで「**ライブラリー**」タブをクリックし、「**変数の追加**」を選択します。 「New Variable Classpath Entry」ウィンドウが表示されます。
   c. 「**変数の構成**」をクリックして、「設定」ウィンドウを開きます。
   d. 「**新規作成**」をクリックして「New Variable Entry」ウィンドウを開きます。
   e. 「**名前**」に ObjectGridPartitionCluster_JAR と入力し、「**ファイル**」をクリックして「JAR Selection」ウィンドウを開きます。

f. WAS_INSTALL_ROOT/lib ディレクトリーを参照し、**wsobjectgrid.jar** を選択します。「**オープン**」をクリックして、「JAR Selection」ウィンドウを閉じます。

g. 「**OK**」をクリックして、「New Variable Entry」ウィンドウを閉じます。ObjectGridPartitionCluster_JAR 変数が、クラスパス変数リストに表示されます。

h. 「**OK**」をクリックして「設定」ウィンドウを閉じます。

i. 変数リストから **ObjectGridPartitionCluster_JAR** 変数を選択し、「**OK**」をクリックして「New Variable Classpath Entry」ウィンドウを閉じます。ObjectGridPartitionCluster_JAR 変数が、「ライブラリー」パネルに表示されます。

j. この手順を繰り返して、wpf.jar ファイルを環境に追加します。

k. wpf.jar ファイルおよび wsobjectgrid.jar ファイルが、ビルド・クラスパスにあることを確認します。

ビルド環境がセットアップされると、ソース・コードを変更し、他の変更を適用することができるようになります。詳しくは、325 ページの『統合された ObjectGrid と区画化機能アプリケーションのビルド』を参照してください。

IBM Rational Application Developer 内の ObjectGridPartitionClusterSample.ear ファイルのエクスポート

サンプル・ファイルを変更した後、ObjectGridPartitionClusterSample アプリケーションをエクスポートして、 WebSphere Extended Deployment サーバーにインストールできるエンタープライズ・アーカイブ (EAR) ファイルを作成することができます。

ソースに変更を加えられるようにするために、ObjectGridPartitionSample.ear ファイルを開発ツールにインポートしておく必要があります。詳しくは、326 ページの『ObjectGrid および区画化機能アプリケーションのビルドの開始』を参照してください。エクスポートする前に、サンプル・アプリケーションの変更を行います。

ObjectGridPartitionClusterSample.ear ファイルを、IBM Rational Application Developer のエンタープライズ・アプリケーション内の ObjectGridPartitionClusterSample プロジェクトからエクスポートすることができます。区画化機能をデプロイした後、エクスポートした ObjectGridPartitionClusterSample.ear ファイルを、任意の WebSphere Extended Deployment バージョン 6.0 サーバーにインストールできます。

1. Java 2 Platform, Enterprise Edition (J2EE) パースペクティブの「Project Explorer」ビューで、「エンタープライズ・アプリケーション」の下にある **ObjectGridPartitionClusterSample** アプリケーションを右クリックします。「**エクスポート**」>「**EAR ファイル**」をクリックします。 「エクスポート」ウィンドウが表示されます。
2. 「**参照**」をクリックして、「**別名保管**」ウィンドウを開きます。ターゲット出力ディレクトリーを位置指定し、ファイル名を ObjectGridPartitionClusterSample と指定して、「**保管**」をクリックします。

3. 「**参照**」をクリックして、「別名保管」ウィンドウを開きます。ターゲット出力ディレクトリーを位置指定し、ファイル名を ObjectGridPartitionClusterSample と指定します。「**保管**」をクリックします。

ObjectGridPartitionClusterSample.ear ファイルが、指定されたターゲット出力ディレクトリーに作成されます。

区画化機能 (WPF) の ObjectGridPartitionClusterSample.ear ファイルをデプロイした後、WebSphere Extended Deployment 内のファイルを実行することができます。詳しくは、『区画化機能と連動するための ObjectGridPartitionClusterSample.ear ファイルのデプロイ』を参照してください。

区画化機能と連動するための ObjectGridPartitionClusterSample.ear ファイルのデプロイ

ObjectGridPartitionClusterSample.ear ファイルを WebSphere Extended Deployment にインストールする予定がある場合は、そのファイルに対して wpf デプロイ操作を実行する必要があります。

既存の ObjectGridPartitionClusterSample.ear ファイルが必要です。既存ファイルを変更するには、325 ページの『統合された ObjectGrid と区画化機能アプリケーションのビルド』を参照してください。

wpf デプロイ操作を実行して、WebSphere Extended Deployment 環境でエンタープライズ・アーカイブ (EAR) ファイルを作成します。

1. DEST_DIR ディレクトリーを作成します。
2. ObjectGridPartitionClusterSample.ear ファイルを DEST_DIR ディレクトリーにコピーします。ObjectGridPartitionClusterSample.ear ファイルの名前を old_ObjectGridPartitionClusterSample.ear ファイルに変更します。
3. 以下のコマンドを実行します。ここで、WORKING_DIR は、ejbdeloy ツールの作業ディレクトリーで、例えば c:¥temp ディレクトリーです。

   ```
   WAS_HOME¥bin¥ejbdeploy.bat|ejbdeploy.sh
   DEST_DIR¥old_ObjectGridPartitionClusterSample.ear WORKING_DIR
   DEST_DIR¥ObjectGridPartitionClusterSample.ear
   ```

4. 以下のコマンドを実行します。ここで、TEMP_DIR はツールの一時ディレクトリーです。 -keep 引数が指定されている場合、 wpfStubUtil ユーティリティーによって作成された一時ディレクトリーは削除されません。

   ```
   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]
   ```

ObjectGridPartitionClusterSample.ear ファイルは、 WebSphere Extended Deployment 環境で実行できる準備ができています。WebSphere Extended Deployment に同梱されている D_ObjectGridPartitionClusterSample.ear サンプル・ファイルは、デプロイ済みです。ソース・コードを変更しなかった場合、アプリケーションをインストールする前に、このファイルをデプロイする必要はありません。

管理コンソールを使用して、ObjectGridPartitionClusterSample.ear ファイルを WebSphere Extended Deployment 環境にインストールします。詳しくは、 325 ページの『統合された ObjectGrid と区画化機能アプリケーションのビルド』を参照してください。

例: ObjectGrid および区画化機能プログラミング

この例は、WebSphere Extended Deployment の Java 2 Platform, Enterprise Edition 環境で、 ObjectGrid と区画化機能の結合機能の使用方法を説明します。

目的

結合した ObjectGrid と区画化機能 (WPF) の機能を説明するほか、この例では ObjectGrid 分散リスナー伝搬と無効化について説明します。

オブジェクトの更新、挿入、除去の要求は、一致する ObjectGrid キーに応じて区画がホストされる特定サーバーに送信されます。ObjectGrid get メソッド要求は、すべてのサーバー間で管理されるワークロードです。

この例はまた、大きな ObjectGrid を多くの小さな ObjectGrids に区画化する方法と、データをプリロードするのに partitionLoadEvent メソッドを使用して、区画化された ObjectGrid が無制限にオブジェクトをホストできるようにする方法を説明しています。

概要

ObjectGridPartitionClusterSampler.ear ファイルは、 ObjectGrid と区画化機能が連動する様子を説明する在庫オブジェクトを作成します。在庫オブジェクトは以下のプロパティーを含んでいます。

- チケット
- 会社
- serialNumber
- 説明
- lastTransaction
- 価格

ここで、lastTransaction プロパティーは在庫が変更された時刻です。lastTransaction プロパティーを使用して別の Java 仮想マシン (JVM) の ObjectGrid にあるオブジェクトの新鮮度を示します。

この例では、 ObjectGrid インスタンスは ObjectGridFactory クラスを使用して Enterprise JavaBeans (EJB) setContext メソッドに作成されます。

ハッシュ・ベースの区画のセットを定義してください。デフォルト値は 10 区画ですが、区画数の変更は可能です。SampleUtility.java ファイルを使用して、在庫チケットをこれらの区画にハッシュしてください。それぞれの区画は ObjectGrid キーと値の組を多数ホストすることができます。

この例は、 ObjectGrid 挿入、更新、除去要求が区画化された特定のサーバーに送られる方法、 ObjectGrid get メソッドがそのキーに応じた特定のサーバーまたはクラ

スター内の任意のサーバーへの経路を要求する方法を説明します。この例では、特定のサーバーでこのキーにより発生した更新、挿入または除去操作のため値を変更した後、異なるサーバーから受信したキーのオブジェクト値が比較されます。

ロケーション

それぞれのサーバーが多数の区画をホストでき、それぞれの区画が異なるキーを持つ多数のオブジェクトをホストできるクラスター環境でこの例を使用してください。

<install_root>¥installableApps¥ ディレクトリーには、次の 2 つの ObjectGrid 区画クラスター・サンプル・ファイルがあります。

- ObjectGridPartitionClusterSampler.ear ファイルにはソース・コードが含まれています。ソースを見るには、EAR ファイルをファイル・システムに展開するか、ソースを開発環境にインポートするかします。詳しくは、325 ページの『統合された ObjectGrid と区画化機能アプリケーションのビルド』を参照してください。
- D_ObjectGridPartitionClusterSample.ear ファイルは、区画化機能のために既にデプロイされています。 README ファイルと指示を読み、このファイルを高速実行してください。

説明

以下のセクションで、 ObjectGrid 区画クラスター・サンプル・アプリケーションについて説明します。

- 『ObjectGrid オペレーション EJB インターフェース』
- 333 ページの『PartitionKey クラス』
- 335 ページの『SampleUtility クラスおよび区画マッピング』
- 337 ページの『エンタープライズ Bean setContext メソッドでの ObjectGrid の作成』
- 339 ページの『Singleton ObjectGridFactory クラス』
- 340 ページの『ObjectGrid 区画のプリロード』

ObjectGrid オペレーション EJB インターフェース

この項目は、取得、区画化されたサーバーからの取得、挿入、更新、除去操作を実行する、ObjectGrid オペレーション Enterprise JavaBeans (EJB) インターフェースについて説明します。

目的

ObjectGrid オペレーション EJB インターフェースは、取得、区画化されたサーバーからの取得、挿入、更新、除去操作を実行します。区画化されたサーバーからの取得メソッドは、要求するキーに対応する区画に送られます。get メソッドはワークロード管理対象ストラテジーで任意のサーバーに送られます。

PFClusterObjectGridEJB インターフェース

以下は PFClusterObjectGridEJB インターフェースのコンテンツです。

```
/**
* Remote interface for Enterprise Bean: PFClusterObjectGridEJB
*/
public interface PFClusterObjectGridEJB extends javax.ejb.EJBObject {
public String PARTITION_PREFIX = "ObjectGridHashPartition";
/**
* Get all Partitions
*
* @return Array of Strings
* @throws java.rmi.RemoteException
*/
public String [] getAllPartitions() throws java.rmi.RemoteException;
/**
* Get where partition is hosted
*
* @param partition
* @return String
* @throws java.rmi.RemoteException
*/
public String getServer(String partition)
throws java.rmi.RemoteException;
/**
* Get Stock object and its server information
* (ServerIDResult) for a stock ticket
* from any server in a cluster (that has had workload management)
*
* @param ticket
* @return
* @throws java.rmi.RemoteException
*/
public ServerIDResult getStock(String ticket)
throws java.rmi.RemoteException;
/**
* Get Stock object and its partitioned server
* information for a stock ticket
* from the partition this ticket key is hashed to
*
* @param ticket
* @return ServerIDResult
* @throws java.rmi.RemoteException
*/
public ServerIDResult getStockOnPartitionedServer(String ticket)
throws java.rmi.RemoteException;
/**
* Update stock in a particular server where the partition is
* active for this stock ticket key.
*
* @param stock
* @return ServerIDResult
* @throws java.rmi.RemoteException
*/
public ServerIDResult updateStock(Stock stock)
throws java.rmi.RemoteException;
/**
* Remove stock in a particular server where the partition is
* active for this stock ticket key.
*
* @param ticket
* @return ServerIDResult
* @throws java.rmi.RemoteException
*/
public ServerIDResult removeStock(String ticket)
throws java.rmi.RemoteException;
/**
* Insert stock in a particular server where the partition is
* active for this stock ticket key.
*
```

```
* @param stock
* @return ServerIDResult
* @throws java.rmi.RemoteException
*/
public ServerIDResult insertStock(Stock stock)
throws java.rmi.RemoteException;
/**
* Retrieve data from all servers and compare values
*
* @param server
* @return ServerObjectGridVerification
* @throws java.rmi.RemoteException
*/
public ServerObjectGridVerification verifyObjectGrid(String server)
throws java.rmi.RemoteException;
}
```

PartitionKey クラス

PartitionKey クラスは区画化機能コンテキスト・ベースのルーティングの振る舞いを制御します。

以下のコードはサンプル区画キー・クラスを説明しています。メソッドが not null を戻すと、それは区画化機能 (WPF) ルーターにより送付されます。メソッドが null を戻した場合は、ワークロード管理 (WLM) ルーターに転送されます。

```
/**
* PartitionKey for Partitioned Stateless Session Bean WPFKeyBasedPartition
*
*/
public class PFClusterObjectGridEJB_PartitionKey {
/**
* Number of Partitions
*
* Default is 10.
*
*/
static int numOfPartitions=10;
/**
* Only once to getPartitionNumbers
*/
static boolean getNumOfPartitions=true;
/**
* Get the number of partitions
*
*/
static void getPartitionNumbers(){
//get only once
if (getNumOfPartitions){
try {
InitialContext ic = new InitialContext();
PFClusterObjectGridEJBHome home =
(PFClusterObjectGridEJBHome) PortableRemoteObject.narrow(
ic.lookup("java:comp/env/ejb/PFClusterObjectGridEJB"),
PFClusterObjectGridEJBHome.class);
final PFClusterObjectGridEJB session = home.create();
String[] PARTITIONS = session.getAllPartitions();
numOfPartitions=PARTITIONS.length;
getNumOfPartitions=false;
}
catch (ClassCastException e) {
e.printStackTrace();
numOfPartitions=10;
}
catch (RemoteException e) {
e.printStackTrace();
```

```
numOfPartitions=10;
}
catch (NamingException e) {
e.printStackTrace();
numOfPartitions=10;
}
catch (CreateException e) {
e.printStackTrace();
numOfPartitions=10;
}
}
}
/**
* Return partition key
*
* @param partition
* @return String
*/
public static String getStock(String key) {
return null;
}
/**
* Return partition key
*
* @param key
* @return String
*/
public static String getServer(String key) {
return key;
}
/**
* Retrieve ObjectGrid data from a partitioned server where
* data's changes happen (the highest quality and integrity).
*
* @param ticket
* @return hashcode of stock ticket
*/
public static String getStockOnPartitionedServer(String ticket) {
if (ticket==null){
return null;
}
getPartitionNumbers();
return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Return partition key
*
* @param stock
* @return hashcode of stock ticket
*/
public static String updateStock(Stock stock) {
getPartitionNumbers();
String ticket=null;
if (stock!=null){
ticket=stock.getTicket();
}
return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Return partition key
*
* @param stock
* @return hashcode of stock ticket
*/
public static String insertStock(Stock stock) {
getPartitionNumbers();
String ticket=null;
```

```
if (stock!=null){
ticket=stock.getTicket();
}
return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Return partition key
*
* @param server
* @return String
*/
public static String verifyObjectGrid(String server) {
return server;
}
/**
* Return partition key
*
* @param stock
* @return hashcode of stock ticket
*/
public static String removeStock(String ticket) {
if (ticket==null){
return null;
}
getPartitionNumbers();
return SampleUtility.hashStockKeyToPartition(ticket, numOfPartitions);
}
/**
* Return partition key
*
* @param partition
* @return
*/
public static String getAllPartitions() {
return null;
}
}
```

それぞれのリモート・メソッドには、有効なストリングまたはヌル値を戻す、対応するメソッドが存在する必要があります。

SampleUtility クラスおよび区画マッピング

SampleUtility.java ファイルを使用して、鍵、在庫チケット、ハッシュ、および区画を取り扱います。このファイルは、ObjectGrid 鍵を区画にマップするためにも使用できます。業務の必要性に合わせて、ObjectGrid 鍵を区画にマップするために、類似したユーティリティー・クラスを作成することができます。ObjectGrid を使用した区画化機能を使用するには、異なる鍵を異なる区画にマップする必要があります。

SampleUtility クラス

以下に ObjectGridPartitionCluster サンプルのユーティリティー・クラスを示します。

```
/**
* Utility class for ObjectGridPartitionCluster sample
*
*
*/
public class SampleUtility {
/**
* Container for recording partitions.
```

```
*/
static Map serverPartitions= new HashMap();
/**
* Partition name prefix
*/
public static String PARTITION_PREFIX = "ObjectGridHashPartition";
/**
* Stock name prefix
*/
public static String STOCK_PREFIX="Stock";
/**
* Retrieve the number part of partition name
*
* @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;
}
/**
* Retrieve the number part of stock ticket
*
* @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;
}
/**
* Hash stock ticket to a given hash base.
*
* @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;
}
/**
* Hash stock key to a partition
*
* @param ticket
* @param base
* @return String - partition name
*/
public static String hashStockKeyToPartition(String ticket, int base){
String p=null;
int hashcode=hashTicket(ticket, base)+1;
p=PARTITION_PREFIX+ padZeroToString(hashcode+"", 6);
return p;
}
```

```
/**
* Record server/partition
*
* @param server
* @param partition
*/
public static void addServer(String server, String partition){
serverPartitions.put(server, partition);
}
/**
* Remove server/partition
*
* @param server
*/
public static void removeServer(String server){
serverPartitions.remove(server);
}
/**
* Get all servers where partitions are active.
*
* @return Iterator - String
*/
public static Iterator getAllServer(){
return serverPartitions.values().iterator();
}
}
```

同じグローバル・ハッシュ・ベースおよび構文解析変数を使用して、ハッシュ・ベースにハッシュする必要があります。次の例について考えてみます。

```
myKey.hashCode % hashBase
```

ハッシュ変数として myKey を構文解析し、異なるサーバー間で同じハッシュ・ベースを保持する必要があります。この例では、Java 環境からの同じ変数が検索されます。 `key1 % 100` は使用できませんが、 `key2 % 90` は使用可能です。

エンタープライズ Bean setContext メソッドでの ObjectGrid の作成

PFClusterObjectGridEJBBean.java ファイルで行ったように、エンタープライズ Bean setContext メソッドに ObjectGrid インスタンスを作成し、プリロード・データを検索してください。

```
/**
* setSessionContext
*
* with ObjectGrid instance
*/
public void setSessionContext(javax.ejb.SessionContext ctx) {
mySessionCtx = ctx;
try {
InitialContext ic = new InitialContext();
//get PartitionManager
ivManager = (PartitionManager)
ic.lookup("java:comp/websphere/wpf/PartitionManager");
// get enableDistribution configuration
boolean enableDistribution = ((Boolean)
ic.lookup("java:comp/env/enableDistribution")).booleanValue();
System.out.println("***** enableDistribution="+ enableDistribution);
// get propagationMode configuration
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;
}
// get propagationVersionOption configuration
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;
}
// get compressionMode configuration
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;
}
// whethere preload is enabled
bPreload = ((Boolean)
ic.lookup("java:comp/env/preload")).booleanValue();
System.out.println("***** enablePreload="+ bPreload);
//whethere remove is enabled
bRemove = ((Boolean)
ic.lookup("java:comp/env/remove")).booleanValue();
System.out.println("***** enableRemove="+ bRemove);
// whethere Loader is enabled
boolean bLoader = ((Boolean)
ic.lookup("java:comp/env/loader")).booleanValue();
System.out.println("***** enableLoader="+ bLoader);
// get file path and name
String filePathandName = (String)
ic.lookup("java:comp/env/filePathandName");
System.out.println("***** fileName="+ filePathandName);
//get ObjectGrid instance
og=ObjectGridFactory.getObjectGrid(ogName,
enableDistribution,  pMode, pVersion,
compressMode, bLoader,
filePathandName);
if (og==null){
throw new RuntimeException
("ObjectGrid insance is null in ObjectGridPartitionClusterSample");
}
System.out.println("Bean Context, getObjectGrid="
+ og + " for name="+ ogName);
if (bPreload && !lock){
System.out.println("Preload data");
PersistentStore store=PersistentStore.getStore(filePathandName);
store.preload(10);
store.verify(10);
lock=true;
preloadData=store.getAllRecords();
}
}
catch (Exception e) {
logger.logp(Level.SEVERE, CLASS_NAME,
```

```
"setSessionContext", "Exception: " + e);
throw new EJBException(e);
}
}
```

Singleton ObjectGridFactory クラス

カスタム設定で ObjectGrid インスタンスをキャッシュに入れるカスタム・ファクトリーにより、 ObjectGrid インスタンスが作成されます。

以下は、ObjectGrid インスタンスのプログラマチックな作成方法、ObjectGridTransformer オブジェクトの設定方法、伝搬イベント・リスナーの構成方法、このリスナーを ObjectGrid インスタンスに設定する方法の例です。 XML ファイルを使用してこの構成を実行することもできます。

```
/**
*
* Create ObjectGrid instance and configure it.
*
*
*/
public class ObjectGridFactory {
/**
* ObjectGrid name
*/
static String ogName="WPFObjectGridSample";
/**
* ObjectGrid instance
*/
static ObjectGrid og=null;
/**
* ObjectGrid session
*/
static Session ogSession=null;
/**
* Map name
*/
static String mapName="SampleStocks";
/**
* ObjectGrid cache
*/
static Map ogCache= new HashMap();
/**
* Get ObjectGrid instance
*
* @param ogn
* @param enableDist
* @param pMode
* @param pVersion
* @param compressMode
* @return
*/
public static synchronized ObjectGrid getObjectGrid(String ogn,
boolean enableDist,
String pMode,
String pVersion,
String compressMode,
boolean loader,
String fileName){
if (ogn!=null){
ogName=ogn;
}
else {
throw new IllegalArgumentException ("ObjectGrid name given is null");
}
```

```
if (ogCache.containsKey(ogName)){
return (ObjectGrid) ogCache.get(ogName);
}
try {
ObjectGridManager manager= ObjectGridManagerFactory.
getObjectGridManager();
og=manager.createObjectGrid(ogName);
if (enableDist){
TranPropListener tpl=new TranPropListener();
if (pMode!=null){
tpl.setPropagateMode(pMode);
}
if (pVersion!=null){
tpl.setPropagateVersionOption(pVersion);
}
if (compressMode!=null) {
tpl.setCompressionMode(compressMode);
}
og.addEventListener(tpl);
}
// Define BackingMap and set the Loader
BackingMap bm = og.defineMap(mapName);
ObjectTransformer myTransformer=
new MyStockObjectTransformer();
bm.setObjectTransformer(myTransformer);
OptimisticCallback myOptimisticCallback=
new MyStockOptimisticCallback();
if (loader){
TransactionCallback tcb=new MyTransactionCallback();
Loader myLoader= new MyCacheLoader(fileName, mapName);
og.setTransactionCallback(tcb);
bm.setLoader(myLoader);
}
og.initialize();
ogCache.put(ogName, og);
}
catch (Exception e) {
}
return og;
}
}
```

ObjectGrid 区画のプリロード

このトピックでは、ObjectGrid インスタンスのプリロードの方法について説明します。

区画が活動状態のときのみ、partitionLoadEvent メソッドを使用してこの区画に関連するオブジェクトをロードしてください。区画化が活動状態のときにオブジェクトをロードすることにより、ObjectGrid を区画化して、ObjectGrid が多数のオブジェクトを処理できるようにします。

```
/**
* This is called when a specific partition is assigned to this server process.
* @param partitionName
* @return
*/
public boolean partitionLoadEvent(String partitionName) {
//preload data
preloadDataForPartition(partitionName);
logger.logp(
Level.FINER,
CLASS_NAME,
"partitionLoadEvent",
"Loading "+ partitionName );
```

```
return true;
}
/**
*
* preload data
*
* @param partition
*/
private synchronized void preloadDataForPartition(String partition){
if (bPreload && (preloadData!=null)){
Iterator itr=preloadData.keySet().iterator();
while (itr.hasNext()){
String ticket= (String) itr.next();
String p=SampleUtility.
hashStockKeyToPartition(ticket, numOfPartitions);
if (partition.equals(p)){
Stock stock= (Stock) preloadData.get(ticket);
System.out.println("preload in partition=" +
partition + " with data ticket="+ ticket);
insertStock(stock);
}
}
}
}
```

大きな ObjectGrid を区画化するために大きな ObjectGrid の区画化されたプリロードを使用する場合、分散更新を使用不可にしなければならないことがあります。分散更新の現行バージョンは区画化することができません。区画化機能 (WPF) コンテキスト・ベースのルーティングは、正しい区画で正しいデータを検索します。

コンテナー管理の Bean と連動する ObjectGrid の構成

WebSphere Application Server バージョン 6.0.2 以降では、外部キャッシュ製品と共にコンテナー管理パーシスタンス (CMP) Bean を使用できます。

このタスクを使用して CMP Bean を使用し、組み込みキャッシュではなく外部キャッシュとしての ObjectGrid を活用します。この機能は、WebSphere Application Server のパーシスタンス・エンジンにより提供されます。

1. JVM 引数を定義して、CacheFactoryManager アダプター、および ObjectGrid XML 構成ファイルのロケーションを定義します。 CacheFactoryManager は、パーシスタンス・エンジンと ObjectGrid の間のアダプターです。
   a. 「**サーバー**」>「**アプリケーション・サーバー**」>「*server_name*」 > 「**Java およびプロセス管理**」>「**プロセス定義**」>「**Java 仮想マシン**」>「**汎用 JVM 引数**」をクリックします。
   b. 以下のプロパティーを追加します。
      - `-Dcom.ibm.ws.pmcache.manager=com.ibm.ws.objectgrid.adapter.pm.CacheFactoryManager`
      - `-Dcom.ibm.ws.pmcache.config=file:/d:/temp/objectGrid.xml`

      `-Dcom.ibm.ws.pmcache.config` プロパティーは ObjectGrid の構成ファイルを指定します。値は ObjectGrid 構成ファイルへの URL です。
2. ObjectGrid XML 構成ファイルを構成します。 構成は `objectGrid.xml` ファイルにあります。 次の例について考えてみます。 Java 2 Platform, Enterprise Edition (J2EE) アプリケーションにはアプリケーションおよびモジュールの各情報が必要です。その情報は、`objectGrid.xml` ファイルに反映されます。ある口座

(Accounts) アプリケーションには、Savings、Checkin、および MoneyMarket という 3 つの CMP Enterprise JavaBeans があります。これらの Enterprise JavaBeans は、PersonalBankingEJB モジュールに含まれています。表示名は *Accounts* であり、*PersonalBankingEJB* はアプリケーション・デプロイメント記述子の EJB モジュールです。 Savings、Checkin、および MoneyMarket は、コンテナー管理のエンティティー Bean (CMP) に対するエンタープライズ Java Bean デプロイメント記述子の ejb-name エレメントで指定される名前です。この構成でのサンプルの断片は以下のとおりです。

```
<ObjectGrids>
<ObjectGrid name="Accounts">
<BackingMap name="PersonalBankingEJB.jar#Savings" readOnly="true"
pluginCollectionRef="default" />
<BackingMap name="PersonalBankingEJB.jar#Checkin" readOnly="true"
pluginCollectionRef="default" />
<BackingMap name="PersonalBankingEJB.jar#MoneyMarket" readOnly="true"
pluginCollectionRef="default" />
</ObjectGrid>
</ObjectGrids>
```

PersonalBankingEJB.jar ファイルは、以下の例のように、アプリケーション・デプロイメント記述子の EJB タグ内で指定されます。

```
<module id="module_1">
<ejb>PersonalBankingEJB.jar</ejb>
</module>
```

3. それぞれの Bean ごとに、外部キャッシュを使用するアプリケーション内で、パーシスタンス・マネージャーの LifeTimeInCache 設定を使用可能にします。ObjectGrid では、デプロイメント記述子でこの設定を使用可能にする必要がありますが、 LifeTimeInCache 設定は無視されます。 ObjectGrid 構成が優先されます。
4. キャッシュからオブジェクトを除去するよう backingMap を明示的に構成します。 objectGrid.xml ファイルの XML コードの断片は以下のとおりです。

```
<backingMapPluginCollection id="TotalTimeToLive">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.TTLEvictor">
<property name="pruneSize" type="int" value="2"
description="set max size for TTL Evictor" />
<property name="numberOfHeaps" type="int" value="1"
description="set number of TTL heaps" />
<property name="sleepTime" type="int" value="1"
description="evictor thread sleep time" />
</bean>
</backingMapPluginCollection>
<backingMapPluginCollection id="LifeTimeInCache">
<bean id="Evictor"
className="com.ibm.websphere.objectgrid.plugins.builtins.TTLEvictor">
<property name="lifeTime" type="int" value="3"
description="lifetime of map entry is 3 seconds" />
<property name="pruneSize" type="int" value="2"
description="set max size for TTL Evictor" />
<property name="numberOfHeaps" type="int" value="1"
description="set number of TTL heaps" />
<property name="sleepTime" type="int" value="1"
description="evictor thread sleep time" />
</bean>
</backingMapPluginCollection>
```

lifeTime プロパティーが Evictor を制御します。

WebSphere Application Server で ObjectGrid を使用する方法について詳しくは、303 ページの『第 10 章 ObjectGrid と WebSphere Application Server の統合』を参照してください。

第 11 章 ObjectGrid パフォーマンスのベスト・プラクティス

次のベスト・プラクティスによって、ObjectGrid Map のパフォーマンスを向上させることができます。これらのベスト・プラクティスは、アプリケーションとそのアーキテクチャーのコンテキスト内でのみ、実行されます。

各アプリケーションと環境で、パフォーマンスに対して別々のソリューションを使用します。 ObjectGrid は、パフォーマンスを向上させるために標準装備のカスタマイズを提供しますが、アプリケーション・アーキテクチャー内でもパフォーマンスを向上させることができます。次の領域ではパフォーマンスの向上を提案しています。

- 『ロック・パフォーマンスのベスト・プラクティス』

  アプリケーションのパフォーマンスに影響を与えることができる、さまざまなロック・ストラテジーの中から選択します。
- 346 ページの『copyMode メソッドのベスト・プラクティス』

  ObjectGrid がエントリーを維持し、コピーする方法を変更するのに使用できる、様々なコピー・モードの中から選択します。
- 351 ページの『ObjectTransformer インターフェースのベスト・プラクティス』

  ObjectTransformer インターフェースによって、アプリケーションへのコールバックが可能になり、普通の操作および、オブジェクト直列化およびオブジェクトに対するディープ・コピーなどの高価な操作のカスタム実装が提供されます。
- 352 ページの『プラグイン Evictor パフォーマンスのベスト・プラクティス』

  least frequently used (LFU) 除去ストラテジーと least recently used (LRU) 除去ストラテジーの中から選択します。
- 354 ページの『デフォルト Evictor のベスト・プラクティス』

  デフォルトの存続時間 (TTL) Evictor のプロパティー、各 backingMap によって作成されるデフォルト Evictor。

ロック・パフォーマンスのベスト・プラクティス

ロック・ストラテジーがご使用のアプリケーションのパフォーマンスに影響する場合があります。

以下のロック・ストラテジーについて詳しくは、134 ページの『ロック』のトピックを参照してください。

ペシミスティック・ロック・ストラテジー

キーがよく衝突する場合のマップの読み取りおよび書き込み操作には、ペシミスティック・ロック・ストラテジーを使用することができます。ペシミスティック・ロック・ストラテジーは、パフォーマンスに最大の影響があります。

オプティミスティック・ロック・ストラテジー

オプティミスティック・ロックはデフォルト構成です。このストラテジーはペシミスティック・ストラテジーに比べ、パフォーマンスにおいてもスケーラビリティーにおいても優れています。ご使用のアプリケーションが若干のオプティミスティック更新の失敗を許容でき、それでもペシミスティック・ストラテジーよりもパフォーマンスに優れている場合は、このストラテジーを使用してください。このストラテジーは、ほとんど読み取りのみで、更新が頻繁ではないアプリケーションに対して大変有効です。

ロックなしストラテジー

ロックなしストラテジーの使用は、読み取り専用アプリケーションの場合に有用です。ロックなしストラテジーはいかなるロックも取得しません。このため、このストラテジーは最も並行性、パフォーマンス、スケーラビリティーに優れています。

copyMode メソッドのベスト・プラクティス

ObjectGrid は、CopyMode 設定に基づいて値をコピーします。 BackingMap API setCopyMode(CopyMode, valueInterfaceClass) メソッドを使用して、com.ibm.websphere.objectgrid.CopyMode クラスで定義される、次の最終の静的フィールドの 1 つに、コピー・モードを設定することができます。

アプリケーションが ObjectMap インターフェースを使用してマップ値への参照を取得する場合、その参照は、それを取得した ObjectGrid トランザクション内でのみ使用することをお勧めします。別の ObjectGrid トランザクションでその参照を使用するとエラーになることがあります。例えば BackingMap に対してペシミスティック・ロック・ストラテジーを使用する場合、get メソッド呼び出しまたは getForUpdate メソッド呼び出しにより、それぞれ S (shared) ロックまたは U (update) ロックを取得します。トランザクションの終了時に get メソッドは値に参照を戻し、取得されているロックは解放されます。トランザクションは get メソッドまたは getForUpdate メソッドを呼び出して、別のトランザクションでマップ・エントリーをロックする必要があります。各トランザクションは、複数のトランザクションで同じ値参照を再利用する代わりに get メソッドまたは getForUpdate メソッドを呼び出すことにより、値への独自の参照を取得する必要があります。

次の情報を使用して、各種コピー・モードの中から選択してください。

COPY_ON_READ_AND_COMMIT モード

COPY_ON_READ_AND_COMMIT モードはデフォルトのモードです。このモードが使用される場合、valueInterfaceClass 引数は無視されます。このモードでは、アプリケーションは BackingMap にある値オブジェクトへの参照を含まず、代わりに、BackingMap にある値のコピーと常に連携しています。
COPY_ON_READ_AND_COMMIT モードでは、BackingMap でキャッシュされるデータをうっかり壊してしまうことはありません。アプリケーションのトランザクションが指定されたキーの ObjectMap.get メソッドを呼び出し、それがそのキーにとって、ObjectMap エントリーへの初めてのアクセスの場合は、値のコピーが戻されます。トランザクションがコミットされると、アプリケーションによってコミットされたすべての変更は BackingMap にコピーされ、アプリケーションが

BackingMap に、コミットされた値への参照を持つことはありません。

COPY_ON_READ モード

COPY_ON_READ モードは、トランザクションがコミットされたときに発生するコピーを除去することによって、COPY_ON_READ_AND_COMMIT モード全体にわたるパフォーマンスを改善します。このモードが使用される場合、valueInterfaceClass 引数は無視されます。 BackingMap データの整合性を保持するために、アプリケーションは、エントリーに対する各参照がトランザクションのコミット後に破棄されることを保証します。このモードでは、ObjectMap.get メソッドは、値への参照の代わりに値のコピーを戻し、アプリケーションが値に対して行った変更は、トランザクションがコミットされるまで BackingMap 値に影響を与えないことを保証します。ただし、トランザクションがコミットすると変更のコピーは行われません。代わりに、ObjectMap.get メソッドによって戻された、コピーへの参照が BackingMap に保管されます。トランザクションがコミットされた後、アプリケーションはすべてのマップ・エントリー参照を破棄します。アプリケーションがこれに失敗すると、データを BackingMap 内でキャッシュし、破棄されるようにします。アプリケーションがこのモードを使用し、問題がある場合は、COPY_ON_READ_AND_COMMIT モードに切り替えてその問題がまだ続いているかどうかを調べます。問題が解消されていれば、トランザクションがコミットされた後でアプリケーションはその参照のすべてを破棄するのに失敗したことになります。

COPY_ON_WRITE モード

COPY_ON_WRITE モードは、指定したキーのトランザクションによって ObjectMap.get メソッドが初めて呼び出されるときに起こるコピーを排除することにより、COPY_ON_READ_AND_COMMIT モードを超えるパフォーマンスを実現します。 ObjectMap.get メソッドは、値オブジェクトへの直接参照の代わりに値のプロキシーを戻します。プロキシーは、アプリケーションが valueInterfaceClass 引数によって指定した値インターフェース上で set メソッドを呼び出さない限り、値のコピーが行われないことを保証します。プロキシーは、*copy on write* インプリメンテーションを提供します。トランザクションがコミットすると、BackingMap はプロキシーを検査して、呼び出される set メソッドの結果としてコピーが行われたかどうかを判別します。コピーが行われた場合は、そのコピーへの参照が BackingMap に保管されます。このモードの大きな利点は、トランザクションが値を変更するために set メソッドを呼び出さない場合には、読み取りまたはコミットの時点で値がコピーされないことです。

COPY_ON_READ_AND_COMMIT および COPY_ON_READ モードはどちらも、値が ObjectMap から検索される場合にディープ・コピーを行います。アプリケーションがトランザクションで検索されたいくつかの値を更新するだけの場合は、このモードは最適ではありません。 COPY_ON_WRITE モードはこの振る舞いを効率的な方法でサポートしますが、アプリケーションがシンプルなパターンを使用する必要があります。インターフェースをサポートするには、値オブジェクトが必要です。アプリケーションは、ObjectGrid セッション内で値と対話する際にこのインターフェース上でメソッドを使用する必要があります。その場合、ObjectGrid はアプリケーションに戻される値のプロキシーを作成します。プロキシーは実際の値になる参照を持ちます。アプリケーションが読み取りだけを行う場合は、その読み取りは常

に実際のコピーに対して実行されます。アプリケーションがオブジェクト上の属性を変更する場合、プロキシーは実際のオブジェクトをコピーして、それからそのコピーに対して変更を行います。プロキシーは次に、そのポイントからコピーを使用します。これにより、アプリケーションによって読み取られるだけのオブジェクトに対するコピー操作は完全に避けることができます。すべての変更操作は設定されたプレフィックスで開始する必要があります。 Enterprise JavaBeans は通常、オブジェクト属性を変更するメソッドに対してこのスタイルのメソッドの名前付けを使用するためにコード化されます。この規則に従わなければいけません。変更されるすべてのオブジェクトは、アプリケーションによって変更時にコピーされます。これは、ObjectGrid がサポートしている、最も効率的な読み取りおよび書き込みのシナリオです。ユーザーは COPY_ON_WRITE を使用するために、次のようにマップを構成できます。この例では、アプリケーションは Map 内の名前を使用して鍵をかけて Person オブジェクトを保管しようとします。Person オブジェクトは次のようなコードの断片に似ています。

```
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;
}
}
```

アプリケーションは、ObjectMap から検索した値と対話する場合にのみ、IPerson インターフェースを使用します。次の例のようにオブジェクトを変更してインターフェースを使用します。

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

それからアプリケーションは、次の例のように、COPY_ON_WRITE モードを使用するために BackingMap を構成する必要があります。

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

最初のセクションはマップ内で Billy と名前を付けられた値を検索するアプリケーションを示しています。このアプリケーションは戻り値を Person オブジェクトではなく、IPerson オブジェクトにキャストします。戻されたプロキシーは次の 2 つのインターフェースを実装するからです。

- BackingMap.setCopyMode メソッド呼び出しで指定されたインターフェース
- com.ibm.websphere.objectgrid.ValueProxyInfo インターフェース

プロキシーを 2 つのタイプにキャストすることができます。先ほどのコードの断片の最後の部分は、COPY_ON_WRITE モードでは許可されないことを示しています。アプリケーションは Bobby レコードを検索し、これを Person オブジェクトにキャストしようとします。このアクションはクラス・キャスト例外により失敗します。戻されるプロキシーが Person オブジェクトではないからです。戻されたプロキシーは IPerson オブジェクトと ValueProxyInfo をインプリメントします。

ValueProxyInfo インターフェースおよび部分更新サポート

このインターフェースはアプリケーションに対して、プロキシーによって参照される、コミットされた読み取り専用の値か、またはこのトランザクション中に変更された属性セットのどちらかの検索を許可します。

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

ibmGetRealValue メソッドは、オブジェクトの読み取り専用のコピーを戻します。アプリケーションはこの値を変更してはいけません。ibmGetDirtyAttributes メソッドは、このトランザクション中にアプリケーションによって変更された属性を示すストリングのリストを戻します。 ibmGetDirtyAttributes は主に、Java database connectivity (JDBC) または CMP ベースのローダーで使用されます。リストに指定

された属性だけを、SQL ステートメントまたはテーブルにマップされたオブジェクト上で更新する必要があります。これにより、Loader により生成される、さらに効率的な SQL が可能です。 copy on write トランザクションがコミットされ、ローダーが接続されると、ローダーは変更されたオブジェクトの値を ValueProxyInfo インターフェースにキャストしてこの情報を取得することができます。

COPY_ON_WRITE またはプロキシーを使用する場合の equals メソッドの処理

例えば、次のコードは Person オブジェクトを構成してから、それを ObjectMap に挿入します。次に、ObjectMap.get メソッドを使用して同じオブジェクトを検索します。値はインターフェースにキャストされます。値が Person インターフェースにキャストされる場合は、ClassCastException 例外が起こります。戻り値が、Person オブジェクトではなく、IPerson インターフェースをインプリメントするプロキシーだからです。== 操作を使用する場合は、等価チェックが失敗します。これらは同じオブジェクトではないからです。

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

equals メソッドをオーバーライドする必要がある場合は、ほかにも考慮しなければならないことがあります。次のコードの断片に示すように、equals メソッドは、引数が IPerson インターフェースをインプリメントし、その引数をキャストして IPerson にするオブジェクトであることを検証する必要があります。引数が、IPerson インターフェースをインプリメントするプロキシーかもしれないので、インスタンス変数が等しいかどうかを比較するときに getAge メソッドと getName メソッドを使用する必要があります。

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

NO_COPY モード

NO_COPY によって、アプリケーションは、ObjectMap.get メソッドを使用して取得した値オブジェクトを、パフォーマンス向上と交換に変更しないことを保証できます。このモードが使用される場合、valueInterfaceClass 引数は無視されます。このモードを使用する場合は、値がコピーされることはありません。アプリケーションが値を変更すると、BackingMap 内のデータが壊れます。 NO_COPY モードは基本

的に、アプリケーションによってデータが変更されることのない、読み取り専用マップで有用です。アプリケーションがこのモードを使用し、問題がある場合は、COPY_ON_READ_AND_COMMIT モードに切り替えてその問題がまだ存在するかどうかを調べます。問題が解消されていれば、トランザクション中またはトランザクションがコミットされた後でアプリケーションは ObjectMap.get メソッドによって戻された値を変更しています。

ObjectTransformer インターフェースのベスト・プラクティス

ObjectTransformer はアプリケーションへのコールバックを使用して、通常の操作およびオブジェクト直列化やオブジェクトに対するディープ・コピーなどの高価な操作のカスタム実装を提供します。

ObjectTransformer インターフェースに関する個々の詳細については、222 ページの『ObjectTransformer プラグイン』のトピックを参照してください。パフォーマンス、および 346 ページの『copyMode メソッドのベスト・プラクティス』のトピックの CopyMode メソッドからの情報の観点から見ると、NO_COPY モードを除き、すべての場合に ObjectGrid が値をコピーするのは明白です。 ObjectGrid 内で採用されているデフォルトのコピー・メカニズムは直列化であり、これは高価な操作として知られています。この状況で ObjectTransformer インターフェースを使用することができます。 ObjectTransformer インターフェースはアプリケーションへのコールバックを使用して、通常の操作およびオブジェクト直列化やオブジェクトに対するディープ・コピーなどの高価な操作のカスタム実装を提供します。

アプリケーションはマップに対する ObjectTransformer インターフェースの実装を提供します。 ObjectGrid は次にこのオブジェクトに対するメソッドを代行し、アプリケーションに基づいてインターフェースの各メソッドの最適なバージョンを提供します。 ObjectTransformer インターフェースは以下のようになります。

```
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);
}
```

次のコード例を使用して、ObjectTransformer インターフェースを BackingMap に関連付けることができます。

```
ObjectGrid g = ...;
BackingMap bm = g.defineMap("PERSON");
MyObjectTransformer ot = new MyObjectTransformer();
bm.setObjectTransformer(ot);
```

オブジェクト直列化およびオブジェクト・インフレーションの調整

オブジェクト直列化は通常、ObjectGrid での最大のパフォーマンス問題です。アプリケーションに ObjectTransformer プラグインが同梱されていない場合、ObjectGrid はデフォルトのシリアライズ可能なメカニズムを使用します。アプリケーションは

Serializable readObject と writeObject の実装を供給するか、または、Externalizable インターフェースを実装するオブジェクトを持つことができますが、後者の方が 10 倍高速です。Map 内のオブジェクトが変更できない場合、アプリケーションは ObjectTransformer を ObjectMap と関連付けます。serialize メソッドおよび inflate メソッドが提供されることにより、アプリケーションは、システムに大きなパフォーマンス効果を与えるこれらの操作を最適化するためのカスタム・コードを提供できます。 serialize メソッドはオブジェクトを直列化し、ストリームを提供します。このメソッドは、提供されるストリームにメソッドを直列化します。 inflate メソッドは入力ストリームを提供し、アプリケーションがオブジェクトを作成し、ストリームでデータを使用してそれをインフレーションし、それからオブジェクトを戻すことを予想します。 serialize メソッドおよび inflate メソッドの実装は、相互にミラーリングする必要があります。

ディープ・コピー操作を調整する

アプリケーションが ObjectMap からオブジェクトを受け取った後で、ObjectGrid はオブジェクト値に対してディープ・コピーを実行し、BaseMap マップ内のコピーが安全であることを保証します。その後アプリケーションはこのオブジェクト値を安全に変更できます。トランザクションがコミットすると、BaseMap マップ内のオブジェクト値のコピーは新しく変更される値に更新され、アプリケーションはその時点からその値の使用を停止します。コミット・フェーズの際に再度オブジェクトをコピーし、プライベート・コピーを行うことも可能でしたが、その場合は、このアクションのパフォーマンス・コストは、トランザクションのコミットの後で値を使用しないようにアプリケーションのプログラマーに伝えることに対してトレードオフされます。デフォルトのオブジェクト・コピー・メカニズムは、クローンあるいは直列化とインフレーションのペアを使用して、コピーを生成しようとします。直列化とインフレーションのペアは、最悪なパフォーマンス・シナリオです。プロファイル作成によって、直列化とインフレーションがご使用のアプリケーションにとって問題であることが判明したら、ObjectTransformer プラグインを提供し、より効率的なオブジェクト・コピーを使用して `copyValue` および `copyKey` メソッドを実行します。

プラグイン Evictor パフォーマンスのベスト・プラクティス

プラグイン Evictor を使用する場合は、これを作成し、バックアップ・マップにこれを使用することを伝えるまで Evictor はアクティブになりません。 LFU Evictor と LRU Evictor について、ベスト・プラクティスとパフォーマンスのヒントを使用してください。

LFU Evictor

LFU Evictor の概念は、頻繁に使用されないマップからエントリーを除去することです。マップのエントリーは、一定量のバイナリー・ヒープを超えて広がります。特定のキャッシュ・エントリーの使用量が増えると、それはヒープの高位に配列されます。Evictor が一連の除去を試行する場合、バイナリー・ヒープの特定のポイントよりも低い位置にあるキャッシュ・エントリーだけを除去します。この結果、頻繁に使用されないエントリーが除去されます。

LRU Evictor

LRU Evictor は LFU Evictor と同じ概念に従いますが、2、3 の点が異なります。主な違いは、LRU ではバイナリー・ヒープのセットの代わりに先入れ先出し (FIFO) キューを使用することです。キャッシュ・エントリーにアクセスされるたびに、そのエントリーはキューの先頭に移動します。この結果、キューの先頭には最後に使用されたマップ・エントリーが含まれ、キューの最後は LRU マップ・エントリーになります。例えば、A キャッシュ・エントリーが 50 回使用され、B キャッシュ・エントリーが A キャッシュ・エントリーの直後に 1 回だけ使用されるとします。この場合、最後に使用された B キャッシュ・エントリーがキューの先頭になり、A キャッシュ・エントリーはキューの最後になります。 LRU Evictor は、キューの末尾にあるキャッシュ・エントリー、すなわち最も古いマップ・エントリーを除去します。

LFU および LRU プロパティーおよびパフォーマンスを向上させるためのベスト・プラクティス

ヒープ数

LFU Evictor を使用する場合は、特定のマップのすべてのキャッシュ・エントリーが指定するヒープ数を超えて配列されます。これによってパフォーマンスが劇的に上がり、また、そのマップのすべての配列を含む、1 つのバイナリー・ヒープ上ですべての除去が同期するのを防ぎます。ヒープが多い場合も、各ヒープのエントリーが少ないので再配列に必要な時間を短縮できます。ご使用の BaseMap でエントリー数の 10% のヒープ数を設定してください。

キューの数

LRU Evictor を使用する場合は、特定のマップのすべてのキャッシュ・エントリーは指定する LRU キューの数を超えて配列されます。これによってパフォーマンスが劇的に上がり、また、そのマップのすべての配列を含む、1 つのキュー上ですべての除去が同期するのを防ぎます。ご使用の BaseMap でエントリー数の 10% のキューの数を設定してください。

MaxSize プロパティー

LFU または LRU Evictor がエントリーの除去を開始すると、MaxSize Evictor プロパティーを使用して、いくつのバイナリー・ヒープまたは LRU キュー・エレメントを除去するかを判別します。例えば、各マップ・キューにおよそ 10 のマップ・エントリーを持つようにヒープまたはキューの数を設定するとします。 MaxSize プロパティーが 7 に設定されている場合は、Evictor は各ヒープまたはキュー・オブジェクトの 3 つのエントリーを除去して、各ヒープまたはキューのサイズを 7 にします。Evictor は、ヒープまたはキューに、エレメントの MaxSize プロパティーの値を超えるエレメントがある場合にのみ、マップ・エントリーをヒープまたはキューから除去します。MaxSize をヒープまたはキュー・サイズの 70% に設定してください。この例の場合、値は 7 に設定されます。ユーザーは、BaseMap エントリーの数を、使用するヒープまたはキューの数で割ることによって、各ヒープまたはキューのおおよそのサイズを得ることができます。

SleepTime プロパティー

Evictor はマップから常にエントリーを除去するわけではありません。その代わり、一定時間スリープし、各 *n* 秒間に 1 回目覚めます。このときの *n*

が SleepTime プロパティーです。このプロパティーでもパフォーマンスを上げることができます。除去スイープをあまり頻繁に実行すると、それを処理するために必要なリソースのためにパフォーマンスが低下します。ただし、Evictor を十分に使用しないと、不要のエントリーのマップが残ります。不要なエントリーでいっぱいのマップは、メモリー所要量にもマップに必要な処理用リソースにも悪影響を与えます。除去スイープを 15 秒に設定すると、ほとんどのマップで良好な実例が得られます。マップが頻繁に書き込まれ、高速のトランザクションで使用される場合は、この値をより低く設定するほうが有益でしょう。ただし、マップへのアクセスが頻繁ではない場合は、この時間をより高い値に設定することができます。

例

以下の例ではマップを定義し、新しい LFU Evictor を作成し、Evictor のプロパティーを設定し、Evictor を使用するようにマップを設定します。

```
//Use ObjectGridManager to create/get the ObjectGrid. Refer to
// the ObjectGridManger section
ObjectGrid objGrid = ObjectGridManager.create.....
BackingMap bMap = objGrid.defineMap("SomeMap");
//Set properties assuming 50,000 map entries
LFUEvictor someEvictor = new LFUEvictor();
someEvictor.setNumberOfHeaps(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

LRU Evictor を使用するのは LFU Evictor を使用するのとよく似ています。以下に例を示します。

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
//Set properties assuming 50,000 map entries
LRUEvictor someEvictor = new LRUEvictor();
someEvictor.setNumberOfLRUQueues(5000);
someEvictor.setMaxSize(7);
someEvictor.setSleepTime(15);
bMap.setEvictor(someEvictor);
```

LFU Evictor の例とは 2 行だけ異なっていることに注意してください。

デフォルト Evictor のベスト・プラクティス

デフォルトの*存続時間* Evictor のベスト・プラクティスです。

352 ページの『プラグイン Evictor パフォーマンスのベスト・プラクティス』のトピックで説明されているプラグイン Evictor のほかに、デフォルトの TTL Evictor が各バックアップ・マップによって作成されます。デフォルトの Evictor は、*存続時間* の概念に基づいてエントリーを除去します。この振る舞いは ttlType 属性で定義されます。以下の 3 つの ttlTypes 属性があります。

- **None**: エントリーの期限切れがないように、それによってマップからエントリーが除去されることがないように設定します。
- **Creation time**: 作成された時に応じてエントリーが除去されるように指定します。

- **Last accessed time**: 最後にアクセスされた時に応じてエントリーが除去されるように指定します。

デフォルトの Evictor のプロパティーとパフォーマンスのためのベスト・プラクティス

TimeToLive プロパティー

このプロパティーは ttlType プロパティーと並んで、パフォーマンスの観点から見ると最も重要です。 CREATION_TIME ttlType を使用している場合、 Evictor は、作成からの時間がその TimeToLive 属性値と等しいときにエントリーを除去します。 TimeToLive 属性値を 10 秒に設定すると、10 秒間経過した後で全マップ内のすべてが除去されます。この値を CREATION_TIME ttlType に設定する場合は注意が必要です。この Evictor は、一定時間にのみ使用される、キャッシュへの妥当な追加量がある場合に、最も有効に使用されます。このストラテジーによって、作成されたものはすべて、一定時間後に除去されます。

以下は、CREATION_TIME の TTL タイプが有効である場合の例です。ユーザーは株価情報を入手する Web アプリケーションを使用しており、最新情報を入手することが重要でないとします。この場合、株価情報は 20 分間 ObjectGrid にキャッシュされます。 20 分後、ObjectGrid マップの有効期限が切れ、除去されます。ほぼ 20 分ごとに、ObjectGrid マップは Loader プラグインを使用してマップ・データをデータベースの新しいデータで更新します。データベースは 20 分ごとに最新の株価情報によって更新されます。つまり、このアプリケーションの場合、TimeToLive 値を 20 分にして使用するのが理想的です。

LAST_ACCESSED_TIME ttlType 属性を使用している場合は、CREATION_TIME ttlType を使用している場合よりも TimeToLive をもっと低い数に設定します。エントリーの TimeToLive 属性は、アクセスされるたびにリセットされるからです。言い換えれば、TimeToLive 属性が 15 で、エントリーが 14 秒間存在し、それからアクセスされた場合、このエントリーはあと 15 秒間有効期限が切れることはありません。TimeToLive を比較的高い数値に設定した場合は、多くのエントリーがまったく除去されなくなる可能性があります。ただし、この値を 15 秒程度に設定すると、エントリーは頻繁にアクセスされない場合に除去されることになります。

以下は、LAST_ACCESSED_TIME の TTL タイプが有効である場合の例です。ObjectGrid マップはクライアントからのセッション・データを保持するために使用されます。セッション・データは、クライアントがそのセッション・データを一定時間使用しない場合は破棄する必要があります。例えば、セッション・データは、クライアントによるアクティビティーが 30 分間なかった後にタイムアウトになるとします。この場合、LAST_ACCESSED_TIME の TTL タイプを使用し、その TimeToLive 属性を 30 分に設定すると、このアプリケーションにまさに必要な条件になります。

例

以下の例ではバックアップ・マップを設定し、そのデフォルト Evictor の ttlType 属性を設定し、TimeToLive プロパティーを設定しています。

```
ObjectGrid objGrid = new ObjectGrid;
BackingMap bMap = objGrid.defineMap("SomeMap");
bMap.setTtlEvictorType(TTLType.LAST_ACCESSED_TIME);
bMap.setTimeToLive(15);
```

ほとんどの Evictor の設定は、ObjectGrid の初期化の前に設定しなければなりません。Evictor のさらに詳細な説明については、199 ページの『Evictor』を参照してください。

第 12 章 ピア Java 仮想マシン間の変更の配布

LogSequence オブジェクトおよび LogElement オブジェクトは、プラグインを使用して ObjectGrid トランザクションで発生した変更を通信します。

Java Message Service (JMS) を使用してトランザクションの変更を配布する方法についての詳細は、361 ページの『トランザクションの変更を配布する Java Message Service』を参照してください。

ObjectGrid インスタンスが ObjectGridManager によりキャッシュされていることが前提条件です。詳しくは 95 ページの『createObjectGrid メソッド』を参照してください。cacheInstance ブール値は、true に設定する必要があります。

オブジェクトは、アプリケーションが、メッセージ・トランスポートを使用してオブジェクト・グリッドで発生した変更をリモート Java 仮想マシン (JVM) 内のピア ObjectGrids に容易にパブリッシュしてから、その変更を JVM 上で適用するための手段を提供します。LogSequenceTransformer クラスは、このサポートを使用可能にするために重要です。この項目では、メッセージを伝搬するために、Java Message Service (JMS) メッセージング・システムを使用して、リスナーを書き込む方法を調べます。

ObjectGrid は、 IBM 提供プラグインを使用した WebSphere Application Server クラスター・メンバー間での ObjectGrid トランザクション・コミットの結果である LogSequences の送信をサポートします。この機能はデフォルトでは使用不可ですが、作動可能に構成することができます。 ただし、コンシューマーまたはプロデューサーのいずれかが WebSphere Application Server ではない場合、外部 JMS メッセージング・システムの使用が必要となる可能性があります。

1. プラグインを初期設定します。 ObjectGrid が開始するときに、ObjectGrid は、ObjectGridEventListener インターフェース・コントラクトの一部である、プラグインの `initialize` メソッドを呼び出します。 `initialize` メソッドは、接続、セッション、およびパブリッシャーを含む JMS リソースを取得し、JMS リスナーであるスレッドを開始する必要があります。初期設定メソッドは、以下の例のように表示されます。

```
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();
// need to start the connection to receive messages.
connection.start();
// the jms session is not transactional (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);
}
}
```

スレッドを開始するためのコードは、Java 2 Platform, Standard Edition (J2SE) スレッドを使用します。WebSphere Application Server バージョン 6.x または WebSphere Application Server バージョン 5.x エンタープライズ・サーバーを実行している場合、非同期 Bean アプリケーション・プログラミング・インターフェース (API) を使用して、このデーモン・スレッドを開始します。共通 API を使用することもできます。以下に、作業マネージャーを使用する同じアクションを示す置換の断片の例を示します。

```
// start the listener thread.
listenerRunning = true;
workManager.startWork(this, true);
```

また、プラグインは、実行可能なインターフェースの代わりに、作業インターフェースをインプリメントする必要があります。また、リリース・メソッドを追加して、listenerRunning 変数を false に設定する必要があります。プラグインは、コンストラクター、または Inversion of Control (IoC) コンテナーを使用している場合は注入によって、 WorkManager インスタンスに提供されている必要があります。

2. 変更を伝送します。 以下は、ObjectGrid 上で行われるローカル変更をパブリッシュするためのサンプル transactionEnd メソッドです。これは JMS を使用しますが、明らかに、信頼性のあるパブリッシュおよびサブスクライブ・メッセージングの機能を持つ任意のメッセージ・トランスポートを使用できます。

```
// This method is synchronized to make sure the
// messages are published in the order the transaction
// were committed. If we started publishing the messages
// in parallel then the receivers could corrupt the Map
// as deletes may arrive before inserts etc.
public synchronized void transactionEnd(String txid,
boolean isWriteThroughEnabled,
boolean committed, Collection changes)
{
try
{
// must be write through and commited.
if(isWriteThroughEnabled && committed)
{
// write the sequences to a byte []
```

```
ByteArrayOutputStream bos = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(bos);
if (publishMaps.isEmpty()) {
// serialize the whole collection
LogSequenceTransformer.serialize(changes, oos, this, mode);
}
else {
// filter LogSequences based on publishMaps contents
Collection publishChanges = new ArrayList();
Iterator iter = changes.iterator();
while (iter.hasNext()) {
LogSequence ls = (LogSequence) iter.next();
if (publishMaps.contains(ls.getMapName())) {
publishChanges.add(ls);
}
}
LogSequenceTransformer.serialize(publishChanges, oos, this,
mode);
}
// make an object message for the changes
oos.flush();
ObjectMessage om = jmsSession.createObjectMessage(
bos.toByteArray());
// set properties
om.setStringProperty(PROP_TX, txid);
om.setStringProperty(PROP_GRIDNAME, myGrid.getName());
// transmit it.
publisher.publish(om);
}
}
catch(Throwable e)
{
throw new ObjectGridRuntimeException("Cannot push changes", e);
}
}
```

このメソッドは、いくつかのインスタンス変数を使用します。

- **jmsSession** 変数: メッセージをパブリッシュするために使用する JMS セッションです。これは、プラグインが初期設定される際に作成されます。
- **mode** 変数: 配布モードです。
- **publishMaps** 変数: パブリッシュする変更内容を持つ各 Map の名前が含まれているセットです。この変数が空の場合、すべての Map がパブリッシュされます。
- **publisher** 変数: プラグイン初期設定メソッド中に作成される TopicPublisher オブジェクトです。

3. 更新メッセージを受信および適用します。 以下は実行メソッドです。このメソッドは、アプリケーションがループを停止するまで、ループ内で実行します。各ループの反復は、JMS メッセージの受信、およびメッセージの ObjectGrid への適用を試行します。

```
private synchronized boolean isListenerRunning()
{
return listenerRunning;
}
public void run()
{
try
{
System.out.println("Listener starting");
// get a jms session for receiving the messages.
// Non transactional.
```

```
TopicSession myTopicSession;
myTopicSession = connection.createTopicSession(false,
javax.jms.Session.AUTO_ACKNOWLEDGE);
// get a subscriber for the topic, true indicates don't receive
// messages transmitted using publishers
// on this connection. Otherwise, we'd receive our own updates.
TopicSubscriber subscriber = myTopicSession.createSubscriber(topic,
null, true);
System.out.println("Listener started");
while(isListenerRunning())
{
ObjectMessage om = (ObjectMessage)subscriber.receive(2000);
if(om != null)
{
// Use Session that was passed in on the initialize...
// very important to use no write through here
mySession.beginNoWriteThrough();
byte[] raw = (byte[])om.getObject();
ByteArrayInputStream bis = new ByteArrayInputStream(raw);
ObjectInputStream ois = new ObjectInputStream(bis);
// inflate the LogSequences
Collection collection = LogSequenceTransformer.inflate(ois,
myGrid);
Iterator iter = collection.iterator();
while (iter.hasNext()) {
// process each Maps changes according to the mode when
// the LogSequence was serialized
LogSequence seq = (LogSequence)iter.next();
mySession.processLogSequence(seq);
}
mySession.commit();
} // if there was a message
} // while loop
// stop the connection
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");
}
```

LogSequenceTransformer クラス、ObjectGridEventListener、 LogSequence および LogElement API により、信頼性のあるパブリッシュおよびサブスクライブを使用して、変更内容を配布し、配布する Map をフィルターに掛けることができます。このタスクの断片は、これらの API を JMS とともに使用して、対等 ObjectGrid をビルドする方法を示します。これは、共通メッセージ・トランスポートを共用するさまざまなプラットフォームのセットでホストされるアプリケーションによって共用されます。

トランザクションの変更を配布する Java Message Service

異なる層または混合しているプラットフォームの環境で変更の配布のために Java Message Service (JMS) を使用します。

JMS は、異なる層または混合しているプラットフォームの環境で配布された変更に理想的なプロトコルです。例えば、ObjectGrid を使用する一部のアプリケーションが Gluecode または Tomcat でデプロイされ、他のアプリケーションは WebSphere Application Server バージョン 6.0 で稼働しているとします。 JMS は、これらの異なる環境の ObjectGrid ピア間で配布された変更に理想的です。 HA マネージャーのメッセージ・トランスポートは非常に高速ですが、単一コア・グループに属する JVM にしか変更を配布できません。 JMS はそれに比較すれば低速ですが、より広範囲で、多様なアプリケーション・クライアントのセットに ObjectGrid を共用させることができます。 JMS は、ファット Swing クライアントと WebSphere Extended Deployment にデプロイされているアプリケーション間の ObjectGrid でデータを共用する場合などに理想的です。

概要

JMS は、ObjectGridEventListener リスナーとして振る舞う Java オブジェクトを使用して、トランザクションの変更を配布するために実装されます。このオブジェクトは、以下の 4 つの方法で状態を伝搬することができます。

無効化 除去、更新、または削除されるエントリーは、メッセージを受け取ると、すべてのピア Java 仮想マシン (JVM) で除去されます。

条件付き無効化
ローカル・バージョンがパブリッシャーのバージョンと同じか、またはそれより古い場合のみ、エントリーが除去されます。

プッシュ
除去、更新、削除または挿入されたエントリーは、JMS メッセージを受信する場合、すべてのピア JVM に追加または上書きされます。

条件付きプッシュ
ローカル・エントリーがパブリッシュされているバージョンより新しくない場合に、エントリーは受信サイドで更新または追加のみ行われます。

パブリッシュする変更の listen

プラグインは、ObjectGridEventListener インターフェースを実装し、transactionEnd イベントをインターセプトします。 ObjectGrid はこのメソッドを呼び出し、プラグインはトランザクションによってタッチされる各 Map の LogSequence リストを JMS メッセージに変換し、それをパブリッシュしようとします。プラグインは、すべての Map または Map のサブセットの変更をパブリッシュするよう構成することができます。 LogSequence オブジェクトは、パブリッシュが使用可能な Map のために処理されます。 LogSequenceTransformer ObjectGrid クラスは、ストリームに対して各 Map のフィルタリングされた LogSequence をシリアライズします。すべての LogSequences がストリームにシリアライズされたら、JMS ObjectMessage が作成され、既知のトピックにパブリッシュされます。

JMS メッセージの listen およびローカル ObjectGrid への適用

同じプラグインはまた、既知のトピックにパブリッシュされるすべてのメッセージを受け取りながら、ループでスピンするスレッドを開始します。メッセージを受け取ると、LogSequenceTransformer クラスにメッセージ・コンテンツを渡し、それを LogSequence オブジェクトのセットに変換します。その後、ノー・ライトスルー・トランザクションが開始されます。各 LogSequence オブジェクトは Session.processLogSequence メソッドに提供され、その変更でローカル Map を更新します。 processLogSequence メソッドは、配布モードを理解しています。トランザクションはコミットされ、ローカル・キャッシュが変更を反映します。

JMS を使用してトランザクションの変更を配布する方法について詳しくは、357 ページの『第 12 章 ピア Java 仮想マシン間の変更の配布』を参照してください。

第 13 章 注入ベースのコンテナー統合

Inversion of Control (IoC) フレームワークを使用して、ObjectGrid を完全に構成することができます。そうすることで、ObjectGrid XML 構成フレームワークを使用する必要がなくなります。

注入ベースのコンテナー

Inversion of Control (IoC) としても知られる、注入ベースのコンテナーは、クライアント・サイドおよびサーバー・サイドでアプリケーションによって使用される共通のパターンです。このようなコンテナーのいくつかのオープン・ソース・インプリメンテーションが存在します。新規 Enterprise JavaBeans (EJB) バージョン 3.0 仕様でも、これらの概念の一部を流用しています。これらのフレームワークのほとんどは Enterprise JavaBean コンテナーで、特定の Bean のインスタンス作成を担当します。これらのフレームワークはまた、プロパティーのセットで Bean を初期化することができ、 Enterprise JavaBean 属性で getter および setter ペアを使用して、必要な他の Enterprise JavaBeans を結び付けます。 ObjectGrid アプリケーション・プログラミング・インターフェース (API) は、このようなコンテナーと適切に作業するように設計されています。 ObjectGridManager.createObjectGrid メソッドを開始すると、ObjectGrid をブートストラップするようにコンテナーを構成し、アプリケーションが作業中の ObjectGrid を参照したり、またはコンテナーに必要に応じて ObjectGrid セッションを提供するように指示することができます。

サポートされるパターン

以下のセクションでは、 IoC フレームワークを使用するアプリケーションが ObjectGrid API を正しく使用できることを検証するために行われたことについて考察します。

ObjectGridManager の使用による ObjectGrid の作成

ObjectGrid API は、IoC フレームワークとうまく機能するように設計されています。このようなフレームワークに使用されるルート・シングルトンは ObjectGridManager インターフェースで、参照を指定された ObjectGrid に戻すいくつかの `createObjectGrid` ファクトリー・メソッドがあります。この ObjectGrid 参照を IoC フレームワークで singleton として設定でき、その Bean の後続の要求が同じ ObjectGrid インスタンスを戻します。

ObjectGrid プラグイン

ObjectGrid のプラグインには、以下のものがあります。

- TransactionCallback
- ObjectGridEventListener
- SubjectSource
- MapAuthorization
- SubjectValidation

各プラグインは、インターフェースを実装する単純な JavaBeans です。 IoC フレームワークを使用して、これらのプラグインを作成し、ObjectGrid インスタンスで適

切な属性にそれらを結び付けることができます。これらのプラグインにはそれぞれ、IoC フレームワークとの正しい統合のために、 ObjectGrid インターフェースに対応する set メソッドがあります。

マップの作成

ObjectGrid インターフェースの createMap ファクトリー・メソッドを使用すると、新たに指定された Map を作成することができます。 BackingMap (createMap によって戻されるオブジェクト) が必要とするすべてのプラグインは、 IoC フレームワークを使用して構成され、適切な属性名を使用して、BackingMap に結び付けられます。 BackingMap は、その他のオブジェクトによって参照されないため、IoC フレームワークに自動構成されることはありません。回避策として、各 BackingMap を主なアプリケーション Bean のダミー属性に結び付けることができます。 setMaps() メソッドを使用して、以前この ObjectGrid で定義されたすべての BackingMap をクリアし、提供されている BackingMap のリストに置き換えます。

マップ・プラグインのバッキング

BackingMap のプラグインは、ObjectGrid のものと同じように振る舞います。各プラグインには、BackingMap インターフェースに対応する `set` メソッドがあります。BackingMap プラグインには以下のようなものがあります。

- Loader
- ObjectTransformer
- OptimisticCallback
- Evictor
- MapEventListener

使用パターン

ObjectGrid 作成のために IoC フレームワークでその構成ファイルをセットアップしたら、`gridName_Session` などの Enterprise Bean を作成します。 ObjectGrid シングルトン Enterprise JavaBean で `getSession` メソッドを呼び出すことによって取得される Enterprise JavaBean としてそれを定義します。その後、そのアプリケーションは IoC フレームワークを使用し、オブジェクトが新規セッションを要求するたびに、 gridName_Session オブジェクトへの参照を取得します。

要約

IoC フレームワークをすでに使用している環境で ObjectGrid を使用して、 Bean をインスタンス化および構成することは簡単です。 IoC フレームワークを使用して、完全に ObjectGrid を構成すると、XML 構成フレームワークを使用する必要がなくなります。 ObjectGrid は、既存の IoC フレームワークとシームレスに稼働します。

第 14 章 トラブルシューティング

このセクションでは、アプリケーション・エラーまたはアプリケーションの設計問題によって起こるトラブルシューティングの問題に関するシナリオを説明します。ObjectGrid に障害があることが疑わしい場合は、ObjectGridManager のトレース・セクションの説明に従って、 ObjectGrid のトレースを使用可能にできます。

偶発的で解明できないエラー

CopyMode セクションで説明したように、アプリケーションは、COPY_ON_READ、 COPY_ON_WRITE または NO_COPY コピー・モードを使用してパフォーマンスを改善しようとします。問題の症状が変化し、その問題が解明できないものまたは予想外のものである場合、アプリケーションに偶発的な問題が発生します。コピー・モードを COPY_ON_READ_AND_COMMIT モードに変更すると偶発的な問題は発生しません。

問題

この問題は、使用したコピー・モードのプログラミング契約にアプリケーションが違反し、その結果発生した ObjectGrid マップ内のデータ破壊に起因する場合があります。データ破壊は、予測不能なエラーが、偶発的または解明不能または予期しない形で発生する原因になることがあります。

ソリューション

ソリューションは、アプリケーションが使用中のコピー・モード用プログラミング契約に従うことです。 COPY_ON_READ および COPY_ON_WRITE コピー・モードの場合、これはアプリケーションが、値参照を取得したトランザクションの有効範囲外の値オブジェクト参照を使用することを意味します。これらのモードを使用するためには、アプリケーションはトランザクションの完了後値オブジェクトへの参照を破棄し、値オブジェクトにアクセスする必要のあるそれぞれのトランザクションの値オブジェクトへの新規参照を獲得することに同意する必要があります。NO_COPY コピー・モードの場合、アプリケーションは値オブジェクトを絶対変更しないことに同意する必要があります。この場合、値オブジェクトを変更しないようにアプリケーションを変更するか、アプリケーションが別のコピー・モードを使用するか、いずれかを行う必要があります。コピー・モード設定の詳細については CopyMode セクションを参照してください。

一般的な例外処理技法

Throwable オブジェクトの根本原因を知ることは、問題の発生源を分離するのに役立ちます。発生した Throwable の根本原因を検出するための例外ハンドラーが使用できるユーティリティー・メソッドの例を以下に示します。

例

```
static public Throwable findRootCause( Throwable t )
{
// Start with Throwable that occurred as the root.
Throwable root = t;
```

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

特定の例外処理技法

重複挿入

この問題は、通常分散トランザクション伝搬環境でのみ発生します。これは頻繁には発生しません。

メッセージ

```
[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)
```

問題

フィルタリングされたログ・シーケンスが 1 つの JVM から他へ広がった場合、外部ログ・シーケンスは 2 番目の JVM で処理されます。このキーのエントリーが存在するか、2 つのログ・シーケンス命令コードが異なっている必要があります。この問題は時折り発生します。

インパクトおよびソリューション

この問題が発生した場合、発生エントリーは他の JVM では更新されず、そのことが ObjectGrid で矛盾を起こすことがあります。しかしこの問題を回避する予備手段があります。オブジェクトの挿入、更新、削除の他、オブジェクト検索の区画化機能 (WPF) を使用できます。この技法の詳細については、WPF と ObjectGrid の統合のセクションを参照してください。

オプティミスティック衝突例外

OptimisticCollisionException 例外は直接受け取ることもできますし、ObjectGridException 例外の受信中に受け取ることもできます。以下のコードは、例外を catch し、そのメッセージを表示する方法の例です。

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

例外の原因

OptimisticCollisionException 例外は、ほとんど同じ時間に 2 つの異なるクライアントが同じマップ・エントリーを更新しようとしたとき作成されます。1 つのクライアント・セッションがコミットされ、マップ・エントリーを更新します。しかし、別のクライアントはコミットする前にデータを読んでしまって、古いデータまたは正しくないデータを含んでいます。この別のクライアントは、例外が作成されるとき、正しくないデータをコミットしようとします。

例外をトリガーしたキーの検索

そうした例外のトラブルシューティングのとき、例外をトリガーしたエントリーに対応するキーを検索すると便利です。OptimisticCollisionException の利点は、キーを表すオブジェクトを戻すメソッド getKey を組み込んでいることです。以下は、OptimisticCollisionException を catch しているとき、キーを検索し印刷する方法の例です。

```
try {
...
} catch (OptimisticCollisionException oce) {
System.out.println(oce.getKey());
}
```

OptimisticCollisionException は ObjectGridException の原因になる場合があります。この場合、以下のコードを使用して例外タイプを判別し、キーを印刷できます。以下のコードは、一般的な例外処理技法のセクションで説明したように、findRootCause ユーティリティー・メソッドを使用しています。

```
try {
...
}
catch (ObjectGridException oe) {
Throwable Root = findRootCause( oe );
if (Root instanceof OptimisticCollisionException) {
OptimisticCollisionException oce = (OptimisticCollisionException)Root;
System.out.println(oce.getKey());
}
}
```

LockTimeoutException 例外

メッセージ

LockTimeoutException 例外は直接 catch することもできますし、ObjectGridException を catch しているときに catch することもできます。以下のコードの断片は、例外を catch し、メッセージを表示する方法を示します。

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

結果は以下の通りです。

```
com.ibm.websphere.objectgrid.plugins.LockTimeoutException: %Message
```

%Message は、例外が作成され、例外プロパティーとメソッドがエラー・メッセージを正確に表示するために使用されるとき、パラメーターとして渡されるストリングを表します。多くの場合、要求されたロックのタイプと、どのマップ・エントリーでトランザクションが活動したかを記述します。

例外の原因

トランザクションまたはクライアントが特定のマップ・エントリーへのロック認可を要求した場合、現行クライアントがロックを解放するのを待たなければならない場合がよくあります。ロック要求が長時間アイドルのままで、ロックが認可を得られない場合、LockTimeoutException が作成されます。以下のセクションで詳細を説明しますが、これはデッドロックを防止します。ロックはトランザクションがコミットされるまで解放されないため、ペシミスティック・ロック・ストラテジーを使用するときにこの例外が起こることが多いです。

ロック要求と例外についての詳細情報

LockTimeoutException には getLockRequestQueueDetails という組み込みメソッドがあります。このメソッドは例外を引き起こした状態の詳細な記述を含むストリングを戻します。以下は例外を catch するコードの例で、エラー・メッセージを表示します。

```
try {
...
}
catch (LockTimeoutException lte) {
System.out.println(lte.getLockRequestQueueDetails());
}
```

出力は以下の結果になります。

```
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]
```

ObjectGridException catch ブロックで例外を取得した場合、以下のコードが例外を決定し、キューの詳細を印刷します。これは、一般的な例外処理技法のセクションで説明した findRootCause ユーティリティー・メソッドを使用します。

```
try {
...
}
catch (ObjectGridException oe) {
Throwable Root = findRootCause( oe );
if (Root instanceof LockTimeoutException) {
LockTimeoutException lte = (LockTimeoutException)Root;
System.out.println(lte.getLockRequestQueueDetails());
}
}
```

考えられるソリューション

LockTimeoutException はご使用のアプリケーションで起こりうるデッドロックを防ぐためのものです。このタイプの例外は設定した時間が経過するとスローされます。しかし、待ち時間は BackingMap で使用可能な setLockTimeout(int) メソッドを使用して設定できます。setLockTimeout メソッドを使用すると LockTimeoutException が除去されます。ご使用のアプリケーションにデッドロックが実際には存在しない場合、ロック・タイムアウトを調整することで LockTimeoutException を回避できます。

以下のコードは、ObjectGrid を作成する方法、マップを定義する方法、LockTimeout 値を 30 秒に設定する方法を示します。

```
ObjectGrid objGrid = new ObjectGrid();
BackingMap bMap = objGrid.defineMap("MapName");
//This will set the amount of time that a
// lock request will wait before an exception is thrown
bMap.setLockTimeout(30);
```

前のコードは、ハードコーディングされた ObjectGrid およびマップ・プロパティーに使用できます。XML ファイルから ObjectGrid を作成した場合、 LockTimeout プロパティーを backingMap タグに設定できます。以下はマップ LockTimeout 値を 30 秒に設定した backingMap タグの例です。

```
<backingMap name="MapName" lockStrategy="PESSIMISTIC" lockTimeout="30">
```

LockDeadlockException

メッセージ

LockDeadLockException は直接 catch することもできますし、 ObjectGridException を catch しているときに取得することもできます。以下は、例外の catch を示すコードの例と表示メッセージです。

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

結果は以下の通りです。

```
com.ibm.websphere.objectgrid.plugins.LockDeadlockException: %Message
```

%Message は例外が作成されスローされるとき、パラメーターとして渡されるストリングを表します。

例外の原因

ペシミスティック・ロック・ストラテジーの使用中、およびそれぞれが特定オブジェクトに共用ロックを持つ 2 つの分離したクライアントを使用中に、最も一般的なタイプのデッドロックが発生します。次に両方のクライアントがそのオブジェクトで排他ロックにプロモートしようとします。以下に、スローされる例外の原因となるトランザクション・ブロックが見られる状況を図示しています。

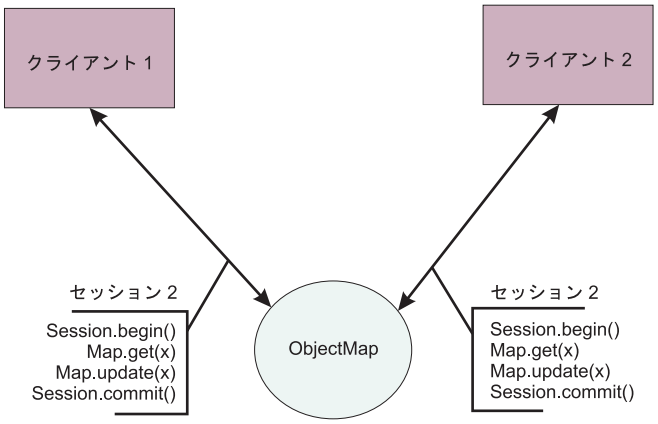


図 23. 潜在的なデッドロック状態の例

これは例外が発生したときご使用のプログラムで何が発生しているのかを示す抽象ビューです。同じ ObjectMap を更新中のスレッドが多数あるアプリケーションでは、この状態が発生する可能性があります。以下に、前図で説明したトランザクション・コード・ブロックを、2 つのクライアントが実行するときの段階的手順例を示します。

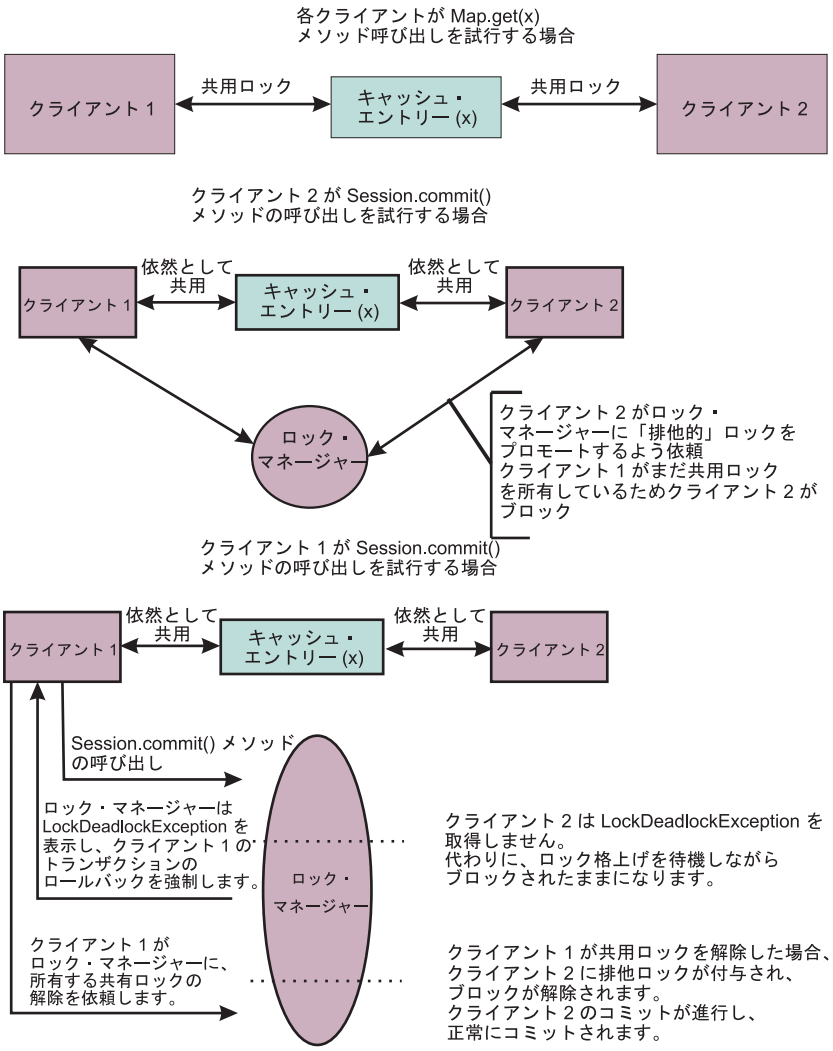


図 24. デッドロック状態

表示の通り、両方のクライアントが排他ロックにプロモートしようとし、引き続き共用ロックを所有している場合、いずれのクライアントもそれを取得できません。これらは常時他方のクライアントが共用ロックを解放するのを待ち、そのため `LockDeadlockException` が発生します。

考えられるソリューション

この例外の受信が正の場合があります。多数のスレッドがあり、そのすべてが特定のマップでトランザクションを実行する場合、前述した状態が発生する可能性があります (図 1)。例外がスローされ、ご使用のプログラムがハングするのを避けます。この例外を catch することにより、自身に対する通知が可能になり、希望すれば、catch ブロックにコードを追加して原因の詳細を見ることができます。ペシミスティック・ロック・ストラテジーではこの例外を表示するだけなので、単純なソリューションとして、オプティミスティック・ロック・ストラテジーを使用します。しかし、ペシミスティックが必要な場合は、get メソッドではなく getForUpdate メソッドを使用できます。これは前述した状態での例外取得を除去します。

XML 構成問題の診断

存在しないプラグイン・コレクションの参照

XML を使用して BackingMap プラグインを定義する場合、backingMap エレメントの pluginCollectionRef 属性は backingMapPluginCollection を参照する必要があります。pluginCollectionRef 属性は backingMapPluginCollection エレメントのいずれか 1 つの ID と一致しなければなりません。

メッセージ

pluginCollectionRef 属性が、 backingMapPluginConfiguration エレメントのどの ID 属性とも一致しない場合、以下のようなメッセージがログに表示されます。

```
[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'..
```

以下のメッセージは、トレースを使用可能にしたログからの抜粋です。

```
[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)
...
```

問題

このエラーの発生に使用された XML ファイルを以下に示します。ブック BackingMap によってその pluginCollectionRef 属性は bookPlugins に設定され、1 つの backingMapPluginCollection が collection1 の ID を持つことに注意してください。

```
<?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>
```

ソリューション

問題を修正するには、それぞれの pluginCollectionRef の値が backingMapPluginCollection エレメントのいずれか 1 つの ID に一致するようにしてください。この例の場合、pluginCollectionRef の名前を collection1 に変えるだけでエラーを回避できます。このほか問題を修正する方法としては、既存の backingMapPluginCollection の ID を pluginCollectionRef に一致するように変更する、または pluginCollectionRef に一致する ID を持つ追加 backingMapPluginCollection を追加するという方法があります。

必須属性の欠落

XML ファイルの多くのエレメントはオプションで幾つかの属性を持っています。ファイルにオプションの属性を追加したり、削除したりすることができます。XML は妥当性検査をどちらの方向にも受け渡すことができます。しかし必須属性がいくつかあります。関連エレメントが使用されるとき必須の属性が存在しない場合、 XML 妥当性検査は失敗します。

メッセージ

必須属性が欠落している場合、以下のようなメッセージがログに表示されます。この例の場合、プロパティー・エレメントから型属性が欠落しています。

```
[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'..
```

以下のメッセージは、トレースを使用可能にしたログからの抜粋です。

```
[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)
...
```

問題

前のエラーの発生に使用された XML ファイルを以下に示します。Evictor のプロパティーが、3 つの必須属性のうち 2 つしか持っていないことに注意してください。名前属性と値属性は存在しますが、型属性が欠落しています。属性の欠落は XML 妥当性検査の失敗の原因になります。

```
<?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>
```

ソリューション

この問題を解決するには、 XML ファイルに必須属性を追加してください。ここで例示した XML ファイルでは、属性タイプを追加し、整数値を割り当てる必要があります。

必須エレメントの欠落

スキーマには数個の XML エレメントが必要です。それが存在しない場合、 XML は妥当性検査に失敗します。

メッセージ

必須エレメントが欠落している場合、以下のようなメッセージがログに表示されます。この場合、objectGrid エレメントが欠落しています。

```
[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..
```

このエラーに関する詳細情報を見るには、トレースを使用可能にしてください。ObjectGridManager のセクションでトレースをオンにする方法を紹介しています。

問題

この問題の発生に使用された XML ファイルを以下に示します。objectGrids エレメントが objectGrid 子エレメントを持っていないことに注意してください。XML スキーマに従って、 objectGrid エレメントは objectGrids タグで少なくとも 1 回は発生する必要があります。このエレメントの欠落は XML 妥当性検査の失敗の原因になります。

```
<?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>
```

ソリューション

この問題を修正するには、必須エレメントが XML ファイルにあることを確認してください。前の例では、 objectGrid エレメントは objectGrids タグに少なくとも 1 回は置かれる必要があります。一度必須エレメントが存在すると、正常に XML ファイルの妥当性検査を行えます。

以下の有効 XML ファイルは存在する必須エレメントを含んでいます。

```
<?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>
```

属性の XML 値が無効

メッセージ

XML ファイルの一部属性は一定の値にのみ割り当てできます。これらの属性はスキーマによって列挙された許容値を持ちます。そうした属性には以下のものがあります。

- objectGrid エレメントの authorizationMechanism 属性
- backingMap エレメントの copyMode 属性
- backingMap エレメントの lockStrategy 属性
- backingMap エレメントの ttlEvictorType 属性
- property エレメントの type 属性

これら属性の 1 つに無効値が割り当てられていると、XML 妥当性検査は失敗します。

属性が、列挙された値のどれにも一致しない値に設定された場合、以下のメッセージがログに表示されます。

```
[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..
```

以下はトレースを使用可能にしたログからの抜粋です。

```
[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)
...
```

問題

特定の値のセット以外の値を割り当てられた属性は、間違った設定をされたことになります。このケースでは、copyMode 属性が列挙された値のどれにも設定されません。INVALID_COPY_MODE に設定されました。このエラーの発生に使用された XML ファイルを以下に示します。

```
<?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>
```

ソリューション

この例では copyMode は無効値を持っています。属性を以下の有効な値の 1 つに設定してください。COPY_ON_READ_AND_COMMIT、COPY_ON_READ、COPY_ON_WRITE、NO_COPY。

インプリメンテーションのサポートなしの XML 妥当性検査

IBM Software Development Kit (SDK) バージョン 1.4.2 は、スキーマに対する XML 妥当性検査に使用できる幾つかの JAXP 機能のインプリメンテーションを含みます。

メッセージ

このインプリメンテーションを含まない SDK を使用する場合、妥当性検査の試みが失敗する場合があります。このインプリメンテーションを含まない SDK を使用して XML の妥当性検査を行いたい場合は、Xerces をダウンロードし、クラスパスに Java アーカイブ (JAR) ファイルを組み込みます。

必要なインプリメンテーションを持たない SDK で XML の妥当性検査をしようとすると、ログに以下のエラーが表示されます。

```
[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)
...
```

問題

使用した SDK は、スキーマに対して XML ファイルの妥当性検査をするのに必要な JAXP 機能のインプリメンテーションを含んでいません。

ソリューション

Apache Xerces をダウンロードし、 JAR をクラスパスに組み込むと、XML ファイルを正常に妥当性検査できます。

ObjectGrid メッセージ

この参照情報は、ObjectGrid の使用中に表示されるメッセージについての詳細情報を提供します。メッセージはメッセージ・キーにより識別され、説明とユーザー応答が示されます。メッセージは通知、警告、エラーのいずれかであり、どれであるかはメッセージ・キーの最後の文字 (I、W、E) で表されます。メッセージの説明部分で、メッセージが発生した理由が説明されます。メッセージのユーザー応答部分で、警告またはエラー・メッセージの場合、どのようなアクションを取るべきかが説明されます。

CWOBJ0001E: Method, {0}, was called after initialization completed.
説明: 初期化の完了後、一部メソッドの起動は受け入れを拒否されます。
ユーザー応答: ご使用のコードを再構築し、ランタイムの使用が開始される前に構成を完了するようにします。

ICWOBJ0002W: ObjectGrid component is ignoring an unexpected exception: {0}.
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ0005W: The thread created an InterruptedException: {0}
説明: InterruptedException が発生しました。
ユーザー応答: 例外メッセージをチェックし、この割り込みが予想通りであるかどうかを確認します。

CWOBJ0006W: An exception occurred: {0}
説明: 実行時に例外が発生しました。
ユーザー応答: 例外メッセージをチェックし、これが予想通りの例外かどうかを確認します。

CWOBJ0007W: The value null was specified for {0}, a default value of {1} is used.
説明: 変数にヌル値が指定されました。デフォルト値が使用されます。
ユーザー応答: 適切な値を設定します。ObjectGrid 文書を参照し、変数またはプロパティーに対する有効な値を確認してください。

CWOBJ0008E: The value {0} provided for the property {1} is invalid.
説明: 変数に無効値が指定されました。
ユーザー応答: 適切な値を設定します。ObjectGrid 文書を参照し、変数またはプロパティーに対する有効な値を確認してください。

CWOBJ0010E: Message key {0} is missing.
説明: メッセージ・リソース・バンドルにメッセージ・キーが欠落しています。
ユーザー応答: CMSG0002

CWOBJ0011W: Cannot deserialize field {0} in class {1}; using the default value.
説明: オブジェクトのデシリアライズ中に、予想通りのフィールドが検出されませんでした。このフィールドは、シリアライズしたバージョンのクラスと異なるバージョンのクラスでオブジェクトがデシリアライズされたため、検出されなかったと考えられます。
ユーザー応答: この警告は、潜在的な問題を示しています。さらにエラーが発生しない限り、ユーザー処置は必要ありません。

CWOBJ0012E: The LogElement type code, {0} ({1}), is not recognized for this operation.
説明: ObjectGrid ランタイムで内部エラーが発生しました。
ユーザー応答: CMSG0002

CWOBJ0013E: An exception occurred while attempting to evict entries from the cache: {0}
説明: キャッシュへのエントリー除外の適用時に問題が発生しました。
ユーザー応答: 例外メッセージをチェックし、これが予想通りの例外かどうかを確認します。

CWOBJ0014E: The ObjectGrid runtime detected an attempt to nest transactions.
説明: トランザクションのネストは許可されません。
ユーザー応答: コードを変更し、トランザクションのネストを避けます。

CWOBJ0015E: An exception occurred while attempting to process a transaction: {0}
説明: トランザクションの処理時に問題が発生しました。
ユーザー応答: 例外メッセージをチェックし、この例外が予想通りであるかどうかを確認します。

CWOBJ0016E: No active transaction is detected for the current operation.
説明: このオペレーションを実行するにはアクティブなトランザクションが必要です。
ユーザー応答: コードを変更し、このオペレーションの実行前にトランザクションが開始されるようにします。

CWOBJ0017E: A duplicate key exception was detected during the processing of the ObjectMap operation: {0}
説明: エントリー用のキーは既にキャッシュにあります。
ユーザー応答: コードを変更し、同じキーを 2 回以上挿入することを避けます。

CWOBJ0018E: The key was not found during the processing of the ObjectMap operation: {0}
説明: エントリー用のキーがキャッシュに存在しません。
ユーザー応答: コードを変更し、このオペレーションを実行する前にエントリーの存在を確認するようにします。

CWOBJ0019W: Did not find data in the cache entry slot reserved for {0} to use for ObjectMap name {1}.
説明: ObjectGrid ランタイムで内部エラーが発生しました。
ユーザー応答: CMSG0002

CWOBJ0020E: Cache entry is not in BackingMap {0}.
説明: ObjectGrid ランタイムの内部エラー。
ユーザー応答: CMSG0002

CWOBJ0021E: A usable ObjectTransformer instance was not found during the deserialization of the LogSequence object for {0} ObjectGrid and {1} ObjectMap.
説明: LogSequence オブジェクト受信サイドが、必要な ObjectTransformer インスタンスをサポートするための適切な構成になっていません。
ユーザー応答: LogSequence オブジェクトの送信サイドおよび受信サイドの両方に対し、ObjectGrid インスタンスの構成を検査します。

CWOBJ0022E: The caller does not own mutex: {0}.
説明: ObjectGrid ランタイムで内部エラーが発生しました。
ユーザー応答: CMSG0002

CWOBJ0023E: The CopyMode ({0}) is not recognized for this operation.
説明: ObjectGrid ランタイムで内部エラーが発生しました。
ユーザー応答: CMSG0002

CWOBJ0024E: Cannot deserialize field {0} in class {1}. Deserialization failed.
説明: オブジェクトのデシリアライズ中に、必要フィールドが検出されませんでした。この問題は、ObjectGrid ランタイム・エラーと考えられます。
ユーザー応答: CMSG0002

CWOBJ0025E: The serialization of the LogSequence object failed. The number of serialized LogElement objects ({0}) does not match the number of read LogElement objects ({1}).
説明: ObjectGrid ランタイムで内部エラーが発生しました。
ユーザー応答: CMSG0002

CWOBJ0026E: The JMX credential type is not right. It should be of type {0}.
説明: JMX クリデンシャル型が正しくありません。基本認証が使用されている場合、期待される型は String[] であり、最初のエレメントはユーザー名、2 番目のエレメントはパスワードです。クライアント証明書が使用されている場合、期待される型は Certificate[] です。
ユーザー応答: 正しいクリデンシャルを使用します。

CWOBJ0027E: Internal runtime error. Clone method not supported: {0}
説明: ObjectGrid ランタイムで内部エラーが発生しました。
CLONE_METHOD_NOT_SUPPORTED_CWOBJ0027.useraction=CMSG0002

CWOBJ0028E: An error occurred in {0} for the map {1}. The key,
{2}, was not found in the map. LogElement type is {3}.
説明: エントリーの除外時に内部エラーが発生しました。
ユーザー応答: CMSG0002

CWOBJ0029E: An error occurred in {0} for the map {1}. CacheEntry is
missing a {2} object for key {3}. LogElement type is {4}.
説明: エントリーの除外時に内部エラーが発生しました。
ユーザー応答: CMSG0002

CWOBJ0900I: The ObjectGrid runtime component is started for server {0}.
説明: ObjectGrid コンポーネントが開始されました。
ユーザー応答: なし。通知です。

CWOBJ0901E: "{0}" system property is required to start ObjectGrid
component for server {1}.
説明: ObjectGrid ランタイム・コンポーネントでは、"{0}" を Java 仮想マシンの
システム・プロパティーとして指定する必要があります。
ユーザー応答: WebSphere 管理コンソールを使用して ObjectGrid で必要なカスタム・
プロパティーを提供する方法について、インフォメーション・センターを
参照してください。

CWOBJ0902W: Error prevented the ObjectGrid runtime component from starting
for server {0}.
説明: 前のエラーにより、ObjectGrid コンポーネントの開始が妨げられました。
ユーザー応答: 前のエラー・メッセージを確認し、何が ObjectGrid コンポーネントの
開始を妨げたか判別します。

CWOBJ0910I: The ObjectGrid runtime component is stopped for server {0}.
説明: ObjectGrid コンポーネントが停止しました。
ユーザー応答: なし。通知です。

CWOBJ0911I: Starting the ObjectGrid runtime component for server {0}.
説明: ObjectGrid コンポーネントが開始中です。
ユーザー応答: なし。通知です。

CWOBJ1001I: ObjectGrid Server {0} is ready to process requests.
説明: ObjectGrid サーバーは要求の処理が可能です。
ユーザー応答: この ObjectGrid サーバーのサービスは使用可能です。

CWOBJ1002E: Server port {0} is already in use.
説明: ObjectGrid サーバーは、ポートの競合のため開始できません。
ユーザー応答: 別のポートを選択する必要があります。

CWOBJ1003I: DCS Adapter service is disabled by configuration,
to enable it, please change your configuration with an endpoint defined.
説明: DCS アダプターがオフにされています。
ユーザー応答: DCS アダプターは、構成を変更することによりオンにできます。

CWOBJ1004E: Server topic is null
説明: サーバー・トピックがヌルです。
ユーザー応答: CMSG0002

CWOBJ1005E: The incoming request queue is null.
説明: クライアント要求ハンドラーが要求を取得できません。
ユーザー応答: CMSG0002

CWOBJ1006E: The outgoing result queue is null.
説明: クライアント要求ハンドラーがクライアントに結果を提供できません。
ユーザー応答: CMSG0002

CWOBJ1007E: ObjectGrid client request is null.
説明: クライアント要求ハンドラーは、要求に関する情報を含まない要求を
処理できません。
ユーザー応答: 要求を確認します。

CWOBJ1008E: ObjectGrid client request TxID is null.
説明: 接続の突き合わせとプールの実行に TXID が使用されるため、TXID はヌルにできません。
ユーザー応答: CMSG0002

CWOBJ1009E: ObjectGrid client received a null response from the server.
説明: サーバーからの応答がヌルでした。
ユーザー応答: CMSG0002

CWOBJ1010I: Shutdown request is processing.
説明: クラスター・サーバーはシャットダウン要求を処理中です。
ユーザー応答: なし。

CWOBJ1011I: Shutdown request is sending.
説明: クラスター・サーバーはシャットダウン要求を処理中です。
ユーザー応答: なし。

CWOBJ1012I: Shutdown request is performed.
説明: クラスター・サーバーはシャットダウン要求を処理中です。
ユーザー応答: なし。

CWOBJ1110I: Starting the transport for ObjectGrid cluster {0} using IP Address {1}, port {2}, transport type {3}.
説明: ObjectGrid クラスター・メンバーのトランスポートが開始中です。
ユーザー応答: なし。通知です。

CWOBJ1111W: Resolution of IP Addresses for host name {0} found only the loopback address. The loopback address will be used.
説明: ホスト名または DNS の解決に問題がある可能性があります。実動に関連するインプリメンテーションでは通常、非ループバック・アドレスが期待されます。
ユーザー応答: ホスト名を変更するか、または DNS に問題があるかどうか判別します。

CWOBJ1112E: An error was encountered while looking up the IP address for the host name of an ObjectGrid cluster member. The host name is {0} and the server name is {1}. The member will be excluded from the cluster.
説明: 指示されたホストの IP アドレスを解決できません。指定されたホストの ObjectGrid クラスター・メンバーは除外されます。
ユーザー応答: ホスト名ルックアップの問題を訂正し、再試行します。

CWOBJ1113E: ObjectGrid cluster transport service on this process is not started. This cluster member is not defined in the configuration.
説明: この ObjectGrid クラスター・メンバーは、クラスターの構成済みメンバーではありません。このクラスター・メンバーが ObjectGrid クラスターのメンバーである必要がある場合は、構成を修復します。
ユーザー応答: 現行の構成について検討します。

CWOBJ1114E: ObjectGrid cluster transport service on this process could not process the incoming message. The message is {0} and the exception is {1}.
説明: 予期しない内部エラーが検出されました。
ユーザー応答: 同様の問題について IBM ObjectGrid インターネット・サポート Web サイトを参照するか、または IBM サービスにお問い合わせください。

CWOBJ1115E: An unrecognized view change event was received from the ObjectGrid cluster transport. The view identifier is {0} and the event is {1}.
説明: イベントのタイプが認識されません。HA マネージャーは、そのイベントへの応答方法を認識していません。
ユーザー応答: 同様の問題について IBM ObjectGrid インターネット・サポート Web サイトを参照するか、または IBM サービスにお問い合わせください。

CWOBJ1116E: An attempt by another process to connect to this process via the ObjectGrid cluster transport has been rejected. The connecting process provided a name of {0}, a target of {1}, a member name of {2} and an IP address of {3}. The error message is {4}.
説明: ObjectGrid クラスター・トランスポートにより、接続の試行が拒否されました。
ユーザー応答: 非認証のパーティーからの接続の試行の可能性があります。

CWOBJ1117E: An attempt to authenticate a connection has failed. The exception is {0}.
説明: ObjectGrid クラスター・トランスポートにより、

接続の試行が拒否されました。
ユーザー応答:
非認証のパーティーからの接続の試行の可能性があります。

CWOBJ1118I: ObjectGrid Server Initializing [Cluster: {0} Server: {1}].
説明: ObjectGrid クラスター・メンバーは初期化中です。
ユーザー応答: なし。通知です。

CWOBJ1119I: ObjectGrid client failed to connect to host: {0} port: {1}.
説明: ObjectGrid クライアントは接続に失敗しました。
ユーザー応答: なし。通知です。

CWOBJ1120I: ObjectGrid Client connected successfully to host: {0} port: {1}.
説明: ObjectGrid クライアントは正常に接続されました。
ユーザー応答: なし。通知です。

CWOBJ1201E: No valid client access end points are defined.
説明: 有効なクライアント・アクセス・エンドポイントが定義されていません。
ユーザー応答: 有効なクライアント・アクセス・エンドポイントを定義します。

CWOBJ1202E: SSL Server Socket failed to initialize. The exception message is {0}
説明: SSL サーバー・ソケットが初期化に失敗しました。SSL 設定が誤っているか、ポート番号が既に使用されている可能性があります。
ユーザー応答: 例外を調査し、何が誤っているか確認します。

CWOBJ1203W: Received a timeout event from the server for transaction: {0}
説明: 構成されたタイムアウトの限度内で、クライアントはサーバーから期待される応答メッセージを受信しませんでした。
ユーザー応答: タイムアウトを説明していると考えられる前のメッセージを探します。見つらなかった場合は、タイムアウトの限度の増加を試みます。

CWOBJ1204W: Received a message of unknown message type.
説明: 予期しない内部エラーが検出されました。
ユーザー応答: 同様の問題について IBM ObjectGrid インターネット・サポート Web サイトを参照するか、または IBM サービスにお問い合わせください。

CWOBJ1205E: SSL Initialization failed. The exception message is {0}
説明: SSL の初期化に失敗しました。SSL 設定が誤っている可能性があります。
ユーザー応答: 例外を調査し、何が誤っているか確認します。

CWOBJ1206W: SSL Initialization failed. The exception message is {0}
説明: SSL の初期化に失敗しました。SSL 設定が誤っている可能性があります。
ユーザー応答: 例外を調査し、何が誤っているか確認します。

CWOBJ1207W: The property {0} on plug-in {1} is using an unsupported property type.
説明: Java プリミティブとその java.lang カウンター・パートのみが、サポートされるプロパティー・タイプです。java.lang.String もサポートされています。
ユーザー応答: プロパティー・タイプを確認し、サポートされているタイプのいずれかに変更します。

CWOBJ1208W: The specified plug-in type, {0}, is not one of the supported plug-in types.
説明: このタイプのプラグインはサポートされていません。
ユーザー応答: サポートされているタイプのプラグインのいずれかを追加します。

CWOBJ1211E: The Performance Monitoring Infrastructure (PMI) creation of {0} failed. The exception is {1}.
説明: ObjectGrid PMI の作成に失敗しました。
ユーザー応答: 例外メッセージ、および First Failure Data Capture (FFDC) ログを調査します。

以下のメッセージは、パネルまたは標準入力においてユーザー ID およびパスワードの収集に使用されます。
LOGIN_PANEL_TITLE=ターゲット・サーバーでログイン
GENERIC_LOGIN_PROMPT=ログイン情報を入力してください

USER_ID=ユーザー ID
PASSWORD=パスワード
OK=OK
CANCEL=キャンセル

CWOBJ1215I: ObjectGrid Transaction Propagation Event Listener is initializing [ObjectGrid {0}].
説明: この通知メッセージは、ObjectGrid トランザクション伝搬イベント・リスナーが初期化中であることを示します。
ユーザー応答: なし。通知です。

CWOBJ1216I: ObjectGrid Transaction Propagation Event Listener is initialized [ObjectGrid {0}].
説明: ObjectGrid トランザクション伝搬イベント・リスナー初期化済み。
ユーザー応答: なし。通知です。

CWOBJ1217I: ObjectGrid Transaction Propagation Service Point Initialized [ObjectGrid {0}].
説明: この通知メッセージは、ObjectGrid トランザクション伝搬イベント・リスナーが初期化されたことを示します。
ユーザー応答: なし。通知です。

CWOBJ1218E: ObjectGrid Transaction Propagation Event Listener failure occurred [ObjectGrid {0} Exception message {1}].
説明: ObjectGrid ランタイムで ObjectGrid トランザクション伝搬の障害が発生しました。
ユーザー応答: 例外を調査し、障害を判別します。

CWOBJ1219E: ObjectGrid Transaction Propagation Service End Point failure occurred [ObjectGrid {0} Exception message {1}].
説明: ObjectGrid ランタイムで ObjectGrid トランザクション伝搬サービス・エンドポイントの障害が発生しました。
ユーザー応答: 例外を調査し、障害を判別します。

CWOBJ1220E: ObjectGrid Transaction Propagation Service is not supported in this environment.
説明: ObjectGrid トランザクション伝搬サービスは、z/OS またはスタンドアロン ObjectGrid サーバーの環境ではサポートされていません。
ユーザー応答: z/OS またはスタンドアロン ObjectGrid サーバー環境では、ObjectGrid トランザクション伝搬サービスを使用しないでください。

CWOBJ1300I: Adapter successfully initialized ObjectGrid.
説明: アダプターにより、正常に ObjectGrid が初期化されました。
ユーザー応答: なし。通知です。

CWOBJ1301E: Adapter failed to initialize ObjectGrid. Exception occurred [Exception message {0}].
説明: ObjectGrid を初期化するアダプターが障害を起こしました。
ユーザー応答: 例外を調査し、障害を判別します。

CWOBJ1302I: Adapter stopped.
説明: アダプターが停止しました。
ユーザー応答: なし。通知のみです。

CWOBJ1303I: Adapter started.
説明: PMA_CWOBJ1303.explanation = アダプターが開始しました。
ユーザー応答: なし。通知のみです。

CWOBJ1304I: ObjectGrid security is enabled.
説明: ObjectGrid セキュリティーは使用可能になりました。
ユーザー応答: なし。

CWOBJ1305I: ObjectGrid security is disabled.
説明: ObjectGrid セキュリティーは使用不可になりました。
ユーザー応答: なし。

CWOBJ1306W: Cannot retrieve the client certificates from the SSL socket.
説明: 何らかの理由で、ランタイムが SSL ソケットからクライアント証明書を取得できません。
ユーザー応答: SSL 構成を確認します。

CWOBJ1307I: Security of the ObjectGrid instance {0} is enabled.
説明: ObjectGrid インスタンス {0} のセキュリティーが使用可能になりました。
ユーザー応答: なし。

CWOBJ1308I: Security of the ObjectGrid instance {0} is disabled.
説明: ObjectGrid インスタンス {0} のセキュリティーが使用不可になりました。
ユーザー応答: なし。

CWOBJ1309E: Unexpected error occured in the
connect token creation: {0}.
説明: 接続トークンの作成で予期しないエラーが
発生しました。
ユーザー応答: セキュリティー構成を確認します。

CWOBJ1310E: An attempt by another process to
connect to this process via the core group transport has been rejected. The
connecting process provided a source core group name of {0}, a target of {1},
a member name of {2} and an IP address of {3}. The error message is {4}.
説明: HA マネージャーにより接続の試行が拒否されました。
ユーザー応答: 非認証のパーティーからの接続の試行の可能性があります。

CWOBJ1400W: Detected multiple ObjectGrid runtime JARS files in the JVM.
Using multiple ObjectGrid runtime JAR files may cause problems.
説明: 通常、JVM にある ObjectGrid ランタイム JAR は
1 つのみです。
ユーザー応答: 適切な ObjectGrid ランタイム JAR の構成を
使用します。

CWOBJ1401E: Detected a wrong ObjectGrid runtime JAR file for this
configuration. Detected configuration is {0}. Expected Jar file is {1}.
説明: それぞれの ObjectGrid ランタイム JAR ファイルは、特定のサポートされる構成に
対応しています。
ユーザー応答: 適切な ObjectGrid ランタイム JAR の構成を
使用します。

CWOBJ1402E: ObjectGrid connection link callback not found for id: {0}.
説明: ObjectGrid ランタイムの内部エラー。
ユーザー応答: CMSG0002

CWOBJ1500E: An exception occurred when attempting to create a
GroupName for HA Group ({0}): {1}.
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ1501E: An exception occurred when member ({0}) attempted
to join HA Group ({1}): {2}.
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ1503E: Cannot access ObjectGrid ({0}) for applying updates
to replica member ({1}).
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ1504E: An exception occurred when attempting to
process the LogSequences for replica ({0}): {1}.
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ1505E: More than one replication group member
reported back as the primary. Only one primary can be active. ({0}).
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ1506E: More than one primary replication group member
exists in this group ({1}). Only one primary can be active. ({0}).
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ1507W: An exception occurred when attempting to end the replication process for BackingMap ({0}): {1}.
説明: 1 次複製グループ・メンバーのシャットダウン時の、クリーンアップの処理中に例外が発生しました。
ユーザー応答: CMSG0002

CWOBJ1508E: An exception occurred when attempting to send message ({0}) from sender ({1}) to receiver ({2}): {3}.
説明: 複製グループ・メンバー間のメッセージ送信時に問題が発生しました。
ユーザー応答: CMSG0002

CWOBJ1509E: An exception occurred when attempting to serialize message ({0}): {1}.
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ1510E: An exception occurred when attempting to inflate message ({0}): {1}.
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ1511I: {0} ({1}) is open for business.
説明: 指定された複製グループ・メンバーは現在、要求を受け入れ可能です。
ユーザー応答: なし。

CWOBJ1512W: {0} already exists in replication group {1}.
説明: 指定された複製グループ・メンバーは、この複製グループにおいて既に活動中です。
ユーザー応答: なし。

CWOBJ1513E: Synchronous replication failed on {0} ({1}). This member is no longer active.
説明: 問題が発生し、同期複製が正常に完了しませんでした。
ユーザー応答: ログ内の前のメッセージを検討し、問題を診断します。指定されたサーバーの停止および再始動が必要な場合があります。

CWOBJ1514I: Primary ({0}) is being downgraded to either a replica or standby.
説明: これは正常な操作ではありませんが、ObjectGrid の処理は継続できます。
ユーザー応答: CMSG0002

CWOBJ1515I: Minimum configuration requirements not satisfied for replication group ({0}).
説明: 最新の複製グループ・メンバーの変更において、必要な 1 次およびレプリカの構成要件が満たされませんでした。
ユーザー応答: 追加のリソースが開始され、この構成が認識されるまで待機します。

CWOBJ1516E: An exception occurred when attempting to activate the replication process for ObjectGrid ({0}): {1}.
説明: 1 次複製グループ・メンバーの開始時の、活動化の処理中に例外が発生しました。
ユーザー応答: CMSG0002

CWOBJ1517E: Synchronous replication failed for transaction {2} on {0} ({1}). This member is no longer active.
説明: 問題が発生し、同期複製が正常に完了しませんでした。
ユーザー応答: ログ内の前のメッセージを検討し、問題を診断します。指定されたサーバーの停止および再始動が必要な場合があります。

CWOBJ1518E: An exception occurred when attempting to commit replica transaction ({0}) for primary transaction ({1}) on Replica ({2}): {3}.
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ1519E: An exception occurred when attempting to rollback the LogSequences for replica ({0}): {1}.
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ1610W: Try to reset a null cluster for {0}.
説明: 複製グループ・クラスターのデータがありません。
ユーザー応答: なし。

CWOBJ1611I: Replication group cluster {0} is open for business.
説明: 複製グループ・クラスターは現在、要求を受け入れ可能です。
ユーザー応答: なし。

CWOBJ1612I: Replication group cluster {0} is closed for business.
説明: 複製グループ・クラスターは現在、要求を受け入れることができません。
ユーザー応答: なし。

CWOBJ1620I: Replacing target for wrongly routed request due to changes in the server. The new target is {0}.
説明: 以前のルーティング・ターゲットは新規ターゲットに置き換えられました。
ユーザー応答: 対象とする複製グループがサービスを休止している場合は、元に戻す必要があります。

CWOBJ1630I: Replication group cannot serve this request {0}.
説明: Domino の影響などのサービスが使用不可である影響のため、ルーティングが拒否されました。
ユーザー応答: 通知のみです。

CWOBJ1632E: Original request does not have a valid ID; no way to forward this request.
説明: 元の要求に有効な ID がないため、この要求を転送する方法がありません。
ユーザー応答: IBM サポートに報告してください。

CWOBJ1634I: Router cannot find the forwarding target; using blind forwarding.
説明: ルーターが転送先を検出できません。
ユーザー応答: なし。

CWOBJ1660I: Replication group member has changed. This server does not host what is requested anymore. The original request is {0}.
説明: 複製グループ・メンバーが変更されました。
ユーザー応答: 対象とする複製グループがサービスを休止している場合は、元に戻す必要があります。

CWOBJ1661I: Cluster data are updated for replication group: {0}
説明: クラスター・データが更新されています。
ユーザー応答: なし。

CWOBJ1663E: Server router cannot verify server routing for {0}, because cluster data for this replication group are null in the server.
説明: 検査する複製グループ・クラスターのデータがありません。
ユーザー応答: IBM サポートに報告してください。

CWOBJ1668W: Request is coming to the server that has not completely started.
説明: サーバーの始動には 60 秒から 120 秒程度必要です。構成に応じて、要求は自動的に再試行されます (デフォルトの構成では、要求は自動的に再試行されます)。
ユーザー応答: 構成を調整するか、またはサーバーの始動の 60 秒から 120 秒後にクライアントを開始します。

CWOBJ1680W: The configured TCP connection timeout is smaller than retryInterval * max(startupRetries, maxRetries), so there is possibility that connection will time out.
説明: TCP 接続タイムアウトの構成は、retryInterval * max(startupRetries, maxRetries) より大きくする必要があります。
ユーザー応答: 構成を調整します。

CWOBJ1682W: The configured transaction timeout is smaller than maxForwards * retryInterval * max(startupRetries, maxRetries), so there is possibility that transaction will time out.
説明: トランザクション・タイムアウトの構成は、maxForwards * retryInterval * max(startupRetries, maxRetries) より大きくする必要があります。
ユーザー応答: 構成を調整します。

CWOBJ1700I: Standalone HAManager
is initialized with coregroup {0}.
説明: スタンドアロン HAManager は正常に初期化されました。
ユーザー応答: なし。

CWOBJ1701I: Standalone HAManager is already initialized.
説明: スタンドアロン HAManager は既に正常に初期化済みです。
ユーザー応答: なし。

CWOBJ1702E: Standalone HAManager is
not initialized, so it cannot be started.
説明: スタンドアロン HAManager は初期化されていません。
ユーザー応答: 開始前に初期化します。

CWOBJ1710I: Standalone HAManager is
started successfully.
説明: スタンドアロン HAManager は正常に開始されました。
ユーザー応答: なし。

CWOBJ1711I: Standalone HAManager is already started successfully.
説明: スタンドアロン HAManager は既に正常に開始済みです。
ユーザー応答: なし。

CWOBJ1712E: Standalone HAManager is not started.
説明: スタンドアロン HAManager は開始されていません。
ユーザー応答: 使用する前に、初期化して開始します。

CWOBJ1713E: Standalone HAManager failed to start.
説明: スタンドアロン HAManager は開始に失敗しました。
ユーザー応答: ポートが既に使用されていないかどうか確認します。

CWOBJ1720I: HAManager Controller detected that
ObjectGrid server is in the WebSphere environment, using WebSphere HAManager
instead of initializing and starting standalone HAManager.
説明: ObjectGrid サーバーは WebSphere 環境で実行中です。
ユーザー応答: なし。

CWOBJ1730I: HAManager Controller detected
that the WebSphere external HAManager is null.
説明: 外部 HAManager を WebSphere から取得できません。
ユーザー応答: なし。

CWOBJ1790I: Need to initialize and
start the standalone HAManager.
説明: 外部 HAManager を WebSphere から取得できません。
スタンドアロン HAManager を初期化して開始する必要があります。
ユーザー応答: なし。

CWOBJ1792I: The maximum number of threads is {0} and the minimum number
of threads is {1}.
説明: スレッド・プールを構成します。
ユーザー応答: 通知のみです。

CWOBJ1800I: Forwarding is required for request {0} with response of {1}.
説明: 転送のルーティングが必要です。
ユーザー応答: なし。自動的に処理されます。

CWOBJ1810I: Forwarding is required for response {0}.
説明: 応答には転送が必要です。
ユーザー応答: なし。

CWOBJ1811E: Forwarding is required, but the
original request cannot be found.
説明: 転送が必要ですが、元の要求が検出できません。
ユーザー応答: なし。

CWOBJ1820E: Forwarding request does not have a
replication group identifier.
説明: この転送要求には複製グループ ID がありません。
ユーザー応答: IBM サポートにお問い合わせください。

CWOBJ1870I: Server service is not available for response {0}.
説明: Domino の影響またはその他のイベントのため、サーバー・サービスは使用不可です。
ユーザー応答: 少なくとも、最小数のサーバーは稼働します。

CWOBJ1871E: Detected unavailable service, received null response, no way to retry.
説明: 使用不可のサービスからのヌル応答です。
ユーザー応答: IBM サポートにお問い合わせください。

CWOBJ1872I: Service is unavailable with response of {0}.
説明: サービスが使用不可です。
ユーザー応答: 少なくとも最小数のサーバーを稼働させるか、またはサーバーの始動が正常であるか確認します。

CWOBJ1890I: Re-routing request {0} due to an un-responsive server.
説明: 対象とするサーバーへの要求の完了に失敗しました。要求は別のサーバーに再送付されます。
ユーザー応答: なし。自動的に処理されます。対象とする複製グループがサービスを休止している場合は、元に戻す必要があります。

CWOBJ1891E: All servers are not available in replication group {0}.
説明: すべてのサーバーが開始されなかったか、または障害を起こしました。使用不可です。
ユーザー応答: 対象とする複製グループがサービスを休止している場合は、元に戻す必要があります。

CWOBJ1898W: Forwarding is required, but router cannot find new available target for response {0}
説明: サービスが使用不可です。
ユーザー応答: サービスを使用可能にしてください。

CWOBJ1899W: Forwarding is required, but router cannot find right replication group for response {0}
説明: 複製グループ ID が失われました。
ユーザー応答: IBM サポートにお問い合わせください。

CWOBJ1900I: Client server remote procedure call service is initialized.
説明: クライアント・サーバーのリモート・プロシージャー・コール・サービスが初期化されました。
ユーザー応答: なし。

CWOBJ1901I: Client server remote procedure call service is started.
説明: クライアント・サーバーのリモート・プロシージャー・コール・サービスが開始されました。
ユーザー応答: なし。

CWOBJ1902I: Client server remote procedure call handler threads are started.
説明: クライアント・サーバーのリモート・プロシージャー・コール・ハンドラー・スレッドが開始されました。
ユーザー応答: なし。

CWOBJ1903I: Configuration network service is initialized.
説明: 構成ネットワーク・サービスが初期化されました。
ユーザー応答: なし。

CWOBJ1904I: Configuration network service is started.
説明: 構成ネットワーク・サービスが開始されました。
ユーザー応答: なし。

CWOBJ1905I: Configuration handler is started.
説明: 構成ハンドラーが開始されました。
ユーザー応答: なし。

CWOBJ1913I: System administration network service is initialized.
説明: システム管理ネットワーク・サービスが初期化されました。
ユーザー応答: なし。

CWOBJ1914I: System administration network service is started.
説明: システム管理ネットワーク・サービスが開始されました。
ユーザー応答: なし。

CWOBJ1915I: System administration handler is started.
説明: システム管理ハンドラーが開始されました。
ユーザー応答: なし。

CWOBJ2000E: No member in this replication group {0}.
説明: この複製グループにはメンバーが検出されません。
ユーザー応答: サーバーが始動済みであるか、またはデータが使用可能であるか確認します。

CWOBJ2001W: No available member in this replication group {0}.
説明: この複製グループには、使用可能なメンバーが検出されません。
ユーザー応答: サーバー・サービスが使用可能であるか確認します。

CWOBJ2002W: No available routing table for this replication group {0}.
説明: この複製グループには、使用可能なルーティング・テーブルがありません。
ユーザー応答: クライアントがルーティング・テーブルにあるか確認します。

CWOBJ2003I: Cannot find routing cache for cache key {0}, creating new routing cache.
説明: 最初のルーティングまたはクラスター変更です。
ユーザー応答: なし。

CWOBJ2010E: Target for this request is null.
説明: 要求に宛先情報がありませんでした。
ユーザー応答: IBM サポートにお問い合わせください。

WOBJ2060I: Client received new version of replication group cluster {0}.
説明: クライアントは、新規バージョンの複製グループ・クラスターを受け取りました。
ユーザー応答: なし。

CWOBJ2068I: Reachability control detected problem in replication group member {0}.
説明: 一部のサーバーに到達できませんが、到達可能性のメカニズムにより処理されます。
ユーザー応答: なし。

CWOBJ2069I: Reachability control timer releases replication group member {0}.
説明: このメンバーはルーティングに使用可能です。
ユーザー応答: なし。

CWOBJ2086I: Routing thread control is activated due to overload for replication group {0}.
説明: スレッド制御が作動しています。
ユーザー応答: なし。

CWOBJ2088I: Reachability control is activated to regulate the server availability for replication group {0}.
説明: 到達可能性が作動しています。
ユーザー応答: なし。

CWOBJ2090W: Cannot find routing table for replication group {0}.
説明: 複製グループ・クラスターがヌルです。
ユーザー応答: なし。

CWOBJ2091W: Routing table is not null, but it does not contain any servers for replication group {0}.
説明: 複製グループ・クラスターが空です。
ユーザー応答: なし。

CWOBJ2092I: Routing table is null in runtime for replication group {0}.
説明: ルーティング・テーブルをランタイムから取得しています。
ユーザー応答: なし。

CWOBJ2093I: Routing table is not null in replication group cluster store for replication group {0}
説明: ルーティング・テーブルをクラスター・ストアから取得しています。
ユーザー応答: なし。

CWOBJ2096I: Routing table was obtained from replication group cluster store for replication group {0}.
説明: 複製グループ・クラスターを複製グループ・クラスター・ストアから取得しました。
ユーザー応答: なし。

CWOBJ2097I: Routing is based on round robin algorithm for replication group {0}.
説明: ルーティングはラウンドロビン・アルゴリズムを基にしています。
ユーザー応答: なし。

CWOBJ2098I: Routing is based on random selection for replication group {0}.
説明: ルーティングはランダム選択を基にしています。
ユーザー応答: なし。

CWOBJ2100I: Connection ({0}) is stale, it cannot be reused.
説明: 接続が不整合です。
ユーザー応答: なし。

CWOBJ2101W: Connection cannot be acquired after the maximum wait time.
説明: プールには接続が残されていません。
ユーザー応答: 構成で最大接続数を増加します。

CWOBJ1600I: ManagementGateway service started on port ({0}).
説明: ManagementGateway サービスは要求の処理が可能です。
ユーザー応答: ManagementGateway サービスは使用可能です。

CWOBJ1601E: ManagementGateway service failed to start on port ({0}).
説明: ManagementGateway サービスは開始に失敗しました。
ユーザー応答: 指定したポートが既に使用されていないことを確認します。

CWOBJ1602E: ManagementGateway service failed to connect to server at ({0}):({1}).
説明: ManagementGateway サービスは、サーバーへの接続に失敗しました。
ユーザー応答: サーバーが実行中であることを確認します。

CWOBJ1603E: Management service failed to respond to ({0}) remote request.
説明: CMSG0001
ユーザー応答: CMSG0002

CWOBJ2400E: Invalid Configuration: backing map {0} is a member of more than one map-set.
説明: backingMap は 1 つのマップ・セットにのみ所属できます。
ユーザー応答: クラスター XML ファイルを編集し、各バッキング・マップが 1 つのマップ・セットにのみ所属するようにします。

CWOBJ2401E: Invalid Configuration: backing map {0} in distributed ObjectGrid {1} is not in a map-set.
説明: 分散 ObjectGrid の各バッキング・マップは、マップ・セットに配置する必要があります。
ユーザー応答: クラスター XML ファイルを編集し、分散 ObjectGrid 内の各バッキング・マップがマップ・セットに所属するようにします。

CWOBJ2402E: Invalid Configuration: map-set has a reference to a {0} map. This backing map does not exist in the ObjectGrid XML file.
説明: マップ・セット内の各マップは、ObjectGrid XML ファイルからのバッキング・マップを参照する必要があります。
ユーザー応答:
XML ファイル (複数可) を編集し、マップ・セット内の各マップが ObjectGrid XML ファイルからのバッキング・マップを参照するようにします。

CWOBJ2403E: The XML file is invalid. A problem has been detected with {0} at line {1}. The error message is {2}.
説明: XML ファイルはスキーマに準拠していません。
ユーザー応答: XML ファイルを編集し、スキーマに準拠するようにします。

CWOBJ2404W: The value specified for {0} is {1}. This is an invalid value. {0} will not be set.
説明: この構成属性の値は無効です。
ユーザー応答: XML ファイルにおいて、構成属性に適正な値を設定します。

CWOBJ2405E: The objectgrid-binding ref {0} in the Cluster XML file does not reference a valid ObjectGrid from the ObjectGrid XML file.
説明: それぞれの objectgrid バインディングは、ObjectGrid XML ファイルからの ObjectGrid を参照する必要があります。
ユーザー応答: XML ファイルを編集し、クラスター XML 内の objectgrid バインディングが ObjectGrid XML 内の有効な ObjectGrid を参照するようにします。

CWOBJ2500E: Failed to start ObjectGrid server {0}.
説明: ObjectGrid サーバーは正常な始動に失敗しました。
ユーザー応答: ログで例外を確認します。

CWOBJ2501I: Launching ObjectGrid server {0}.
説明: ObjectGrid サーバーは始動中です。
ユーザー応答: なし。

CWOBJ2502I: Starting ObjectGrid server using ObjectGrid XML file URL "{0}" and Cluster XML file URL "{1}".
説明: ObjectGrid サーバーは、クラスター XML ファイルおよび ObjectGrid XML ファイルを使用して始動中です。
ユーザー応答: なし。

CWOBJ2503I: Bootstrapping to a peer Objectgrid server on host {0} and port {1}.
説明: この ObjectGrid サーバーはピア・サーバーにブートストラップし、始動に必要な情報を取得します。
ユーザー応答: なし。

COBJ2504I: Attempting to bootstrap to a peer ObjectGrid server using the following host(s) and port(s) "{0}".
説明: この ObjectGrid サーバーは、ピア ObjectGrid サーバーへの接続の試行時に提供された、ホストおよびポートのリストを使用します。
ユーザー応答: なし。

CWOBJ2505I: Attempting to bootstrap to a peer ObjectGrid server using the Cluster XML file URL "{0}".
説明: この ObjectGrid サーバーは、ピア ObjectGrid サーバーへの接続の試行時の、クラスター XML ファイル内のサーバーのリストを使用します。
ユーザー応答: なし。

CWOBJ2506I: Trace is being logged to {0}.
説明: トレース・ファイルがコマンド行で設定されました。
ユーザー応答: ObjectGrid サーバーの始動トレースについては、指定されたトレース・ファイルを参照します。

CWOBJ2507I: Trace specification is set to {0}.
説明: トレース仕様がコマンド行で設定されました。
ユーザー応答: なし。

CWOBJ2508I: A security properties file "{0}" has been specified and will be used to start the server.
説明: セキュリティー・プロパティー・ファイルが提供され、セキュア・サーバーが始動されます。
ユーザー応答: なし。

CWOBJ2509E: Timed out after waiting {0} seconds for the server to start.
説明: ObjectGrid サーバーがタイムアウトのインターバル内には開始しませんでした。
ユーザー応答: ログで例外を確認します。

CWOBJ2510I: Stopping ObjectGrid server {0}.
説明: ObjectGrid サーバーを停止中です。
ユーザー応答:

CWOBJ2511I: Waiting for the server to stop.
説明: ObjectGrid サーバーの停止を待機中です。
ユーザー応答: なし。

CWOBJ2512I: ObjectGrid server {0} stopped.
説明: ObjectGrid サーバーは停止済みです。
ユーザー応答: なし。

CWOBJ2513E: Timed out after waiting {0} seconds for the server to stop.
説明: ObjectGrid サーバーがタイムアウトのインターバル内には停止しませんでした。
ユーザー応答: ログで例外を確認します。

CWOBJ2514I: Waiting for ObjectGrid server activation to complete.
説明: ObjectGrid サーバーが起動されました。サーバーの活動化の完了を待機中です。
ユーザー応答: なし。

CWOBJ2515E: The arguments provided are invalid. Here are the valid arguments.{0}{1}
説明: このスクリプトに提供された引数は無効です。
ユーザー応答: 有効な引数を入力します。

CWOBJ2516I: ObjectGrid server has completed activation.
説明: ObjectGrid サーバーは活動中であり、要求の処理が可能です。
ユーザー応答: なし。

CWOBJ2517I: Successfully bootstrapped to peer Objectgrid server on host {0} and port {1}.
説明: この ObjectGrid サーバーはピア・サーバーに正常にブートストラップし、このサーバーの始動に必要な情報を取得しました。
ユーザー応答: なし。

CWOBJ2407W: The {0} property on the {1} plug-in class could not be set. The exception is {2}.
説明: このプラグインのプロパティーは設定できませんでした。
ユーザー応答: 詳細情報については例外を参照します。

特記事項

本書に記載の製品、プログラム、またはサービスが日本においては提供されていない場合があります。日本で利用可能な製品、プログラム、またはサービスについては、日本アイ・ビー・エムの営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。IBM 製品、プログラムまたはサービスに代えて、 IBM の知的所有権を侵害することのない機能的に同等のプログラムまたは製品を使用することができます。ただし、IBM によって明示的に指定されたものを除き、他社の製品と組み合わせた場合の動作の評価と検証はお客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
Mail Station P300
522 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

商標

以下は、IBM Corporation の商標です。

- AIX
- CICS
- Cloudscape
- DB2
- Domino
- IBM
- Lotus
- RACF
- Redbooks
- Tivoli
- WebSphere
- z/OS

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

LINUX は、Linus Torvalds の米国およびその他の国における商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。