WebSphere® eXtreme Scale バージョン 7.1
管理ガイド

WebSphere eXtreme Scale 管理ガイド

IBM

本書は、WebSphere eXtreme Scale のバージョン 7、リリース 1、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

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® eXtreme Scale Version7.1
Administration Guide
WebSphere eXtreme Scale
Administration Guide

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2010.7

目次

図

表

管理ガイド 情報

WebSphere® eXtreme Scale の資料セットには、WebSphere eXtreme Scale 製品の使用、プログラミング、および管理に必要な情報を提供する 3 つのボリュームがあります。

WebSphere eXtreme Scale ライブラリー

WebSphere eXtreme Scale ライブラリーには、以下の資料が含まれます。

- *管理ガイド* には、アプリケーション・デプロイメント計画の作成方法、容量計画の作成方法、製品のインストールと構成方法、サーバーの始動と停止方法、環境のモニター方法、環境の保護方法など、システム管理者に必要な情報が含まれます。
- *プログラミング・ガイド* には、掲載されている API 情報を使用して WebSphere eXtreme Scale 用のアプリケーションを開発する方法に関する、アプリケーション開発者のための情報が含まれます。
- *製品概要* には、ユース・ケース・シナリオ、およびチュートリアルなど、WebSphere eXtreme Scale 概念の高水準の観点が含まれます。

これらの資料をダウンロードするには、WebSphere eXtreme Scale ライブラリー・ページにアクセスしてください。

このライブラリーと同じ情報は、WebSphere eXtreme Scale インフォメーション・センターからも入手することができます。

本書の対象者

本書は、主にシステム管理者、セキュリティー管理者、およびシステム・オペレーターの方々を対象としています。

本書の構成

本書には、以下の主要なトピックに関する情報が入っています。

- **第 1 章**には、入門に関する情報が含まれています。
- **第 2 章**には、アプリケーション・デプロイメントの計画に関する情報が含まれています。
- **第 3 章**には、キャパシティー・プランニングに関する情報が含まれています。
- **第 4 章**には、製品のインストールに関する情報が含まれています。
- **第 5 章**には、z/OS プラットフォームのカスタマイズに関する情報が含まれています。
- **第 6 章**には、製品の構成に関する情報が含まれています。
- **第 7 章**には、環境の管理に関する情報が含まれています。
- **第 8 章**には、デプロイメント環境のモニターに関する情報が含まれています。
- **第 9 章**には、デプロイメント環境のセキュアに関する情報が含まれています。
- **第 10 章**には、環境のトラブルシューティングに関する情報が含まれています。

- **第 11 章**には、製品の用語集が含まれています。

本書の更新の取得

本書の更新は、WebSphere eXtreme Scale ライブラリー・ページから最新のバージョンをダウンロードすることで取得できます。

第 1 章 WebSphere eXtreme Scale 入門

WebSphere eXtreme Scale をスタンドアロン環境にインストールすると、以下の手順でメモリー内のデータ・グリッドとしてのその機能を簡単に導入できます。

スタンドアロンでインストールした WebSphere eXtreme Scale に組み込まれているサンプルを使用すると、インストールした製品を検証し、単純な eXtreme Scale グリッドおよびクライアントの使用方法を実際に確かめることができます。この入門サンプルは *installRoot*/ObjectGrid/gettingstarted ディレクトリーにあります。

この入門サンプルは、eXtreme Scale の機能および基本的な操作を紹介するものです。このサンプルは、シェル・スクリプトおよびバッチ・スクリプトからなります。これらは、ほんの少しカスタマイズするだけで単純なグリッドを開始できるよう設計されています。 さらに、この基本的なグリッドに対して単純な作成、読み取り、更新、および削除 (CRUD) 機能を実行するためのクライアント・プログラムが、ソースを含めて提供されています。

スクリプトとその機能

このサンプルには、以下の 4 つのスクリプトがあります。

env.sh|bat: このスクリプトは、他のスクリプトから呼び出されて、必要な環境変数を設定します。通常は、このスクリプトを変更する必要はありません。

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

runcat.sh|bat: このスクリプトは、ローカル・システム上で eXtreme Scale カタログ・サービス・プロセスを開始します。

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

runcontainer.sh|bat: このスクリプトは、コンテナー・サーバー・プロセスを開始します。指定した固有のサーバー名を使用してこのスクリプトを複数回実行すれば、いくつでもコンテナーを開始できます。それらのインスタンスが一緒に、グリッド内の区画化された冗長な情報をホストすることができます。

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

runclient.sh|bat: このスクリプトは、単純な CRUD クライアントを実行し、指定された操作を開始します。

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

command には、以下のいずれかのオプションを使用します。

- *value2* を、キー *value1* を持つグリッドに挿入するには、`i` と指定します。
- *value1* のキーのオブジェクトを *value2* に更新するには、`u` と指定します。
- *value1* のキーのオブジェクトを削除するには、`d` と指定します。
- *value1* のキーのオブジェクトを検索して表示するには、`g` を指定します。

注: *installRoot*`/ObjectGrid/ gettingstarted/src/Client.java` ファイルは、カタログ・サーバーへの接続、ObjectGrid インスタンスの取得、および ObjectMap API の使用をどのように行うのかを示すクライアント・プログラムです。

基本的な手順

次の手順で最初のグリッドを開始し、グリッドと対話するクライアントを実行します。

1. 端末セッションまたはコマンド行ウィンドウを開きます。
2. 次のコマンドを使用して、`gettingstarted` ディレクトリーに移動します。

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

   installRoot の部分は、eXtreme Scale インストール・ルート・ディレクトリーへのパス、または、抽出した eXtreme Scale trial *installRoot* のルート・ファイル・パスに置き換えてください。
3. 次のスクリプトを実行して、ローカル・ホストでカタログ・サービス・プロセスを開始します。

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

   カタログ・サービス・プロセスは、現行の端末ウィンドウで実行されます。
4. 別の端末セッションまたはコマンド行ウィンドウを開き、次のコマンドを実行して、コンテナー・サーバー・インスタンスを開始します。

   - UNIX Linux
     ```
     ./runcontainer.sh server0
     ```
   - Windows
     ```
     runcontainer.bat server0
     ```

   コンテナー・サーバーは、現行の端末ウィンドウで実行されます。複製をサポートするためにさらに多くのコンテナー・サーバー・インスタンスを開始したい場合は、ステップ 5 と 6 を繰り返すことができます。
5. クライアント・コマンドを実行するため、別の端末セッションまたはコマンド行ウィンドウを開きます。
   - グリッドへのデータの追加:
     - UNIX Linux
       ```
       ./runclient.sh i key1 helloWorld
       ```
     - Windows
       ```
       runclient.bat i key1 helloWorld
       ```
   - 値を検索して表示:
     - UNIX Linux
       ```
       ./runclient.sh g key1
       ```

- Windows runclient.bat g key1

- 値の更新:
  - UNIX Linux ./runclient.sh u key1 goodbyeWorld
  - Windows runclient.bat u key1 goodbyeWorld
- 値の削除:
  - UNIX Linux ./runclient.sh d key1
  - Windows runclient.bat d key1

6. <ctrl+c> を使用して、カタログ・サービス・プロセスおよびコンテナー・サーバーをそれぞれのウィンドウで停止します。

ObjectGrid の定義

サンプルでは、コンテナー・サーバーを開始するため、 *installRoot*/ObjectGrid/gettingstarted/xml ディレクトリーにある objectgrid.xml ファイルと deployment.xml ファイルが使用されます。objectgrid.xml ファイルは ObjectGrid 記述子 XML ファイルであり、deployment.xml ファイルは ObjectGrid デプロイメント・ポリシー記述子 XML ファイルです。両方のファイルが一緒になって、分散 ObjectGrid トポロジーが定義されます。

ObjectGrid 記述子 XML ファイル

ObjectGrid 記述子 XML ファイルは、アプリケーションによって使用される ObjectGrid の構造を定義するのに使用されます。このファイルには、BackingMap 構成のリストが含まれます。 これらの BackingMap はキャッシュ・データ用の実際のデータ・ストレージです。以下の例は、objectgrid.xml ファイルのサンプルです。ファイルの最初の数行には、各 ObjectGrid XML ファイルの必須ヘッダーが含まれています。このサンプル・ファイルは、Map1 と Map2 という BackingMap がある、Grid ObjectGrid を定義しています。

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

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

</objectGridConfig>
```

デプロイメント・ポリシー記述子 XML ファイル

デプロイメント・ポリシー記述子 XML ファイルは、始動時に ObjectGrid コンテナー・サーバーに渡されます。デプロイメント・ポリシーは ObjectGrid XML ファイルと一緒に使用する必要があり、一緒に使用される ObjectGrid XML と互換でなければなりません。デプロイメント・ポリシー内の各 objectgridDeployment 要素ごとに、対応する 1 つの ObjectGrid 要素が ObjectGrid XML 内に必要です。objectgridDeployment 要素内に定義された backingMap 要素は、ObjectGrid XML 内

にある backingMap と整合していなければなりません。すべての backingMap は、1 つの mapSet 内のみで参照する必要があります。

デプロイメント・ポリシー記述子 XML ファイルは、対応する ObjectGrid XML である objectgrid.xml ファイルと対で使用されることを想定しています。以下の例では、deployment.xml ファイルの最初の数行には、各デプロイメント・ポリシー XML ファイルの必須ヘッダーが含まれています。このファイルは、objectgrid.xml ファイル内に定義された Grid ObjectGrid の objectgridDeployment 要素を定義しています。 Grid ObjectGrid 内に定義された Map1 と Map2 の両 BackingMap は、mapSet mapSet に含まれていて、そこでは numberOfPartitions、 minSyncReplicas、および maxSyncReplicas 属性が構成されています。

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

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

</deploymentPolicy>
```

mapSet エレメントの numberOfPartitions 属性は、mapSet の区画の数を指定します。これはオプションの属性であり、デフォルトは 1 です。この数は、グリッドに予想される容量に適した値である必要があります。

mapSet の minSyncReplicas 属性は、mapSet 内の各区画の同期複製の最小数を指定します。これはオプションの属性であり、デフォルトは 0 です。この同期複製の最小数をドメインがサポートできるまでは、プライマリーおよび複製は配置されません。minSyncReplicas 値をサポートするには、minSyncReplicas の値よりも 1 つだけ多いコンテナーが必要です。同期複製の数が minSyncReplicas の値よりも小さくなると、その区画に対しては書き込みトランザクションを行えなくなります。

mapSet の maxSyncReplicas 属性は、mapSet 内の各区画の同期複製の最大数を指定します。これはオプションの属性であり、デフォルトは 0 です。ある特定の区画でこの同期複製数にドメインが達すると、それ以降は、他の同期複製がその区画に対して配置されることはありません。まだ maxSyncReplicas 値を満たしていない場合には、この ObjectGrid をサポートできるコンテナーを追加すると、同期複製の数を増やすことができます。上のサンプルでは maxSyncReplicas は 1 に設定されていますが、これは、ドメインが最大 1 つの同期複製を置くことを意味しています。複数のコンテナー・サーバー・インスタンスを開始する場合、それらのコンテナー・サーバー・インスタンスの 1 つに、同期複製が 1 つだけ置かれます。

ObjectGrid の使用

installRoot/ObjectGrid/gettingstarted/src/ ディレクトリーにある Client.java ファイルは、カタログ・サーバーへの接続、ObjectGrid インスタンスの取得、および ObjectMap API の使用をどのように行うのかを示すクライアント・プログラムです。

クライアント・アプリケーションの観点から、WebSphere eXtreme Scale の使用は以下のステップに分解できます。

1. ClientClusterContext インスタンスを取得することによって、カタログ・サービスに接続する。
2. クライアント ObjectGrid インスタンスを取得する。
3. Session インスタンスを取得する。
4. ObjectMap インスタンスを取得する。
5. ObjectMap メソッドを使用する。

1. **ClientClusterContext インスタンスを取得することによって、カタログ・サービスに接続する**

   カタログ・サーバーに接続するには、ObjectGridManager API の connect メソッドを使用します。使用されている connect メソッドが必要とするのは、`hostname:port` という形式のカタログ・サーバー・エンドポイントのみです (例えば `localhost:2809`)。カタログ・サーバーへの接続が成功すれば、connect メソッドは ClientClusterContext インスタンスを戻します。この ClientClusterContext インスタンスは、ObjectGridManager API から ObjectGrid を取得するのに必要です。以下のコード・スニペットは、カタログ・サーバーへの接続方法と ClientClusterContext インスタンスの取得方法を示します。

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

2. **ObjectGrid インスタンスを取得する**

   ObjectGrid インスタンスを取得するには、ObjectGridManager API の getObjectGrid メソッドを使用します。 getObjectGrid メソッドは、ClientClusterContext インスタンスと、 ObjectGrid インスタンスの名前との両方を必要とします。ClientClusterContext インスタンスは、カタログ・サーバーへの接続中に取得されます。ObjectGrid 名の名前は、`objectgrid.xml` ファイルに指定されている `Grid` です。以下のコード・スニペットは、ObjectGridManager API の getObjectGrid メソッドを呼び出すことによって ObjectGrid を取得する方法を示します。

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

3. **Session インスタンスを取得する**

   取得した ObjectGrid インスタンスから、Session を取得することができます。Session インスタンスは、ObjectMap インスタンスの取得とトランザクション区分の実行のために必要です。以下のコード・スニペットは、ObjectGrid API の getSession メソッドを呼び出すことによって Session インスタンスを取得する方法を示します。

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

4. **ObjectMap インスタンスを取得する**

   Session を取得した後、Session API の getMap メソッドを呼び出すことによって、Session インスタンスから ObjectMap インスタンスを取得することができます。ObjectMap インスタンスを取得するため、マップ名を getMap メソッドにパラメーターとして渡す必要があります。以下のコード・スニペットは、Session API の getMap メソッドを呼び出すことによって ObjectMap を取得する方法を示します。

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

5. **ObjectMap メソッドを使用する**

   ObjectMap を取得した後、ObjectMap API を使用できます。ObjectMap インターフェースはトランザクション・マップであり、Session API の begin メソッドおよび commit メソッドを使用することによるトランザクション区分を必要とすることに注意してください。アプリケーション内で明示的なトランザクション区分がない場合、ObjectMap 操作は自動コミット・トランザクションで実行します。

   以下のコード・スニペットは、自動コミット・トランザクションでの ObjectMap API の使用方法を示しています。

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

   以下のコード・スニペットは、明示的なトランザクション区分がある場合の ObjectMap API の使用方法を示しています。

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

追加情報

このサンプルは、カタログ・サーバーおよびコンテナー・サーバーを開始する方法と、スタンドアロン環境での ObjectMap API の使用を示しています。EntityManager API を使用することもできます。

WebSphere eXtreme Scale がインストールされているか使用可能にされている WebSphere Application Server 環境では、最も一般的なシナリオは、ネットワーク接続されたトポロジーです。 ネットワーク接続トポロジーでは、カタログ・サーバーは WebSphere Application Server デプロイメント・マネージャー・プロセス内でホストされ、各 WebSphere Application Server インスタンスが 1 つの eXtreme Scale サーバーを自動的にホストします。Java™ Platform, Enterprise Edition アプリケーションは、ObjectGrid 記述子 XML ファイルと ObjectGrid デプロイメント・ポリシー記述子 XML ファイルの両方を、任意のモジュールの `META-INF` ディレクトリーに含めることのみを必要とし、ObjectGrid は自動的に使用可能になります。 その後、アプリケーションはローカルで使用可能なカタログ・サーバーに接続し、使用する ObjectGrid インスタンスを取得できます。

ディレクトリー規則

このトピックでは、*wxs_install_root* や *wxs_home* などの特殊ディレクトリーを参照する必要がある、多くの例およびコマンド行構文について説明します。これらのディレクトリーは以下のように定義されます。

wxs_install_root

wxs_install_root ディレクトリーは、WebSphere eXtreme Scale 製品ファイルがインストールされているルート・ディレクトリーです。これは、試用版の zip ファイルが解凍されたディレクトリー、または eXtreme Scale 製品がインストールされているディレクトリーになる可能性があります。

- 試用版を解凍した場合の例:

  ```
  /opt/IBM/WebSphere/eXtremeScale
  ```

- eXtreme Scale がスタンドアロン・ディレクトリーにインストールされている場合の例:

  ```
  /opt/IBM/eXtremeScale
  ```

- eXtreme Scale が WebSphere Application Server に統合されている場合の例:

  ```
  /opt/IBM/WebSphere/AppServer
  ```

wxs_home

wxs_home ディレクトリーは、WebSphere eXtreme Scale 製品のライブラリー、サンプル、およびコンポーネントのルート・ディレクトリーです。これは、試用版を解凍した場合は、*wxs_install_root* ディレクトリーと同じです。スタンドアロン・インストール済み環境の場合は、これは *wxs_install_root* 内の ObjectGrid サブディレクトリーになります。 WebSphere Application Server に統合されているインストール済み環境の場合は、これは、*wxs_install_root* 内の `optionalLibraries/ObjectGrid` ディレクトリーになります。

- 試用版を解凍した場合の例:

  ```
  /opt/IBM/WebSphere/eXtremeScale
  ```

- eXtreme Scale がスタンドアロン・ディレクトリーにインストールされている場合の例:

  ```
  /opt/IBM/eXtremeScale/ObjectGrid
  ```

- eXtreme Scale が WebSphere Application Server に統合されている場合の例:

  ```
  /opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid
  ```

was_root

was_root ディレクトリーは、WebSphere Application Server インストール済み環境のルート・ディレクトリーです。

```
/opt/IBM/WebSphere/AppServer
```

restservice_home

restservice_home ディレクトリーは、eXtreme Scale REST データ・サービスのライブラリーおよびサンプルが配置されるディレクトリーです。このディレクトリーは *restservice* という名前で、*wxs_home* ディレクトリー内のサブディレクトリーです。

- スタンドアロン・デプロイメントの場合の例:

  ```
  /opt/IBM/WebSphere/eXtremeScale/ObjectGrid/restservice
  ```

- WebSphere Application Server 統合デプロイメントの場合の例:

  ```
  /opt/IBM/WebSphere/AppServer/optionalLibraries/ObjectGrid/restservice
  ```

tomcat_root

tomcat_root は、Apache Tomcat インストール済み環境のルート・ディレクトリーです。

```
/opt/tomcat5.5
```

wasce_root

wasce_root は、WebSphere Application Server Community Edition インストール済み環境のルート・ディレクトリーです。

```
/opt/IBM/WebSphere/AppServerCE
```

java_home

java_home は、Java Runtime Environment (JRE) インストール済み環境のルート・ディレクトリーです。

```
/opt/IBM/WebSphere/eXtremeScale/java
```

第 2 章 キャパシティー・プランニング

初期データ・セット・サイズおよび予測されるデータ・セット・サイズがわかっている場合、WebSphere eXtreme Scale を実行するために必要なキャパシティーを計画できます。こうした計画は、将来の変更に向けて eXtreme Scale を効率よくデプロイするのに役立ちますが、eXtreme Scale の弾力性を最大限に生かすことができます。これにより、メモリー内のデータベースや他のタイプのデータベースなど、異なるシナリオを考える必要がなくなります。

スケーラビリティーの概念の概要

スケーラビリティーによって、WebSphere eXtreme Scale のデプロイメント内のデータを、ユーザーの構成選択に基づいて一連のサーバー (コンテナー) に配布することができます。

メモリー・サイズ設定および区画数の計算

特定の構成に必要なメモリーの容量および区画数を計算できます。

WebSphere eXtreme Scale は、データを Java 仮想マシン (JVM) のアドレス・スペースに保管します。各 JVM は、JVM に保管されたデータの作成、取得、更新、および削除の呼び出しをサービスするためのプロセッサー・スペースを提供します。さらに、各 JVM は、データ・エントリーおよびレプリカ用のメモリー・スペースを提供します。Java オブジェクトのサイズはさまざまなので、必要なメモリー量を見積もるための測定が必要です。

必要なメモリーのサイズを設定するには、アプリケーション・データを 1 つの JVM にロードします。ヒープ使用量が 60% に達している場合、使用されるオブジェクトの数に注意してください。この数値は、Java 仮想マシン のそれぞれの推奨されるオブジェクトの最大数です。最も的確なサイズ設定を行うには、現実に近いデータを使用します。索引もまたメモリーを消費するので、定義されているすべての索引をサイズ設定に含めてください。メモリー使用量のサイズ見積もりを行う最良の方法は、ガーベッジ・コレクション verbosegc 出力を実行することです。この出力によって、ガーベッジ・コレクション後の数値が分かるからです。ヒープ使用量については MBean またはプログラムでいつでも照会できますが、この方法では消費メモリーは正確には示されません。そういった照会ではヒープの現行スナップショットしか取得できず、未収集のガーベッジが含まれている可能性があるからです。

構成の拡張

区画当たりの断片数 (numShardsPerPartition 値)

区画当たりの断片数である numShardsPerPartition 値を計算するには、プライマリー断片の 1 に、必要な複製断片の総数を加算します。

```
numShardsPerPartition = 1 + total_number_of_replicas
```

Java 仮想マシン 数 (minNumJVMs 値)

構成を拡張するには、まず保管する必要のある合計オブジェクトの最大数を決定します。必要な Java 仮想マシンの数を決めるには、次の式を使用します。

```
minNumJVMS=(numShardsPerPartition * numObjs) / numObjsPerJVM
```

この値を切り上げて、最も近い整数値にしてください。

断片数 (numShards 値)

最終的な増加サイズでは、それぞれの JVM に 10 個の断片が使用されます。前述したように、各 JVM には、1 つのプライマリー断片と (N-1) 個の複製断片があります。この場合、9 個のレプリカがあります。データ保管用に既に多数の Java 仮想マシンがあるため、Java 仮想マシンの数に 10 を掛けて、断片の数を決定することができます。

```
numShards = minNumJVMs * 10 shards/JVM
```

区画数

区画に 1 つのプライマリー断片と 1 つの複製断片がある場合、区画には 2 つの断片 (プライマリーとレプリカ) があります。区画数は、断片数を 2 で割って、一番近い素数に丸めたものです。区画に 1 つのプライマリーと 2 つのレプリカがある場合、区画数は、断片数を 3 で割って、一番近い素数に丸めたものになります。

```
numPartitions = numShards / numShardsPerPartition
```

拡張の例

この例では、エントリー数は当初 250,000,000 であるとします。毎年、エントリー数は 14% 増加します。7 年後には、エントリーの合計数が 500,000,000 となるため、それに応じた容量計画が必要になります。高可用性を実現するため、1 つのレプリカが必要になります。レプリカを使用すると、エントリー数は 2 倍、すなわち 1,000,000,000 となります。テストとして、2,000,000 のエントリーを各 JVM に保管できます。このシナリオの計算を使用すると、以下の構成が必要になります。

- 最終的な数のエントリーを保管するために 500 の Java 仮想マシン。
- 5000 の断片 (Java 仮想マシン数の 500 に 10 を掛けたもの)。
- 2500 の区画、すなわち次位の素数である 2503 (5000 の断片を 2 (プライマリー断片と複製断片) で割ったもの)。

構成の開始

前述の計算に基づいて、250 の Java 仮想マシンから始めて、5 年間で 500 の Java 仮想マシンに増やします。そうすると、最終的なエントリー数に達するまで段階的な増加を管理できます。

この構成では、区画ごとに約 200,000 (500,000,000 のエントリーを区画数の 2503 で割ったもの) のエントリーが保管されます。numberOfBuckets パラメーターを次位の素数 (この例では 70887) までのエントリーを収めるマップで設定します。これにより約 3 の比率が保たれます。

Java 仮想マシンの最大数に達した場合

500 という Java 仮想マシンの最大数に達しても、グリッドを拡張できます。Java 仮想マシン の数が 500 を超えるまでになると、各 JVM について断片数が 10 (推奨数) を下回り始めます。つまり断片が大きくなり始めます。これは問題となる場合があります。将来の成長を考慮しながら、サイズ設定処理を繰り返し、区画数を設定し直す必要があります。この作業では、グリッド全体の再始動が必要になります。さもないとグリッド不足となります。

サーバー数

重要: どのような場合にも、サーバーについてはページングは使用しないでください。

1 つの JVM は、ヒープ・サイズを超えるメモリーを使用します。例えば、JVM の 1 GB のヒープでは、実際には 1.4 GB の実メモリーが使用されます。サーバー上の使用可能な空き RAM を判別してください。RAM の容量を JVM あたりのメモリーで割って、サーバー上の Java 仮想マシンの最大数を計算してください。

トランザクションの区画ごとの CPU 見積もり

eXtreme Scale の主要な機能は、その弾力的なスケーリングですが、拡大のための CPU のサイズ変更の考慮および理想的な CPU 数の調整も重要です。

プロセッサーのコストには、以下が含まれます。

- クライアントからの作成、取得、更新、および削除操作にサービスを提供するコスト。
- 他の Java 仮想マシンの複製のコスト。
- 無効化のコスト。
- 除去ポリシーのコスト。
- ガーベッジ・コレクションのコスト。
- アプリケーション・ロジックのコスト。

サーバーごとの Java 仮想マシン

サーバーは 2 台使用し、サーバーごとに最大の JVM 数を開始します。前のセクションで計算した区画数を使用します。その後、それら 2 台のコンピューターに収まるだけの十分なデータと Java 仮想マシンをプリロードします。クライアントには別のサーバーを使用してください。この 2 台のサーバーからなるグリッドに対し、現実的なトランザクション・シミュレーションを実行します。

ベースラインを計算するには、プロセッサー使用が飽和状態になるようにしてください。プロセッサーを飽和状態にできない場合は、ネットワークが飽和している可能性があります。ネットワークが飽和している場合は、ネットワーク・カードをさらに追加し、複数のネットワーク・カードで Java 仮想マシンをラウンドロビンさせてください。

プロセッサー使用量 60% でコンピューターを実行し、作成、取得、更新、および削除トランザクションの速度を測定します。この測定により、2 台のサーバーでのスループットがわかります。この数値は、サーバー 4 台で倍になり、サーバー 8 台でさらにその倍になり、以降も同様の割合で大きくなります。この拡張の前提

は、ネットワーク容量とクライアントのキャパシティーも同様に拡大可能であることです。

結果的に、eXtreme Scale 応答時間は、サーバーの数が増しても安定しています。トランザクション・スループットは、グリッドにコンピューターが追加されるにつれて直線的に増加します。

並列トランザクションの場合の CPU のサイズ設定

グリッドが拡張するにつれ、単一区画トランザクションのスループットが直線的に増加します。並列トランザクションは、サーバーの集合 (これはすべてのサーバーの可能性があります) にタッチするので、単一区画トランザクションとは異なります。

トランザクションがすべてのサーバーにタッチする場合、スループットは、トランザクションを開始したクライアントまたはタッチされた最低速のサーバーのスループットに制限されます。グリッドが大きくなれば、データはより広く分散され、提供されるプロセッサー・スペース、メモリー、ネットワークなどが拡張されます。ただし、クライアントは最低速のサーバーの応答を待たなければならなくなり、クライアントはトランザクションの結果を消費しなければならなくなります。

トランザクションがサーバーのサブセットにタッチする場合、N 個のサーバーのうちの M 個が要求を受け取ります。この結果、スループットは、最低速のサーバーのスループットより、N/M 倍した分だけ速くなります。例えば、20 のサーバーがある場合に、トランザクションが 5 つのサーバーにタッチすると、スループットは、グリッド内の最低速のサーバーのスループットを 4 倍したものになります。

並列トランザクションが完了すると、結果がそのトランザクションを開始したクライアント・スレッドに送信されます。ここでこのクライアントは、単一スレッド化された結果を集約する必要があります。トランザクションでタッチされるサーバーの数が増えると、この集約時間が増加します。ただし、グリッドが拡大すると、各サーバーが戻す結果が小さくなる可能性もあるので、この時間はアプリケーションに依存します。

一般的には、グリッド全体で区画が均等に配分されるので、並列トランザクションはグリッド内のすべてのサーバーにタッチします。この場合、スループットは、最初の事例に制限されます。

要約

このサイズ設定により、以下の 3 つのメトリックが得られます。

- 区画の数。
- 必須メモリーに必要なサーバーの数。
- 必須スループットに必要なサーバーの数。

メモリー所要量に対して 10 個のサーバーが必要であるが、プロセッサーが飽和状態であるため必要とするスループットの 50% しか得られない場合、2 倍の数のサーバーが必要になります。

最高の安定度を実現するために、60% のプロセッサー負荷でサーバー、さらに 60% のヒープ負荷で JVM ヒープを稼働してください。スパイクでプロセッサー使用量を 80% から 90% の間に引き上げることができますが、通常はこのレベルより高いレベルでサーバーを稼働しないようにしてください。

キャパシティー・プランニングと高可用性 (動的キャッシング)

WebSphere Application Server にデプロイされた Java EE アプリケーションでは、動的キャッシュ API を使用できます。動的キャッシュは、ビジネス・データや生成された HTML をキャッシュに入れるために、または、データ複製サービス (DRS) を使用してセル内のキャッシュ・データを同期化するために利用できます。

概説

WebSphere eXtreme Scale 動的キャッシュ・プロバイダーで作成されたすべての動的キャッシュ・インスタンスは、デフォルトで、高い可用性を持ちます。高可用性のレベルおよびメモリー・コストは、使用されるトポロジーによって変わります。

組み込みトポロジーを使用している場合、キャッシュ・サイズは、1 つのサーバー・プロセス内の空きメモリーの量に制限され、各サーバー・プロセスはキャッシュの完全コピーを保管します。1 つのサーバー・プロセスが実行し続けている限り、キャッシュは存続します。キャッシュ・データが失われるのは、キャッシュにアクセスするすべてのサーバーがシャットダウンされた場合のみです。

組み込み区画化トポロジーを使用するキャッシングの場合、キャッシュ・サイズの上限は、すべてのサーバー・プロセス内にある空きスペースの総計です。デフォルトでは、eXtreme Scale 動的キャッシュ・プロバイダーは各プライマリー断片ごとに 1 つの複製を使用します。したがって、各キャッシュ・データは 2 回ずつ保管されます。

組み込み区画化キャッシュの容量を判定するには、以下の式 A を使用してください。

式 A

F * C / (1 + R) = M

各部の意味は、次のとおりです。

- F = コンテナー・プロセス当たりの空きメモリー
- C = コンテナーの数
- R = 複製の数
- M = キャッシュの合計サイズ

各プロセスに 256 MB の使用可能なスペースがあり、サーバー・プロセスが合計 4 つある WebSphere Network Deployment グリッドの場合、それらの全サーバーにまたがるキャッシュ・インスタンスは 512 メガバイトまでのデータを保管できます。このモードでは、キャッシュは、1 つのサーバーが破損しても、データを失うことなく存続できます。また、最大 2 つのサーバーが順次シャットダウンしても、データを失うことはありません。この例では、上の式は次のようになります。

256mb * 4 コンテナー/ (1 プライマリー + 1 複製) = 512mb

リモート・トポロジーを使用するキャッシュのサイジング特性は、組み込み区画化トポロジーを使用するキャッシュと似ていますが、すべての eXtreme Scale コンテナー・プロセス内の使用可能なスペースの総量に制限されます。

リモート・トポロジーでは、複製の数を増やすことによって、メモリーのオーバーヘッドが余分にかかる代わりにアベイラビリティーのレベルを上げることが可能です。これは大部分の動的キャッシュ・アプリケーションでは不必要ですが、dynacache-remote-deployment.xml ファイルを編集して複製の数を増やすことができます。

以下の式 B と C を使用して、キャッシュの高可用性のために複製を追加することの影響を判定できます。

式 B

N = Minimum(T -1, R)

各部の意味は、次のとおりです。

- N = 同時に破損してもかまわないプロセスの数
- T = コンテナーの総数
- R = 複製の総数

式 C

Ceiling(T/ (1+N)) = m

各部の意味は、次のとおりです。

- T = コンテナーの総数
- N = 複製の総数
- m = キャッシュ・データをサポートするのに必要な最小コンテナー数

動的キャッシュ・プロバイダーでのパフォーマンス・チューニングについては、動的キャッシュ・プロバイダーのチューニングを参照してください。

キャッシュのサイズ見積もり

WebSphere eXtreme Scale 動的キャッシュ・プロバイダーを使用するアプリケーションをデプロイする前に、前のセクションに記述されている一般的な規則に、実動システムの環境データを組み合わせて検討する必要があります。まず最初に確定する必要がある数値は、コンテナー・プロセスの総数と、キャッシュ・データを保持するための各プロセス内の使用可能メモリーの量です。組み込みトポロジーを使用している場合、キャッシュ・コンテナーは WebSphere Application Server プロセスの内部の同一場所に配置され、キャッシュを共有する各サーバーごとにコンテナーが 1 つずつある状態になります。 キャッシングを使用可能にしていないアプリケーションと WebSphere Application Server のメモリー・オーバーヘッドを判定することが、プロセス内で使用可能なスペース量を計算する最良の方法です。これは、詳細ガーベッジ・コレクション・データを分析することで行えます。リモート・トポロジーを使用している場合、この情報は、新しく開始され、キャッシュ・データがま

だ設定されたことのないスタンドアロン・コンテナーの、詳細ガーベッジ・コレクション出力を見れば分かります。 キャッシュ・データ用に使用可能な、プロセス当たりのスペース量を計算する際には、ガーベッジ・コレクション用にいくらかのヒープ・スペースを確保しておくことにも注意が必要です。コンテナー (WebSphere Application Server またはスタンドアロン) のオーバーヘッドに、キャッシュ用に予約されたサイズを加えた結果は、合計ヒープの 70 % 以下になっているべきです。

この情報を収集できたら、前述の式 A に値を挿入して、区画化されたキャッシュの最大サイズを判定してください。最大サイズが判明したら、次のステップは、サポートできるキャッシュ・エントリー総数を判定することです。これには、キャッシュ・エントリー当たりの平均サイズを決定することが必要です。これを行う簡単な方法は、カスタマー・オブジェクトのサイズに 10% 追加することです。動的キャッシュを使用している場合のキャッシュ・エントリーのサイズ見積もりについて詳しくは、動的キャッシュおよびデータ複製サービスのチューニング・ガイドを参照してください。

圧縮が有効にされている場合、圧縮はカスタマー・オブジェクトのサイズには影響しますが、キャッシング・システムのオーバーヘッドには影響しません。圧縮を使用している場合のキャッシュ・オブジェクトのサイズを判定するには、以下の式を使用します。

S = O * C + O * 0.10

各部の意味は、次のとおりです。

- S = キャッシュ・オブジェクトの平均サイズ
- O = 圧縮されていないカスタマー・オブジェクトの平均サイズ
- C = 分数で表した圧縮率

つまり、2 を 1 にする場合の圧縮率は 1/2 = 0.50 です。これは小さいほど良い値です。保管されるオブジェクトが通常の POJO で、大部分がプリミティブ型の場合、圧縮率を 0.60 から 0.70 に想定してください。キャッシュされるオブジェクトが、サーブレット、JSP、または WebServices オブジェクトの場合、圧縮率を判定する最適の方法は、代表的なサンプルを ZIP 圧縮ユーティリティーで圧縮することです。これが不可能な場合、このタイプのデータの一般的な圧縮率は 0.2 から 0.35 です。

次に、この情報を使用して、サポートできるキャッシュ・エントリーの総数を判定します。以下の式 D を使用してください。

式 D

T = S / A

各部の意味は、次のとおりです。

- T = キャッシュ・エントリーの総数
- S = 式 A を使用して算出され、キャッシュ・データ用に使用可能な合計サイズ
- A = 各キャッシュ・エントリーの平均サイズ

最後に、動的キャッシュ・インスタンスにキャッシュ・サイズを設定して、この制限を強制する必要があります。WebSphere eXtreme Scale 動的キャッシュ・プロバイ

ダーは、この点で、デフォルトの動的キャッシュ・プロバイダーと異なります。以下の式を使用して、動的キャッシュ・インスタンスのキャッシュ・サイズに設定する値を判定してください。以下の式 E を使用してください。

式 E

Cs = Ts / Np

各部の意味は、次のとおりです。

- Ts = キャッシュの合計目標サイズ
- Cs = 動的キャッシュ・インスタンスに設定するキャッシュ・サイズ設定値
- Np = 区画の数。デフォルトは 47 です。

キャッシュ・インスタンスを共有する各サーバーで、動的キャッシュ・インスタンスのサイズを、式 E で計算した値に設定してください。

第 3 章 WebSphere eXtreme Scale のインストールおよびデプロイ

WebSphere eXtreme Scale は、複数のサーバーにまたがるアプリケーション・データおよびビジネス・ロジックの区画化、複製、および管理を動的に行うために使用できるメモリー内のデータ・グリッドです。デプロイメントの目的および要件を決定した後に、 eXtreme Scale をシステムにインストールします。

始める前に

- WebSphere eXtreme Scale が現行トポロジーにどのように適合するかを設定します。詳しくは、*製品概要*の WebSphere eXtreme Scale アーキテクチャーとトポロジーの概要を参照してください。
- ご使用の環境が eXtreme Scale をインストールするための前提条件を満たしていることを確認してください。詳しくは、 69 ページの『ハードウェアおよびソフトウェアの要件』を参照してください。

このタスクについて

7.1+ 2 つのインストール・タイプ

- WebSphere eXtreme Scale フルインストール: このインストールを使用して、サーバーまたはクライアント、あるいはその両方をインストールできます。
- WebSphere eXtreme Scale クライアント インストール: このインストールを使用して、特定のプラットフォームにクライアントをインストールできます。

サポートされる環境

eXtreme Scale のインストールおよびデプロイ先として、オペレーティング・システムの特定レベルの要件はありません。 Java Platform, Standard Edition (J2SE) および Java Platform, Enterprise Edition (JEE) のそれぞれのインストールでは、異なるオペレーティング・システムのレベルまたはフィックスが必要です。

この製品は、JEE および J2SE 環境にインストールしてデプロイできます。また、クライアント・コンポーネントを WebSphere Application Server に統合せずに、直接 JEE アプリケーションにバンドルすることができます。 WebSphere eXtreme Scale は、Java ランタイム環境 (JRE) バージョン 1.4.2 以降および WebSphere Application Server バージョン 6.0.2 以降をサポートします。

手順

- スタンドアロン WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアント をインストールします。

  WebSphere Application Server または WebSphere Application Server Network Deployment を含まない環境に、スタンドアロンの eXtreme Scale をインストールすることができます。スタンドアロン・オプションで、eXtreme Scale サーバーをインストールする新規インストール・ロケーションを定義します。

重要: スタンドアロン環境で、WebSphere eXtreme Scale の非 root (非管理者) プロファイルを使用することもできます。スタンドアロン環境は、WebSphere Application Server を使用しない環境です。非 root プロファイルを使用するには、`ObjectGrid` ディレクトリーの所有者を非 root プロファイルに変更する必要があります。その後、その非 root プロファイルでログインして、通常の root (管理者) プロファイルの場合と同様に eXtreme Scale を操作できます。

- WebSphere Application Server または WebSphere Application Server Network Deployment と製品を統合します。

  eXtreme Scale をインストールして、既存の WebSphere Application Server または WebSphere Application Server Network Deployment のインストールと統合することができます。フルインストールでは、eXtreme Scale のクライアントとサーバーの両方をインストールするか、クライアントのみをインストールするかを選択できます。
- プロファイルを作成および拡張します。

  eXtreme Scale フィーチャーを使用するためのプロファイルを作成し、拡張します。WebSphere Application Server バージョン 6.1 またはバージョン 7.0 を実行している場合、プロファイル管理ツール・プラグインまたは manageprofiles コマンドが使用できます。WebSphere Application Server バージョン 6.0.2 を実行している場合、プロファイルの作成および拡張には、wasprofile コマンドを使用する必要があります。
- 保守を適用します。

  ご使用の環境に保守を適用するには、IBM® Update Installer バージョン 7.0.0.4 以降を使用してください。

関連概念

69 ページの『ハードウェアおよびソフトウェアの要件』
ハードウェア要件およびオペレーティング・システム要件の概要をご覧ください。WebSphere eXtreme Scale に対して使用するハードウェアまたはオペレーティング・システムのレベルについて、特定のレベルの要件はありませんが、公式にサポートされるハードウェアおよびソフトウェアのオプションに関するオペレーティング・システム別の詳しいリストを、製品サポート・サイトのシステム要件ページから入手できます。 インフォメーション・センターの情報と「システム要件」ページの情報に違いがある場合は、Web サイトの情報を優先してください。インフォメーション・センターの前提条件の情報は、便宜上提供されているだけです。

関連資料

59 ページの『インストール・パラメーター』
製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

WebSphere eXtreme Scale バージョン 7.1 へのマイグレーション

WebSphere eXtreme Scale インストーラーでは、前のインストール済み環境をアップグレードすることも変更することもできません。新しいバージョンをインストールする前に前のバージョンをアンインストールする必要があります。構成ファイルは後方互換性があるため、マイグレーションする必要はありません。ただし、製品に

付属のスクリプト・ファイルのいずれかを変更した場合には、更新したスクリプト・ファイルにその変更を再適用する必要があります。

始める前に

ご使用のシステムが、マイグレーションおよびインストールしようとしている製品バージョンの最小限の要件を満たしていることを確認してください。詳細については、69 ページの『ハードウェアおよびソフトウェアの要件』を参照してください。

このタスクについて

変更済みの製品スクリプト・ファイルを /bin ディレクトリーにある新規の製品スクリプト・ファイルとマージして、変更の保守を行います。

ヒント: 製品にインストールされているスクリプト・ファイルを変更しなかった場合は、以下のマイグレーション・ステップを実行する必要はありません。代わりに、以前のバージョンをアンインストールしてから同じディレクトリーに新規バージョンをインストールすることで、バージョン 7.1 にアップグレードできます。

手順

1. eXtreme Scale を使用しているすべてのプロセスを停止します。
   - スタンドアロン eXtreme Scale 環境で実行しているすべてのプロセスを停止するには、 スタンドアロン・サーバーの停止を参照してください。
   - WebSphere Application Server または WebSphere Application Server Network Deployment 環境で実行しているすべてのプロセスを停止するには、 コマンド行ユーティリティーを参照してください。
2. 現行インストール・ディレクトリーから変更済みのすべてのスクリプトを一時ディレクトリーに保存します。
3. 製品をアンインストールします。
4. eXtreme Scale バージョン 7.1 をインストールします。 詳しくは、17 ページの『第 3 章 WebSphere eXtreme Scale のインストールおよびデプロイ』を参照してください。
5. 一時ディレクトリーにあるファイルから、/bin ディレクトリーにある新規の製品スクリプト・ファイルに必要な変更をマージします。
6. すべての eXtreme Scale プロセスを開始して、製品の使用を開始します。 詳細については、341 ページの『管理に関する用語』を参照してください。

関連概念

69 ページの『ハードウェアおよびソフトウェアの要件』
ハードウェア要件およびオペレーティング・システム要件の概要をご覧ください。

WebSphere eXtreme Scale に対して使用するハードウェアまたはオペレーティング・システムのレベルについて、特定のレベルの要件はありませんが、公式にサポートされるハードウェアおよびソフトウェアのオプションに関するオペレーティング・システム別の詳しいリストを、製品サポート・サイトのシステム要件ページから入手できます。 インフォメーション・センターの情報と「システム要件」ページの情報に違いがある場合は、Web サイトの情報を優先してください。インフォメーション・センターの前提条件の情報は、便宜上提供されているだけです。

スタンドアロン WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントのインストール

スタンドアロン WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントを、 WebSphere Application Server も WebSphere Application Server Network Deployment も含まれていない環境にインストールできます。

始める前に

- ターゲット・インストール・ディレクトリーが空であるか、存在していないことを確認します。

  重要: バージョン 7.1 のインストール先に指定したディレクトリーに、以前のバージョンの WebSphere eXtreme Scale または ObjectGrid コンポーネントが存在していると、この製品はインストールされません。例えば、既存の `<wxs_install_root>/ObjectGrid` フォルダーが存在する場合があります。他のインストール・ディレクトリーを選択するか、あるいは、インストールを取り消すことができます。次に、以前のインストールをアンインストールしてから、再度ウィザードを実行してください。

- **7.1+** IBM Runtime Environment は、 *<wxs_install_home>*`/java` フォルダーにスタンドアロン・インストールの一部としてインストールされます。

このタスクについて

この製品をスタンドアロンとしてインストールする場合、WebSphere eXtreme Scale クライアントとサーバーを別々にインストールします。スタンドアロン・モードの WebSphere eXtreme Scale クライアントのインストールでは、データ・グリッド内のデータにアクセスするクライアントをインストールします。 したがって、サーバーとクライアントのプロセスは、必要なすべてのリソースにローカル・アクセスします。また、スクリプトと Java アーカイブ (JAR) ファイルを使用して、WebSphere eXtreme Scale を既存の Java Platform, Standard Edition (J2SE) アプリケーションに組み込むこともできます。

表 1. WebSphere eXtreme Scale フルインストールのランタイム・ファイル： WebSphere eXtreme Scale は、ObjectGrid プロセスおよび関連 API に依存しています。次の表に、このインストールに含まれる JAR ファイルをリストします。

ファイル名	環境	インストールの場所	説明
wxsdynacache.jar	クライアントおよびサーバー	dynacache/lib	wxsdynacache.jar ファイルには、動的キャッシュ・プロバイダーと一緒に使用するために必要なクラスが含まれています。提供されているスクリプトを使用すると、このファイルは自動的にサーバー・ランタイム環境に組み込まれます。
wxshyperic.jar	ユーティリティー	hyperic/lib	SpringSource Hyperic モニター・エージェントの WebSphere eXtreme Scale サーバー検出プラグイン。
objectgrid.jar	ローカル、クライアント、およびサーバー	lib	objectgrid.jar ファイルは、J2SE バージョン 1.4.2 以降のサーバー・ランタイム環境によって使用されます。提供されているスクリプトを使用すると、このファイルは自動的にサーバー・ランタイム環境に組み込まれます。
ogagent.jar	ローカル、クライアント、およびサーバー	lib	ogagent.jar ファイルには、EntityManager API と一緒に使用される Java インスツルメンテーション・エージェントの実行に必要なランタイム・クラスが含まれています。
ogclient.jar	ローカルおよびクライアント	lib	ogclient.jar ファイルには、ローカル・ランタイム環境とクライアント・ランタイム環境のみが含まれています。このファイルは、J2SE バージョン 1.4.2 以降で使用することができます。
ogspring.jar	ローカル、クライアント、およびサーバー	lib	ogspring.jar ファイルには、SpringSource Spring フレームワーク統合用のサポート・クラスが含まれています。
wsogclient.jar	ローカルおよびクライアント	lib	wsogclient.jar ファイルは、WebSphere Application Server バージョン 6.0.2 以降を含む環境を使用した場合にインストールされます。このファイルには、ローカル・ランタイム環境およびクライアント・ランタイム環境のみが含まれています。
wxssizeagent.jar	ローカル、クライアント、およびサーバー	lib	wxssizeagent.jar ファイルは、Java ランタイム環境 (JRE) バージョン 1.5 以上の使用時に、より正確なキャッシュ・エントリー・サイジング情報を提供するために使用されます。
ibmcfw.jar ibmext.jar ibmorb.jar ibmorbapi.jar	クライアントおよびサーバー	lib/endorsed	このファイル・セットには、Java SE プロセスでアプリケーションを実行するために使用されるオブジェクト・リクエスト・ブローカー (ORB) ランタイムが含まれています。
restservice.ear	クライアント	restservice/lib	restservice.ear ファイルには、WebSphere Application Server 環境用の eXtreme Scale REST データ・サービス・アプリケーション・エンタープライズ・アーカイブが含まれています。
restservice.war	クライアント	restservice/lib	restservice.war ファイルには、別のベンダーから取得されたアプリケーション・サーバー用の eXtreme Scale REST データ・サービス Web アーカイブが含まれています。
xsadmin.jar	ユーティリティー	samples	xsadmin.jar ファイルには、eXtreme Scale 管理サンプル・ユーティリティーが含まれています。
sessionobjectgrid.jar	クライアントおよびサーバー	session/lib	sessionobjectgrid.jar ファイルには、eXtreme Scale HTTP セッション管理ランタイムが含まれています。
splicerlistener.jar	ユーティリティー	session/lib	splicerlistener.jar ファイルには、eXtreme Scale バージョン 7.1 HTTP セッション・リスナー用のスプライサー・ユーティリティーが含まれています。

表 1. WebSphere eXtreme Scale フルインストールのランタイム・ファイル (続き):
WebSphere eXtreme Scale は、ObjectGrid プロセスおよび関連 API に依存しています。次の表に、このインストールに含まれる JAR ファイルをリストします。

ファイル名	環境	インストールの場所	説明
xsgbean.jar	サーバー	wasce/lib	xsgbean.jar ファイルには、eXtreme Scale サーバーを WebSphere Application Server Community Edition アプリケーション・サーバーに組み込むための GBean が含まれています。
splicer.jar	ユーティリティー	legacy/session/lib	WebSphere eXtreme Scale バージョン 7.0 HTTP セッション・マネージャー・フィルター用のスプライサー・ユーティリティー

表 2. WebSphere eXtreme Scale クライアント用のランタイム・ファイル： WebSphere eXtreme Scale クライアント は、ObjectGrid プロセスおよび関連 API に依存しています。次の表に、このインストールに含まれる JAR ファイルをリストします。

ファイル名	環境	インストールの場所	説明
wxsdynacache.jar	クライアントおよびサーバー	dynacache/lib	wxsdynacache.jar ファイルには、動的キャッシュ・プロバイダーと一緒に使用するために必要なクラスが含まれています。提供されているスクリプトを使用すると、このファイルは自動的にサーバー・ランタイム環境に組み込まれます。
wxshyperic.jar	ユーティリティー	hyperic/lib	SpringSource Hyperic モニター・エージェントの WebSphere eXtreme Scale サーバー検出プラグイン。
ogagent.jar	ローカル、クライアント、およびサーバー	lib	ogagent.jar ファイルには、EntityManager API と一緒に使用される Java インスツルメンテーション・エージェントの実行に必要なランタイム・クラスが含まれています。
ogclient.jar	ローカルおよびクライアント	lib	ogclient.jar ファイルには、ローカル・ランタイム環境とクライアント・ランタイム環境のみが含まれています。このファイルは、J2SE バージョン 1.4.2 以降で使用することができます。
ogspring.jar	ローカル、クライアント、およびサーバー	lib	ogspring.jar ファイルには、SpringSource Spring フレームワーク統合用のサポート・クラスが含まれています。
wsogclient.jar	ローカルおよびクライアント	lib	wsogclient.jar ファイルは、WebSphere Application Server バージョン 6.0.2 以降を含む環境を使用した場合にインストールされます。このファイルには、ローカル・ランタイム環境およびクライアント・ランタイム環境のみが含まれています。
wxssizeagent.jar	ローカル、クライアント、およびサーバー	lib	wxssizeagent.jar ファイルは、Java ランタイム環境 (JRE) バージョン 1.5 以上の使用時に、より正確なキャッシュ・エントリー・サイジング情報を提供するために使用されます。
ibmcfw.jar ibmext.jar ibmorb.jar ibmorbapi.jar	クライアントおよびサーバー	lib/endorsed	このファイル・セットには、Java SE プロセスでアプリケーションを実行するために使用されるオブジェクト・リクエスト・ブローカー (ORB) ランタイムが含まれています。
restservice.ear	クライアント	restservice/lib	restservice.ear ファイルには、WebSphere Application Server 環境用の eXtreme Scale REST データ・サービス・アプリケーション・エンタープライズ・アーカイブが含まれています。
restservice.war	クライアント	restservice/lib	restservice.war ファイルには、別のベンダーから取得されたアプリケーション・サーバー用の eXtreme Scale REST データ・サービス Web アーカイブが含まれています。

表 2. *WebSphere eXtreme Scale* クライアント用のランタイム・ファイル *(続き)*： WebSphere eXtreme Scale クライアント は、ObjectGrid プロセスおよび関連 API に依存しています。次の表に、このインストールに含まれる JAR ファイルをリストします。

ファイル名	環境	インストールの場所	説明
xsadmin.jar	ユーティリティー	samples	xsadmin.jar ファイルには、eXtreme Scale 管理サンプル・ユーティリティーが含まれています。
sessionobjectgrid.jar	クライアントおよびサーバー	session/lib	sessionobjectgrid.jar ファイルには、eXtreme Scale HTTP セッション管理ランタイムが含まれています。
splicerlistener.jar	ユーティリティー	session/lib	splicerlistener.jar ファイルには、eXtreme Scale バージョン 7.1 HTTP セッション・リスナー用のスプライサー・ユーティリティーが含まれています。
splicer.jar	ユーティリティー	legacy/session/lib	WebSphere eXtreme Scale バージョン 7.0 HTTP セッション・マネージャー・フィルター用のスプライサー・ユーティリティー

手順

1. ウィザードを使用して、インストールします。
   - 以下のスクリプトを実行して、WebSphere eXtreme Scale フルインストール用のウィザードを開始します。
     - Linux UNIX *dvd_root*/install
     - Windows *dvd_root*¥install.bat
   - 以下のスクリプトを実行して、WebSphere eXtreme Scale クライアントのインストール用のウィザードを開始します。
     - Linux UNIX root/*WXS_Client*/install
     - Windows root¥*WXS_Client*¥install.bat
2. ウィザードのプロンプトに従って、「**終了**」をクリックします。

制約事項: オプション・フィーチャー・パネルに、インストールを選択できるフィーチャーがリストされます。ただし、製品のインストール後に、その製品環境にフィーチャーを順次追加することはできません。初期の製品インストール時にフィーチャーのインストールを選択しなかった場合、そのフィーチャーを追加するには、製品をアンインストールしてから再インストールする必要があります。

タスクの結果

Windows WebSphere eXtreme Scale クライアントを Windows® にインストールした場合には、インストールの結果で以下のテキストが表示されることがあります。

```
Success: The installation of the following product was successful:
WebSphere eXtreme Scale Client.  Some configuration steps have errors.
For more information, refer to the following log file:
<WebSphere Application Server install root>¥logs¥wxs_client¥install¥log.txt"
Review the installation log (log.txt) and review the deployment manager
augmentation log.
```

iscdeploy.sh ファイルで障害が発生しても、エラーを無視して構いません。このエラーでは、問題は生じません。

次のタスク

eXtreme Scale の構成を参照して、クライアント・アプリケーション・プロセスおよびサーバー・プロセスをセットアップします。

関連資料

59 ページの『インストール・パラメーター』
製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントの WebSphere Application Server との統合

WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアント を WebSphere Application Server または WebSphere Application Server Network Deployment がインストールされている環境にインストールできます。 WebSphere Application Server または WebSphere Application Server Network Deployment の既存のフィーチャーを使用して、 eXtreme Scale アプリケーションを拡張できます。

始める前に

- WebSphere Application ServerまたはWebSphere Application Server Network Deploymentをインストールします。詳しくは、アプリケーション・サービス提供環境のインストールを参照してください。
- インストールするバージョン (バージョン 6.0.x、バージョン 6.1、またはバージョン 7.0) に基づいて、WebSphere Application Server または WebSphere Application Server Network Deployment の最新のフィックスパックを適用して、製品レベルを更新してください。詳しくは、WebSphere Application Server の最新フィックスパックを参照してください。
- ターゲット・インストール・ディレクトリーに WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアント の既存インストールがないことを確認します。
- WebSphere Application Server または WebSphere Application Server Network Deployment 環境で実行中のすべてのプロセスを停止します。
  stopManager、stopNode、および stopServer の各コマンドの詳細については、 コマンド行ツールの使用を参照してください。

  注意:
  すべての実行中のプロセスを必ず停止してください。実行中のプロセスを停止しなくてもインストールは続行しますが、一部のプラットフォームでは予測不能な結果が生じて、インストールが不確定状態になります。

重要: WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアント をインストールする際には、WebSphere Application Server をインストールしたのと同じディレクトリーにインストールする必要があります。例えば、WebSphere Application Server を `C:¥<was_root>` にインストールした場合には、WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアント のインストールのターゲット・ディレクトリーにも、`C:¥<was_root>` を選択する必要があります。

このタスクについて

eXtreme Scale を WebSphere Application Server または WebSphere Application Server Network Deployment に統合して、eXtreme Scale の機能をご使用の Java Platform, Enterprise Edition アプリケーションに適用します。Java EE アプリケーションはデータ・グリッドをホストし、クライアント接続を使用してそのデータ・グリッドにアクセスします。

表 3. WebSphere eXtreme Scale用のランタイム・ファイル： 次の表に、このインストールに含まれる Java アーカイブ (JAR) ファイルをリストします。

ファイル名	環境	インストールの場所	説明
wxsdynacache.jar	クライアントおよびサーバー	lib	wxsdynacache.jar ファイルには、動的キャッシュ・プロバイダーと一緒に使用するために必要なクラスが含まれています。
wsobjectgrid.jar	ローカルおよびクライアント	lib	wsobjectgrid.jar には、eXtreme Scale 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。
ogagent.jar	ローカル、クライアント、およびサーバー	lib	ogagent.jar ファイルには、EntityManager API と一緒に使用される Java インスツルメンテーション・エージェントの実行に必要なランタイム・クラスが含まれています。
ogsip.jar	サーバー	lib	ogsip.jar ファイルには、WebSphere Application Server バージョン 6.1.x との互換性がある、 eXtreme Scale Session Initiation Protocol (SIP) セッション管理ランタイムが含まれています。
sessionobjectgrid.jar	クライアントおよびサーバー	lib	sessionobjectgrid.jar ファイルには、eXtreme Scale HTTP セッション管理ランタイムが含まれています。
sessionobjectgridsip.jar	サーバー	lib	sessionobjectgridsip.jar ファイルには、 WebSphere Application Server バージョン 7.x との互換性がある、eXtreme Scale SIP セッション管理ランタイムが含まれています。
wsogclient.jar	ローカルおよびクライアント	lib	wsogclient.jar ファイルは、WebSphere Application Server バージョン 6.0.2 以降を含む環境を使用した場合にインストールされます。このファイルには、ローカル・ランタイム環境およびクライアント・ランタイム環境のみが含まれています。
wxssizeagent.jar	ローカル、クライアント、およびサーバー	lib	wxssizeagent.jar ファイルは、Java ランタイム環境 (JRE) バージョン 1.5 以上の使用時に、より正確なキャッシュ・エントリー・サイジング情報を提供するために使用されます。
oghibernate-cache.jar	クライアントおよびサーバー	optionalLibraries/ObjectGrid	oghibernate-cache.jar ファイルには、 JBoss Hibernate 用の eXtreme Scale レベル 2 キャッシュ・プラグインが含まれています。
ogspring.jar	ローカル、クライアント、およびサーバー	optionalLibraries/ObjectGrid	ogspring.jar ファイルには、SpringSource Spring フレームワーク統合用のサポート・クラスが含まれています。
xsadmin.jar	ユーティリティー	optionalLibraries/ObjectGrid	xsadmin.jar ファイルには、eXtreme Scale 管理サンプル・ユーティリティーが含まれています。
ibmcfw.jar ibmext.jar ibmorb.jar ibmorbapi.jar	クライアントおよびサーバー	optionalLibraries/ObjectGrid/endorsed	このファイル・セットには、Java SE プロセスでアプリケーションを実行するために使用されるオブジェクト・リクエスト・ブローカー (ORB) ランタイムが含まれています。
wxshyperic.jar	ユーティリティー	optionalLibraries/ObjectGrid/hyperic/lib	SpringSource Hyperic モニター・エージェントの WebSphere eXtreme Scale サーバー検出プラグイン。

表 3. WebSphere eXtreme Scale用のランタイム・ファイル (続き)： 次の表に、このインストールに含まれる Java アーカイブ (JAR) ファイルをリストします。

ファイル名	環境	インストールの場所	説明
restservice.ear	クライアント	optionalLibraries/ObjectGrid/restservice/lib	restservice.ear ファイルには、WebSphere Application Server 環境用の eXtreme Scale REST データ・サービス・アプリケーション・エンタープライズ・アーカイブが含まれています。
restservice.war	クライアント	optionalLibraries/ObjectGrid/restservice/lib	restservice.war ファイルには、別のベンダーから取得されたアプリケーション・サーバー用の eXtreme Scale REST データ・サービス Web アーカイブが含まれています。
splicerlistener.jar	ユーティリティー	optionalLibraries/ObjectGrid/session/lib	splicerlistener.jar ファイルには、eXtreme Scale HTTP セッション・マネージャー・フィルター用のスプライサー・ユーティリティーが含まれています。
splicer.jar	ユーティリティー	optionalLibraries/ObjectGrid/legacy/session/lib	splicer.jar には、eXtreme Scale HTTP セッション・マネージャー・フィルター用のバージョン 7.0 スプライサー・ユーティリティーが含まれています。

表 4. WebSphere eXtreme Scale クライアント用のランタイム・ファイル： 次の表に、このインストールに含まれる Java アーカイブ (JAR) ファイルをリストします。

ファイル名	環境	インストールの場所	説明
wxsdynacache.jar	クライアントおよびサーバー	lib	wxsdynacache.jar ファイルには、動的キャッシュ・プロバイダーと一緒に使用するために必要なクラスが含まれています。
ogagent.jar	ローカル、クライアント、およびサーバー	lib	ogagent.jar ファイルには、EntityManager API と一緒に使用される Java インスツルメンテーション・エージェントの実行に必要なランタイム・クラスが含まれています。
ogsip.jar	サーバー	lib	ogsip.jar ファイルには、WebSphere Application Server バージョン 6.1.x との互換性がある、eXtreme Scale Session Initiation Protocol (SIP) セッション管理ランタイムが含まれています。
sessionobjectgrid.jar	クライアントおよびサーバー	lib	sessionobjectgrid.jar ファイルには、eXtreme Scale HTTP セッション管理ランタイムが含まれています。
sessionobjectgridsip.jar	サーバー	lib	sessionobjectgridsip.jar ファイルには、WebSphere Application Server バージョン 7.x との互換性がある、eXtreme Scale SIP セッション管理ランタイムが含まれています。
wsogclient.jar	ローカルおよびクライアント	lib	wsogclient.jar ファイルは、WebSphere Application Server バージョン 6.0.2 以降を含む環境を使用した場合にインストールされます。このファイルには、ローカル・ランタイム環境およびクライアント・ランタイム環境のみが含まれています。
wxssizeagent.jar	ローカル、クライアント、およびサーバー	lib	wxssizeagent.jar ファイルは、Java ランタイム環境 (JRE) バージョン 1.5 以上の使用時に、より正確なキャッシュ・エントリー・サイジング情報を提供するために使用されます。
oghibernate-cache.jar	クライアントおよびサーバー	optionalLibraries/ObjectGrid	oghibernate-cache.jar ファイルには、 JBoss Hibernate 用の eXtreme Scale レベル 2 キャッシュ・プラグインが含まれています。
ogspring.jar	ローカル、クライアント、およびサーバー	optionalLibraries/ObjectGrid	ogspring.jar ファイルには、SpringSource Spring フレームワーク統合用のサポート・クラスが含まれています。
xsadmin.jar	ユーティリティー	optionalLibraries/ObjectGrid	xsadmin.jar ファイルには、eXtreme Scale 管理サンプル・ユーティリティーが含まれています。

表 4. *WebSphere eXtreme Scale* クライアント用のランタイム・ファイル *(続き)*： 次の表に、このインストールに含まれる Java アーカイブ (JAR) ファイルをリストします。

ファイル名	環境	インストールの場所	説明
ibmcfw.jar ibmext.jar ibmorb.jar ibmorbapi.jar	クライアントおよびサーバー	optionalLibraries/ObjectGrid/endorsed	このファイル・セットには、Java SE プロセスでアプリケーションを実行するために使用されるオブジェクト・リクエスト・ブローカー (ORB) ランタイムが含まれています。
wxshyperic.jar	ユーティリティー	optionalLibraries/ObjectGrid/hyperic/lib	SpringSource Hyperic モニター・エージェントの WebSphere eXtreme Scale サーバー検出プラグイン。
restservice.ear	クライアント	optionalLibraries/ObjectGrid/restservice/lib	restservice.ear ファイルには、WebSphere Application Server 環境用の eXtreme Scale REST データ・サービス・アプリケーション・エンタープライズ・アーカイブが含まれています。
restservice.war	クライアント	optionalLibraries/ObjectGrid/restservice/lib	restservice.war ファイルには、別のベンダーから取得されたアプリケーション・サーバー用の eXtreme Scale REST データ・サービス Web アーカイブが含まれています。
splicerlistener.jar	ユーティリティー	optionalLibraries/ObjectGrid/session/lib	splicerlistener.jar ファイルには、eXtreme Scale HTTP セッション・マネージャー・フィルター用のスプライサー・ユーティリティーが含まれています。
splicer.jar	ユーティリティー	optionalLibraries/ObjectGrid/legacy/session/lib	splicer.jar には、eXtreme Scale HTTP セッション・マネージャー・フィルター用のバージョン 7.0 スプライサー・ユーティリティーが含まれています。

手順

1. ウィザードを使用して、インストールを完了します。
   - 以下のスクリプトを実行して、WebSphere eXtreme Scale フルインストール用のウィザードを開始します。
     - Linux UNIX *dvd_root*/install
     - Windows *dvd_root*¥install.bat
   - 以下のスクリプトを実行して、WebSphere eXtreme Scale クライアントのインストール用のウィザードを開始します。
     - Linux UNIX root/*WXS_Client*/install
     - Windows root¥*WXS_Client*¥install.bat
2. ウィザードのプロンプトに従います。

   オプション・フィーチャー・パネルに、インストールを選択できるフィーチャーがリストされます。ただし、製品のインストール後に、その製品環境にフィーチャーを順次追加することはできません。初期の製品インストール時にフィーチャーのインストールを選択しなかった場合、そのフィーチャーを追加するには、製品をアンインストールしてから再インストールする必要があります。

   プロファイル拡張パネルには、eXtreme Scale のフィーチャーで拡張するために選択できる既存プロファイルがリストされます。既に使用中の既存プロファイルを選択すると、警告パネルが表示されます。インストールを続行するには、その

プロファイルに構成されているサーバーを停止するか、「**戻る**」をクリックして選択からそのプロファイルを除去します。

タスクの結果

Windows WebSphere eXtreme Scale クライアントを Windows にインストールした場合には、インストールの結果で以下のテキストが表示されることがあります。

```
Success: The installation of the following product was successful:
WebSphere eXtreme Scale Client.  Some configuration steps have errors.
For more information, refer to the following log file:
<WebSphere Application Server install root>¥logs¥wxs_client¥install¥log.txt"
Review the installation log (log.txt) and review the deployment manager
augmentation log.
```

`iscdeploy.sh` ファイルで障害が発生しても、エラーを無視して構いません。このエラーでは、問題は生じません。

次のタスク

WebSphere Application Server バージョン 6.1 またはバージョン 7.0 を実行している場合、プロファイル管理ツール・プラグインまたは manageprofiles コマンドが使用できます。WebSphere Application Server バージョン 6.0.2 を実行している場合、プロファイルの作成および拡張には、wasprofile コマンドを使用する必要があります。

ご使用の WebSphere Application Server 環境で、アプリケーションのデプロイ、カタログ・サービスの開始、およびコンテナーの開始を実行します。詳細については、369 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

関連資料

59 ページの『インストール・パラメーター』
製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

Installation Factory プラグインを使用したカスタマイズ・パッケージの作成およびインストール

カスタマイズ・インストール・パッケージ (CIP) または統合インストール・パッケージ (IIP) を作成するには、 IBM Installation Factory plug-in for WebSphere eXtreme Scale を使用します。 CIP には、単一の製品インストール・パッケージと各種のオプション資産が含まれます。 IIP は、1 つ以上のインストール・パッケージを組み合わせて、自分でデザインした 1 つのインストール・ワークフローにします。

始める前に

eXtreme Scale のカスタマイズ・パッケージを作成してインストールする前に、次の製品をまずダウンロードしてください。

- IBM Installation Factory for WebSphere Application Server
- IBM Installation Factory plug-in for WebSphere eXtreme Scale

このタスクについて

Installation Factory を使用して、単一の製品コンポーネントを保守パッケージ、カスタマイズ・スクリプト、その他のファイルと組み合わせることによって、CIP が作成できます。 IIP を作成した場合、個別のコンポーネントまたはインストール・パッケージを単一のインストール・パッケージに集約します。

ビルド定義ファイル

ビルド定義ファイルは、カスタマイズ・インストール・パッケージ (CIP) または統合インストール・パッケージ (IIP) をビルドしてインストールする方法を指定する XML 文書です。IBM Installation Factory for WebSphere eXtreme Scale は、ビルド定義ファイルのパッケージ詳細を読み取り、CIP または IIP を生成します。

CIP または IIP を作成する前に、各カスタマイズ・パッケージのビルド定義ファイルを作成する必要があります。ビルド定義ファイルは、インストールする製品コンポーネントまたはインストール・パッケージ、CIP または IIP のロケーション、組み込む保守パッケージ、インストール・スクリプト、および組み込むように選択したその他のファイルを説明します。IIP のビルド定義ファイル内で Installation Factory が各インストール・パッケージをインストールする順序を指定することもできます。

ビルド定義ウィザードによって、ビルド定義ファイルの作成プロセスを実行することができます。また、ウィザードを使用して、既存のビルド定義ファイルを変更することもできます。ビルド定義ウィザードの各パネルには、パッケージ ID、ビルド定義のインストール・ロケーション、カスタマイズ・パッケージのインストール・ロケーションなど、カスタマイズ・パッケージに関する情報を求めるプロンプトが出ます。この情報はすべて新規のビルド定義ファイルに保存されるか、変更されて既存のビルド定義ファイルに保存されます。詳しくは、CIP ビルド定義ウィザード・パネルおよび IIP ビルド定義ウィザード・パネルを参照してください。

ビルド定義ファイルのみを作成するには、コマンド行インターフェース・ツールを使用して、GUI の外部でカスタマイズ・パッケージを生成することができます。詳しくは、37 ページの『CIP または IIP のサイレント・インストール』を参照してください。

ビルド定義ファイルの作成と CIP の生成

IBM Installation Factory plug-in for WebSphere eXtreme Scale は、ビルド定義ファイルに指定した詳細に従って、カスタマイズ・インストール・パッケージ (CIP) を生成します。ビルド定義では、インストールする製品パッケージ、CIP のロケーション、インストールに組み込む保守パッケージ、インストール・スクリプト・ファイル、および CIP に組み込むその他のファイルを指定します。

このタスクについて

ビルド定義ウィザードを使用して、ビルド定義ファイルを作成し、CIP を生成します。

手順

1. *IF_HOME*/bin ディレクトリーから次のスクリプトを実行して、 Installation Factory を開始します。

- UNIX Linux `ifgui.sh`
- Windows `ifgui.bat`

「**ビルド定義の新規作成**」アイコンをクリックします。

2. ビルド定義ファイルに組み込む製品を選択して「**終了**」をクリックし、ビルド定義ウィザードを開始します。
3. ウィザードのプロンプトに従います。

「インストール・スクリプトとアンインストール・スクリプト」パネルで、「**スクリプトの追加...**」をクリックして、テーブルにカスタマイズ・インストール・スクリプトを取り込みます。スクリプト・ファイルのロケーションを入力し、エラー・メッセージが表示された場合に続行するチェック・ボックスをクリアします。デフォルトでは、操作は停止されます。「**OK**」をクリックしてパネルに戻ります。

タスクの結果

これで、ビルド定義ファイルが作成されてカスタマイズされ、接続モードでの作業を選択している場合には、CIP が生成されます。

ビルド定義ファイルから CIP を生成するオプションが、ビルド定義ウィザードにない場合は、 *IF_HOME*`/bin` ディレクトリーから `ifcli.sh|bat` スクリプトを実行して生成することができます。

次のタスク

CIP をインストールします。詳しくは、『CIP のインストール』を参照してください。

CIP のインストール:

カスタマイズ・インストール・パッケージ (CIP) をインストールすることで、製品インストール処理を簡素化できます。 CIP は、1 つ以上の保守パッケージ、構成スクリプト、およびその他のファイルを含むことができる単一の製品インストール・イメージです。

始める前に

CIP をインストールする前に、ビルド定義ファイルを作成して、CIP に組み込むオプションを指定する必要があります。詳しくは、 29 ページの『ビルド定義ファイルの作成と CIP の生成』を参照してください。

このタスクについて

CIP は、単一の製品コンポーネントを保守パッケージ、カスタマイズ・スクリプト、その他のファイルと組み合わせてインストールします。

手順

1. インストールの準備を行うワークステーション上で実行されているすべてのプロセスを停止します。 デプロイメント・マネージャーを停止するには、次のスクリプトを実行します。

- Linux UNIX *profile_root*/bin/stopManager.sh
- Windows *profile_root*¥bin¥stopManager.bat

ノードを停止するには、次のスクリプトを実行します。

- Linux UNIX *profile_root*/bin/stopNode.sh
- Windows *profile_root*¥bin¥stopNode.bat

2. 次のスクリプトを実行して、インストールを開始します。
   - Linux UNIX *CIP_home*/bin/install
   - Windows *CIP_home*¥bin¥install.bat
3. ウィザードのプロンプトに従って、インストールを完了します。

   オプション・フィーチャー・パネルに、インストールを選択できるフィーチャーがリストされます。ただし、製品のインストール後に、その製品環境にフィーチャーを順次追加することはできません。初期の製品インストール時にフィーチャーのインストールを選択しなかった場合、そのフィーチャーを追加するには、製品をアンインストールしてから再インストールする必要があります。

   プロファイル拡張パネルには、eXtreme Scale のフィーチャーで拡張するために選択できる既存プロファイルがリストされます。既に使用中の既存プロファイルを選択すると、警告パネルが表示されます。インストールを続行するには、そのプロファイルに構成されているサーバーを停止するか、「**戻る**」をクリックして選択からそのプロファイルを除去します。

タスクの結果

CIP が正常にインストールされました。

次のタスク

WebSphere Application Server バージョン 6.1 またはバージョン 7.0 を実行している場合、プロファイル管理ツール・プラグインまたは manageprofiles コマンドを使用して、プロファイルを作成および拡張することができます。 WebSphere Application Server バージョン 6.0.2 を実行している場合、プロファイルの作成および拡張には、wasprofile コマンドを使用する必要があります。詳しくは、45 ページの『WebSphere eXtreme Scale のプロファイルの作成および拡張』を参照してください。

インストール処理中に eXtreme Scale のプロファイルを拡張した場合には、ご使用の WebSphere Application Server 環境で、アプリケーションのデプロイ、カタログ・サービスの開始、およびコンテナーの開始を実行できます。詳しくは、369 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

既存の製品インストール済み環境に保守を適用するための CIP のインストール:

カスタマイズ・インストール・パッケージ (CIP) をインストールすることによって、既存の製品インストール済み環境に保守パッケージを適用することができます。 CIP で既存のインストール済み環境に保守を適用する処理を一般的にスリップ・インストール と呼びます。

始める前に

ビルド定義ファイルを作成して、CIP に組み込むオプションを指定します。詳しくは、 29 ページの『ビルド定義ファイルの作成と CIP の生成』を参照してください。

このタスクについて

リフレッシュ・パックまたはフィックスパック、あるいはその両方を含む CIP で保守を適用する場合、以前にインストールされたすべてのプログラム診断依頼書 (APAR) はウィザードによってアンインストールされます。CIP が製品と同じレベルの場合、以前にインストールされた APAR は、CIP 内にパッケージ化されている場合に限り、その状態のままです。既存のインストール済み環境に保守を正常に適用するには、インストールされているフィーチャーを CIP に含める必要があります。

手順

1. インストールの準備を行うワークステーション上で実行されているすべてのプロセスを停止します。 デプロイメント・マネージャーを停止するには、次のスクリプトを実行します。
   - Linux UNIX *profile_root*/bin/stopManager.sh
   - Windows *profile_root*¥bin¥stopManager.bat

   ノードを停止するには、次のスクリプトを実行します。
   - Linux UNIX *profile_root*¥bin¥stopNode.sh
   - Windows *profile_root*¥bin¥stopNode.bat
2. 次のスクリプトを実行して、インストールを開始します。
   - Linux UNIX *CIP_home*/bin/install
   - Windows *CIP_home*¥bin¥install.bat
3. ウィザードのプロンプトに従って、インストールを完了します。

   インストール・プレビュー要約には、結果として得られた製品バージョン、適用可能なフィーチャーおよび暫定修正がリストされます。次に、ウィザードが保守を正常に適用し、製品のフィーチャーを更新します。

タスクの結果

製品のバイナリー・ファイルが *was_home*/properties/version/nif/backup ディレクトリーにコピーされます。 IBM Update Installer を使用して更新をアンインストールし、ワークステーションを復元することができます。詳しくは、 33 ページの『既存の製品インストール済み環境からの CIP 更新のアンインストール』を参照してください。

既存の製品インストール済み環境からの CIP 更新のアンインストール:

製品全体を除去せずに、既存の製品インストール済み環境から CIP の更新を除去することができます。 CIP の更新をアンインストールするには、IBM Update Installer バージョン 7.0.0.4 を使用します。このタスクは、*スリップ・アンインストール* とも呼ばれます。

始める前に

製品の、少なくとも 1 つ以上の既存コピーがシステムにインストールされている必要があります。

手順

1. Update Installer のバージョン 7.0.0.4 を次の FTP サイトからダウンロードします。

   ftp://ftp.software.ibm.com/software/websphere/cw/process_server/FEP/UPDI/7004
2. Update Installer をインストールします。 詳しくは、WebSphere Application Server インフォメーション・センターの Update Installer for WebSphere Software のインストールを参照してください。
3. CIP のインストール後に環境に追加したフィックスパック、リフレッシュ・パック、または暫定修正があれば、それをアンインストールします。
4. スリップ・インストールに組み込んだ暫定修正があれば、それをアンインストールします。 これは、単一のフィックスパックまたはリフレッシュ・パックをアンインストールするプロセスと同じです。ただし、CIP に組み込まれていた保守は現在、単一の操作で組み込まれるようになりました。
5. Update Installer を使用して CIP をアンインストールします。 保守レベルは更新前の状態に戻り、CIP は、ファイル名に接頭部として追加される CIP ID によって示されます。 次の例では、保守パッケージの選択パネルで、他の通常の保守パッケージとは異なって CIP がどのように表示されるかを示しています。

   CIP

   `com.ibm.ws.cip.7000.wxs.primary.ext.pak`

タスクの結果

既存の製品インストール済み環境から CIP 更新を正常に除去しました。

ビルド定義ファイルの作成と IIP の生成

IBM Installation Factory plug-in for WebSphere eXtreme Scale は、ビルド定義ファイルによって指定されるプロパティーに基づいて、IIP を生成します。ビルド定義ファイルには、IIP に含めるインストール・パッケージ、Installation Factory が各パッケージをインストールする順序、IIP の場所などの情報が含まれています。

このタスクについて

ビルド定義ウィザードを使用して、ビルド定義ファイルを作成し、IIP を生成します。

手順

1. *IF_HOME*/bin ディレクトリーから次のスクリプトを実行して、 Installation Factory を開始します。
   - UNIX Linux `ifgui.sh`
   - Windows `ifgui.bat`
2. 「**統合インストール・パッケージの新規作成**」アイコンをクリックして、ビルド定義ウィザードを開始します。
3. ウィザードのプロンプトに従います。
   a. 「IIP の構成 (Construct the IIP)」パネルで、サポートされるインストール・パッケージをリストから選択し、「**インストーラーの追加**」をクリックして、インストール・パッケージを IIP に追加します。 パッケージ名、パッケージ ID、パッケージ・プロパティーを示したパネルが表示されます。選択したパッケージに関する固有情報を表示するには、「**インストール・パッケージの情報を表示**」をクリックします。「**変更**」をクリックして、各オペレーティング・システムのインストール・パッケージのディレクトリー・パスを入力します。 WebSphere Extended Deployment のインストール・パッケージを現在追加している場合には、サポートされるすべてのオペレーティング・システムに同じパッケージを使用するオプションのチェック・ボックスを選択します。「**OK**」をクリックして、「IIP の構成 (Construct the IIP)」パネルに戻ります。デフォルトで、呼び出しが作成されます。
      - インストール・パッケージのディレクトリー・パスを変更するには、 IIP リストに使用されているインストール・パッケージからパッケージを選択し、「**変更**」をクリックします。
      - 呼び出しを変更するには、呼び出しを選択して、「**変更**」をクリックします。各オペレーティング・システムにおける呼び出しのデフォルトのインストール場所を指定します。デフォルトのインストール・モードにサイレント・インストールを選択する場合には、その場所を応答ファイルに指定します。
      - 「**呼び出しの追加**」をクリックして、インストール・パッケージに呼び出しコントリビューションを追加します。呼び出しのプロパティーを指定できるパネルが表示されます。
      - インストール・パッケージまたは呼び出しを除去するには、「**除去**」をクリックします。
4. 選択の要約を確認し、「**ビルド定義ファイルを保存し、統合インストール・パッケージを生成する**」オプションを選択してから、「**終了**」をクリックします。

   あるいは、IIP を生成せずにビルド定義ファイルを保存することもできます。このオプションを使用した場合には、*IF_home*/bin/ ディレクトリーから `ifcli.bat | ifcli.sh` スクリプトを実行することによって、ウィザードの外で IIP を実際に生成します。

タスクの結果

これで、IIP のビルド定義ファイルが作成され、カスタマイズされます。

次のタスク

IIP をインストールします。

IIP のインストール:

統合インストール・パッケージ (IIP) をインストールするには、IBM Installation Factory plug-in for WebSphere eXtreme Scale を使用します。 IIP は、1 つ以上のインストール・パッケージを組み合わせて、自分でデザインした 1 つのワークフローにします。

始める前に

CIP をインストールする前に、ビルド定義ファイルを作成して、CIP に組み込むオプションを指定する必要があります。詳しくは、33 ページの『ビルド定義ファイルの作成と IIP の生成』を参照してください。

このタスクについて

IIP には、1 つ以上の一般出荷可能インストール・パッケージ、 1 つ以上の CIP、その他のオプションのファイルおよびディレクトリーを含めることができます。 IIP をインストールすることによって、複数のインストール・パッケージ、つまりコントリビューション を 1 つのパッケージに集約し、コントリビューションを特定の順序でインストールしてすべてのインストールを完了します。

手順

1. 以下のスクリプトを実行して、ウィザードを開始します。
   - Linux UNIX *IIP_home*/bin/install
   - Windows *IIP_home*¥bin¥install.bat
2. 「ようこそ」パネルの「**製品情報 (About)**」をクリックして、パッケージ ID、サポートされるオペレーティング・システム、組み込まれるインストール・パッケージなど、IIP の詳細を表示します。

   オプション: 各パッケージのインストール・オプションを変更するには、「**変更**」をクリックします。

   オプション: ウィザード・パネルには、「**ログの表示**」ボタンが 2 つ表示されます。各パッケージのログを表示する場合は、インストール・パッケージをリストした表の横に表示される「**ログの表示**」ボタンをクリックします。 IIP の全体のログ詳細を表示する場合は、状況情報の横に表示される「**ログの表示**」ボタンをクリックします。
3. 実行するインストール・パッケージを選択し、「**インストール**」をクリックします。 IIP に含まれるすべてのコントリビューションのリストが、呼び出し順に表示されます。インストール中に実行しないコントリビューション呼び出しを指定するには、「**インストール名**」フィールドの横にあるチェック・ボックスをクリアします。

タスクの結果

IIP が正常にインストールされました。

IIP の既存ビルド定義ファイルの変更:

IIP のプロパティーに編集や追加を行って、インストールをさらにカスタマイズすることができます。

このタスクについて

IIP のプロパティーを変更するには、既存のビルド定義ファイルを変更します。

手順

1. *IF_HOME*/bin ディレクトリーから次のスクリプトを実行して、 Installation Factory を開始します。
   - UNIX Linux `ifgui.sh`
   - Windows `ifgui.bat`
2. 「**ビルド定義を開く**」アイコンをクリックし、変更するビルド定義ファイルを選択します。
3. 変更する IIP の具体的なプロパティーを選択します。 以下のリストには、可能な変更が含まれています。
   - 現行モード選択を変更します。接続モードでは、現行ワークステーションから、使用するビルド定義を作成し、また、オプションで IIP を生成します。切断モードでは、別のワークステーションで使用するビルド定義ファイルを作成します。
   - IIP がサポートする既存のオペレーティング・システムを追加または除去します。
   - IIP の既存の ID およびバージョンを編集します。
   - ビルド定義ファイルのターゲット・ロケーションを編集します。
   - IIP のターゲット・ロケーションを編集します。
   - IIP のインストール・ウィザードを表示するかどうかを変更します。ウィザードでは、IIP に関する情報と、IIP 実行時のインストール・オプションが示されます。
   - IIP に含まれるインストール・パッケージを追加、除去、および編集します。

     重要: サポートされるオペレーティング・システムを追加し、IIP 内のインストール・パッケージのプロパティーを更新していないと、選択したコントリビューションに、IIP がサポートするすべてのオペレーティング・システムに識別されているインストール・パッケージが含まれないことを示す警告メッセージを受け取ります。続行する場合には「**はい**」、インストール・パッケージを編集する場合には「**いいえ**」をクリックします。
4. 選択の要約を確認し、「**ビルド定義ファイルを保存し、統合インストール・パッケージを生成する**」を選択し、「**終了**」をクリックします。

CIP または IIP のサイレント・インストール

ニーズに具体的に対応して構成する完全修飾応答ファイル、またはコマンド行に受け渡すパラメーターのいずれかを使用して、製品のカスタマイズ・インストール・パッケージ (CIP) または統合インストール・パッケージ (IIP) をサイレントにインストールすることができます。

始める前に

CIP または IIP のビルド定義ファイルを作成します。詳しくは、29 ページの『ビルド定義ファイルの作成と CIP の生成』を参照してください。

このタスクについて

サイレント・インストールは、グラフィカル・ユーザー・インターフェース (GUI) バージョンが使用するのと同じインストール・プログラムを使用します。ただし、ウィザード・インターフェースを表示する代わりに、サイレント・インストールは、カスタマイズされたファイルから、あるいはコマンド行にパスされたパラメーターからすべての応答を読み取ります。 IIP をサイレントにインストールする場合、コントリビューションの呼び出しには、応答ファイルに指定したオプションのほかに、コマンド行に直接指定したオプションを組み合わせて使用できます。ただし、コマンド行にコントリビューション・オプションを渡すと、IIP インストーラーは、特定のコントリビューションの応答ファイルに指定されたオプションをすべて無視します。詳しくは、詳細な IIP インストール・オプションを参照してください。

注: 完全修飾応答ファイル名を指定してください。相対パスを指定すると、エラーが発生したことをまったく示さずにインストールが失敗します。

手順

1. オプション: 応答ファイルを使用して CIP または IIP をインストールする場合は、まずファイルをカスタマイズします。
   a. 応答ファイル `wxssetup.response.txt` を製品 DVD からディスク・ドライブにコピーします。
   b. 任意のテキスト・エディターで応答ファイルを開き、編集します。 このファイルには、構成プロセスを支援するコメントが含まれています。ファイルには、次のパラメーターを組み込む必要があります。
      - ご使用条件
      - 製品インストールのロケーション

      ヒント: インストーラーは、インストールに選択したロケーションを使用して WebSphere Application Server インスタンスのインストール場所を判別します。複数の WebSphere Application Server インスタンスが含まれるノードにインストールする場合、そのロケーションを明確に定義してください。
   c. 次のスクリプトを実行して、カスタマイズ応答ファイルを開始します。
      - Linux UNIX `install -options /absolute_path/response_file.txt -silent`

- Windows `install.bat -options C:¥drive_path¥response_file.txt -silent`

2. オプション: 特定のパラメーターをコマンド行に渡すことによって CIP または IIP をインストールする場合は、次のスクリプトを実行してインストールを開始します。
   - Linux UNIX `install -silent -OPT silentInstallLicenseAcceptance=true -OPT installLocation=install_location`
   - Windows `install.bat -silent -OPT silentInstallLicenseAcceptance=true -OPT installLocation=install_location`

   ここで、*install_location* は、既存の WebSphere Application Server インストールのロケーションです。
3. 結果ログを検討して、エラーやインストールの失敗を調べます。

タスクの結果

CIP または IIP がサイレントにインストールされました。

次のタスク

WebSphere Application Server バージョン 6.1 またはバージョン 7.0 を実行している場合、プロファイル管理ツール・プラグインまたは manageprofiles コマンドを使用して、プロファイルを作成および拡張することができます。 WebSphere Application Server バージョン 6.0.2 を実行している場合、プロファイルの作成および拡張には、wasprofile コマンドを使用する必要があります。

インストール処理中に eXtreme Scale のプロファイルを拡張した場合には、ご使用の WebSphere Application Server 環境で、アプリケーションのデプロイ、カタログ・サービスの開始、およびコンテナーの開始を実行できます。詳しくは、369 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

関連資料

59 ページの『インストール・パラメーター』
製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

wxssetup.response.txt ファイル:

完全修飾応答ファイルを使用して、WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアント をサイレント・インストールできます。

注意:
インストール場所のパスの最後に、/ や \ など、末尾のスラッシュを追加しないでください。これらのパスは、installLocation 属性で指定されます。インストール場所の最後にスラッシュを追加すると、インストールが失敗することがあります。例えば次のパスは、インストール失敗の原因となります。

```
-OPT installLocation="/usr/IBM/WebSphere/eXtremeScale/"
```

このパスは次のように指定する必要があります。

```
-OPT installLocation="/usr/IBM/WebSphere/eXtremeScale"
```

WebSphere eXtreme Scale フルインストール用の応答ファイル

```
################################################################################
#
# IBM WebSphere eXtreme Scale V7.1.0 InstallShield オプション・ファイル
#
# ウィザード名: インストール
# ウィザード・ソース: setup.jar
#
# ウィザードを「-options」コマンド行オプションを指定して実行した場合に、この
# ファイルを使用して、以下で指定したオプションでインストールを構成できます。
# 値の変更方法については、各設定の説明を参照してください。
# すべての値を一対の二重引用符で囲ってください。
#
# オプション・ファイルの一般的用途は、サイレント・モードでのウィザード実行です。
# オプション・ファイルの作成者は、グラフィカル・モードまたはコンソール・モードで
# ウィザードを実行せずに、ウィザード設定を指定できます。サイレント・モードの
# 実行でこのオプション・ファイルを使用するには、ウィザード実行時に以下の
# コマンド行引数を使用します。
#
#    -options "D:¥installImage¥WXS¥wxssetup.response" -silent
#
# なお、完全修飾応答ファイル名を使用する必要があります。
#
################################################################################

################################################################################
#
# ご使用条件の受け入れ
#
# 有効値:
#  true - ご使用条件を受け入れます。製品をインストールします。
#  false - ご使用条件に同意しません。インストールされません。
#
# インストールが行われない場合は、ユーザーの一時ディレクトリーにある
# 一時ログ・ファイルにそのことが記録されます。
#
# この応答ファイルの silentInstallLicenseAcceptance プロパティーを true に
# 変更することで、このプログラムに付属の IBM プログラムのご使用条件を確認して
# 同意したことになります。このご使用条件は、
# CD_ROOT¥XD¥wxs.primary.pak¥repository¥legal.xs¥license.xs
# にあります。
# このご使用条件に同意しない場合には、値を変更しないでください。また、
# プログラムをダウンロード、インストール、コピー、アクセス、
# 使用しないでください。入手元へプログラムおよびライセンス証書を速やかに
# 返却して、支払額を払い戻してください。
#
-OPT silentInstallLicenseAcceptance="false"


################################################################################
# ノンブロッキング前提条件検査
#
# ノンブロッキング前提条件検査を使用不可にする場合は、以下の行のコメントを
```

```
# 外してください。これにより、前提条件検査が失敗した場合でも、インストールを
# 進め、警告を記録するように、インストーラーに指示します。
#
#-OPT disableNonBlockingPrereqChecking="true"


###############################################################################
#
# インストール場所
#
# 製品のインストール場所。製品をインストールする有効なディレクトリーを指定
# します。ディレクトリーにスペースが含まれている場合には、以下の Windows の
# 例のように、二重引用符でディレクトリーを囲みます。なお、インストール場所に
# スペースを含められるのは、Windows オペレーティング・システムのみです。
# Windows の場合、最大パス長は 60 文字です。
#
# 以下に、root ユーザーでインストールする場合の各オペレーティング・システムの
# デフォルト・インストール場所のリストを示します。デフォルトでは、この応答
# ファイルでは、Windows のインストール場所が使用されています。別の
# オペレーティング・システムのデフォルト・インストール場所を使用する場合は、
# 以下で、該当デフォルト・インストール場所項目のコメントを外し (「#」を削除)、
# Windows オペレーティング・システムの項目をコメント化 (「#」を追加) します。
#
# インストール場所は、WebSphere eXtreme Scale をスタンドアロン・デプロイメント
# としてインストールするのか、既存の WebSphere Application Server インストール
# 済み環境に統合するのかを決定するために使用されます。
#
# 指定場所が既存の WebSphere Application Server または WebSphere Network
# Deployment インストール済み環境の場合、eXtreme Scale は既存の WebSphere
# Application Server に統合されます。指定場所が新規または空のディレクトリー
# の場合は、WebSphere eXtreme Scale は、スタンドアロン・デプロイメントとして
# インストールされます。
#
# 注: 指定インストール場所に WebSphere eXtreme Scale、WebSphere eXtended
# Deployment DataGrid、または ObjectGrid の前のインストール済み環境が
# 存在する場合には、インストールは失敗します。
#
# AIX のデフォルト・インストール場所:
#
#  -OPT installLocation="/usr/IBM/WebSphere/eXtremeScale"
#
# HP-UX、Solaris、または Linux のデフォルト・インストール場所:
#
#  -OPT installLocation="/opt/IBM/WebSphere/eXtremeScale"
#
#
# Windows のデフォルト・インストール場所:
#
# -OPT installLocation="C:¥Program Files¥IBM¥WebSphere¥eXtremeScale"

#
# Unix で非 root ユーザーとして、または Windows で非管理者としてインストール
# する場合は、以下のデフォルト・インストール場所をお勧めします。選択する
# インストール場所に対する書き込み許可が備わった状態にしてください。
#
# AIX のデフォルト・インストール場所:
#
#  -OPT installLocation="<ユーザーの home>/IBM/WebSphere/eXtremeScale"
#
# HP-UX、Solaris、または Linux のデフォルト・インストール場所:
#
#  -OPT installLocation="<ユーザーの home>/IBM/WebSphere/eXtremeScale"
#
# Windows のデフォルト・インストール場所:
#
# -OPT installLocation="C:¥IBM¥WebSphere¥eXtremeScale"
```

```
################################################################################
#  オプション・フィーチャーのインストール
#
# 目的の各フィーチャーを「true」に設定することで、インストールするオプション・
# フィーチャーを指定します。インストールしないオプション・フィーチャーは、
# 「false」に設定します。
#
# オプション selectServer、selectClient、selectPF、および selectXSStreamQuery
# は、上述の installLocation オプションに WebSphere Application Server の
# インストール済み環境が含まれている場合にのみ有効です。これらのオプションは、
# WebSphere eXtreme Scale スタンドアロン・インストール済み環境では無視されます。
#
# WebSphere eXtreme Scale スタンドアロン・インストール済み環境では、eXtreme
# Scale サーバーおよびクライアントは自動的にインストールされます。eXtreme
# Scale スタンドアロン・インストール済み環境のフィーチャー・オプションは、
# selectXSConsoleOther および selectXSStreamQueryOther です。

#
# このオプションを選択すると、WebSphere eXtreme Scale サーバーおよび eXtreme
# Scale 動的キャッシュ・サービス・プロバイダーの実行に必要なコンポーネントが
# インストールされます。このオプションを選択した場合は、WebSphere eXtreme
# Scale クライアントも選択する (コメントを外して値を「true」に設定する) 必要が
# あります。
# さもないと、サイレント・インストールが失敗します。
#
# -OPT selectServer="true"

#
# このオプションを選択すると、WebSphere eXtreme Scale クライアント・
# アプリケーションの実行に必要なコンポーネントがインストールされます。上の
# Server オプションを選択している場合は、このオプションも選択 (コメントを外して
# 値を「true」に設定) する必要があります。そうしないとサイレント・インストール
# が失敗します。
#
-OPT selectClient="true"

#
# このオプションを選択すると、WebSphere eXtreme Scale コンソールの実行に
# 必要なコンポーネントがインストールされます。このオプションを選択する場合は、
# コンソール・オプションは WebSphere eXtreme Scale スタンドアロン・デプロイメント
# でのみ有効であるため、上で指定するインストールの場所は新規ディレクトリー
# または空のディレクトリーにする必要があります。
# このオプションをインストールするには、以下のオプション行のコメントを外して、
# 値を「true」に設定します。
#-OPT selectXSConsoleOther="false"

#
# 以下のオプションを選択すると、非推奨機能がインストールされます。
#
# このオプションで、WebSphere Partition Facility がインストールされます。
# この機能は非推奨です。このオプションをインストールするには、以下の
# オプション行のコメントを外して、値を「true」に設定します。
#
#-OPT selectPF="false"

#
# このオプションにより、WebSphere eXtreme Scale StreamQuery for WAS が
# インストールされます。この機能は非推奨です。このオプションをインストール
# するには、以下のオプション行のコメントを外して、値を「true」に設定します。
# このオプションを選択した場合には、WebSphere eXtreme Scale クライアントも
# 選択 (コメントを外して、値を「true」に設定) する必要があります。
# さもないと、サイレント・インストールが失敗します。
#
#-OPT selectXSStreamQuery="false"

#
```

```
# このオプションにより、WebSphere eXtreme Scale StreamQuery for J2SE が
# インストールされます。この機能は非推奨です。このオプションをインストール
# するには、以下のオプション行のコメントを外して、値を「true」に設定します。
# このオプションを選択した場合には、WebSphere eXtreme Scale クライアントも
# 選択 (コメントを外して、値を「true」に設定) する必要があります。
# さもないと、サイレント・インストールが失敗します。
#
#-OPT selectXSStreamQueryOther="false"


###############################################################################
# 拡張用プロファイル・リスト
#
# 拡張する既存プロファイルを指定するか、行をコメント化してインストールで
# 検出されたすべての既存プロファイルを拡張します。
#
# 複数のプロファイルを指定するには、コンマで各プロファイル名を区切ります。
# 例えば、「AppSrv01,Dmgr01,Custom01」。リスト内でスペースは使用できません。
#
# -OPT profileAugmentList=""


###############################################################################
# トレース制御
#
# このオプションで、トレース出力フォーマットを制御できます。
# -OPT traceFormat=ALL
#
# フォーマットとして、「text」および「XML」を選択できます。デフォルトでは、
# 両方のフォーマットが 2 つの異なるトレース・ファイルに生成されます。
#
# 片方のフォーマットのみが必要な場合は、以下のように、traceFormat オプションで
# フォーマットを指定します。
#
# 有効値:
#
#  text -  簡単に読めるように、トレース・ファイルの行をプレーン・テキスト・
#          フォーマットで生成します。
#  XML  -  トレース・ファイルの行は、標準 Java ロギング XML フォーマットで生成
#          されます。テキスト・エディターまたは XML エディター、あるいは以下の
#          URL にある Apache のチェーンソー・ツールを使用して表示できます。
#          (http://logging.apache.org/log4j/docs/chainsaw.html)
#
# 取り込まれるトレース情報の量は、以下のオプションで制御できます。
# -OPT traceLevel=INFO
#
# 有効値:
#
# トレース    数値
# レベル      レベル   説明
# -------  ---------  -------------------------------------------------------
# OFF         0       トレース・ファイルは生成されません。
# SEVERE      1       重大エラーのみがトレース・ファイルに出力されます。
# WARNING     2       致命的ではない例外および警告に関するメッセージが、
#                     トレース・ファイルに追加されます。
# INFO        3       通知メッセージがトレース・ファイルに追加されます。
#                     (これは、デフォルト・トレース・レベルです)
# CONFIG      4       構成関連メッセージがトレース・ファイルに追加されます。
# FINE        5       public メソッドのメソッド呼び出しをトレースします。
# FINER       6       getter と setter を除く非 public メソッドの
#                     メソッドをトレースします。
# FINEST      7       すべてのメソッド呼び出しをトレースします。トレースの
#                     出入り口に、パラメーターおよび戻り値が含まれます。
```

WebSphere eXtreme Scale クライアント インストール用の応答ファイル

```
#############################################################################
#
# IBM WebSphere eXtreme Scale クライアント V7.1.0 InstallShield オプション・ファイル
#
# ウィザード名: インストール
# ウィザード・ソース: setup.jar
#
# ウィザードを「-options」コマンド行オプションを指定して実行した場合に、この
# ファイルを使用して、以下で指定したオプションでインストールを構成できます。
# 値の変更方法については、各設定の説明を参照してください。
# すべての値を一対の二重引用符で囲ってください。
#
# オプション・ファイルの一般的用途は、サイレント・モードでのウィザード実行です。
# オプション・ファイルの作成者は、グラフィカル・モードまたはコンソール・モードで
# ウィザードを実行せずに、ウィザード設定を指定できます。サイレント・モードの
# 実行でこのオプション・ファイルを使用するには、ウィザード実行時に以下の
# コマンド行引数を使用します。
#
#    -options "D:¥installImage¥WXS_Client¥wxssetup.response" -silent
#
# なお、完全修飾応答ファイル名を使用する必要があります。
#
#############################################################################

#############################################################################
#
# ご使用条件の受け入れ
#
# 有効値:
#  true - ご使用条件を受け入れます。製品をインストールします。
#  false - ご使用条件に同意しません。インストールされません。
#
# インストールが行われない場合は、ユーザーの一時ディレクトリーにある
# 一時ログ・ファイルにそのことが記録されます。
#
# この応答ファイルの silentInstallLicenseAcceptance プロパティーを true に
# 変更することで、このプログラムに付属の IBM プログラムのご使用条件を確認して
# 同意したことになります。このご使用条件は、
# CD_ROOT\WXS_Cleint\wxs.client.primary.pak\repository\legal.xs.client\license.xs
# にあります。
# このご使用条件に同意しない場合には、値を変更しないでください。また、
# プログラムをダウンロード、インストール、コピー、アクセス、
# 使用しないでください。入手元へプログラムおよびライセンス証書を速やかに
# 返却して、支払額を払い戻してください。
#
# -OPT silentInstallLicenseAcceptance="false"


#############################################################################
# ノンブロッキング前提条件検査
#
# ノンブロッキング前提条件検査を使用不可にする場合は、以下の行のコメントを
# 外してください。これにより、前提条件検査が失敗した場合でも、インストールを
# 進め、警告を記録するように、インストーラーに指示します。
#
# -OPT disableNonBlockingPrereqChecking="true"


#############################################################################
#
# インストール場所
#
# 製品のインストール場所。製品をインストールする有効なディレクトリーを指定
# します。ディレクトリーにスペースが含まれている場合には、以下の Windows の
# 例のように、二重引用符でディレクトリーを囲みます。なお、インストール場所に
```

```
# スペースを含められるのは、Windows オペレーティング・システムのみです。
# Windows の場合、最大パス長は 60 文字です。
#
# 以下に、root ユーザーでインストールする場合の各オペレーティング・システムの
# デフォルト・インストール場所のリストを示します。デフォルトでは、この応答
# ファイルでは、Windows のインストール場所が使用されています。別の
# オペレーティング・システムのデフォルト・インストール場所を使用する場合は、
# 以下で、該当デフォルト・インストール場所項目のコメントを外し (「#」を削除)、
# Windows オペレーティング・システムの項目をコメント化 (「#」を追加) します。
#
# インストール場所は、WebSphere eXtreme Scale をスタンドアロン・デプロイメント
# としてインストールするのか、既存の WebSphere Application Server インストール
# 済み環境に統合するのかを決定するために使用されます。
#
# 指定場所が既存の WebSphere Application Server または WebSphere Network
# Deployment インストール済み環境の場合、eXtreme Scale は既存の WebSphere
# Application Server に統合されます。指定場所が新規または空のディレクトリー
# の場合は、WebSphere eXtreme Scale は、スタンドアロン・デプロイメントとして
# インストールされます。
#
# 注: 指定インストール場所に WebSphere eXtreme Scale、WebSphere eXtended
# Deployment DataGrid、または ObjectGrid の前のインストール済み環境が
# 存在する場合には、インストールは失敗します。
#
# AIX のデフォルト・インストール場所:
#
#  -OPT installLocation="/usr/IBM/WebSphere/eXtremeScale"
#
# HP-UX、Solaris、または Linux のデフォルト・インストール場所:
#
#  -OPT installLocation="/opt/IBM/WebSphere/eXtremeScale"
#
#
# Windows のデフォルト・インストール場所:
#
# -OPT installLocation="C:¥Program Files¥IBM¥WebSphere¥eXtremeScale"

#
# Unix で非 root ユーザーとして、または Windows で非管理者としてインストール
# する場合は、以下のデフォルト・インストール場所をお勧めします。選択する
# インストール場所に対する書き込み許可が備わった状態にしてください。
#
# AIX のデフォルト・インストール場所:
#
#  -OPT installLocation="<ユーザーの home>/IBM/WebSphere/eXtremeScale"
#
# HP-UX、Solaris、または Linux のデフォルト・インストール場所:
#
#  -OPT installLocation="<ユーザーの home>/IBM/WebSphere/eXtremeScale"
#
# Windows のデフォルト・インストール場所:
#
# -OPT installLocation="C:¥IBM¥WebSphere¥eXtremeScale"


################################################################################
# 拡張用プロファイル・リスト
#
# 拡張する既存プロファイルを指定するか、行をコメント化してインストールで
# 検出されたすべての既存プロファイルを拡張します。
#
# 複数のプロファイルを指定するには、コンマで各プロファイル名を区切ります。
# 例えば、「AppSrv01,Dmgr01,Custom01」。リスト内でスペースは使用できません。
#
# -OPT profileAugmentList=""
```

```
################################################################################
# トレース制御
#
# このオプションで、トレース出力フォーマットを制御できます。
# -OPT traceFormat=ALL
#
# フォーマットとして、「text」および「XML」を選択できます。デフォルトでは、
# 両方のフォーマットが 2 つの異なるトレース・ファイルに生成されます。
#
# 片方のフォーマットのみが必要な場合は、以下のように、traceFormat オプションで
# フォーマットを指定します。
#
# 有効値:
#
#  text -  簡単に読めるように、トレース・ファイルの行をプレーン・テキスト・
#          フォーマットで生成します。
#  XML  -  トレース・ファイルの行は、標準 Java ロギング XML フォーマットで生成
#          されます。テキスト・エディターまたは XML エディター、あるいは以下の
#          URL にある Apache のチェーンソー・ツールを使用して表示できます。
#          (http://logging.apache.org/log4j/docs/chainsaw.html)
#
# 取り込まれるトレース情報の量は、以下のオプションで制御できます。
# -OPT traceLevel=INFO
#
# 有効値:
#
# トレース    数値
# レベル      レベル    説明
# -------  ---------  ------------------------------------------------------
# OFF         0       トレース・ファイルは生成されません。
# SEVERE      1       重大エラーのみがトレース・ファイルに出力されます。
# WARNING     2       致命的ではない例外および警告に関するメッセージが、
#                     トレース・ファイルに追加されます。
# INFO        3       通知メッセージがトレース・ファイルに追加されます。
#                     (これは、デフォルト・トレース・レベルです)
# CONFIG      4       構成関連メッセージがトレース・ファイルに追加されます。
# FINE        5       public メソッドのメソッド呼び出しをトレースします。
# FINER       6       getter と setter を除く非 public メソッドの
#                     メソッドをトレースします。
# FINEST      7       すべてのメソッド呼び出しをトレースします。トレースの
#                     出入り口に、パラメーターおよび戻り値が含まれます。
```

WebSphere eXtreme Scale のプロファイルの作成および拡張

製品のインストール後、WebSphere eXtreme Scale の固有のタイプのプロファイルを作成し、既存のプロファイルを拡張します。

始める前に

WebSphere eXtreme Scale をインストールします。詳しくは、24 ページの『WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントの WebSphere Application Server との統合』を参照してください。

WebSphere eXtreme Scale で使用するプロファイルの拡張はオプションですが、以下の使用シナリオでは必須です。

- WebSphere Application Server プロセスでカタログ・サービスまたはコンテナーを自動始動する場合。サーバー・プロファイルを拡張しないと、サーバーを始動するには、必ずプログラマチックに行う (ServerFactory API を使用するか、別プロセスとして startOgServer スクリプトを使用する) 必要があります。
- Performance Monitoring Infrastructure (PMI) を使用して、WebSphere eXtreme Scale メトリックをモニターする場合。

- WebSphere Application Server 管理コンソールで WebSphere eXtreme Scale のバージョンを表示する場合。

このタスクについて

WebSphere Application Server バージョン 6.0.2 での実行

ご使用の環境に WebSphere Application Server バージョン 6.0.2 が含まれている場合、次の例に示すように、 wasprofile コマンドを使用して WebSphere eXtreme Scale のプロファイルを作成または拡張します。

```
install_root/bin/wasprofile.sh|bat -augment -profileName dmgr_01
-templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
```

詳しくは、WebSphere Application Server インフォメーション・センターの wasprofile コマンドを参照してください。

WebSphere Application Server バージョン 6.1 またはバージョン 7.0 での実行

ご使用の環境に WebSphere Application Server バージョン 6.1 またはバージョン 7.0 が含まれている場合、プロファイル管理ツール・プラグインまたは manageprofiles コマンドを使用して、プロファイルを作成および拡張することができます。

次のタスク

実行するタスクに応じて、ファースト・ステップ・コンソールを起動して、製品環境の構成とテストを支援します。ファースト・ステップ・コンソールは、<*install_root*>¥firststeps¥wxs¥firststeps.bat ディレクトリーにあります。また、前のタスクのいずれかを繰り返すことによって、追加プロファイルを作成または拡張することもできます。

関連資料

59 ページの『インストール・パラメーター』
製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

49 ページの『manageprofiles コマンド』
manageprofiles ユーティリティーで、 WebSphere eXtreme Scale テンプレートを使用してプロファイルを作成したり、 eXtreme Scale 拡張テンプレートを使用して既存のアプリケーション・サーバー・プロファイルの拡張および拡張解除を行えます。製品のこの機能を使用するには、ご使用の環境に含まれる少なくとも 1 つのプロファイルが製品用に拡張されている必要があります。

プロファイルを作成するグラフィカル・ユーザー・インターフェースの使用

プロファイル管理ツール・プラグインで提供されているグラフィカル・ユーザー・インターフェース (GUI) を使用して WebSphere eXtreme Scale のプロファイルを作成します。プロファイルは、ランタイム環境を定義するファイル・セットです。

始める前に

注: WebSphere Application Server バージョン 6.0.2 または WebSphere Application Server Network Deployment バージョン 6.0.2 を実行している場合、次の例に示すように、wasprofile コマンドを使用して WebSphere eXtreme Scale のプロファイルを作成または拡張します。

```
install_root/bin/wasprofile.sh|bat -create -profileName dmgr_01
-templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
```

詳しくは、WebSphere Application Server バージョン 6.0 インフォメーション・センターの wasprofile コマンドを参照してください。

このタスクについて

製品フィーチャーを使用するために、プロファイル管理ツール・プラグインでは、GUI を使用してプロファイル (WebSphere Application Server プロファイル、デプロイメント・マネージャーのプロファイル、セルのプロファイル、およびカスタム・プロファイルなど) のセットアップを行うことができます。

手順

プロファイル管理ツール GUI を使用してプロファイルを作成します。 ウィザードを開始するには、以下のオプションのいずれかを選択します。

- ファースト・ステップ・コンソールから「**プロファイル管理ツール**」を選択します。
- 「**スタート**」メニューからプロファイル管理ツールにアクセスします。
- *install_root*/bin/ProfileManagement ディレクトリーから ./pmt.sh|bat スクリプトを実行します。

次のタスク

追加のプロファイルを作成したり、既存のプロファイルを拡張したりできます。プロファイル管理ツールを再始動するには、*install_root*/bin/ProfileManagement ディレクトリーから ./pmt.sh|bat コマンドを実行するか、ファースト・ステップ・コンソールで「**プロファイル管理ツール**」を選択します。

ご使用の WebSphere Application Server 環境で、カタログ・サービスの開始、コンテナーの開始、および TCP ポートの構成を実行します。詳細については、369 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

関連資料

49 ページの『manageprofiles コマンド』
manageprofiles ユーティリティーで、 WebSphere eXtreme Scale テンプレートを使用してプロファイルを作成したり、 eXtreme Scale 拡張テンプレートを使用して既存のアプリケーション・サーバー・プロファイルの拡張および拡張解除を行えます。製品のこの機能を使用するには、ご使用の環境に含まれる少なくとも 1 つのプロファイルが製品用に拡張されている必要があります。

プロファイルを拡張するグラフィカル・ユーザー・インターフェースの使用

製品をインストールした後で、既存のプロファイルを拡張し、 WebSphere eXtreme Scale と互換性を持たせることができます。

始める前に

注: WebSphere Application Server バージョン 6.0.2 または WebSphere Application Server Network Deployment バージョン 6.0.2 を実行している場合、次の例に示すように、wasprofile コマンドを使用して WebSphere eXtreme Scale のプロファイルを作成または拡張します。

```
install_root/bin/wasprofile.sh|bat -augment -profileName dmgr_01
-templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
```

詳しくは、WebSphere Application Server インフォメーション・センターの wasprofile コマンドを参照してください。

このタスクについて

既存のプロファイルを拡張する場合、製品固有の拡張テンプレートを適用してプロファイルの変更をします。例えば、WebSphere eXtreme Scale サーバーは、サーバー・プロファイルが xs_augment テンプレートで拡張されていない限り、自動始動されません。

- eXtreme Scale クライアントまたは、クライアントとサーバーをインストールしている場合、xs_augment テンプレートを使用してプロファイルを拡張します。
- 区画化機能をインストールしている場合のみ、pf_augment テンプレートを使用してプロファイルを拡張します。
- ご使用の環境に eXtreme Scale クライアントと区画化機能が含まれている場合は、両方のテンプレートを適用します。

手順

プロファイル管理ツール GUI を使用して、eXtreme Scale のプロファイルを拡張します。 ウィザードを開始するには、以下のオプションのいずれかを選択します。

- ファースト・ステップ・コンソールから「**プロファイル管理ツール**」を選択します。
- 「**スタート**」メニューからプロファイル管理ツールにアクセスします。
- *install_root*/bin/ProfileManagement ディレクトリーから ./pmt.sh|bat スクリプトを実行します。

次のタスク

追加のプロファイルを拡張することもできます。プロファイル管理ツールを再始動するには、*install_root*/bin/ProfileManagement ディレクトリーから ./pmt.shlbat コマンドを実行するか、ファースト・ステップ・コンソールで「**プロファイル管理ツール**」を選択します。

ご使用の WebSphere Application Server 環境で、カタログ・サービスの開始、コンテナーの開始、および TCP ポートの構成を実行します。詳しくは、 369 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

関連資料

『manageprofiles コマンド』

manageprofiles ユーティリティーで、 WebSphere eXtreme Scale テンプレートを使用してプロファイルを作成したり、 eXtreme Scale 拡張テンプレートを使用して既存のアプリケーション・サーバー・プロファイルの拡張および拡張解除を行えます。製品のこの機能を使用するには、ご使用の環境に含まれる少なくとも 1 つのプロファイルが製品用に拡張されている必要があります。

manageprofiles コマンド

manageprofiles ユーティリティーで、 WebSphere eXtreme Scale テンプレートを使用してプロファイルを作成したり、 eXtreme Scale 拡張テンプレートを使用して既存のアプリケーション・サーバー・プロファイルの拡張および拡張解除を行えます。製品のこの機能を使用するには、ご使用の環境に含まれる少なくとも 1 つのプロファイルが製品用に拡張されている必要があります。

- プロファイルを作成および拡張する前に、 eXtreme Scale をインストールする必要があります。詳しくは、 24 ページの『WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントの WebSphere Application Server との統合』を参照してください。
- ご使用の環境に WebSphere Application Server バージョン 6.0.2 が含まれている場合、次の例に示すように、 wasprofile コマンドを使用して eXtreme Scale のプロファイルを作成したり拡張する必要があります。

  ```
  install_root/bin/wasprofile.sh|bat -augment -profileName dmgr_01
  -templatePath "C:/ProgramFiles/IBM/WebSphere/AppServer/profileTemplates/xs_augment/dmgr"
  ```

  詳しくは、 WebSphere Application Server インフォメーション・センターの wasprofile コマンドを参照してください。

目的

manageprofiles コマンドは、プロファイルと呼ばれる一連のファイルに、製品プロセスのランタイム環境を作成します。プロファイルは、ランタイム環境を定義します。manageprofiles コマンドを使用して、以下のアクションを行うことができます。

- デプロイメント・マネージャー・プロファイルの作成および拡張
- カスタム・プロファイルの作成および拡張
- スタンドアロン・アプリケーション・サーバー・プロファイルの作成および拡張
- セル・プロファイルの作成および拡張
- 任意のタイプのプロファイルの拡張解除

既存のプロファイルを拡張する場合、製品固有の拡張テンプレートを適用してプロファイルの変更をします。

- eXtreme Scale のクライアント、または、そのクライアントとサーバーの両方をインストールしている場合、xs_augment テンプレートを使用してプロファイルの拡張をします。
- 区画化機能のみをインストールしている場合は、pf_augment テンプレートを使用してプロファイルの拡張をします。
- ご使用の環境に eXtreme Scale クライアントと区画化機能が含まれている場合は、両方のテンプレートを適用します。

ロケーション

このコマンド・ファイルは、*install_root*/bin ディレクトリーにあります。

使用法

詳しいヘルプが必要な場合、以下のように **-help** パラメーターを使用してください。

```
./manageprofiles.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/dmgr -help
```

以降のセクションで、 manageprofiles コマンドを使用して実行できる各タスクを、必須パラメーターのリストと共に説明します。各タスクに指定するオプション・パラメーターに関する詳細は、WebSphere Application Server インフォメーション・センターの manageprofiles コマンドを参照してください。

デプロイメント・マネージャー・プロファイルの作成

manageprofiles コマンドを使用して、 デプロイメント・マネージャー・プロファイルを作成できます。デプロイメント・マネージャーはセルに統合されているアプリケーション・サーバーを管理します。

パラメーター

-create
プロファイルを作成します。(必須)

-templatePath *template_path*
テンプレートへのファイル・パスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/dmgr
```

ここで、*template_type* は `xs_augment` または `pf_augment` です。

例

- xs_augment テンプレートを使用する場合:

  ```
  ./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/dmgr
  ```

- pf_augment テンプレートを使用する場合:

  ```
  ./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/pf_augment/dmgr
  ```

カスタム・プロファイルの作成

manageprofiles コマンドを使用して、 はカスタム・プロファイルを作成します。カスタム・プロファイルは、アプリケーション・サーバー、クラスター、またはその他の Java プロセスを組み込むようにデプロイメント・マネージャーを介してカスタマイズする空のノードです。

パラメーター

-create
プロファイルを作成します。(必須)

-templatePath *template_path*
テンプレートへのファイル・パスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/managed
```

ここで、*template_type* は `xs_augment` または `pf_augment` です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/managed
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/pf_augment/managed
```

スタンドアロン・アプリケーション・サーバー・プロファイルの作成

manageprofiles コマンドを使用して、 はスタンドアロン・アプリケーション・サーバー・プロファイルを作成します。

パラメーター

-create
プロファイルを作成します。(必須)

-templatePath *template_path*
テンプレートへのファイル・パスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/default
```

ここで、*template_type* は `xs_augment` または `pf_augment` です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/default
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/pf_augment/default
```

セル・プロファイルの作成

manageprofiles コマンドを使用して、デプロイメント・マネージャーおよびアプリケーション・サーバーからなるセル・プロファイルを作成します。

パラメーター

デプロイメント・マネージャー・テンプレートに以下のパラメーターを指定します。

-create
プロファイルを作成します。(必須)

-templatePath *template_path*
テンプレートへのファイル・パスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/cell/dmgr
```

ここで、*template_type* は xs_augment または pf_augment です。

アプリケーション・サーバー・テンプレートを使用して以下のパラメーターを指定します。

-create
プロファイルを作成します。(必須)

-templatePath *template_path*
テンプレートへのファイル・パスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/cell/default
```

ここで、*template_type* は xs_augment または pf_augment です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/cell/dmgr
-nodeProfilePath install_root/profiles/AppSrv01 -cellName cell01dmgr -nodeName node01dmgr
-appServerNodeName node01

./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/xs_augment/cell/default
-dmgrProfilePath install_root/profiles/Dmgr01 -portsFile
install_root/profiles/Dmgr01/properties/portdef.props -nodePortsFile
install_root/profiles/Dmgr01/properties/nodeportdef.props -cellName cell01dmgr
-nodeName node01dmgr -appServerNodeName node01
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/pf_augment/cell/dmgr
-nodeProfilePath install_root/profiles/AppSrv01 -cellName cell01dmgr -nodeName node01dmgr
-appServerNodeName node01

./manageprofile.sh|bat -create -templatePath install_root/profileTemplates/pf_augment/cell/default
-dmgrProfilePath install_root/profiles/Dmgr01 -portsFile
install_root/profiles/Dmgr01/properties/portdef.props -nodePortsFile
install_root/profiles/Dmgr01/properties/nodeportdef.props -cellName cell01dmgr
-nodeName node01dmgr -appServerNodeName node01
```

デプロイメント・マネージャー・プロファイルの拡張

manageprofiles コマンドを使用して、デプロイメント・マネージャー・プロファイルを拡張します。

パラメーター

-augment
既存のプロファイルを拡張します。(必須)

-profileName
プロファイルの名前を指定します。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/dmgr
```

ここで、*template_type* は xs_augment または pf_augment です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01
-templatePath install_root/profileTemplates/xs_augment/dmgr
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01
-templatePath install_root/profileTemplates/pf_augment/dmgr
```

カスタム・プロファイルの拡張

manageprofiles コマンドを使用して、カスタム・プロファイルを拡張します。

パラメーター

-augment
既存のプロファイルを拡張します。(必須)

-profileName
プロファイルの名前を指定します。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/managed
```

ここで、*template_type* は xs_augment または pf_augment です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root
   /profileTemplates/xs_augment/managed
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root
   /profileTemplates/pf_augment/managed
```

スタンドアロン・アプリケーション・サーバー・プロファイルの拡張

manageprofiles コマンドを使用して、スタンドアロン・アプリケーション・サーバー・プロファイルを拡張します。

パラメーター

-augment
既存のプロファイルを拡張します。(必須)

-profileName
プロファイルの名前を指定します。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/default
```

ここで、*template_type* は xs_augment または pf_augment です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root
    /profileTemplates/xs_augment/default
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root
    /profileTemplates/pf_augment/default
```

セル・プロファイルの拡張

manageprofiles コマンドを使用して、セル・プロファイルを拡張します。

パラメーター

デプロイメント・マネージャー・プロファイルに以下のパラメーターを指定します。

-augment
既存のプロファイルを拡張します。(必須)

-profileName
プロファイルの名前を指定します。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/cell/dmgr
```

ここで、*template_type* は xs_augment または pf_augment です。

アプリケーション・サーバー・プロファイルに以下のパラメーターを指定します。

-augment
既存のプロファイルを拡張します。(必須)

-profileName
プロファイルの名前を指定します。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(必須)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/cell/default
```

ここで、*template_type* は `xs_augment` または `pf_augment` です。

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root
    /profileTemplates/xs_augment/cell/dmgr

./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root
    /profileTemplates/xs_augment/cell/default
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root
    /profileTemplates/pf_augment/cell/dmgr

./manageprofile.sh|bat -augment -profileName profile01 -templatePath install_root
    /profileTemplates/pf_augment/cell/default
```

プロファイルの拡張解除

プロファイルを拡張解除する場合は、必須の **-unaugment** パラメーターと **-profileName** パラメーターを指定した上で、 **-ignoreStack** パラメーターを **-templatePath** パラメーターと共に指定します。

パラメーター

-unaugment
前に拡張されたプロファイルを拡張解除します。(必須)

-profileName
プロファイルの名前を指定します。このパラメーターは、値が指定されていない場合にデフォルトで発行されます。(必須)

-templatePath *template_path*
インストール・ルート・ディレクトリーにあるテンプレート・ファイルへのパスを指定します。(オプション)

以下のフォーマット設定を使用します。

```
-templatePath install_root/profileTemplates/template_type/profile_type
```

ここで、*template_type* は `xs_augment` または `pf_augment` で、*profile_type* は次の 4 つのプロファイル・タイプのいずれかです。

- `dmgr`: デプロイメント・マネージャー・プロファイル
- `managed`: カスタム・プロファイル
- `default`: スタンドアロン・アプリケーション・サーバー・プロファイル
- `cell`: セル・プロファイル

-ignoreStack
拡張されている特定のプロファイルを拡張解除するために、**-templatePath** パラメーターとともに使用されます。(オプション)

例

- xs_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -unaugment -profileName profile01 -ignoreStack
-templatePath install_root/profileTemplates/xs_augment/profile_type
```

- pf_augment テンプレートを使用する場合:

```
./manageprofile.sh|bat -unaugment -profileName profile01 -ignoreStack
-templatePath install_root/profileTemplates/pf_augment/profile_type
```

関連タスク

45 ページの『WebSphere eXtreme Scale のプロファイルの作成および拡張』
製品のインストール後、WebSphere eXtreme Scale の固有のタイプのプロファイルを作成し、既存のプロファイルを拡張します。

46 ページの『プロファイルを作成するグラフィカル・ユーザー・インターフェースの使用』
プロファイル管理ツール・プラグインで提供されているグラフィカル・ユーザー・インターフェース (GUI) を使用して WebSphere eXtreme Scale のプロファイルを作成します。プロファイルは、ランタイム環境を定義するファイル・セットです。

48 ページの『プロファイルを拡張するグラフィカル・ユーザー・インターフェースの使用』
製品をインストールした後で、既存のプロファイルを拡張し、 WebSphere eXtreme Scale と互換性を持たせることができます。

非 root プロファイル

非 root ユーザーが製品のプロファイルを作成できるように、非 root ユーザーにファイルおよびディレクトリーへのアクセス権を付与してください。非 root ユーザーは、root ユーザー、別の非 root ユーザー、または同じ非 root ユーザーが作成したプロファイルを拡張することもできます。

WebSphere Application Server 環境では、非 root (非管理者) ユーザーは、それぞれの環境において可能なプロファイルの作成、および使用が制限されています。プロファイル管理ツール・プラグインでは、非 root ユーザーに対して固有名とポート値は使用不可になっています。非 root ユーザーは、プロファイル名、ノード名、セル名、およびポートの割り当てについて、プロファイル管理ツールのデフォルト・フィールド値を変更する必要があります。各フィールドで、非 root ユーザーに一定範囲の値を割り当てることを検討してください。非 root ユーザーに対して、適切な値の範囲を守る責任と、独自の定義の整合性を維持する責任を割り当てることができます。

用語「インストーラー」は、root ユーザーまたは非 root ユーザーのいずれかを指します。インストーラーとして、非 root ユーザーにプロファイルを作成し、独自の製品環境を確立する許可を与えることができます。例えば、非 root ユーザーが所有するプロファイルを持ったアプリケーション・デプロイメントをテストする製品環境を作成する場合があります。非 root ユーザーにプロファイルの作成を許可するために完了する具体的なタスクには、次の項目があります。

- 非 root ユーザーが特定のプロファイルの場合に WebSphere Application Server を開始できるように、プロファイルを作成し、プロファイル・ディレクトリーの所有権を非 root ユーザーに割り当てます。

- 非 root ユーザーに適切なファイルおよびディレクトリーの書き込み許可を与えます。これにより、非 root ユーザーはプロファイルを作成できるようになります。このタスクで、プロファイルの作成を許可されたユーザーのグループを作成したり、個々のユーザーがプロファイルを作成できるようにすることができます。
- 製品の保守パッケージをインストールします。これには、非ユーザーにより所有されている既存のプロファイルに必要なサービスが含まれます。インストーラーであれば、保守パッケージが作成するすべての新規ファイルの所有者です。

非 root ユーザーのプロファイルの作成について詳しくは、非 root ユーザーのプロファイルの作成を参照してください。

インストーラーとして、非 root ユーザーがプロファイルを拡張する許可を与えることもできます。例えば、非 root ユーザーはインストーラーによって作成されたプロファイルを拡張したり、作成するプロファイルを拡張したりすることができます。WebSphere Application Server Network Deployment 非 root ユーザー拡張プロセスに従ってください。

ただし、インストーラーによって作成されたプロファイルを非 root ユーザーが拡張する際に、非 root ユーザーは拡張前に以下のファイルを作成する必要はありません。 以下のファイルは、プロファイル作成プロセス中に作成済みです。

- *app_server_root*/logs/manageprofiles.xml
- *app_server_root*/properties/fsdb.xml
- *app_server_root*/properties/profileRegistry.xml

root 以外のユーザーが作成したプロファイルを自ら拡張する場合は、eXtreme Scale プロファイル・テンプレート内に位置する文書の権限を変更する必要があります。

重要: WebSphere Application Server の外側、スタンドアロン環境で WebSphere eXtreme Scale の非 root (非管理者) プロファイルを使用することもできます。ObjectGrid ディレクトリーの所有者を非 root プロファイルに変更してください。その後、その非 root プロファイルでログインして、通常の root (管理者) プロファイルの場合と同様に eXtreme Scale を操作できます。

WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントのサイレント・インストール

必要に応じて具体的に構成することができる完全修飾応答ファイルを使用するか、またはコマンド行にパラメーターを渡して WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアント をサイレント・インストールします。

始める前に

- WebSphere Application Server または WebSphere Application Server Network Deployment 環境で実行中のすべてのプロセスを停止します。stopManager、stopNode、および stopServer の各コマンドの詳細については、 コマンド行ツールの使用を参照してください。

注意:
すべての実行中のプロセスを必ず停止してください。実行中のプロセスを停止しなくてもインストールは続行しますが、一部のプラットフォームでは予測不能な結果が生じて、インストールが不確定状態になります。

- ターゲット・インストール・ディレクトリーが空であるか、存在していないことを確認します。

  重要: バージョン 7.1 のインストール先に指定したディレクトリーに、以前のバージョンの WebSphere eXtreme Scale または ObjectGrid コンポーネントが存在していると、この製品はインストールされません。例えば、既存の `<wxs_install_root>/ObjectGrid` フォルダーが存在する場合があります。他のインストール・ディレクトリーを選択するか、あるいは、インストールを取り消すことができます。次に、以前のインストールをアンインストールしてから、再度ウィザードを実行してください。

このタスクについて

サイレント・インストールは、グラフィカル・ユーザー・インターフェース (GUI) バージョンが使用するのと同じインストール・プログラムを使用します。ただし、ウィザード・インターフェースを表示する代わりに、サイレント・インストールは、カスタマイズされたファイルから、あるいはコマンド行にパスされたパラメーターからすべての応答を読み取ります。各オプションの説明が含まれている、 38 ページの『`wxssetup.response.txt` ファイル』の例を参照してください。

手順

1. オプション: 応答ファイルを使用した WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントのインストールを選択する場合には、まず `wxssetup.response.txt` ファイルをカスタマイズします。

   要確認: 完全修飾応答ファイル名を指定してください。相対パスを指定すると、エラーが発生したことをまったく示さずにインストールが失敗します。

   a. カスタマイズする応答ファイルのコピーを作成します。

      WebSphere eXtreme Scale フルインストールの場合は、応答ファイルを製品 DVD からディスク・ドライブにコピーします。

      WebSphere eXtreme Scale クライアント の場合は、WebSphere eXtreme Scale クライアント zip ファイルをハード・ディスクに解凍して、応答ファイルを見つけます。

   b. 任意のテキスト・エディターで応答ファイルを開き、編集します。 前の応答ファイルの例には、各パラメーターの指定方法の詳細が示されています。以下のパラメーターを指定する必要があります。

      - ご使用条件
      - インストール・ディレクトリー

      ヒント: WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアント を WebSphere Application Server 環境にインストールする場合、インストーラーはインストール・ディレクトリーを使用して、既存の WebSphere Application Server インスタンスがインストールされている場所を判別しま

す。複数の WebSphere Application Server インスタンスが含まれるノードにインストールする場合、そのロケーションを明確に定義してください。

c. 次のスクリプトを実行して、インストールを開始します。

WebSphere eXtreme Scale フルインストールの場合:

```
./install.sh|bat -options C:/drive_path/response_file.txt -silent
```

WebSphere eXtreme Scale クライアントのインストールの場合:

```
./WXS_Client/install.sh|bat -options C:/drive_path/response_file.txt -silent
```

GUI インストールを実行する場合にも応答ファイルを使用できます。 GUI インストールで応答ファイルを使用して、サイレント・インストールでは分からなかった問題をデバッグできます。 GUI インストールまたはサイレント・インストールで wxssetup.response ファイルを指定する際には、完全修飾パスを使用する必要があります。以下のスクリプトを実行し、応答ファイルを使用して GUI インストールを実行します。

- Linux UNIX `<install_home>/install.sh -options <full_install_path_required>/wxssetup.response`
- Windows `<install_home>¥install.exe -options c:¥<full_install_path_required>¥wxssetup.response`

2. オプション: 特定のパラメーターをコマンド行に渡すことによって eXtreme Scale をインストールする場合は、次のスクリプトを実行してインストールを開始します。

WebSphere eXtreme Scale フルインストールの場合:

```
./install.sh|bat -silent -OPT silentInstallLicenseAcceptance=true -OPT installLocation=install_location
```

WebSphere eXtreme Scale クライアントのインストールの場合:

```
./WXS_Client/install.sh|bat -silent -OPT silentInstallLicenseAcceptance=true -OPT installLocation=install_location
```

関連資料

『インストール・パラメーター』
製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

インストール・パラメーター

製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

注: 完全修飾応答ファイル名を指定してください。相対パスを指定すると、エラーが発生したことをまったく示さずにインストールが失敗します。

パラメーター

コマンド行で、または製品のオプション・ファイルのインストール中に、次のパラメーターを渡すことができます。

-silent
グラフィカル・ユーザー・インターフェース (GUI) を抑止します。**-options** パラメーターを指定して、インストーラーに、カスタマイズしたオプション・ファ

イルに従ってインストールを実行するように指示します。 **-options** パラメーターを指定しないと、代わりにデフォルト値が使用されます。

使用例

```
./install.sh|bat -silent -options options_file.txt
```

-options *path_name/file_name*
サイレント・インストールを実行するためにインストーラーが使用するオプション・ファイルを指定します。コマンド行のプロパティーが優先されます。

使用例

```
./install.sh|bat -options c:/path_name/options_file.txt
```

-log #!file_name @*event_type*
次のイベント・タイプを記録するインストール・ログ・ファイルを生成します。

- err
- wrn
- msg1
- msg2
- dbg
- ALL

使用例

```
./install.sh|bat -log # !c:/temp/logfiles.txt @ALL
```

-is:log *path_name/file_name*
GUI の始動中に、インストーラーの Java 仮想マシン (JVM) 検索を含むログ・ファイルを作成します。ログ・ファイルは、指定しないと作成されません。

使用例

```
./install.sh|bat -is:log c:/logs/javalog.txt
```

-is:javaconsole
インストール・プロセス中にコンソール・ウィンドウを表示します。

使用例

```
./install.sh|bat -is:javaconsole
```

-is:silent
インストーラーの開始時に表示される Java 初期化ウィンドウを抑止します。

使用例

```
./install.sh|bat -is:silent
```

-is:tempdir *path_name*
インストーラーがインストール時に使用する一時ディレクトリーを指定します。

使用例

```
./install.sh|bat -is:tempdir c:/temp
```

関連タスク

20 ページの『スタンドアロン WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントのインストール』
スタンドアロン WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントを、 WebSphere Application Server も WebSphere Application Server Network Deployment も含まれていない環境にインストールできます。

62 ページの『WebSphere eXtreme Scale のアンインストール』
ご使用の環境から WebSphere eXtreme Scale を削除するには、ウィザードを使用するか、または、製品をサイレント・アンインストールすることができます。

24 ページの『WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントの WebSphere Application Server との統合』
WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアント を WebSphere Application Server または WebSphere Application Server Network Deployment がインストールされている環境にインストールできます。 WebSphere Application Server または WebSphere Application Server Network Deployment の既存のフィーチャーを使用して、 eXtreme Scale アプリケーションを拡張できます。

45 ページの『WebSphere eXtreme Scale のプロファイルの作成および拡張』
製品のインストール後、WebSphere eXtreme Scale の固有のタイプのプロファイルを作成し、既存のプロファイルを拡張します。

57 ページの『WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントのサイレント・インストール』
必要に応じて具体的に構成することができる完全修飾応答ファイルを使用するか、またはコマンド行にパラメーターを渡して WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアント をサイレント・インストールします。

17 ページの『第 3 章 WebSphere eXtreme Scale のインストールおよびデプロイ』
WebSphere eXtreme Scale は、複数のサーバーにまたがるアプリケーション・データおよびビジネス・ロジックの区画化、複製、および管理を動的に行うために使用できるメモリー内のデータ・グリッドです。デプロイメントの目的および要件を決定した後に、 eXtreme Scale をシステムにインストールします。

37 ページの『CIP または IIP のサイレント・インストール』
ニーズに具体的に対応して構成する完全修飾応答ファイル、またはコマンド行に受け渡すパラメーターのいずれかを使用して、製品のカスタマイズ・インストール・パッケージ (CIP) または統合インストール・パッケージ (IIP) をサイレントにインストールすることができます。

Update Installer を使用して保守パッケージをインストールする

IBM Update Installer を使用して、ご使用の WebSphere eXtreme Scale 環境をさまざまなタイプの保守 (暫定修正、フィックスパック、リフレッシュ・パックなど) で更新します。

このタスクについて

IBM Update Installer を使用して、WebSphere eXtreme Scale のさまざまなタイプの保守パッケージをインストールして、適用します。 Update Installer は定期的に保守されるため、そのツールの最新バージョンを使用する必要があります。

手順

1. ご使用の環境で実行中のすべてのプロセスを停止します。
   - スタンドアロン eXtreme Scale 環境で実行中のすべてのプロセスを停止するには、詳細について、 363 ページの『スタンドアロン eXtreme Scale サーバーの停止』を参照してください。
   - WebSphere Application Server 環境で実行中のすべてのプロセスを停止する場合は、コマンド行ユーティリティー (Command-line utilities) を参照してください。
2. Update Installer の最新バージョンをダウンロードします。 詳しくは、推奨フィックスを参照してください。
3. Update Installer をインストールします。 詳しくは、WebSphere Application Server インフォメーション・センターの Update Installer for WebSphere Software のインストールを参照してください。
4. インストールしようとする保守パッケージを *updi_root*/maintenance ディレクトリーにダウンロードします。 詳しくは、 サポート・サイトを参照してください。
5. Update Installer を使用して、暫定修正、修正パッケージ、またはリフレッシュ・パックをインストールします。 保守パッケージのインストールは、グラフィカル・ユーザー・インターフェース (GUI) を実行するか、Update Installer をサイレント・モードで実行することで行うことができます。

   updi_root ディレクトリーから次のコマンドを実行して、GUI を開始します。

   - Linux UNIX `update.sh`
   - Windows `update.bat`

   updi_root ディレクトリーから次のコマンドを実行して、Update Installer をサイレント・モードで実行します。

   - Linux UNIX `./update.sh -silent -options responsefile/file_name`
   - Windows `update.bat -silent -options responsefile¥file_name`

   インストール・プロセスが失敗した場合、 *updi_root*/logs/update/tmp ディレクトリーにある一時ログ・ファイルを参照してください。 Update Installer は、インストール・ログ・ファイルが入れられる *install_root*/logs/update/*maintenance_package*.install ディレクトリーを作成します。

WebSphere eXtreme Scale のアンインストール

ご使用の環境から WebSphere eXtreme Scale を削除するには、ウィザードを使用するか、または、製品をサイレント・アンインストールすることができます。

始める前に

重要: アンインストーラーは、すべてのバイナリー・ファイルと、フィックスパックやインテリム・フィックスなどのすべての保守を同時に削除します。

手順

1. eXtreme Scale を実行中のすべてのプロセスを停止します。

   注意:
   すべての実行中のプロセスを必ず停止してください。実行中のプロセスを停止しなくてもアンインストールは続行しますが、一部のプラットフォームでは予測不能な結果が生じて、アンインストールが不確定状態になります。

   - スタンドアロン eXtreme Scale をインストールしている場合は、スタンドアロン・サーバーの停止を参照して、プロセスを停止します。
   - eXtreme Scale を WebSphere Application Server の既存のインストール環境にインストールしている場合は、 WebSphere Application Server プロセスの停止の詳細について、 コマンド行ユーティリティー を参照してください。

2. 製品をアンインストールします。 GUI アンインストールまたはサイレント・アンインストールを実行できます。

   注: サイレント・アンインストール/インストールまたは GUI アンインストール/インストールで応答ファイル `wxssetup.response` を指定する際には、常に完全修飾パスを指定する必要があります。 GUI アンインストールでは応答ファイルはオプションです。

   - **GUI を使用してアンインストールを実行するには、以下のようにします。**
     - Linux UNIX `<install_home>/uninstall_wxs/uninstall`
     - Windows `<install_home>¥uninstall_wxs¥uninstall.exe`

     `wxssetup.response` ファイルを使用して GUI アンインストールを実行する場合には、以下のコマンドのいずれかを使用します。

     - Linux UNIX

       ```
       <install_home>/uninstall_wxs/uninstall -options
       <full_install_path_required>/wxssetup.response
       ```

     - Windows

       ```
       <install_home>¥uninstall_wxs¥uninstall.exe -options
       <full_install_path_required>¥wxssetup.response
       ```

   - **応答ファイル `wxssetup.response` スクリプトを使用してサイレント・アンインストールを実行する場合には、以下のようにします。**
     - Linux UNIX

       ```
       <install_home>/uninstall_wxs/uninstall -options
       <full_install_path_required>/wxssetup.response -silent
       ```

     - Windows

       ```
       <install_home>¥uninstall_wxs¥uninstall.exe -options
       <full_install_path_required>¥wxssetup.response -silent
       ```

タスクの結果

これで、ご使用の環境から eXtreme Scale の削除ができました。

関連資料

59 ページの『インストール・パラメーター』
製品のインストールをカスタマイズし、構成するためのパラメーターをコマンド行に指定します。

第 4 章 WebSphere eXtreme Scale for z/OS のカスタマイズ

WebSphere カスタマイズ・ツールを使用し、カスタマイズ・ジョブを生成して実行し、 WebSphere eXtreme Scale for z/OS® をカスタマイズできます。

始める前に

- システムに最新レベルの WebSphere Application Server Network Deployment が含まれていることを確認します。
  - バージョン 6.1 を実行している場合、最小でも Fix Pack 31 がシステムに必要です。詳しくは、 バージョン 6.1 のアプリケーション・サービス提供環境のインストール を参照してください。
  - バージョン 7.0 を実行している場合、最小でも Fix Pack 9 がシステムに必要です。詳しくは、 バージョン 7.0 のアプリケーション・サービス提供環境のインストール を参照してください。
- WebSphere eXtreme Scale for z/OS をインストールします。詳しくは、Library ページの WebSphere eXtreme Scale *WebSphere eXtreme Scale Program Directory* を参照してください。

このタスクについて

WebSphere カスタマイズ・ツールを使用して、カスタマイズ定義を生成し、カスタマイズ・ジョブをアップロードして実行し、 WebSphere eXtreme Scale for z/OS をカスタマイズします。詳しくは、次のトピックを参照してください。

手順

- 『WebSphere カスタマイズ・ツールのインストール』
- 67 ページの『カスタマイズ定義の生成』
- 68 ページの『カスタマイズ・ジョブのアップロードおよび実行』

WebSphere カスタマイズ・ツールのインストール

WebSphere eXtreme Scale for z/OS 環境をカスタマイズするには、 WebSphere カスタマイズ・ツールのバージョン 7.0.0.6 以降をインストールします。

始める前に

WebSphere eXtreme Scale for z/OS をインストールします。詳しくは、Library ページの *WebSphere eXtreme Scale Program Directory* を参照してください。

このタスクについて

WebSphere カスタマイズ・ツールはワークステーション・ベースのグラフィック・ツールで、WebSphere eXtreme Scale for z/OS ランタイム環境を構築するカスタマイズ・ジョブの作成に使用します。

手順

1. FTP を使用して、xs.wct と xspf.wct の拡張ファイルを z/OS システムから WebSphere カスタマイズ・ツールをインストールするワークステーションにコピーします。 拡張ファイルは、z/OS システム上の /usr/lpp/zWebSphereXS/util/V7R1/WCT ディレクトリーにあります。
2. WebSphere カスタマイズ・ツールのバージョン 7.0.0.6 以降を該当する Web サイトからダウンロードしてインストールします。
   - Windows WebSphere Customization Tools for Windows
   - Linux WebSphere Customization Tools for Linux®
3. xs.wct ファイルを WebSphere カスタマイズ・ツール・アプリケーションにアップロードします。
   a. ワークステーションで WebSphere カスタマイズ・ツール・アプリケーションを開始します。
   b. 「**ヘルプ**」 → 「**ソフトウェアも更新**」 → 「**拡張のインストール**」をクリックします。
   c. 「WebSphere カスタマイズ・ツール拡張ロケーション」パネルから「**新規拡張ロケーションのインストール**」をクリックします。
   d. 「ソース・アーカイブ・ファイル」パネルから「**参照**」をクリックし、ステップ 1 で xs.wct ファイルをコピーしたディレクトリーにナビゲートし、「**開く**」をクリックします。
   e. 「要約」パネルで「**次へ**」をクリックします。

      注: 「インストールが正常に完了しました (Install Successful)」のパネルが表示されます。「**終了**」をクリックする前に、ロケーション・フィールドからデータをコピーして保存してください。

      ```
      C:¥Documents and Settings¥Administrator¥WCT¥workspace¥configuration¥
      com.ibm.ws.pmt.update¥com.ibm.ws390.pmt.xs_7.1.0.0¥eclipse
      ```

   f. 「製品構成 (Product Configuration)」パネルから「**拡張ロケーションの追加**」をクリックします。 前のステップでコピーしたデータを「ロケーション」フィールドに貼り付けて、「**OK**」をクリックします。
   g. 「**はい**」をクリックして WebSphere カスタマイズ・ツールを再始動します。
4. xspf.wct ファイルを WebSphere カスタマイズ・ツール・アプリケーションにアップロードします。
   a. 「**ヘルプ**」 → 「**ソフトウェアも更新**」 → 「**拡張のインストール**」をクリックします。
   b. 「WebSphere カスタマイズ・ツール拡張ロケーション」パネルから「**新規拡張ロケーションのインストール**」をクリックします。
   c. 「ソース・アーカイブ・ファイル」パネルから「**参照**」をクリックし、ステップ 1 で xspf.wct ファイルをコピーしたディレクトリーにナビゲートし、「**開く**」をクリックします。
   d. 「要約」パネルで「**次へ**」をクリックします。

      注: 「インストールが正常に完了しました (Install Successful)」のパネルが表示されます。「**終了**」をクリックする前に、ロケーション・フィールドからデータをコピーして保存してください。

```
C:¥Documents and Settings¥Administrator¥WCT¥workspace¥configuration¥
com.ibm.ws.pmt.update¥com.ibm.ws390.pmt.xs_7.1.0.0¥eclipse
```

e. 「製品構成 (Product Configuration)」パネルから「**拡張ロケーションの追加**」をクリックします。 前のステップでコピーしたデータを「ロケーション」フィールドに貼り付けて、「**OK**」をクリックします。

f. 「**はい**」をクリックして WebSphere カスタマイズ・ツールを再始動します。

次のタスク

両方の拡張ファイルをアップロードして WebSphere カスタマイズ・ツールを再始動したら、プロファイル管理ツールを使用して、 eXtreme Scale for z/OS のカスタマイズ定義を生成することができます。詳しくは、『カスタマイズ定義の生成』を参照してください。

カスタマイズ定義の生成

WebSphere カスタマイズ・ツール内のプロファイル管理ツール機能を使用して、カスタマイズ定義を生成し、 WebSphere eXtreme Scale for z/OS のカスタマイズ・ジョブを作成します。

始める前に

WebSphere カスタマイズ・ツールをインストールし、 `xs.wct` および `xspf.wct` 拡張ファイルをアップロードします。詳しくは、65 ページの『WebSphere カスタマイズ・ツールのインストール』を参照してください。

このタスクについて

WebSphere カスタマイズ・ツールで提供されるプロファイル管理ツールを使用して、カスタマイズ定義を生成できます。*カスタマイズ定義* とは、WebSphere eXtreme Scale for z/OS を構成するカスタマイズ・ジョブを作成するために使用する一連のファイルです。

手順

1. プロファイル管理ツールを開始します。

   - Windows 「**スタート**」 → 「**プログラム**」 → 「**IBM WebSphere**」 → 「**WebSphere カスタマイズ・ツール**」をクリックします。アプリケーションが開始したら、「**プロファイル管理ツール**」タブをクリックします。
   - Linux *operating_system_menus* → 「**IBM WebSphere**」 → 「**WebSphere カスタマイズ・ツール**」をクリックします。アプリケーションが開始したら、「**プロファイル管理ツール**」タブをクリックします。

2. 既存のロケーションを追加するか、作成するカスタマイズ定義のロケーションを作成します。 「**カスタマイズのロケーション (Customization Locations)**」タブで「**追加**」をクリックします。ロケーションを作成した場合、「バージョン」ボックスは、ご使用の z/OS システムにインストールされている既存の WebSphere Application Server 製品のバージョンを指します。

   注: 他の eXtreme Scale カスタマイズ定義に使用しているものと同じロケーションを使用しないでください。

3. カスタマイズ定義を生成します。 「**カスタマイズの定義**」タブで「**拡張**」をクリックします。
4. 作成する定義環境のタイプを選択します。
   - スタンドアロン・アプリケーション・サーバー・ノード
   - デプロイメント・マネージャー
   - アプリケーション・サーバー
   - 管理対象 (カスタム) ノード
5. パネルのフィールドに入力します。 z/OS システムの作成に使用されるパラメーターの値を指定します。
6. 「**拡張**」をクリックして、カスタマイズ定義を生成します。

次のタスク

ターゲットの z/OS システムにカスタマイズ・ジョブをアップロードします。詳しくは、『カスタマイズ・ジョブのアップロードおよび実行』を参照してください。

カスタマイズ・ジョブのアップロードおよび実行

カスタマイズ定義を生成したら、定義に関連付けられたカスタマイズ・ジョブを WebSphere eXtreme Scale for z/OS システムにアップロードして実行できます。

始める前に

z/OS システムにアップロードするジョブのカスタマイズ定義を生成します。詳しくは、 67 ページの『カスタマイズ定義の生成』を参照してください。

このタスクについて

WebSphere カスタマイズ・ツールを使用して作成したカスタマイズ・ジョブをアップロードして実行し、 WebSphere eXtreme Scale for z/OS 環境の管理およびモニターを行います。

手順

1. カスタマイズ・ジョブをアップロードします。 「**カスタマイズの定義**」タブで、アップロードするジョブを選択し、「**プロセス**」をクリックします。
2. z/OS システム上の FTP サーバーにジョブをアップロードします。 「**カスタマイズ定義のアップロード**」パネルに必要な情報を指定します。
3. 「**終了**」をクリックします。
4. カスタマイズ・ジョブを実行します。 「**カスタマイズの指示**」タブをクリックし、各ジョブのカスタマイズの指示に従います。

第 5 章 アプリケーション・デプロイメントの計画

WebSphere eXtreme Scale にアプリケーションをデプロイする前に、ハードウェア要件およびソフトウェア要件、ネットワーキングとチューニングの設定、デプロイメント構成などを検討する必要があります。運用チェックリストを使用して、アプリケーションをデプロイできる環境になっているかどうかを確認することもできます。

使用する WebSphere eXtreme Scale アプリケーションを設計する際に利用できるベスト・プラクティスについては、developerWorks®: ハイパフォーマンスで高い回復力を持つ WebSphere eXtreme Scale アプリケーションを作成するための原則とベスト・プラクティス (Principles and best practices for building high performing and highly resilient WebSphere eXtreme Scale application) の記事を参照してください。

アプリケーション・デプロイメントの概要

WebSphere eXtreme Scaleを実稼働環境で使用する前に、デプロイメントを最適化するための以下の問題を検討してください。

アプリケーション・デプロイメントの計画

次のリストに、検討項目を示します。

- システムおよびプロセッサーの数: 環境内には物理マシンとプロセッサーがいくつ必要ですか?
- サーバーの数: いくつの eXtreme Scale サーバーが eXtreme Scale マップをホストしますか?
- 区画の数: マップ内に保管されるデータの量は、必要な区画の数を決定する 1 つの要因です。
- レプリカの数: ドメイン内の各プライマリーに対してレプリカがいくつ必要ですか?
- 同期または非同期複製: データがきわめて重要であるため、同期複製が必要ですか? それとも、パフォーマンスに高い優先度を置くため、非同期複製が適切な選択ですか?
- ヒープ・サイズ: 各サーバーには、どれほどのデータが保管されますか?

ハードウェアおよびソフトウェアの要件

ハードウェア要件およびオペレーティング・システム要件の概要をご覧ください。WebSphere eXtreme Scale に対して使用するハードウェアまたはオペレーティング・システムのレベルについて、特定のレベルの要件はありませんが、公式にサポートされるハードウェアおよびソフトウェアのオプションに関するオペレーティング・システム別の詳しいリストを、製品サポート・サイトのシステム要件ページから入手できます。 インフォメーション・センターの情報と「システム要件」ページの情報に違いがある場合は、Web サイトの情報を優先してください。インフォメーション・センターの前提条件の情報は、便宜上提供されているだけです。

「システム要件」ページを参照してください。

ハードウェア要件

WebSphere eXtreme Scale では、ハードウェアの具体的なレベルの要件はありません。ハードウェア要件は、WebSphere eXtreme Scale を実行するのに使用される Java Platform, Standard Edition のインストール済み環境でサポートされるハードウェアによって異なります。 eXtreme Scale を WebSphere Application Server または別の Java Platform, Enterprise Edition 実装環境で使用する場合、これらのプラットフォームのハードウェア要件は WebSphere eXtreme Scale にとって十分です。

オペレーティング・システム要件

eXtreme Scale では、オペレーティング・システムの具体的なレベルの要件はありません。各 Java SE および Java EE 実装は、それぞれ異なるオペレーティング・システム・レベル、または、Java 実装のテスト中に発見された問題に対するフィックスを必要とします。これらの実装に必要なレベルは、eXtreme Scale にとって十分です。

WebSphere Application Server 要件

分散環境で稼働する eXtreme Scale のクライアントとサーバー、およびローカルのメモリー内の ObjectGrid は、WebSphere Application Server バージョン 6.0.2 以降でサポートされます。

注: 動的キャッシュ・プロバイダーを使用する場合、システムは以下の最小要件のいずれかを満たしていなければなりません。

- WebSphere Application Server バージョン 6.1.0.25 以上および暫定修正 PK85622
- WebSphere Application Server バージョン 7.0.0.3 以上および暫定修正 PK85622

詳しくは、WebSphere Application Server の推奨フィックスを参照してください。

その他のアプリケーション・サーバー要件

その他の Java EE 実装は、ローカル・インスタンスとして、または、eXtreme Scale サーバーへのクライアントとして、eXtreme Scale ランタイムを使用できます。Java SE を実装する場合は、バージョン 1.4.2 以降を使用する必要があります。

関連タスク

18 ページの『WebSphere eXtreme Scale バージョン 7.1 へのマイグレーション』
WebSphere eXtreme Scale インストーラーでは、前のインストール済み環境をアップグレードすることも変更することもできません。新しいバージョンをインストールする前に前のバージョンをアンインストールする必要があります。構成ファイルは後方互換性があるため、マイグレーションする必要はありません。ただし、製品に付属のスクリプト・ファイルのいずれかを変更した場合には、更新したスクリプト・ファイルにその変更を再適用する必要があります。

341 ページの『管理に関する用語』
WebSphere eXtreme Scale を管理するにあたっては、以下の事実を理解してください。

363 ページの『スタンドアロン eXtreme Scale サーバーの停止』
`stopOgServer` スクリプトを使用して、サーバー・プロセスを停止できます。

17 ページの『第 3 章 WebSphere eXtreme Scale のインストールおよびデプロイ』
WebSphere eXtreme Scale は、複数のサーバーにまたがるアプリケーション・データおよびビジネス・ロジックの区画化、複製、および管理を動的に行うために使用できるメモリー内のデータ・グリッドです。デプロイメントの目的および要件を決定した後に、 eXtreme Scale をシステムにインストールします。

Java Platform, Enterprise Edition の考慮事項

WebSphere eXtreme Scale を Java Platform, Enterprise Edition 環境に統合する準備をするときは、構成オプション、要件と制約、およびアプリケーションのデプロイメントと管理などを考慮します。

Java EE 環境での eXtreme Scale アプリケーションの実行

Java EE アプリケーションは、eXtreme Scale のリモート・アプリケーションに接続できます。さらに、WebSphere Application Server 環境は、アプリケーションがアプリケーション・サーバーで開始するときに eXtreme Scale サーバーの始動をサポートします。

ObjectGrid インスタンスの作成に XML ファイルを使用する場合、かつ XML ファイルがエンタープライズ・アーカイブ (EAR) ファイルのモジュール内にある場合、getClass().getClassLoader().getResource("META-INF/objGrid.xml") メソッドを使用してファイルにアクセスし、ObjectGrid インスタンスの作成に使用する URL オブジェクトを取得してください。 メソッド呼び出しで使用している XML ファイルの名前に置き換えます。

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その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・ロケーションと 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。 リモート・インターフェースには start メソッドと stop メソッドという 2 つのメソッドがあります。start メソッドを使用してインスタンスをブートストラップし、stop メソッドを使用してインスタンスを破棄します。 アプリケーションは ObjectGridManager.getObjectGrid メソッドを使用して、イ

ンスタンスへの参照を保持します。 詳しくは、「*プログラミング・ガイド*」の ObjectGridManager を使用した ObjectGrid へのアクセスに関する情報を参照してください。

クラス・ローダーの使用

別のクラス・ローダーを使用するアプリケーション・モジュールが Java EE アプリケーションの単一 ObjectGrid インスタンスを共有する場合、eXtreme Scale に保管されるオブジェクトと製品のプラグインがアプリケーションの共通ローダーにあることを確認してください。

サーブレット内の ObjectGrid インスタンスのライフサイクルの管理

サーブレットで ObjectGrid インスタンスのライフサイクルを管理するためには、init メソッドを使用してインスタンスを作成したり、destroy メソッドを使用してインスタンスを除去することができます。 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を行います。 詳しくは、「*プログラミング・ガイド*」の ObjectGridManager を使用した ObjectGrid へのアクセスに関する情報を参照してください。

キャッシング・トポロジー: メモリー内のキャッシングおよび分散キャッシング

WebSphere eXtreme Scale を使用して、アーキテクチャーはローカルのメモリー内でのデータ・キャッシング、または分散クライアント/サーバーでのデータ・キャッシングを使用できます。

WebSphere eXtreme Scale を作動させるには、最低限の追加インフラストラクチャーが必要です。インフラストラクチャーは、サーバー上で Java Platform, Enterprise Edition アプリケーションをインストール、開始、および停止するためのスクリプトで構成されます。キャッシュ・データは eXtreme Scale サーバー内に保管され、クライアントはリモート側でサーバーに接続します。

分散キャッシュは、より高いパフォーマンス、可用性、およびスケーラビリティーをもたらすもので、動的トポロジーを使用して構成できます。こうした構成では、サーバーのバランスが自動的に取られます。また、既存の eXtreme Scale サーバーを再始動せずに、別のサーバーを追加することもできます。単純なデプロイメントを作成することも、数千ものサーバーが必要になる大規模なテラバイト・サイズのデプロイメントを作成することもできます。

ローカルのメモリー内のキャッシュ

最も単純なケースでは、eXtreme Scale は、ローカルの (非分散型の) メモリー内のデータ・グリッド・キャッシュとして使用できます。 ローカルのケースは、特に複数のスレッドにより一時データにアクセスして変更する必要がある、高い並行性を持つアプリケーションで有効になります。 ローカル eXtreme Scale グリッドに保持されるデータは、索引を付け、WebSphere eXtreme Scale の照会サポートを使用して検索することができます。データ照会を可能にする機能は、Java 仮想マシン (JVM) が提供するそのままの状態で作動可能な制限付きデータ構造サポートに比べ、開発者がメモリー内の大量のデータ・セットを処理する場合に非常に役に立ちます。

eXtreme Scale でのローカルのメモリー内キャッシュ・トポロジーは、単一 Java 仮想マシン内で、一時データへの整合したトランザクション・アクセスを可能にするために使用されます。

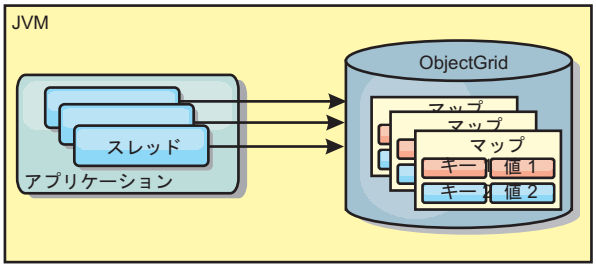


図 1. ローカルのメモリー内のキャッシュ・シナリオ

利点

- セットアップが簡単: ObjectGrid は、プログラマチックに作成することも、ObjectGrid デプロイメント記述子 XML ファイルまたは Spring などのその他のフレームワークを使用して宣言的に作成することもできます。
- 高速: 各 BackingMap は、最適のメモリー使用効率および並行性が得られるように独立して調整できます。
- 扱うデータ・セットが小さい単一 Java 仮想マシン・トポロジー、また頻繁にアクセスされるデータのキャッシングに最適。
- トランザクション型。BackingMap 更新は、単一の作業単位にまとめることができ、Java Transaction Architecture (JTA) トランザクションなどの 2 フェーズ・トランザクションの最終参加者として統合することができます。

欠点

- フォールト・トレラントでない。
- データは複製されない。メモリー内キャッシュは読み取り専用参照データに最適。
- スケーラブルでない。データベースが必要とするメモリーの量が Java 仮想マシンを圧倒するおそれがある。
- Java 仮想マシンを追加するときに、次のような問題が発生する。
  - データを簡単には区画化できない
  - Java 仮想マシン間で状態を手動で複製しなければならない。そうしないと、各キャッシュ・インスタンスが同一データの別バージョンを保持するようになります
  - 無効化にかかるコストが高い。
  - 各キャッシュは個別にウォームアップが必要になる。ウォームアップは、有効なデータがキャッシュに設定されるようにデータをロードする期間です。

使用する場合

ローカルのメモリー内キャッシュのデプロイメント・トポロジーは、キャッシュに入れるデータ量が小さく (1 つの Java 仮想マシンに収まる場合)、比較的安定している場合に限って使用するようにしてください。このアプローチの場合、不整合デ

ータの存在を許容する必要があります。Evictor を使用して、最も使用頻度が高いデータまたは最近使用されたデータをキャッシュに保持するようにすると、キャッシュ・サイズを小さく維持し、データの関連性を高くすることができます。

ピア複製されるローカルのメモリー内キャッシュ

ローカル WebSphere eXtreme Scale キャッシュでは、独立したキャッシュ・インスタンスに複数のプロセスがある場合、確実にキャッシュが同期されるようにする必要があります。 そのために、JMS を使用するピア複製キャッシュを使用可能にしてください。

WebSphere eXtreme Scale には、ピア ObjectGrid インスタンス間にトランザクション変更を自動的に伝搬する 2 つのプラグインがあります。
JMSObjectGridEventListener プラグインは、Java Messaging Service (JMS) を使用して、eXtreme Scale 変更を自動的に伝搬します。

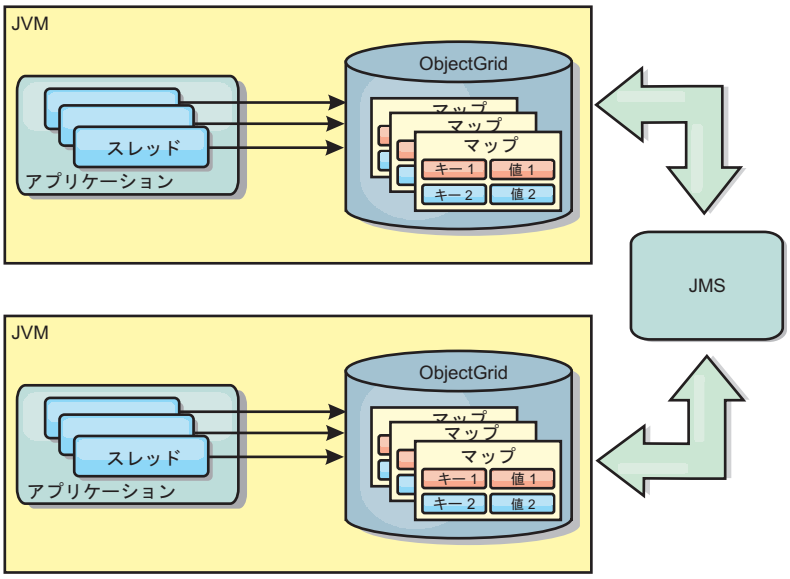


図 2. JMS によって変更が伝搬されるピア複製キャッシュ

WebSphere Application Server 環境を実行している場合は、TranPropListener プラグインも使用可能です。 TranPropListener プラグインは、高可用性 (HA) マネージャーを使用して、各ピア eXtreme Scale キャッシュ・インスタンスに変更を伝搬します。

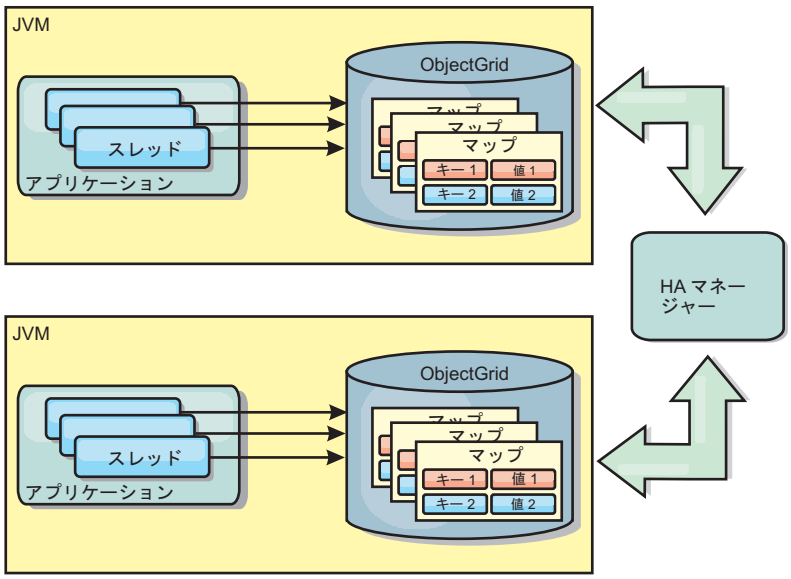


図 3. HA マネージャーによって変更が伝搬されるピア複製キャッシュ

利点

- より頻繁にデータが更新されるため、データが有効な場合が増えます。
- TranPropListener プラグインを使用すると、ローカル環境と同様、 eXtreme Scale デプロイメント記述子 XML ファイルや他のフレームワーク (Spring など) を使用して、eXtreme Scale をプログラマチックまたは宣言的に作成できます。HA マネージャーとの統合は自動的に行われます。
- 最適のメモリー使用効率および並行性が得られるように、各 BackingMap を独立して調整できます。
- BackingMap 更新は、単一の作業単位にまとめることができ、Java Transaction Architecture (JTA) トランザクションなどの 2 フェーズ・トランザクションの最終参加者として統合することができます。
- 十分小さなデータ・セットの少数 JVM トポロジー、または頻繁にアクセスされるデータのキャッシングに最適です。
- eXtreme Scale に対する変更は、すべてのピア eXtreme Scale インスタンスに複製されます。 変更は、永続サブスクリプションが使用されている限り、整合性が保たれます。

欠点

- JMSObjectGridEventListener の構成および保守は、複雑になる場合があります。eXtreme Scale は、eXtreme Scale デプロイメント記述子 XML ファイルまたは Spring などのその他のフレームワークを使用して、プログラマチックまたは宣言的に作成できます。
- スケーラブルではありません。データベースが必要とするメモリー量が、JVM の負担になる場合があります。
- Java 仮想マシンを追加する場合に不適切な機能:
  - データを簡単には区画化できない
  - 無効化にコストがかかります。
  - 各キャッシュは個別にウォームアップが必要になります。

使用する場合

このデプロイメント・トポロジーは、キャッシュに入れるデータ量が小さく (1 つの JVM に収まる)、かつ比較的安定している場合に限って使用するようにしてください。

分散キャッシュ

WebSphere eXtreme Scale は、共用キャッシュとして使用されることが最も多く、これまで使用されていたような従来のデータベースに代わり、データへのトランザクション・アクセスを複数のコンポーネントに提供します。共用キャッシュにより、データベースを構成する必要がなくなります。

すべてのクライアントがキャッシュ内の同じデータを見るので、キャッシュはコヒーレントです。各データはキャッシュ内の 1 つのサーバーのみに保管されるため、さまざまなバージョンのデータを保管することになりかねない、レコードの無駄なコピーが防止されます。コヒーレントなキャッシュは、より多くのサーバーがグリッドに追加されるにつれて、より多くのデータを保持することができ、グリッドのサイズが増えるのにつれて直線的に増加します。クライアントはこのグリッドからのデータに、リモート・プロシージャー・コールを使用してアクセスするので、このキャッシュはリモート・キャッシュ (または、ファー・キャッシュ) とも呼ばれます。データの区画化により、各プロセスは、全データ・セットの中から固有のサブセットを保持します。グリッドが大きいほどより多くのデータを保持でき、そのデータに対するより多くの要求にサービスを提供できます。コヒーレントであることによって、失効データが存在しないため、グリッドの周囲で無効化データをプッシュする必要がなくなります。コヒーレント・キャッシュは、各データの最新コピーのみを保持します。

WebSphere Application Server 環境を実行している場合は、TranPropListener プラグインも使用可能です。TranPropListener プラグインは、WebSphere Application Server 高可用性コンポーネント (HA マネージャー) を使用して、変更を各ピア ObjectGrid キャッシュ・インスタンスに伝搬します。

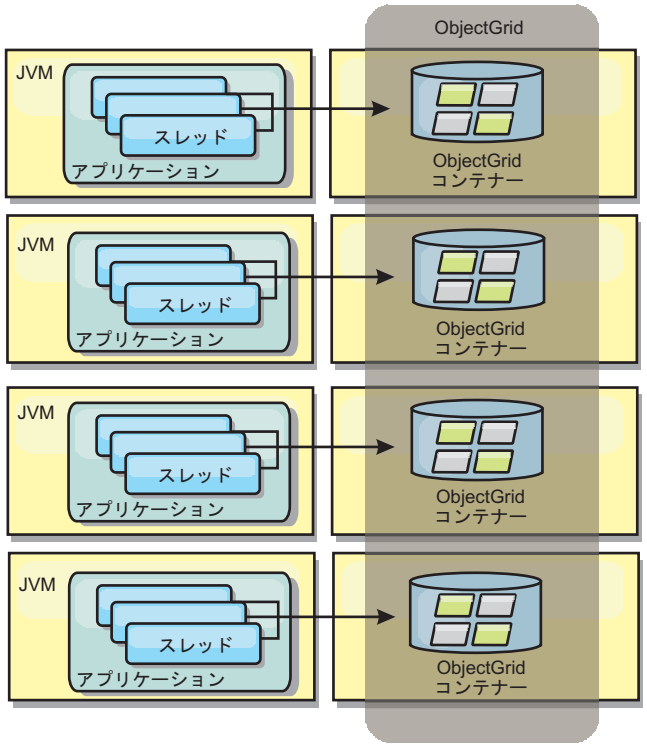


図 4. 分散キャッシュ

ニア・キャッシュ

クライアントは、eXtreme Scale が分散トポロジーで使用されている場合、オプションでローカルのインライン・キャッシュを持つことができます。オプションのこのキャッシュはニア・キャッシュと呼ばれます。これは、各クライアントにある独立した ObjectGrid であり、リモート用のキャッシュ (サーバー・サイド・キャッシュ) として機能します。ニア・キャッシュは、ロックがオプティミスティックまたはロックなしに構成されている場合、デフォルトで使用可能にされており、ロックがペシミスティックに構成されている場合は使用することができません。

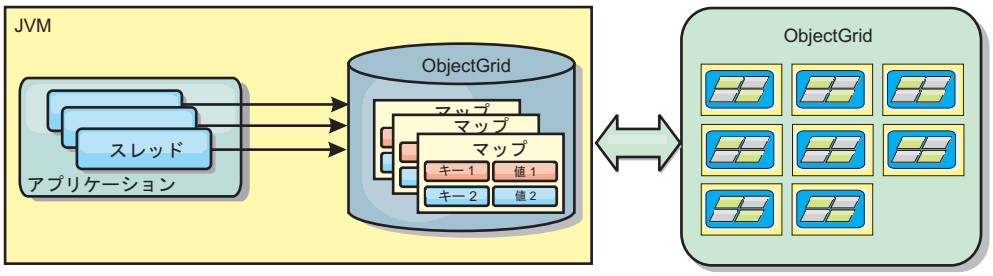


図 5. ニア・キャッシュ

ニア・キャッシュは、リモート側で eXtreme Scale サーバーに保管されているキャッシュ・データ・セット全体のサブセットへのメモリー内アクセスを可能にするため、非常に高速です。ニア・キャッシュは区画化されず、任意のリモート eXtreme Scale 区画からのデータを含みます。WebSphere eXtreme Scale は、以下のように、3 つまでのキャッシュ層を持つことができます。

1. トランザクション層キャッシュには、単一トランザクションのすべての変更が含まれます。トランザクション・キャッシュは、トランザクションがコミットされるまで、データの作業用コピーを保持します。クライアント・トランザクションが ObjectMap のデータを要求すると、最初にトランザクションがチェックされます。
2. クライアント層のニア・キャッシュは、サーバー層のデータのサブセットを保持します。トランザクション層にデータがない場合、データはニア・キャッシュにあればニア・キャッシュから取り出され、トランザクション・キャッシュに挿入されます。
3. サーバー層のグリッドには大半のデータが含まれ、すべてのクライアント間で共用されます。サーバー層は区画に分割できるので、大量のデータをキャッシュに入れることができます。クライアントのニア・キャッシュにデータが存在しないと、サーバー層からデータがフェッチされ、クライアント・キャッシュに挿入されます。サーバー層は、ローダー・プラグインを保持することもできます。グリッドに要求されたデータがない場合、Loader が呼び出され、結果のデータがバックエンドのデータ・ストアからグリッドに挿入されます。

ニア・キャッシュを使用不可にするには、クライアント・オーバーライド eXtreme Scale 記述子構成で numberOfBuckets 属性を 0 に設定します。eXtreme Scale のロック・ストラテジーについて詳しくは、マップ・エントリーのロックに関するトピックを参照してください。 ニア・キャッシュは、クライアント・オーバーライド eXtreme Scale 記述子構成を使用して、別の除去ポリシーや異なるプラグインを使用するように構成することもできます。

利点

- データへのアクセスがすべてローカルで行われるため、応答時間が速くなります。

欠点

- 失効したデータの期間が増大します。
- メモリー不足を回避するため、Evictor を使用してデータを無効化する必要があります。

使用する場合

応答時間が重要で、失効したデータは許容できる場合に使用します。

組み込みキャッシュ

eXtreme Scale グリッドは、組み込み eXtreme Scale サーバーとして既存のプロセス内で実行することも、外部プロセスとして管理することもできます。組み込みグリッドは、WebSphere Application Server などのアプリケーション・サーバー内で実行する場合に便利です。組み込まれていない eXtreme Scale サーバーは、コマンド行スクリプトを使用し、Java プロセスで実行することによって開始できます。

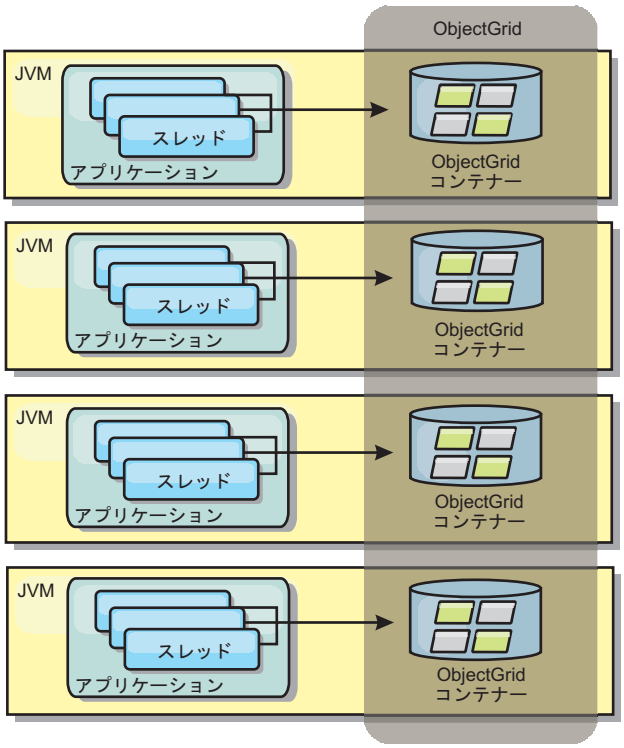


図 6. 組み込みキャッシュ

利点

- 管理するプロセスが減るため、管理が簡単になります。
- グリッドがクライアント・アプリケーションのクラス・ローダーを使用しているため、アプリケーションのデプロイメントが簡単になります。
- 区画化と高可用性をサポートします。

欠点

- すべてのデータがプロセス内に連結されるため、クライアント・プロセスのメモリー占有スペースが増えます。
- クライアント要求にサービスを提供するための CPU 使用率が高くなります。
- クライアントがサーバーと同じアプリケーション Java アーカイブ・ファイルを使用しているため、アプリケーション・アップグレードの処理がさらに難しくなります。
- 柔軟性が低くなります。クライアントとグリッド・サーバーは、同じレートで拡張することができません。サーバーを外部で定義すると、プロセス数の管理の柔軟性が増します。

使用する場合

組み込みグリッドは、クライアント・プロセスにグリッド・データおよび潜在的なフェイルオーバー・データ用の空きメモリーが豊富にある場合に使用します。

詳しくは、*管理ガイド*のクライアント無効化メカニズムの使用可能化に関するトピックを参照してください。

マルチ・マスター・グリッド複製トポロジー (AP)

マルチ・マスター非同期複製機能を使用すると、2 つ以上のグリッドを、他のグリッドの正確なミラーにすることができます。 このミラーリングは、非同期複製を使用し、グリッドをまとめて接続するリンク間で実行されます。 各グリッドは完全に独立した「ドメイン」内でホストされ、独自のカタログ・サービス、コンテナー・サーバー、および固有のドメイン・ネームを所有します。 マルチ・マスター非同期複製機能では、これらのドメインの集合を相互接続するリンクを使用し、リンク上の複製を使用してドメインを同期させることができます。eXtreme Scale では、ドメイン間のリンクの定義はユーザーに任されているため、ほとんどのトポロジーを構成できます。

7.1+ マルチ・マスター・グリッド複製は、バージョン 7.1 の重要な新機能です。

ドメイン: 固有の特性を持つグリッド

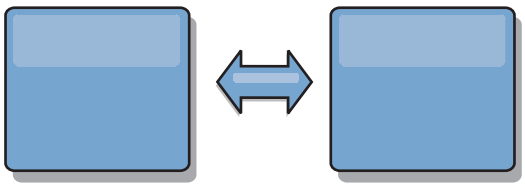
マルチ・マスター複製トポロジーで使用されるグリッドは、*ドメイン*と呼ばれます。各ドメインは、以下の特性を持つ必要があります。

- 固有のドメイン・ネームを持つ専用カタログ・サービスがある
- ドメイン内の他のグリッドと同じグリッド名である
- ドメイン内の他のグリッドと同じ数の区画がある
- FIXED_PARTITION グリッドである (PER_CONTAINER グリッドは複製不可)
-
- ドメイン内の他のグリッドと同じデータ・タイプが複製される
- ドメイン内の他のグリッドと同じマップ・セット名、マップ名、および動的マップ・テンプレートがある

トポロジーのドメインが開始された後、上記の特性を持つ任意のグリッドが複製されます。 グリッドの複製ポリシーは無視されることに注意してください。

ドメインを接続するリンク

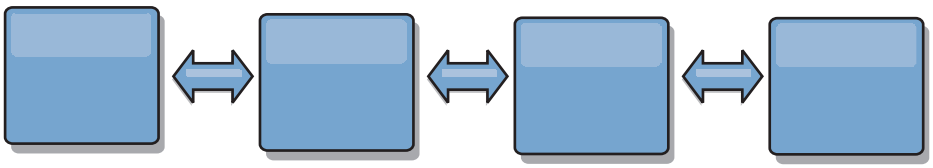
複製グリッドのインフラストラクチャーは、ドメイン間を双方向のリンクで接続したドメインのグラフです。 リンクによって、2 つのドメインはデータ変更を交換することができます。例えば、最も単純なトポロジーはドメイン間に単一のリンクを持つ 1 対のドメインです。 ドメインの名前は、左から「A」で始まって次は「B」というように付けられます。 リンクは、遠距離にわたる広域ネットワーク (WAN) を経由する場合もあります。 リンクが中断した場合は、いずれのドメインでもデータに対する変更を行うことができます。 その変更は後で、リンクがドメインを再接続すると調整されます。 ネットワーク接続が中断されると、リンクは自動的に再接続しようとします。



リンクのセットアップ後、eXtreme Scale は、まずすべてのドメインを同一にしようとします。そして、任意のドメインで変更が行われると同一状態を維持しようとします。 eXtreme Scale の目標は、各ドメインがリンクで接続されたすべての他のドメインの正確なミラーになることです。 ドメイン間の複製リンクは、1 つのドメインで行われたすべての変更を確実に他のドメインにコピーするのに役立ちます。

ライン・トポロジー

ライン・トポロジーは最も単純なトポロジーの 1 つですが、かなりのリンク品質を実証します。 まず、変更を受信するために、ドメインは直接すべての他のドメインに接続する必要はありません。 ドメイン B はドメイン A から変更をプルします。ドメイン C は、ドメイン A と C を接続するドメイン B を介してドメイン A から変更を受信します。同様に、ドメイン D はドメイン C を介して他のドメインから変更を受信します。この機能によって、変更のソースから変更を配布する負荷が分散されます。



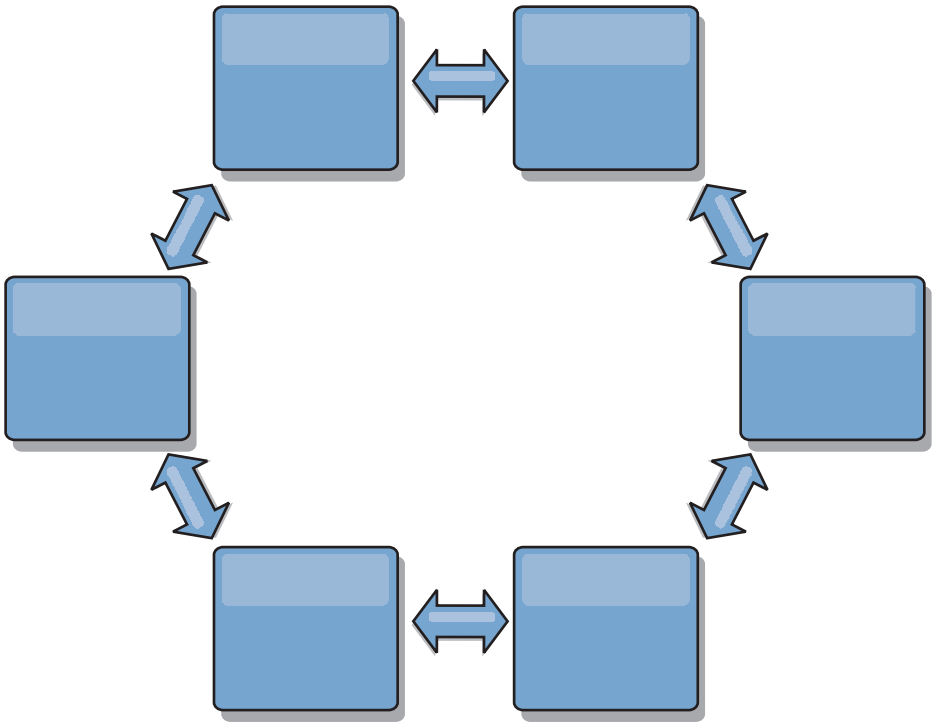
ドメイン C に障害が起こった場合、以下のイベントの発生が考えられることに注意してください。

1. ドメイン D は、ドメイン C が再開されるまで孤立します。
2. ドメイン C は、ドメイン A のコピーであるドメイン B と自分自身を同期させます。
3. ドメイン D は、ドメイン D が孤立していた間 (ドメイン C がダウンしていた間) にドメイン A と B で発生した変更を、ドメイン C を使って自分自身と同期させます。

最後に、ドメイン A、B、C、および D はすべて、他のドメインと再び同一になります。

リング・トポロジー

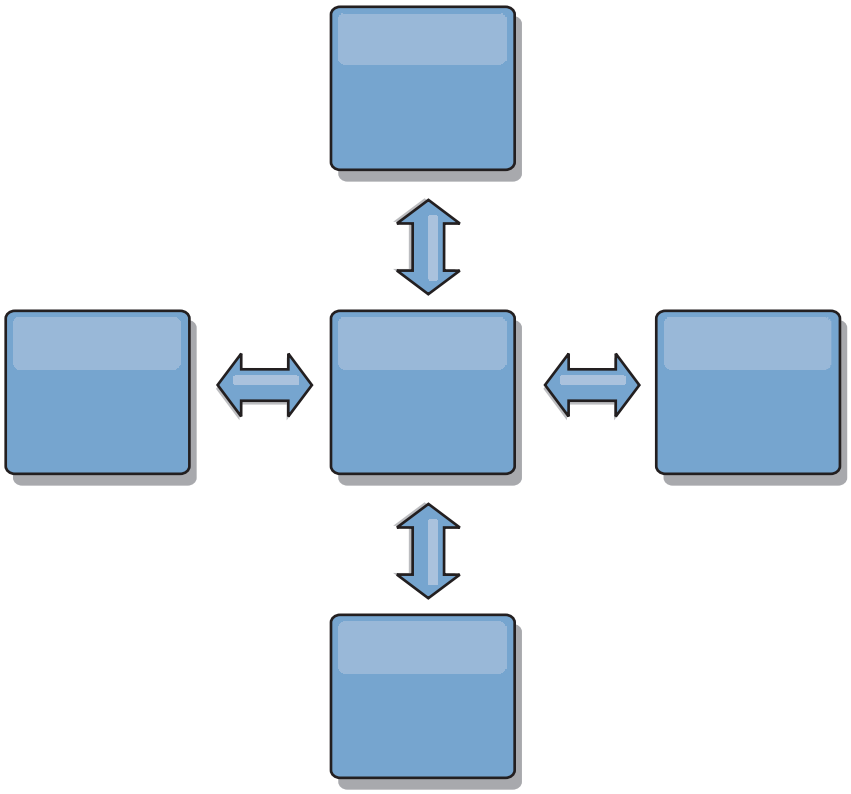
リング・トポロジーは、より回復力のあるトポロジーの例です。 1 つのドメインまたは単一リンクに障害が起こっても、残存しているドメインは、障害を避けてリング内を伝わる変更をそのまま取得できます。 各ドメインには、他のドメインへつながる 2 つのリンクがあります。 リング・トポロジーの大きさには関係なく、各ドメインは最大 2 つのリンクを持ちます。すべてのドメインがすべての変更を見るまで、特定のドメインからの変更をいくつものドメインの間で伝えなければならない場合があるため、変更を伝搬する待ち時間は長くなる可能性があります。 ライン・トポロジーにも同じ問題があります。



リングの中心に置いたルート・ドメインを使った、より洗練されたリング・トポロジーを想像してください。 ルート・ドメインは中央クリアリングハウスとして機能し、他のドメインはルート・ドメインで発生する変更に対してリモート・クリアリングハウスとして機能します。 ルート・ドメインはドメイン間の変更をアービトレーションすることができます。ルート・ドメインを囲む複数のリングがリング・トポロジーに含まれている場合、ルート・ドメインは最も内側にあるリング内のドメイン間の変更のみをアービトレーションすることができます。ただし、アービトレーションの結果は他のリングのドメインにも広がります。

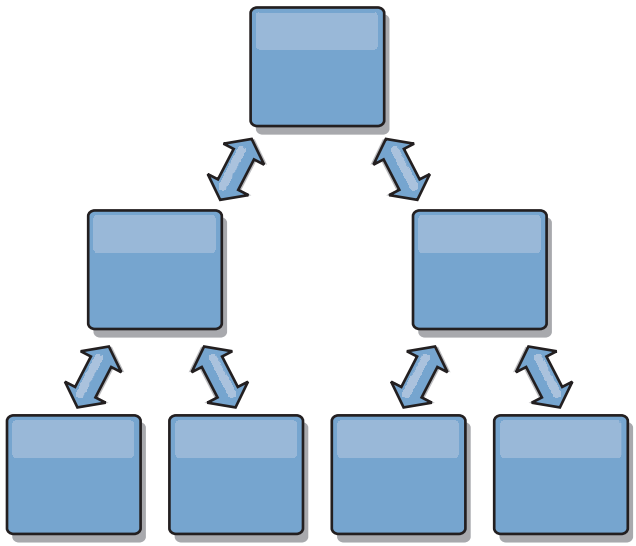
ハブ・アンド・スポーク・トポロジー

ハブ・アンド・スポーク・トポロジーでは、待ち時間が改善されます。したがって、変更は最大 1 つの中間ドメイン (ハブ) に伝わりますが、他の問題が出てきます。 このトポロジーにはハブとして機能する中央ドメインがあります。 この「ハブ・ドメイン」は、リンクを使用してすべての「スポーク・ドメイン」に接続されます。 明らかに、ドメイン間に変更を配布する負担はハブにかかります。 ハブは衝突のクリアリングハウスとして機能し、あるシナリオではセットアップが重要になってきます。 更新速度の速い環境では、ハブは、それについて行けるようにスポークよりも多くのハードウェア上で稼働する必要があります。 eXtreme Scale は、直線的に拡大するように設計されています。つまり、問題なく、必要に応じてハブをさらに大きくすることができます。 ただし、ハブに障害が起こった場合は、変更はハブが再始動するまで配布されません。 スポーク・ドメイン上の変更は、ハブが再接続された後に配布されます。



ツリー・トポロジー

最後のもう 1 つのトポロジーの例は、非循環有向ツリーです。 非循環とは、循環やループがないという意味です。有向とは、リンクが親と子の間にのみ存在するという意味です。この構成は、すべての可能なスポークに中央ハブを接続することが実用的ではないほどドメインをたくさん持つトポロジーの場合、あるいは、ルート・ドメインを更新することなく子ドメインを追加する機能を必要となるトポロジーの場合に役立ちます。



このトポロジーもルート・ドメインに中央クリアリングハウスを持つことができますが、第 2 レベルは、自分より下のドメインで発生する変更に対してリモート・ク

リアリングハウスとして機能することができます。 ルート・ドメインは、第 2 レベルにあるドメイン間の変更のみをアービトレーションすることができます。N 進ツリーも可能です。 N 進ツリーには 各レベルに N 個の子があります。 各ドメインは、N 個ずつ展開します。

トポロジー設計におけるアービトレーションの考慮事項

同じレコードが 2 個所で同時に変更される可能性がある場合には、変更の競合が生じることがあります。各ドメインが、同程度の CPU、メモリー、ネットワーク・リソースを持つようにセットアップしてください。変更の衝突処理 (アービトレーション) を実行しているドメインは、他のドメインよりも多くのリソースを使用することに気付くことがあります。 衝突は、自動的に検出されます。 衝突は、以下の 2 つのうちの 1 つのメカニズムを使って解決されます。

- **デフォルトの衝突アービター。** デフォルトのプロトコルは、字句的に最も小さい名前の付いたドメインからの変更を使用します。 例えば、ドメイン A と B によってレコードの競合が生じる場合には、ドメイン B の変更は無視されます。ドメイン A はそのバージョンを保持し、ドメイン B のレコードはドメイン A からのレコードに一致するように変更されます。これは、ユーザーやセッションが正常にバインドされているアプリケーション、またはユーザーやセッションがグリッドの 1 つにアフィニティーを持つ対象となるアプリケーションにも同様に適用されます。
- **カスタムの衝突アービター。** アプリケーションはカスタム・アービターを提供することができます。ドメインは、衝突を検出するとアービターを呼び出します。「優れた」カスタム・アービターの作成について詳しくは、マルチ・マスター複製用カスタム・アービターの作成を参照してください。

衝突が起こる可能性のあるトポロジーに対しては、ハブ・アンド・スポーク・トポロジーまたはツリー・トポロジーの使用を検討してください。 これらの 2 つのトポロジーは、以下のような場合に発生する可能性のある、エンドレスな衝突の回避につながります。

1. 複数のドメインで衝突が発生します。
2. 各ドメインが衝突をローカルで解決し、改訂を生成します。
3. 改訂が衝突し、その結果、改訂の改訂をもたらします。
4. このように、同期を取ろうとするさまざまなドメイン間に改訂が伝搬していきます。

エンドレスな衝突を回避するには、ドメインのサブセット用衝突ハンドラーとして特定のドメイン - アービトレーション・ドメイン – を選択してください。 例えば、ハブ・アンド・スポーク・トポロジーはハブを衝突ハンドラーとして使用する場合があります。 スポーク衝突ハンドラーは、スポーク・ドメインで検出された衝突を無視します。 ハブ・ドメインは改訂を作成し、制御できない衝突改訂を防ぎます。 衝突を処理するように割り当てられたドメインは、衝突の解決に責任を持つすべてのドメインにリンクしていなければなりません。 ツリー・トポロジーでは、内部の親ドメインが自分の直接の子の衝突を解決します。 対照的に、リング・トポロジーを使用すると、リング内の 1 つのドメインをリゾルバーとして指定することはできません。

次の表に、さまざまなトポロジーと互換性のあるアービトレーション・アプローチをまとめました。

表 5. アービトレーション・アプローチ： この表は、アプリケーション・アービトレーションがさまざまなトポロジーと互換性があるかどうかについて記述します。

トポロジー	アプリケーション・アービトレーションとの互換性	注
2 つのドメインのライン	あり	1 つのドメインをアービターとして選択します。
3 つのドメインのライン	あり	真ん中のドメインがアービターでなければなりません。 真ん中のドメインが、単純なハブ・アンド・スポーク・トポロジーのハブだと考えてください。
3 つより多いドメインのライン	なし	アプリケーション・アービトレーションはサポートされません。
N 個のスポークを持つハブ	あり	すべてのスポークへのリンクを持つハブがアービトレーション・ドメインでなければなりません。
N 個のドメインのリング	なし	アプリケーション・アービトレーションはサポートされません。
非循環有向ツリー (N 進ツリー)	あり	すべてのルート・ノードは、自分の直接の子孫のみをアービトレーションする必要があります。

トポロジー設計におけるリンクの考慮事項

変更待ち時間、フォールト・トレランス、およびパフォーマンス特性におけるトレードオフを最適化している間、トポロジーにはリンクの最小数が含まれているのが理想的です。

- **変更待ち時間**

  変更待ち時間は、変更が特定のドメインに到着する前に経由しなければならない中間ドメインの数によって決まります。

  トポロジーが、すべてのドメインを他のすべてのドメインにリンクすることによって中間ドメインを除去すれば、トポロジーの変更待ち時間は最善になります。ただし、ドメインはそのリンク数に比例して複製作業を実行しなければなりません。 大規模トポロジーの場合、非常に多くのリンクが定義され、管理が負担になることが考えられます。

  変更が他のドメインにコピーされる速度は、以下の追加要因によって異なります。

  – ソース・ドメイン上の CPU とネットワーク帯域幅
  – ソース・ドメインとターゲット・ドメインの間の中間ドメイン数とリンク数
  – ソース・ドメイン、ターゲット・ドメイン、および中間ドメインで使用可能な CPU とネットワーク・リソース

- **フォールト・トレランス**

フォールト・トレランスは、変更の複製のために、2 つのドメイン間に存在するパス数によって決定します。

ドメイン間に単一リンクしか存在しない場合、リンクに障害が起こると変更は伝搬されません。 1 つのドメインから別のドメインへの単一リンクが中間ドメインを経由する場合、いずれかの中間ドメインがダウンすると変更は伝搬されません。

3 つのドメイン A、B、および C を持つライン・トポロジーを考えてみます。

```
A <-> B <-> C
```

以下のいくつかの状態のままであれば、ドメイン C は A からの変更はまったく見えません。

- ドメイン A が稼働中でドメイン B がダウン
- A と B の間のリンクがダウン
- B と C の間のリンクがダウン

対照的に、リング・トポロジーでは、各ドメインはいずれかの方向から変更をプルすることができます。

```
A <-> B <-> C <-> A に戻る
```

例えば、ドメイン B がダウンしている場合、ドメイン C は引き続き変更を直接ドメイン A からプルできます。

ハブ・アンド・スポークの設計は、すべての変更がハブを介してプッシュされるので、ダウンしているハブの影響を受けやすくなります。 しかし、単一ドメインは、WAN や物理データ・センターの問題などの、コースに障害が起こる可能性のある完全なフォールト・トレラントなグリッドのままだと覚えていることは価値があります。

- **パフォーマンス**

ドメイン上に定義されるリンク数は、パフォーマンスに影響します。リンクが多いと使われるリソースも多くなり、結果的に複製のパフォーマンスが落ちる場合もあります。 他のドメインを介してドメイン A の変更をプルする機能は、そのトランザクションをどこにでも複製するドメイン A の負荷を効果的に軽減します。 *ドメイン上の変更配布の負荷は、ドメインが使用するリンクの数に制限されます。 トポロジー内にあるドメイン数には関係ありません。* このプロパティーによって、スケーラビリティーが提供され、変更配布の負荷は、単一ドメインに負荷をかけるのではなく、トポロジー内の複数のドメインによって共有することができます。

1 つのドメインが他のドメインを介して間接的に変更をプルできます。 5 つのドメインを持つライン・トポロジーを考えてみます。

```
A <=> B <=> C <=> D <=> E
```

- A は、B、C、D、および E から B を介して変更をプルします。
- B は、A と C からは直接、D と E からは C を介して変更をプルします。
- C は、B と D からは直接、A からは B を介して、E からは D を介して変更をプルします。
- D は、C と E からは直接、A と B からは C を介して変更をプルします。

– E は、D からは直接、A、B、および C からは D を介して変更をプルします。

ドメイン A および E は、それぞれ単一ドメインへのリンクのみを持っているので、配布の負荷は最も低くなります。 ドメイン B、C、および D はそれぞれ 2 つのドメインへのリンクを持っているので、ドメイン B、C、および D 上の配布の負荷は、ドメイン A および E 上の負荷の 2 倍になります。負荷は、トポロジー内の全体のドメイン数ではなく、各ドメインのリンク数によって異なるため、この負荷の分散は、ラインに 1000 ドメインを含んだとしても一定のままです。

パフォーマンスの考慮事項

マルチ・マスター複製トポロジーを使う際は、以下の制限を考慮してください。

- **変更配布の調整** (前述のとおり)
- **複製の待ち時間** (前述のとおり)
- **複製リンクのパフォーマンス** eXtreme Scale は、任意の一対の JVM 間で、単一の TCP/IP ソケットを作成します。それらの JVM 間のトラフィックはすべてそのソケット上で発生し、マルチ・マスター複製も含まれます。 ドメインは少なくとも N 個のコンテナー JVM でホストされ、少なくとも N 個の TCP リンクをピア・ドメインに提供しているため、コンテナー数をより多く持つドメインには、より高い複製のパフォーマンス・レベルがあります。 より多くのコンテナーとは、より多くの CPU とネットワーク・リソースを意味します。
- **TCP スライディング・ウィンドウのチューニングおよび RFC 1323** リンクの両端の RFC 1323 サポートを使用可能にすると、より多くのデータが往復でき、この結果、より高いスループットが実現されます。 この技法は、約 16,000 の要因でウィンドウの容量を拡張します。

  TCP ソケットが、スライディング・ウィンドウのメカニズムを使用して大量データのフローを制御することを思い出してください。これは通常、往復のインターバルのソケットを 64 KB に制限します。 往復のインターバルが 100 ミリ秒の場合、追加チューニングをすることなく帯域幅は 640 KB/秒に制限されます。 リンクで使用可能な帯域幅を完全に使用する場合は、オペレーティング・システムに固有のチューニングが必要になることがあります。 ほとんどのオペレーティング・システムにはチューニング・パラメーターがあり、高度な待ち時間リンクのスループットを向上させる RFC 1323 オプションも含まれます。

  以下の複数の要因が複製のパフォーマンスに影響する可能性があります。

  – eXtreme Scale が変更をプルする速度。
  – eXtreme Scale がプル複製要求をサービスする速度。
  – スライディング・ウィンドウの容量。
  – リンクの両端のネットワーク・バッファーをチューニングすると、eXtreme Scaleは、可能な限り速くソケット上の変更をプルできます。
- **オブジェクト・シリアライゼーション** すべてのデータはシリアライズ可能でなければなりません。 ドメインが COPY_TO_BYTES を使用していない場合、そのドメインは Java のシリアライゼーションまたは ObjectTransformers を使用してシリアライゼーション・パフォーマンスを最適化する必要があります。

- **圧縮** eXtreme Scale は、デフォルトでドメイン間で送信されるすべてのデータを圧縮します。 現行リリースにおいて、圧縮を無効にするオプションはありません。
- **メモリー・チューニング** *マルチ・マスター複製トポロジーのメモリー使用量は、トポロジー内のドメイン数とはほとんど関係ありません。*

  マルチ・マスター複製を使用可能にすると、バージョン管理を扱うマップ・エントリーごとに一定のオーバーヘッドが追加されます。 各コンテナーはトポロジー内の各ドメインの一定量のデータも追跡します。 2 つのドメインを持つトポロジーは、50 ドメインを持つトポロジーとほぼ同じメモリーを使用します。 eXtreme Scale は、その実装環境のリプレイ・ログや類似のキューを使用しません。すなわち、複製リンクがかなりの期間使用できない場合、データ構造のサイズは増大せず、リンクが再始動するときに複製が再開されるのを待ちます。

FIXED_PARTITION の複数データ・センター

現在、複数のデータ・センター間で FIXED_PARTITION グリッドを使用できます。各データ・センターには、マルチ・マスター複製用語で、独自のドメインが必要です。 各データ・センターは、ローカル・ドメインからのデータの読み取り、およびローカル・ドメインへのデータの書き込みができます。 これらの変更は、定義したリンクを使って他のデータ・センターに伝搬されます。

完全複製クライアント

このトポロジー変化には、ハブとして稼働する 1 対の eXtreme Scale サーバーが含まれます。 各クライアントは、クライアント JVM のカタログを使って、必要なものを完備した単一コンテナー・グリッドを作成します。 クライアントは、そのグリッドを使用してハブ・カタログに接続します。これにより、クライアントはハブへの接続を取得すると、すぐにハブと同期するようになります。

クライアントによって行われた変更は、クライアントに対してローカルで、非同期でハブに複製されます。 ハブはアービトレーション・ドメインとして機能し、すべての接続されたクライアントに変更を配布します。完全複製クライアントのトポロジーは、OpenJPA などのオブジェクト・リレーショナル・マッパーに適した L2 キャッシュを提供します。変更はハブを介してクライアント JVM 間に迅速に配布されます。 キャッシュ・サイズをクライアントの使用可能なヒープ・スペース内に含むことができる限り、このトポロジーは L2 のこのスタイルに適したアーキテクチャーです。

必要であれば、複数の区画を使用して、複数の JVM 上にハブ・ドメインを拡張します。 すべてのデータはまだ単一のクライアント JVM に収まらなければならないため、複数の区画を使用してハブの容量を増加させ、変更の配布とアービトレーションを行いますが、単一ドメインの容量は変更しません。

制限

マルチ・マスター複製トポロジーを使うかどうか、およびその使用方法について決定する際は、以下の制限を考慮してください。

- **複数ドメインを使ったクラス・ローダーの構成には気をつけてください**

ドメインは、キーおよび値として使用されるクラスすべてへのアクセス権限を持たなければなりません。すべての依存関係は、すべてのドメインのグリッド・コンテナー JVM に対するすべてのクラスパスに反映されなければなりません。CollisionArbiter プラグインがキャッシュ・エントリーの値を取得する場合、その値に対するクラスはアービターを呼び出すドメインに存在しなければなりません。

- **ローダーの使用はお勧めしません**

  ローダーを使用して、グリッドとデータベースの間の変更をインターフェースで連結することができます。 トポロジー内のすべてのグリッド (ドメイン) が、地理的に同じデータベースに連結されているということは考えられません。 WAN の待ち時間および他の要因で、このユース・ケースが望ましくない状態になる場合があります。

  グリッドのプリロードは、設計に慎重を要する別の問題です。 通常、グリッドは再始動すると、もう一度プリロードされます。 マルチ・マスター複製の使用時は、プリロードは必要ない、または望ましくさえありません。 ドメインは、オンラインになると、すぐに自動的に自分がリンクされているドメインの内容とともに自分自身を再ロードします。 結果的に、マルチ・マスター複製トポロジー内のドメインであるグリッドの手動プリロードを開始する必要はありません。

  ローダーは、通常、挿入規則および更新規則に従います。 マルチ・マスター複製を使用すると、挿入はマージとして扱う必要があります。 ドメインの再始動後にリモート側でデータがプルされている場合、既存データはローカル・ドメインに「挿入」されます。 このデータは既にローカル・データベースにあると考えられるため、標準的な挿入はデータベースの重複キー例外で失敗します。 代わりに、マージ・セマンティクスを使用してください。

  eXtreme Scale を構成し、Loader プラグインでプリロード・メソッドを使用して断片ベースのプリロードを行うことができます。この技法をマルチ・マスター複製トポロジーで使用しないでください。代わりに、トポロジーを始動するとき (最初に) は、クライアント・ベースのプリロードを使用します。トポロジー内の他のドメインに保管されたものの現行コピーを使用して、マルチ・マスター・トポロジーが再始動したドメインをリフレッシュできるようにします。ドメインが開始した後、マルチ・マスター・トポロジーは責任を持ってそれらのドメインを同期させます。

- **EntityManager はサポートされません**

  エンティティー・マップを含むマップ・セットは、ドメインを介して複製されません。

- **バイト配列マップはサポートされません**

  COPY_TO_BYTES で構成されたマップを含むマップ・セットは、ドメインを介して複製されません。

- **後書きはサポートされません**

  後書きサポートで構成されたマップを含むマップ・セットは、ドメインを介して複製されません。

eXtreme Scale のデプロイメント構成

WebSphere eXtreme Scale は、ローカルまたは分散の 2 つのタイプの環境にデプロイできます。必要な構成は、デプロイメント環境のタイプによって異なります。

ローカル環境

ローカル環境にデプロイすると、eXtreme Scale は単一 Java 仮想マシン内で稼働するため、複製されません。ローカル環境では、ObjectGrid XML ファイルまたは ObjectGrid API が必要です。

分散環境

分散環境にデプロイすると、eXtreme Scale は Java 仮想マシンのセット内で稼働します。この構成では、データの複製および区画化の使用が可能です。分散環境は、必要に応じてサーバーを追加できる動的デプロイメント・トポロジーにさらに分類できます。ローカル環境の場合と同じように、分散環境でも ObjectGrid XML ファイル、または同等のプログラマチック構成が必要です。さらに eXtreme Scale メモリー内のデータ・グリッドの構成詳細を持つデプロイメント・ポリシー XML ファイルを提供する必要があります。

高可用性カタログ・サービス

カタログ・サービス・ドメインは、使用しているカタログ・サーバーのグリッドであり、eXtreme Scale 環境内のすべてのコンテナーのトポロジー情報を保持します。カタログ・サービスは、すべてのクライアントの平衡化とルーティングを制御します。 eXtreme Scale をメモリー内のデータベース処理スペースとしてデプロイするには、高可用性を実現するためにカタログ・サービスをカタログ・サービス・ドメインにクラスター化する必要があります。

カタログ・サービス・ドメインのコンポーネント

複数のカタログ・サーバーが始動すると、サーバーのいずれか 1 つがマスター・カタログ・サーバーとして選択されます。マスター・カタログ・サーバーは、Internet Inter-ORB Protocol (IIOP) ハートビートを受信し、カタログ・サービスまたはコンテナーの変更に応じて、システム・データの変更を処理します。

クライアントがいずれかのカタログ・サーバーにアクセスすると、カタログ・サービス・ドメインのルーティング・テーブルは、共通オブジェクト・リクエスト・ブローカー・アーキテクチャー (CORBA) サービス・コンテキストを通してクライアントに伝搬されます。

少なくとも 3 つのカタログ・サーバーを構成します。構成がゾーンに分かれている場合、ゾーンごとに 1 つずつカタログ・サーバーを構成することができます。

eXtreme Scale サーバーおよびコンテナーが、カタログ・サーバーの 1 つにアクセスすると、カタログ・サービス・ドメインのルーティング・テーブルが、CORBA サービス・コンテキストを通して eXtreme Scale サーバーおよびコンテナーにも伝搬されます。また、アクセスされたカタログ・サーバーがその時点でマスター・カタログ・サーバーでなかった場合、要求は現行マスター・カタログ・サーバーに自動的に転送され、カタログ・サーバーのルーティング・テーブルも更新されます。

注: カタログ・サーバー・グリッドとコンテナー・サーバー・グリッドは、まったく別のものです。カタログ・サービス・ドメインは、システム・データの高可用性のためのものです。コンテナー・グリッドは、ユーザーのデータの高可用性、スケーラビリティー、およびワークロード管理を目的としています。したがって、カタログ・サービス・ドメインのルーティング・テーブルとサーバー・グリッド断片のルーティング・テーブルという 2 つの異なるルーティング・テーブルが存在します。

カタログの責務は、一連のサービスに分割されます。コア・グループ・マネージャーは、ヘルスをモニターするためのピアのグループ化を実行し、配置サービスは、割り振りを行い、管理サービスは、管理のためのアクセスを提供し、ロケーション・サービスは局所性を管理します。

カタログ・サービス・ドメインのデプロイメント

コア・グループ・マネージャー

カタログ・サービスは、高可用性マネージャー (HA マネージャー) を使用して、可用性モニタリングのためにプロセスをグループ化します。各プロセス・グループが、コア・グループです。eXtreme Scale では、コア・グループ・マネージャーが動的にプロセスをグループ化します。これらのプロセスは、スケーラビリティーのために小さく維持されます。各コア・グループはそれぞれリーダーを選出します。リーダーには、個々のメンバーに障害が発生したときに状況をコア・グループ・マネージャーに送信するという責任が追加されます。同じ状況のメカニズムは、グループのすべてのメンバーで障害が起こったときに (これにより、リーダーとの通信に障害が発生します)、それを検出するために使用されます。

コア・グループ・マネージャーは完全に自動化されたサービスです。コンテナーを少数のサーバーからなるグループに編成する責務を持ち、そのグループは自動で緩やかに統合して ObjectGrid を形成します。コンテナーは、カタログ・サービスへの初回接続時、新規または既存のグループに割り当てられるまで待機します。eXtreme Scale デプロイメントはそうした多くのグループから構成されており、このグループ化は重要なスケーラビリティー・イネーブラーです。 各グループは Java 仮想マシンで構成されるグループであり、ハートビートを使用して、他のグループの可用性をモニターします。これらのグループ・メンバーの 1 つがリーダーに選出され、リーダーには可用性情報をカタログ・サービスにリレーする責務が追加され、再割り振りとルート転送により障害に対処できるようにします。

配置サービス

カタログ・サービスは、使用可能なすべてのコンテナーでの断片の配置を管理します。物理リソース間でのリソース・バランスの維持は、配置サービスの担当です。配置サービスは、個々の断片をホスト・コンテナーに割り振る責任を担います。配置サービスは、データ・グリッド内で N 個の中から 1 つ選ばれたサービスとして実行されるため、サービスのインスタンスは 1 つだけが実行されます。 そのインスタンスに障害が起こると、別のプロセスが選出され、それが引き継ぎます。予備のために、カタログ・サービスの状態は、カタログ・サービスをホスティングするすべてのサーバーに複製されます。

管理

カタログ・サービスは、システム管理のための論理的なエントリー・ポイントでもあります。カタログ・サービスは、Managed Bean (MBean) をホストし、サービスが管理しているすべてのサーバーの Java Management Extensions (JMX) URL を提供します。

ロケーション・サービス

ロケーション・サービスは、探しているアプリケーションをホストするコンテナーを検索しているクライアント、およびホストされるアプリケーションを配置サービスに登録しようとしているコンテナーを検索しているクライアントの両方に対し、タッチ・ポイントとしての役割を果たします。ロケーション・サービスは、この機能をスケールアウトするために、すべてのグリッド・メンバーで実行されます。

カタログ・サービスは、一般的に定常状態でアイドルになるロジックをホストします。その結果として、カタログ・サービスがスケーラビリティーに与える影響はごくわずかです。サービスは、同時に使用可能になる多数のコンテナーにサービスを提供するために作成されています。可用性のために、カタログ・サービスをグリッドに構成します。

計画

カタログ・サービス・ドメインが開始されると、グリッドのメンバーは相互にバインドされます。カタログ・サービス・ドメイン構成を実行時に変更することはできないので、カタログ・サービス・ドメインのトポロジーは慎重に計画してください。エラー防止のため、グリッドはできるだけ広範囲に分散させてください。

カタログ・サービス・ドメインの開始

カタログ・サービス・ドメインの作成について詳しくは、を参照してください。

WebSphere Application Server に組み込まれている eXtreme Scale コンテナーのスタンドアロン・カタログ・サービス・ドメインへの接続

WebSphere Application Server 環境に組み込まれている eXtreme Scale コンテナーを構成して、スタンドアロン・カタログ・サービス・ドメインに接続できます。

- 7.1+ カタログ・サービス・ドメインは管理コンソールで作成できます。詳しくは、を参照してください。
-  (非推奨) 過去のリリースでは、カスタム・プロパティーを作成することによって、カタログ・サービスをカタログ・サービス・ドメインに接続しました。 このプロパティーをそのまま使用することはできますが、推奨されません。 このカスタム・プロパティーについて詳しくは、 「*管理ガイド*」にある WebSphere Application Server でのカタログ・サービス・プロセスの開始に関する情報を参照してください。

  注: サーバー名の競合: このプロパティーは、eXtreme Scale カタログ・サーバーの開始だけでなく、そこに接続する目的でも使用されるため、カタログ・サーバーの名前をどの WebSphere Application Server とも同じ名前にすることはできません。

詳しくは、*製品概要*のカタログ・サーバー・クォーラムに関する説明を参照してください。

カタログ・サーバー・クォーラム

クォーラムとは、データ・グリッドに対して配置操作を実行するために必要なカタログ・サーバーの最小数のことです。この最小数は、クォーラムの値がオーバーライドされていない限り、カタログ・サーバーのフルセットです。

重要な用語

以下は、WebSphere eXtreme Scale に関するクォーラムの考慮事項に関連する用語のリストです。

- **ブラウン・アウト:** ブラウン・アウトとは、1 つ以上のサーバー間における一時的な接続喪失のことです。
- **ブラック・アウト:** ブラック・アウトとは、1 つ以上のサーバー間における完全な接続喪失のことです。
- **データ・センター:** データ・センターとは、地理的に配置されたサーバーの一群のことで、通常はサーバー同士がローカル・エリア・ネットワーク (LAN) で接続されています。
- **ゾーン:** ゾーンとは、何らかの物理的特性を共有するサーバーを 1 つのグループにまとめるために使用される構成オプションのことです。サーバーのグループに対するゾーンの例として、上の箇条書きの中で説明したデータ・センター、エリア・ネットワーク、ビル、ビルの 1 フロアなどがあります。
- **ハートビート:** ハートビートすることは、指定された Java 仮想マシン (JVM) が稼働中かどうかを判別するために使用されるメカニズムです。

トポロジー

このセクションでは、WebSphere eXtreme Scale が信頼性の低いコンポーネントを含むネットワークでどう稼働するかについて説明します。このようなネットワークの例としては、複数のデータ・センターにまたがるネットワークがあります。

IP アドレス・スペース

WebSphere eXtreme Scale では、ネットワーク上のアドレス可能なすべての要素がネットワーク上のアドレス可能な他のすべての要素にスムーズに接続できるようなネットワークが必要となります。つまり、WebSphere eXtreme Scale は、フラット IP アドレス・ネーミング・スペースを必要とするとともに、WebSphere eXtreme Scale の要素をホスティングする Java 仮想マシン (JVM) が使用する IP アドレスとポートの間をすべてのトラフィックが流れることを許可するよう、すべてのファイアウォールに要求します。

接続された LAN

各 LAN には WebSphere eXtreme Scale 要件のゾーン ID が割り当てられます。WebSphere eXtreme Scale は単一ゾーン内の JVM を活発にハートビートします。カタログ・サービスがクォーラムを持っている場合、欠落ハートビートがあるとフェイルオーバー・イベントが発生します。

カタログ・サービス・ドメインとコンテナー・サーバー

カタログ・サービス・ドメインとは類似した JVM の集合をいいます。カタログ・サービス・ドメインはカタログ・サーバーから成るグリッドで、そのサイズは固定されています。ただし、コンテナー・サーバーの数は動的です。コンテナー・サーバーはオンデマンドで追加したり除去したりすることができます。データ・センターが 3 つある構成では、WebSphere eXtreme Scale はデータ・センターごとにカタログ・サービス JVM を 1 つずつ必要とします。

カタログ・サービス・ドメインはフル・クォーラム・メカニズムを使用します。このフル・クォーラム・メカニズムにより、データ・グリッドのすべてのメンバーが任意のアクションに合意する必要があります。

コンテナー・サーバー JVM はゾーン ID でタグ付けされます。コンテナー JVM のグリッドは JVM から成る小さいコア・グループに自動的に分割されます。 1 つのコア・グループには同じゾーンからの JVM のみが含まれます。いかなる時点でも、異なるゾーンからの JVM が同じコア・グループに存在することはありません。

コア・グループはそのメンバー JVM の障害の検出を活発に試みます。いかなる時点でも、コア・グループのコンテナー JVM は広域ネットワークと同様にリンクによって接続された複数の LAN にまたがってはなりません。つまり、コア・グループは異なるデータ・センターで実行されている同じゾーン内のコンテナーを持つことができません。

サーバー・ライフ・サイクル

カタログ・サーバーの始動

カタログ・サーバーは startOgServer コマンドを使用して始動されます。デフォルトではクォーラム・メカニズムが使用不可になっています。クォーラムを使用可能にするためには、startOgServer コマンドで `-quorum` 使用可能フラグを渡すか、または `enableQuorum=true` プロパティーをプロパティー・ファイルに追加してください。すべてのカタログ・サーバーに対して同じクォーラム設定を指定する必要があります。

```
# bin/startOgServer cat0 –serverProps objectGridServer.properties
```

objectGridServer.properties ファイル

```
catalogClusterEndPoints=cat0:cat0.domain.com:6600:6601,
cat1:cat1.domain.com:6600:6601
catalogServiceEndPoints= cat0.domain.com:2809, cat1.domain.com:2809
enableQuorum=true
```

コンテナー・サーバーの始動

コンテナー・サーバーは startOgServer コマンドを使用して始動されます。これらのサーバーは、複数のデータ・センターにまたがるデータ・グリッドを実行する際、ゾーン・タグを使用してサーバーが存在するデータ・センターを識別する必要があります。グリッド・サーバー上にゾーンを設定することにより、WebSphere eXtreme Scale はデータ・センターまでの範囲内にあるサーバーのヘルスをモニターし、データ・センター間のトラフィックを最小化することができます。

```
# bin/startOgServer gridA0 –serverProps objectGridServer.properties –
objectgridfile xml/objectgrid.xml –deploymentpolicyfile xml/
deploymentpolicy.xml
```

objectGridServer.properties ファイル

```
catalogServiceEndPoints= cat0.domain.com:2809, cat1.domain.com:2809
zoneName=ZoneA
```

グリッド・サーバーのシャットダウン

グリッド・サーバーは stopOgServer コマンドを使用して停止されます。保守のためにデータ・センター全体をシャットダウンする際は、そのゾーンに属するすべてのサーバーのリストを渡してください。そうすると、分解されているゾーンから残存しているゾーンへの滑らかな状態遷移が可能になります。

```
# bin/stopOgServer gridA0,gridA1,gridA2 –catalogServiceEndPoints
cat0.domain.com:2809,cat1.domain.com:2809
```

障害検出

WebSphere eXtreme Scale は異常ソケット閉鎖イベントを通じてプロセス・デスを検出します。プロセスが終了すると、そのことが直ちにカタログ・サービスに知らされます。ブラック・アウトは欠落ハートビートを通じて検出されます。 WebSphere eXtreme Scale は、クォーラム実装を使用することで、データ・センターでのブラウン・アウト条件から防護します。

ハートビート実装

このセクションでは、WebSphere eXtreme Scale で活性検査がどのようにして実装されるかについて説明します。

コア・グループ・メンバーのハートビート

カタログ・サービスはコンテナー JVM を限られたサイズのコア・グループに配置します。コア・グループは、2 つの方法を使用して、そのメンバーの障害の検出を試みます。 JVM のソケットが閉じられていると、その JVM はデッド状態にあると見なされます。さらに、各メンバーは構成によって決定されたペースでこれらのソケットをハートビートします。ある JVM が構成された最大時間内にこれらのハートビートに応答しないと、その JVM はデッド状態にあると見なされます。

コア・グループのメンバーの中から常に 1 つだけがリーダーに選ばれます。コア・グループ・リーダー (CGL) は、コア・グループが活動中であることをカタログ・サービスに定期的に知らせるとともに、メンバーシップの変更をカタログ・サービスに報告する必要があります。メンバーシップの変更としては、JVM の障害や新しく追加された JVM のコア・グループへの参加などが考えられます。

コア・グループ・リーダーがカタログ・サービス・ドメインのメンバーと連絡できないと、コア・グループ・リーダーは再試行し続けます。

カタログ・サービス・ドメインのハートビート

カタログ・サービス・ドメインは静的メンバーシップおよびクォーラム・メカニズムを持つ専用コア・グループに似ています。そして、通常のコア・グループと同じ

方法で障害検出を行います。ただし、その振る舞いはクォーラム・ロジックを含むように変更されています。さらに、カタログ・サービスではそれほど活発でないハートビートの構成が使用されます。

コア・グループのハートビート

カタログ・サービスは、コンテナー・サーバーに障害が発生したとき、そのことを知る必要があります。各コア・グループは、コンテナー JVM の障害を突き止め、コア・グループ・リーダーを通じてこれをカタログ・サービスに報告する役割を担っています。コア・グループの全メンバーが完全に障害を起こす可能性もあります。コア・グループ全体で障害が起こった場合は、カタログ・サービスがその障害を検出しなければなりません。

カタログ・サービスがあるコンテナー JVM を障害と判断した後で、そのコンテナーが活動中と報告された場合は、このコンテナー JVM に対して WebSphere eXtreme Scale コンテナー・サーバーをシャットダウンするよう指示が出されます。この状態の JVM は xsadmin コマンド照会では不可視です。コンテナー JVM のログのメッセージは、コンテナー JVM が失敗したことを示します。これらの JVM は、手動で再始動する必要があります。

クォーラム損失イベントが発生した場合は、クォーラムが再確立されるまでハートビートは中断されます。

カタログ・サービス・クォーラムの振る舞い

通常、カタログ・サービスのメンバーは完全な接続性を備えています。カタログ・サービス・ドメインは JVM の静的集合です。WebSphere eXtreme Scale は、カタログ・サービスのすべてのメンバーが常時オンラインであることを想定しています。カタログ・サービスは、カタログ・サービスがクォーラムを持っている間だけコンテナー・イベントに応答します。

カタログ・サービスがクォーラムを失うと、カタログ・サービスはクォーラムが再確立されるのを待ちます。カタログ・サービスは、クォーラムを持っていない間は、コンテナー・サーバーからのイベントを無視します。 WebSphere eXtreme Scale はクォーラムが再確立されることを想定しているため、コンテナー・サーバーはクォーラム不在の間にカタログ・サーバーが拒否した要求を再試行します。

次のメッセージはクォーラムが失われたことを示しています。このメッセージはご使用のカタログ・サービス・ログに入っています。

```
CWOBJ1254W: カタログ・サービスがクォーラムを待機しています。
```

WebSphere eXtreme Scale は、以下の理由によってクォーラムの損失を予想します。

- カタログ・サービス JVM メンバーの障害
- ネットワークのブラウン・アウト
- データ・センター損失

`stopOgServer` を使用してカタログ・サーバー・インスタンスを停止しても、システムはこのサーバー・インスタンスが停止したこと (これは JVM 障害やブラウン・アウトとは異なる) を知っているため、これがクォーラム損失の原因となることはありません。

JVM 障害によるクォーラム損失

障害を起こしたカタログ・サーバーはクォーラム損失の原因となります。そうなった場合は、できるだけ迅速にクォーラムがオーバーライドされるようにしてください。障害を起こしたカタログ・サービスは、クォーラムがオーバーライドされるまで、グリッドに再参加できません。

ネットワーク・ブラウン・アウトによるクォーラム損失

WebSphere eXtreme Scale はブラウン・アウトの可能性を予想できる設計になっています。ブラウン・アウトとは、データ・センター間の接続が一時的に失われた状態をいいます。これは通常、本質的に一過性のものであり、数秒あるいは数分足らずでブラウン・アウトは解消するはずです。ブラウン・アウト中、WebSphere eXtreme Scale は通常動作の維持を試みますが、ブラウン・アウトは 1 つの障害イベントと見なされます。この障害は修正されることが想定されており、WebSphere eXtreme Scale のアクションを必要とせずに通常動作が再開されます。

長時間に及ぶブラウン・アウトは、ユーザーの介入がある場合にのみブラック・アウトに分類できます。このイベントをブラック・アウトに分類するためには、ブラウン・アウトの一方の側でクォーラムをオーバーライドする必要があります。

カタログ・サービス JVM の循環

stopOgServer を使用してカタログ・サーバーが停止された場合は、クォーラムを持つサーバーが 1 つ減少します。つまり、残りのサーバーはまだクォーラムを持っています。このカタログ・サーバーを再始動すると、クォーラムは元の数に戻ります。

クォーラム損失の影響

クォーラムが失われると同時にコンテナー JVM が障害を起こした場合は、ブラウン・アウトが回復するまでリカバリーが行われないか、または (ブラック・アウトの場合) お客様がクォーラム・オーバーライド・コマンドを実行します。
WebSphere eXtreme Scale はクォーラム損失イベントとコンテナー障害を二重障害と見なしますが、これはまれにしか起こりません。 つまり、クォーラムが復元されてリカバリーが正常に行われるようになるまで、アプリケーションは障害を起こした JVM に保管されたデータへの書き込みアクセスを失う可能性があります。

同様に、クォーラム損失イベントの発生中にコンテナーを開始しようとしても、コンテナーは開始されません。

クォーラムの損失中に完全クライアント接続が許可されます。クォーラム損失イベント中にコンテナー障害も接続問題も起こらなければ、クライアントはコンテナー・サーバーと完全に対話することができます。

ブラウン・アウトが発生すると、クライアントによっては、ブラウン・アウトが解消されるまで、データのプライマリーまたは複製コピーにアクセスできない場合があります。

ブラウン・アウト・イベント中でもクライアントが少なくとも 1 つのカタログ・サービス JVM に到達できるように各データ・センターにはカタログ・サービス JVM が存在するはずなので、新規のクライアントを開始することができます。

クォーラムのリカバリー

何らかの理由によってクォーラムが失われた場合は、クォーラムが再確立される際、リカバリー・プロトコルが実行されます。クォーラム損失イベントが発生すると、コア・グループに対する活性検査はすべて中断され、障害報告も無視されます。クォーラムが元に戻ると、カタログ・サービスはすべてのコア・グループに対して活性検査を行い、直ちにそれらのメンバーシップを決定します。障害として報告されたコンテナー JVM でそれまでホストされていた断片は、いずれもこの時点でリカバリーされます。プライマリー断片が失われた場合は、残存している複製がプライマリーに昇格します。複製断片が失われた場合は、残存物の上に追加の複製が作成されます。

クォーラムのオーバーライド

これは、データ・センター障害が発生した場合にだけ使用するようにしてください。カタログ・サービス JVM の障害またはネットワークのブラウン・アウトのためにクォーラムが失われた場合は、カタログ・サービス JVM が再始動されるか、またはネットワークのブラウン・アウトが解消されると、リカバリーが自動的に行われます。

データ・センターの障害に通じているのは管理者のみです。 WebSphere eXtreme Scale はブラウン・アウトとブラック・アウトを同様に扱います。これらの障害が発生した場合は、クォーラムをオーバーライドする xsadmin コマンドを使用して eXtreme Scale 環境に通知する必要があります。そうすると、カタログ・サービスに対して、現在のメンバーシップでクォーラムが達成されて、完全リカバリーが行われると想定するように指示が出されます。クォーラム・オーバーライド・コマンドを発行したときは、障害のあるデータ・センター内の JVM が実際に障害を起こしていて、しかもリカバリーされないことを保証していることになります。

以下のリストは、クォーラムのオーバーライドに関するいくつかのシナリオを考慮したものです。 A、B、および C という 3 つのカタログ・サーバーがあるとします。

- ブラウン・アウト: C が一時的に分離されているブラウン・アウトがあるとします。カタログ・サービスはクォーラムを失い、ブラウン・アウトが解消されるのを待ちます。解消された時点で、C はカタログ・サービス・ドメインに再参加し、クォーラムが再確立されます。この間、アプリケーションはいかなる問題も検出しません。
- 一時障害: ここでは、C が障害を起こし、カタログ・サービスがクォーラムを失ったため、クォーラムをオーバーライドする必要があります。クォーラムが再確立されると、C を再始動することができます。 C は、再始動されたとき、カタログ・サービス・ドメインに再参加します。この間、アプリケーションはいかなる問題も検出しません。
- データ・センター障害: データ・センターが実際に障害を起こし、しかもネットワーク上で分離されていることを確認します。次に、xsadmin クォーラム・オーバーライド・コマンドを発行します。そうすると、残存している 2 つのデータ・

センターが、障害を起こしたデータ・センターでホストされていた断片を置き換えることによって完全リカバリーを実行します。カタログ・サービスは現在、A および B のフル・クォーラムで実行しています。アプリケーションは、ブラック・アウトが始まってからクォーラムがオーバーライドされるまでの間に、遅延や例外を検出することがあります。クォーラムがオーバーライドされると、グリッドがリカバリーされ、通常動作が再開されます。

- データ・センター・リカバリー: 残存しているデータ・センターは、オーバーライドされたクォーラムで既に実行されています。 C を含むデータ・センターが再始動されたならば、そのデータ・センターにあるすべての JVM も再始動する必要があります。そうすると、C は既存のカタログ・サービス・ドメインに再参加し、クォーラムはユーザーの介入なしで通常状態に戻ります。
- データ・センターの障害およびブラウン・アウト: C を含むデータ・センターが障害を起こしました。残りのデータ・センターでは、クォーラムがオーバーライドされてリカバリーされます。 A と B の間でブラウン・アウトが発生した場合は、通常のブラウン・アウト・リカバリー・ルールが適用されます。ブラウン・アウトが解消されると、クォーラムが再確立され、クォーラム損失からの必要なリカバリーが実行されます。

コンテナーの振る舞い

このセクションでは、クォーラムが失われてからリカバリーされるまでの間、コンテナー・サーバー JVM がどのように振る舞うかについて説明します。

コンテナーは 1 つ以上の断片をホストします。断片は、特定の区画のプライマリーであるか複製であるか、そのいずれかです。カタログ・サービスが断片をコンテナーに割り当てると、カタログ・サービスから新しい指示が送られてくるまで、コンテナーはこの割り当てに従います。つまり、ブラウン・アウトのためにコンテナー内のプライマリー断片が複製断片と通信できない場合は、カタログ・サービスから新しい指示を受け取るまで、コンテナーは再試行し続けます。

ネットワーク・ブラウン・アウトが発生し、プライマリー断片が複製との通信を失うと、カタログ・サービスが新しい指示を出すまでコンテナーは接続を再試行します。

同期複製の振る舞い

接続が中断されている間でも、マップ・セットの minsync プロパティーと少なくとも同数の複製がオンラインである限り、プライマリーは新しいトランザクションを受け入れることができます。同期複製へのリンクが中断されているときにプライマリーで新しいトランザクションが処理された場合は、リンクが再確立された時点で、複製はクリアされ、プライマリーの現在の状態と再同期されます。

データ・センター間や WAN スタイルのリンクに対しては、同期複製を決して推奨しません。

非同期複製の振る舞い

接続が中断されている間でも、プライマリーは新しいトランザクションを受け入れることができます。プライマリーは変更内容を限界までバッファーに入れます。この限界に達する前に複製との接続が再確立された場合は、バッファーに入れられた

変更内容を使用して複製が更新されます。限界に達すると、プライマリーはバッファーに入れられたリストを破棄します。そして複製が再接続されると、複製はクリアされて再同期されます。

クライアントの振る舞い

カタログ・サービス・ドメインがクォーラムを持っていてもいなくても、常時、クライアントはカタログ・サーバーに接続して、グリッドをブートストラップすることができます。クライアントは、経路テーブルを取得した上でグリッドと対話するために、カタログ・サーバー・インスタンスへの接続を試みます。ネットワークの接続性により、ネットワーク・セットアップが原因でクライアントが一部の区画と対話できなくなる場合があります。クライアントはローカル複製に接続してリモート・データを取得できますが、その場合はクライアントがそうするように構成されていなければなりません。クライアントは、該当するプライマリー区画が使用可能でない場合、データを更新できません。

xsadmin を含むクォーラム・コマンド

このセクションでは、クォーラムのさまざまな状況に役立つ xsadmin コマンドについて説明します。

クォーラム状況の照会

カタログ・サーバー・インスタンスのクォーラム状況は、xsadmin コマンドを使用して問い合わせることができます。

```
xsadmin –ch cathost –p 1099 –quorumstatus
```

起こり得る結果は 5 つあります。

- クォーラムが使用不可である: カタログ・サーバーがクォーラム使用不可モードで実行されています。これは開発モードまたは単一専用データ・センター・モードです。複数データ・センター構成の場合は、これをお勧めしません。
- クォーラムが使用可能で、かつカタログ・サーバーがクォーラムを持っている: クォーラムが使用可能で、しかもシステムが正常に機能しています。
- クォーラムは使用可能だが、カタログ・サーバーがクォーラムを待機している: クォーラムが使用可能で、かつクォーラムが失われています。
- クォーラムが使用可能で、かつクォーラムがオーバーライドされている: クォーラムが使用可能で、しかもクォーラムがオーバーライドされました。
- クォーラム状況が禁止である: ブラウン・アウトが発生したとき、カタログ・サーバーが 2 つの区画 (A および B) に分割され、カタログ・サーバー A のクォーラムがオーバーライドされました。ネットワーク区画は分解され、B 区画にあるサーバーが禁止されているため、JVM の再始動が必要です。この状況は、ブラウン・アウト中に B にあるカタログ JVM が再始動されてから、ブラウン・アウトが解消された場合にも起こります。

クォーラムのオーバーライド

xsadmin コマンドを使用してクォーラムをオーバーライドすることができます。残存しているカタログ・サーバー・インスタンスを使用することができます。ある残

存物に対してクォーラムをオーバーライドする指示が出されると、それがすべての残存物に通知されます。これを行うための構文は次のとおりです。

```
xsadmin –ch cathost –p 1099 –overridequorum
```

診断コマンド

- クォーラム状況: 前のセクションで説明したとおりです。
- コア・グループ・リスト: これはすべてのコア・グループのリストを表示します。コア・グループのメンバーとリーダーが表示されます。

  ```
  xsadmin –ch cathost –p 1099 –coregroups
  ```
- サーバーの分解: このコマンドはグリッドからサーバーを手動で除去します。障害サーバーとして検出されたサーバーは自動的に除去されるので、これは通常不要ですが、このコマンドは IBM サポート・ヘルプのもとで使用するために提供されています。

  ```
  xsadmin –ch cathost –p 1099 –g Grid –teardown server1,server2,server3
  ```
- 経路テーブルの表示: このコマンドは、グリッドへの新しいクライアント接続をシミュレートすることによって、現在の経路テーブルを表示します。また、すべてのコンテナー・サーバーが経路テーブル内でのそれぞれのロール (どの区画のどのタイプの断片であるかなど) を認識していることを確認することによって、経路テーブルの妥当性検査も行います。

  ```
  xsadmin –ch cathost –p 1099 –g myGrid -routetable
  ```
- 未割り当て断片の表示: グリッドに配置できない断片がある場合は、これを使用してそれらの断片をリストすることができます。これは、配置サービスに配置を妨げる制約があるときのみ現れます。例えば、実動モードのとき、1 つの物理ボックスで JVM を始動した場合は、プライマリー断片のみを配置できます。 2 つ目のボックスで JVM が始動されるまで、複製は未割り当てとなります。配置サービスでは、プライマリー断片をホストしている JVM とは異なる IP アドレスを持つ JVM にのみ複製が配置されます。ゾーンに JVM が存在しない場合も、断片が未割り当てとなることがあります。

  ```
  xsadmin –ch cathost –p 1099 –g myGrid –unassigned
  ```
- トレースの設定: このコマンドは、xsadmin コマンドに対して指定されたフィルターと一致するすべての JVM について、トレースを設定します。この設定によって、別のコマンドが使用されるか、修正された JVM が障害を起こすか、または停止されるまでの間に、トレース設定のみが変更されます。

  ```
  xsadmin –ch cathost –p 1099 –g myGrid –fh host1 –settracespec
  ObjectGrid*=event=enabled
  ```

  これにより、指定されたホスト名 (この場合は host1) を持つボックスにあるすべての JVM に対して、トレースが使用可能となります。
- マップ・サイズの検査: マップ・サイズ・コマンドは、キー分布がキー内の断片に均等であることを確認するために役立ちます。他のコンテナーより著しく多いキーを持っているコンテナーがある場合は、キー・オブジェクトに対するハッシュ関数の分布が不完全である可能性があります。

```
xsadmin -ch cathost -p 1099 -g myGrid -m myMapSet -mapsizes myMap
```

トランスポート・セキュリティーの考慮事項

データ・センターは、通常、地理的に異なるロケーションにデプロイされるため、ユーザーは、安全上の理由からデータ・センター間のトランスポート・セキュリティーを使用可能にしたい場合があります。

「*管理ガイド*」のトランスポート層セキュリティーについて参照してください。

運用チェックリスト

この運用チェックリストを使用して、WebSphere eXtreme Scale のデプロイ用に環境を準備してください。

表 6. 運用チェックリスト

チェックリスト項目	詳細情報

AIX® を使用している場合、以下のオペレーティング・システムの設定を調整してください。
 	
TCP_KEEPINTVL
TCP_KEEPINTVL 設定は、ネットワーク障害の検出を可能にする、ソケットのキープアライブ・プロトコルの一部です。このプロパティーは、接続を検証するために送信されるパケット間の間隔を指定します。 WebSphere eXtreme Scale を指定している場合、この値は 10 に設定します。現行設定を確認するには、次のコマンドを実行します。
`# no –o tcp_keepintvl`
現行設定を変更するには、次のコマンドを実行します。
`# no –o tcp_keepintvl=10`
TCP_KEEPINTVL 設定は、0.5 秒単位で設定します。
 	
TCP_KEEPINIT
TCP_KEEPINIT 設定は、ネットワーク障害の検出を可能にする、ソケットのキープアライブ・プロトコルの一部です。このプロパティーは、TCP 接続の初期タイムアウト値を指定します。 WebSphere eXtreme Scale を使用している場合、この値は 40 に設定します。現行設定を確認するには、次のコマンドを実行します。
`# no –o tcp_keepinit`
現行設定を変更するには、次のコマンドを実行します。
`# no –o tcp_keepinit=40`
TCP_KEEPINIT 設定は、0.5 秒単位で設定します。
	• AIX のシステム調整について詳しくは、 AIX システムの調整を参照してください。

orb.properties ファイルを更新すると、グリッドのトランスポート動作を変更できます。 orb.properties ファイルは、java/jre/lib ディレクトリーにあります。	206 ページの『ORB プロパティー・ファイル』

表 6. 運用チェックリスト (続き)

チェックリスト項目	詳細情報
startOgServer スクリプトのパラメーターを使用します。特に、以下のパラメーターを使用します。 • **-jvmArgs** パラメーターを使用してヒープ設定を設定します。 • **-jvmArgs** パラメーターを使用してアプリケーション・クラスパスとプロパティーを設定します。 • エージェント・モニターの構成用に **-jvmArgs** パラメーターを設定します。 **ポートの設定** WebSphere eXtreme Scale は、一部のトランスポートの通信用にポートを開く必要があります。これらのポートはすべて動的に定義されます。ただし、コンテナー間のファイアウォールが使用中の場合はポートを指定する必要があります。ポートに関する以下の情報を使用してください。 **リスナー・ポート** プロセス間の通信に使用するポートを指定する場合、**-listenerPort** 引数を使用できます。 **コア・グループ・ポート** 障害検出に使用するポートを指定する場合、**-haManagerPort** 引数を使用できます。この引数は、peerPort と同じです。コア・グループは、ゾーンをまたいで通信する必要がないことに注意してください。したがって、ファイアウォールが単一ゾーンのすべてのメンバーに対してオープンである場合は、このポートを設定する必要はありません。 **JMX サービス・ポート** JMX サービスが使用するポートを指定する場合、**-JMXServicePort** 引数を使用できます。 **SSL ポート** `-Dcom.ibm.CSI.SSLPort=1234` を **-jvmArgs** 引数として引き渡すと、SSL ポートが `1234` に設定されます。この SSL ポートは、リスナー・ポートと対等のセキュア・ポートです。 **クライアント・ポート** カタログ・サービスのみで使用されます。この値は、**-catalogServiceEndPoints** 引数を使用して指定できます。このパラメーターの値は次の形式になります。 `serverName:hostName:clientPort:peerPort`	360 ページの『startOgServer スクリプト』
次のセキュリティー設定が正しく構成されていることを検証します。 • トランスポート (SSL) • アプリケーション (認証と許可) セキュリティー設定を検証するには、悪意のあるクライアントを使用してご使用の構成に接続を試みてください。例えば、SSL が必要な設定が構成されている場合に、TCP_IP 設定があるクライアント、あるいは、正しくないトラストストアのクライアントが、サーバーに接続できてはいけません。認証が必要な場合に、クレデンシャル (例えば、ユーザー ID とパスワードなど) を持たないクライアントは、サーバーに接続できてはいけません。許可が実行されている場合に、アクセス許可を持たないクライアントは、サーバー・リソースへのアクセスを認可されるべきではありません。	397 ページの『外部プロバイダーとのセキュリティー統合』

表 6. 運用チェックリスト (続き)

チェックリスト項目	詳細情報
ご使用の環境をモニターする方法を選択します。 • xsAdmin – カタログ・サーバーの JMX ポートが、XSAdmin ツールから表示可能になっている必要があります。コンテナー・ポートも、コンテナーからの情報を収集する一部のコマンドにとってアクセス可能である必要があります。 • 以下のベンダー・モニター・ツールから選択できます。 – Tivoli® Enterprise Monitoring Agent – CA Wily Introscope – Hyperic HQ	• 413 ページの『xsAdmin サンプル・ユーティリティーによるモニター』 • 394 ページの『Java Management Extensions (JMX) セキュリティー』 • 438 ページの『IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale のモニター』 • 449 ページの『Hyperic HQ による eXtreme Scale のモニター』 • 445 ページの『CA Wily Introscope による eXtreme Scale アプリケーションのモニター』

第 6 章 デプロイメント環境の構成

スタンドアロン環境で実行する場合は WebSphere eXtreme Scale を構成し、WebSphere Application Server または WebSphere Application Server Network Deployment を使用した環境で実行する場合は、eXtreme Scale を構成します。eXtreme Scale のデプロイメントでサーバー・グリッド・サイドの構成変更を反映させるには、動的な適用ではなく、プロセスを再始動して変更を有効にする必要があります。一方、クライアント・サイドでは、既存のクライアント・インスタンスの構成設定は変更できませんが、 XML ファイルを使用するか、またはプログラムによって、新しいクライアントを必要な設定で作成できます。クライアントの作成時には、現行のサーバー構成からのデフォルト設定をオーバーライド可能です。

構成方式

XML ファイルおよびプロパティー・ファイルは、製品を構成するための最も一般的な、プログラムによらない方式です。アプリケーションおよびシステムのプログラミング・インターフェース、プラグイン、管理対象 Bean についても含め、代替方式については、プログラミング・ガイドを参照してください。

XML ファイルを使用した構成

WebSphere eXtreme Scale は、XML ファイルのコレクションにより構成されます。

XML ファイルを使用して、eXtreme Scale 用に以下の構成を作成することができます。

- デプロイメント・ポリシー - デプロイメント・ポリシーを構成するには、デプロイメント・ポリシー記述子 XML ファイルを使用します。さまざまな要求に対して記述子 XML ファイルのエレメントおよび属性を定義するには、デプロイメント・ポリシー記述子 XML ファイルを参照してください。
- ObjectGrid 記述子 - 個々の ObjectGrid インスタンスの詳細を構成するには、記述子 XML ファイルを使用します。詳しくは、 151 ページの『ObjectGrid 記述子 XML ファイル』を参照してください。
- エンティティーの構成 - 論理スキーマで定義された、デプロイメント内のエンティティーに対するニーズに基づき、アノテーション付き Java クラス、XML、またはそれらの組み合わせを使用して、エンティティーを構成することができます。定義されたエンティティーは、eXtreme Scale サーバーに登録され、BackingMap、索引、およびその他のプラグインにバインドされます。エンティティー・スキーマとは、1 組のエンティティーとそれらエンティティーの間のリレーションシップのことです。構成オプションを用いたエンティティー・スキーマの最適化について詳細は、 229 ページの『エンティティーの構成』 を参照してください。
- セキュリティーの構成 - XML 構成ファイルを使用して、特定のデプロイメントに対してセキュリティーを使用可能にすることができます。セキュリティーの使用可能化の構成オプションについて詳細は、 402 ページの『セキュリティー記述子 XML ファイル』 を参照してください。

- クライアントの構成 - プロパティー・ファイルとその他のメソッドを使用して、クライアントのホスト名、ポート、セキュリティーなどの情報を指定します。XML ファイルを使用したクライアントのカスタマイズについて詳細は、215 ページの『クライアントの構成』 を参照してください。
- Spring 統合の構成 - Spring Framework を使用可能にし、XML スクリプトおよびスキーマ定義を拡張 Bean や名前空間サポートなどの他のメソッドと併用して eXtreme Scale デプロイメント環境で作業できるようにします。Spring Framework を用いた eXtreme Scale の使用について詳しくは、293 ページの『Spring 統合の構成』 を参照してください。

XML 構成のトラブルシューティング

eXtreme Scale の構成時に、XML 構成で予想外の動作が発生する場合があります。

以下のセクションに、発生する可能性のあるさまざまな XML 構成上の問題を示します。

デプロイメント・ポリシーと ObjectGrid XML ファイルの不一致

デプロイメント・ポリシーと ObjectGrid XML ファイルは一致している必要があります。一致する ObjectGrid 名およびマップ名がない場合には、エラーが発生します。

正しくない BackingMap およびマップ参照

ObjectGrid XML ファイル内の backingMap リストがデプロイメント・ポリシー XML ファイル内のマップ参照リストと一致しない場合は、カタログ・サーバーでエラーが発生します。

例えば、以下の ObjectGrid XML ファイルおよびデプロイメント・ポリシー XML ファイルはコンテナー・プロセスを開始する場合に使用されます。デプロイメント・ポリシー・ファイルには、ObjectGrid XML ファイルでリストされているマップ参照よりも多くのマップ参照があります。

ObjectGrid.xml - incorrect example

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

deploymentPolicy.xml - incorrect example

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
    <objectgridDeployment objectgridName="accounting">
        <mapSet name="mapSet1" numberOfPartitions="4" minSyncReplicas="1"
     maxSyncReplicas="2" maxAsyncReplicas="1">
            <map ref="payroll"/>
            <map ref="ledger"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

メッセージ

デプロイメント・ポリシーが ObjectGrid XML ファイルと非互換である場合、SystemOut.log ファイル内にエラー・メッセージが発生します。上記の例では、以下のようなメッセージが発生します。

```
CWOBJ3179E: ObjectGrid アカウンティング・デプロイメント記述子ファイルの mapSet mapSet1
内にあるマップ元帳参照が、ObjectGrid XML の有効なバックアップ・マップを参照していません。
```

デプロイメント・ポリシーで、ObjectGrid XML ファイル内にリストされた backingMap へのマップ参照が欠落している場合、SystemOut.log ファイル内にエラー・メッセージが発生します。以下に例を示します。

```
CWOBJ3178E: ObjectGrid XML 内で参照された ObjectGrid アカウンティングのマップ元帳が
デプロイメント記述子ファイル内で見つかりませんでした。
```

問題

ObjectGrid XML ファイル の backingMap リストとデプロイメント・ポリシーのマップ参照は一致している必要があります。

解決策

使用中の環境に対してどのリストが正しいかを決定し、正しくないリストが含まれている XML ファイルを訂正します。

正しくない ObjectGrid 名

ObjectGrid の名前は ObjectGrid XML ファイルおよびデプロイメント・ポリシー XML ファイルの両方で参照されます。

メッセージ

IncompatibleDeploymentPolicyException の例外が原因となって、ObjectGridException が発生します。以下に例を示します。

原因: com.ibm.websphere.objectgrid.IncompatibleDeploymentPolicyException: objectGridName が accountin の objectgridDeployment には、ObjectGrid XML 内に対応する objectGrid がありません。

問題

ObjectGrid XML ファイルは ObjectGrid 名のマスター・リストです。デプロイメント・ポリシーに ObjectGrid XML ファイルに含まれていない ObjectGrid 名がある場合、エラーが発生します。

解決策

ObjectGrid 名のスペルを検証します。ObjectGrid XML ファイルまたはデプロイメント・ポリシー XML ファイルに対し、余分な名前を除去するか、または欠落している ObjectGrid 名を追加します。このメッセージ例では、objectGridName が「accounting」のところが、ミススペルのため「accountin」になっています。

属性の XML 値が無効

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。これらの属性には、スキーマによって列挙された許容値が含まれています。以下のリストには、属性の一部が挙げられています。

- 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
- objectGrid エレメントの initialState
- backingMap エレメントの evictionTriggers

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。以下の XML ファイルの例では、正しくない値 INVALID_COPY_MODE が使用されています。

INVALID_COPY_MODE example

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

次のメッセージがログに表示されます。

```
CWOBJ2403E: The XML file is invalid. A problem has been detected
with < null > at line 5. The error message is 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, COPY_TO_BYTES]'.
It must be a value from the enumeration.
```

属性またはタグの欠落

XML ファイルで正しい属性またはタグが欠落している場合には、エラーが発生します。例えば、以下の ObjectGrid XML ファイルでは閉じる < /objectGrid > タグが欠落しています。

missing attributes - example 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="accounting">
            <backingMap name="payroll" />

   </objectGrids>
</objectGridConfig>
```

次のメッセージがログに表示されます。

```
CWOBJ2403E: The XML file is invalid. A problem has been detected with
< null > at line 7. The error message is The end-tag for element type "objectGrid"
must end with a '>' delimiter.
```

無効な XML ファイルに関する ObjectGridException が、XML ファイルの名前で発生します。

構文エラー

XML ファイルが、正しくない構文または欠落している構文でフォーマット済みである場合、CWOBJ2403E がログに表示されます。例えば、XML 属性の 1 つで引用符が欠落している場合、以下のメッセージが表示されます。

```
CWOBJ2403E: The XML file is invalid. A problem has been detected with
< null > at line 7. The error message is Open quote is expected for attribute
"maxSyncReplicas" associated with an element type "mapSet".
```

また、無効な XML ファイルに関する ObjectGridException も発生します。

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 と一致している必要があります。

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は、以下のメッセージまたは同様の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'.
```

問題

以下の XML ファイルを使用すると、エラーが生じます。BackingMap の名前 book 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。

referencing a non-existent attribute XML - example

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

解決策

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。このエラーが発生しないように、pluginCollectionRef の名前を collection1 に変更するだけです。あるいは、既存の backingMapPluginCollection の ID を pluginCollectionRef に一致するように変更するか、または pluginCollectionRef に一致する ID を持つ追加の backingMapPluginCollection を追加してエラーを訂正します。

実装のサポートなしの 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いくつかの Java API for XML Processing (JAXP) 機能の実装を含みます。

この実装を含まない 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。この実装を含まない SDK を使用して XML の妥当性検査を行う場合は、Apache Xerces をダウンロードして、その Java アーカイブ (JAR) ファイルをクラスパスに組み込みます。

必要な実装を持たない 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。

```
XmlConfigBuild XML validation is enabled
SystemErr R com.ibm.websphere.objectgrid
SystemErr R at com.ibm.ws.objectgrid.ObjectGridManagerImpl.getObjectGridConfigurations
 (ObjectGridManagerImpl.java:182)
SystemErr R at com.ibm.ws.objectgrid.ObjectGridManagerImpl.createObjectGrid(ObjectGridManagerImpl.java:309)
SystemErr R at com.ibm.ws.objectgrid.test.config.DocTest.main(DocTest.java:128)
SystemErr R Caused by: java.lang.IllegalArgumentException: No attributes are implemented
SystemErr R at org.apache.crimson.jaxp.DocumentBuilderFactoryImpl.setAttribute(DocumentBuilderFactoryImpl.java:93)
SystemErr R at com.ibm.ws.objectgrid.config.XmlConfigBuilder.<init>XmlConfigBuilder.java:133)
SystemErr R at com.ibm.websphere.objectgrid.ProcessConfigXML$2.runProcessConfigXML.java:99)...
```

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 機能の実装を含んでいません。

この問題を回避するために、Xerces をダウンロードして JAR ファイル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。

プロパティー・ファイルの解説

サーバー・プロパティー・ファイルには、カタログ・サーバーとコンテナー・サーバーを実行するための設定が含まれています。サーバー・プロパティー・ファイルは、スタンドアロンに対して、あるいは WebSphere Application Server 構成に対して 1 つ指定することができます。クライアント・プロパティー・ファイルには、クライアントの設定が含まれます。

サンプル・プロパティー・ファイル

extremescale_root¥properties ディレクトリーにある以下のサンプル・プロパティー・ファイルを使用して、プロパティー・ファイルを作成することができます。

- sampleServer.properties
- sampleClient.properties

非推奨システム・プロパティー

-Dcom.ibm.websphere.objectgrid.CatalogServerProperties
このプロパティーは、WebSphere eXtreme Scale バージョン 7.0 で使用すべきではありません。**-Dobjectgrid.server.props** プロパティーを使用してください。

-Dcom.ibm.websphere.objectgrid.ClientProperties
このプロパティーは、WebSphere eXtreme Scale バージョン 7.0 で使用すべきではありません。**-Dobjectgrid.client.props** プロパティーを使用してください。

-Dobjectgrid.security.server.prop
このプロパティーは、WebSphere eXtreme Scaleバージョン 6.1.0.3 で使用すべきではありません。 **-Dobjectgrid.server.prop** プロパティーを使用してください。

-serverSecurityFile
この引数は、WebSphere eXtreme Scale バージョン 6.1.0.3 で使用すべきではありません。このオプションは、 `startOgServer` スクリプトに渡されます。 **-serverProps** 引数を使用してください。

グリッドの構成

ObjectGrid 記述子 XML ファイルを使用して、グリッド、バックアップ・マップ、プラグインなどを構成します。WebSphere® eXtreme Scale を構成するには、ObjectGrid ディスクリプター XML ファイルおよび ObjectGrid API を使用します。分散トポロジーには、ObjectGrid 記述子 XML ファイルだけでなくデプロイメント・ポリシー XML ファイルも必要となります。

ローカル・デプロイメントの構成

ローカルのメモリー内 eXtreme Scale 構成は、ObjectGrid 記述子 XML ファイルまたは eXtreme Scale API を使用して作成できます。

このタスクについて

以下の `companyGrid.xml` ファイルは、ObjectGrid 記述子 XML の例です。ファイルの最初の数行には、各 ObjectGrid XML ファイルの必須ヘッダーが含まれています。このファイルは、ObjectGrid インスタンス（「CompanyGrid」）および複数の BackingMap（「Customer」、「Item」、「OrderLine」、および「Order」）を定義しています。

companyGrid.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="CompanyGrid">
   <backingMap name="Customer" />
   <backingMap name="Item" />
   <backingMap name="OrderLine" />
   <backingMap name="Order" />
```

```
  </objectGrid>
 </objectGrids>

</objectGridConfig>
```

ObjectGridManager インターフェース内で createObjectGrid メソッドのうちの 1 つに XML ファイルを受け渡します。以下のコード・サンプルは、XML スキーマに対して companyGrid.xml ファイルを妥当性検査し、「CompanyGrid」という ObjectGrid インスタンスを作成しています。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 companyGrid = objectGridManager.createObjectGrid("CompanyGrid",
 new URL("file:etc/test/companyGrid.xml"), true, false);
```

または、XML を使用せずに、プログラマチックに ObjectGrid インスタンスを作成することもできます。例えば、前の XML およびコードの代わりに、以下のコード・スニペットを使用できます。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid ("CompanyGrid", false);
BackingMap customerMap= companyGrid.defineMap("Customer");
BackingMap itemMap= companyGrid.defineMap("Item");
BackingMap orderLineMap= companyGrid.defineMap("OrderLine");
BackingMap orderMap = companyGrid.defineMap("Order");
```

ObjectGrid XML ファイルについて詳しくは、eXtreme Scale 構成の解説書を参照してください。

HashIndex の構成

HashIndex (組み込み com.ibm.websphere.objectgrid.plugins.index.HashIndex クラス) は、静的索引または動的索引を作成するために BackingMap に追加可能な MapIndexPlugin プラグインです。 HashIndex は MapIndex インターフェースと MapRangeIndex インターフェースの両方をサポートします。 索引を適切に定義および使用すると、照会のパフォーマンスをかなり改善できます。

索引付けについては、索引付けおよび複合 HashIndex を参照してください。索引付けの使用方法について詳しくは、非キー・データ・アクセスでの索引付けの使用および複合 HashIndex を参照してください。

HashIndex を構成するための属性

ObjectGrid デプロイメント記述子 XML ファイルを使用して、またはプログラマチックに HashIndex を構成するには、以下の属性が使用できます。

Name 索引の名前を指定します。名前は各マップで固有でなければなりません。この名前は、BackingMap の ObjectMap インスタンスから索引オブジェクトを取り出すのに使用されます。

AttributeName
索引に対する属性の名前をコンマで区切ったリストを指定します。フィールド・アクセス索引の場合、属性名はフィールド名と同じです。プロパティー・アクセス索引の場合、属性名は JavaBean 互換のプロパティー名です。HashIndex は、属性名が 1 つだけであれば単一属性索引であり、その属性がリレーションシップの場合はリレーションシップ索引でもあります。複数の属性名が含まれている場合は、HashIndex は複合索引です。

FieldAccessAttribute

非エンティティー・マップに使用されます。true の場合、フィールドを使用してオブジェクトに直接アクセスします。指定されていないか、false の場合、データのアクセスには、属性の getter メソッドが使用されます。

POJOKeyIndex

非エンティティー・マップに使用されます。true の場合、索引はマップのキー部分でオブジェクトをイントロスペクトします。これは、キーが複合キーで、値にキーが組み込まれていない場合に役立ちます。指定されていないか、false の場合、索引はマップの値部分でオブジェクトをイントロスペクトします。

RangeIndex

true の場合、範囲索引付けが使用可能にされ、アプリケーションは取り出された索引オブジェクトを MapRangeIndex インターフェースにキャストできます。RangeIndex プロパティーが false と構成されている場合は、アプリケーションは取り出された索引オブジェクトを MapIndex インターフェースにしかキャストできません。

BackingMap への HashIndex の追加

HashIndex を BackingMap に追加するために使用できる方法はあまりありません。以下の例で、静的索引プラグインの追加による XML 構成アプローチを示します。

HashIndex を BackingMap に追加するための XML 構成アプローチ

```
<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>
```

この XML 構成例では、組み込み HashIndex クラスが索引プラグインとして使用されています。HashIndex は、ユーザーが構成できるプロパティーをサポートしています。上の例にある Name、RangeIndex、 AttributeName などです。

- Name プロパティーは、この索引プラグインを識別するストリングである CODE と構成されています。Name プロパティー値は、BackingMap の有効範囲内で固有でなければならず、BackingMap の ObjectMap インスタンスから名前で索引オブジェクトを取り出すのに使用できます。
- RangeIndex プロパティーは true と構成されています。これが意味するのは、取り出された索引オブジェクトをアプリケーションが MapRangeIndex インターフェースにキャストできるということです。RangeIndex プロパティーが false と構成されている場合は、アプリケーションは取り出された索引オブジェクトを MapIndex インターフェースにしかキャストできません。MapRangeIndex は、範囲関数 greater than や less than、あるいは両方を使用するデータ検出をサポートしますが、MapIndex は equals 関数のみをサポートします。索引が照会によって使用される場合、RangeIndex プロパティーは、単一属性索引に対して true と構

成されていなければなりません。リレーションシップ索引および複合索引に対しては、RangeIndex プロパティーは `false` と構成される必要があります。

- AttributeName プロパティーは `employeeCode` と構成されています。これは、キャッシュに入れられたオブジェクトの employeeCode 属性を使用して、単一属性索引が構築されることを意味しています。複数の属性を持つ、キャッシュに入れられたオブジェクトをアプリケーションが検索する必要がある場合、AttributeName プロパティーには、属性をコンマで区切ったリストを設定でき、そうすると複合索引が生成されます。

つまり、上記の例では、単一属性範囲 HashIndex を定義しています。 これは、単一属性 HashIndex です。 また、範囲 HashIndex でもあります。

単一属性 HashIndex と複合 HashIndex

HashIndex の AttributeName プロパティーに複数の属性名が含まれている場合、HashIndex は複合索引です。そうではなく、含まれている属性名が 1 つのみの場合は、単一属性索引です。例えば、AttributeName プロパティー値が `city,state,zipcode` であるような、複合 HashIndex が考えられます。この例では、3 つの属性がコンマで区切られています。もし AttributeName プロパティー値が単に `zipcode` であれば、属性は 1 つだけなので、単一属性 HashIndex であるということになります。前記の例は、AttributeName プロパティーの値が 1 つの属性名だけを含む「employeeCode」なので、単一属性 HashIndex です。

複合 HashIndex は、検索条件に多くの属性が関係するような場合に、キャッシュに入れられたオブジェクトを検索する効果的な方法を提供します。ただし、範囲索引はサポートしないため、 RangeIndex プロパティーは「false」に設定されている必要があります。

管理ガイド の複合 HashIndex に関するトピックを参照してください。

リレーションシップ HashIndex

単一属性 HashIndex の索引属性が、単一値または複数値のリレーションシップの場合、その HashIndex はリレーションシップ HashIndex です。リレーションシップ HashIndex の場合、 HashIndex の RangeIndex プロパティーは「false」に設定されている必要があります。

リレーションシップ HashIndex は、循環参照を使用する照会や、IS NULL、IS EMPTY、SIZE、および MEMBER OF 照会フィルターを使用する照会を速くすることができます。「*プログラミング・ガイド*」で、索引を使用する照会最適化についての説明を参照してください。

キー HashIndex

非エンティティー・マップの場合、HashIndex の POJOKeyIndex プロパティーが `true` に設定されていると、HashIndex はキー HashIndex であり、エントリーのキー部分が索引付けに使用されます。 HashIndex の AttributeName プロパティーが指定されていないと、キー全体に索引が付けられます。指定されていると、キー HashIndex は単一属性 HashIndex にしかなりません。

例えば、以下のプロパティーを前記の例に追加すると、POJOKeyIndex プロパティーの値が true のため、HashIndex はキー HashIndex になります。

```
<property name="POJOKeyIndex" type="boolean" value="true"
description="indicates if POJO key HashIndex" />
```

前記のキー索引の例では、AttributeName プロパティーの値が employeeCode として指定されているため、索引付き属性はマップ・エントリーのキー部分の employeeCode フィールドです。 キー索引をマップ・エントリーのキー部分全体で作成する場合は、AttributeName プロパティーを除去してください。

範囲 HashIndex

HashIndex の RangeIndex プロパティーが true に設定されている場合、その HashIndex は範囲索引であり、MapRangeIndex インターフェースをサポートできます。MapRangeIndex は、範囲関数 greater than や less than、あるいは両方を使用するデータ検出をサポートしますが、MapIndex は equals 関数のみをサポートします。単一属性索引の場合、RangeIndex プロパティーを true に設定できるのは、索引付けられる属性のタイプが Comparable の場合に限ります。単一属性索引が照会によって使用される場合、RangeIndex プロパティーは true に設定されていなければならず、索引付けられる属性のタイプは Comparable でなければなりません。リレーションシップ HashIndex および複合 HashIndex の場合、RangeIndex プロパティーは false に設定されていなければなりません。

RangeIndex プロパティーの値が true なので、前記の例は範囲 HashIndex です。

以下の表に、範囲索引の使用についての要約を示します。

表 7. 範囲索引のサポート： HashIndex のタイプが範囲索引をサポートするかどうかを記述します。

HashIndex タイプ	**範囲索引のサポート**
単一属性 HashIndex: 索引付けられるキーまたは属性のタイプは Comparable である	あり
単一属性 HashIndex: 索引付けられるキーまたは属性のタイプは Comparable でない	なし
複合 HashIndex	なし
リレーションシップ HashIndex	なし

HashIndex を使用した照会の最適化

索引を適切に定義および使用すると、照会のパフォーマンスをかなり改善できます。WebSphere eXtreme Scale 照会は、組み込み HashIndex プラグインを使用して、照会のパフォーマンスを向上させることができます。索引の使用は、照会パフォーマンスを大きく向上させることができますが、トランザクションのマップ操作のパフォーマンスに影響を与えることもあります。

Evictor の構成

Evictor の構成は、ObjectGrid 記述子 XML ファイルを使用するか、プログラムで行います。

このタスクについて

XML を使用した Evictor の構成に関する参照情報については、 151 ページの『ObjectGrid 記述子 XML ファイル』 を参照してください。

TimeToLive (TTL) Evictor

WebSphere eXtreme Scale には、キャッシュ・エントリーを除去するためのデフォルトのメカニズムと、カスタム Evictor を作成するためのプラグインが用意されています。Evictor は、各 BackingMap インスタンスのエントリーのメンバーシップを制御します。

プログラマチックな TTL Evictor の使用可能化

TTL Evictor は、BackingMap インスタンスと関連しています。デフォルトの Evictor は、各 BackingMap インスタンスに対して存続時間 (TTL) 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、時間ではなく、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。

以下のコード・スニペットでは、BackingMap インターフェースを使用して、各エントリーの有効期限の時間をエントリー作成の 10 分後に設定しています。

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

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 になっています。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。有効期限の時間に達すると、TTL Evictor はそのエントリーを除去します。

有効期限の時間を最終アクセス時刻プラス 10 分に設定するには、setTtlEvictorType メソッドに渡される引数を TTLType.CREATION_TIME から TTLType.LAST_ACCESS_TIME に変更します。 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。 (更新を伴ったかどうかに関わらず) 単純に最終 アクセス をベースにするのではなく、最終 *更新* をベースにして有効期限を設定するには、TTLType.LAST_ACCESS_TIME 設定を TTLType.LAST_UPDATE_TIME 設定に置き換えてください。

TTLType.LAST_ACCESS_TIME または TTLType.LAST_UPDATE_TIME 設定を使用する場合、ObjectMap インターフェースおよび 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。前述のコード・スニペットが ttlType 属性を

LAST_ACCESS_TIME に設定し、存続時間の値を 10 分に設定したと想定します。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・スニペットでは、key1 キーを持つエントリーの有効期限の時間は、ObjectMap インスタンスの 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 インスタンスの 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 インスタンスの 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。 アプリケーションは、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ながら、もっと後の時点でもこれらのマップ・エントリーを更新することができます。 以下の例では、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 キーには 1800 秒、key2 キーには 1200 秒の存続時間値が保管されます。これらの値が、前のトランザクションでこれらのマップ・エントリーが挿入されたときに使用されたためです。

前の例では、key3 キーの新規マップ・エントリーの挿入も示されています。この場合、USE_DEFAULT 特殊値は、このマップの存続時間値に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 インスタンスに存続時間属性を定義する方法について詳しくは、BackingMap インターフェース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。この資料では、BackingMap.getTtlEvictorType メソッドから TTLType.LAST_ACCESS_TIME または

TTLType.LAST_UPDATE_TIME 以外の値が戻された場合は、IllegalStateException 例外が生じることを説明しています。ObjectMap インターフェースおよび JavaMap インターフェースは、TTL Evictor タイプに LAST_ACCESS_TIME または TTLType.LAST_UPDATE_TIME 設定を使用している場合のみ、存続時間の値をオーバーライドすることができます。setTimeToLive メソッドは、Evictor タイプ設定に CREATION_TIME または NONE 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 構成を使用した TTL Evictor の使用可能化

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 インスタンスを構成することができます。以下のコードは、3 つの異なる BackingMap マップに対してこれらの属性を設定する方法を示しています。

enabling time-to-live evictor using XML

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

前掲の例は、map1 BackingMap インスタンスが NONE TTL Evictor タイプを使用することを示しています。 map2 BackingMap インスタンスでは、LAST_ACCESS_TIME または LAST_UPDATE_TIME の TTL Evictor タイプ (これらの設定うちのいずれかを指定してください) が使用され、存続時間の値は 1800 秒、すなわち 30 分になります。map3 BackingMap インスタンスは、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。

プラグ可能エビクターのプラグイン

Evictor は BackingMap に関連付けられているため、BackingMap インターフェースを使用してプラグ可能 Evictor を指定します。

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に基づいて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 Evictor 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、最長未使用時間 (LRU) アルゴリズムを使用します。
- LFUEvictor Evictor 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、最少使用頻度 (LFU) アルゴリズムを使用します。

BackingMap は、トランザクション内でエントリーが作成、変更、または除去されると Evictor に通知します。 BackingMap は、これらのエントリーをトラッキングし、BackingMap インスタンスから 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 インスタンスには、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、除去ストラテジーを作成できます。

オプションのプラグ可能 Evictor の使用

オプションのプラグ可能 Evictor を BackingMap 構成に追加する場合、プログラマチック構成または XML 構成を使用できます。

プラグ可能 Evictor のプログラマチックなプラグイン

Evictor は BackingMap に関連付けられているため、BackingMap インターフェースを使用してプラグ可能 Evictor を指定します。次のコード・スニペットは、map1 BackingMap インスタンス用に LRUEvictor Evictor、および map2 BackingMap インスタンス用に LFUEvictor Evictor を指定する例です。

```
plugging in an evictor programmatically
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectGridManager;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor;
import com.ibm.websphere.objectgrid.plugins.builtins.LFUEvictor;
ObjectGridManager ogManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid og = ogManager.createObjectGrid( "grid" );
BackingMap bm = og.defineMap( "map1" );
LRUEvictor evictor = new LRUEvictor();
evictor.setMaxSize(1000);
evictor.setSleepTime( 15 );
evictor.setNumberOfLRUQueues( 53 );
bm.setEvictor(evictor);
bm = og.defineMap( "map2" );
LFUEvictor evictor2 = new LFUEvictor();
evictor2.setMaxSize(2000);
evictor2.setSleepTime( 15 );
evictor2.setNumberOfHeaps( 211 );
bm.setEvictor(evictor2);
```

上記のスニペットは、概算最大エントリー数が 53,000 (53 x 1000) である map1 BackingMap に使用される LRUEvictor Evictor を示しています。 LFUEvictor

Evictor は、概算最大エントリー数が 422,000 (211 x 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。 setMaxSize メソッドは、各キューで許容されるエントリーの数を示します。キューがその最大エントリー数に達すると、キュー内の最も使用頻度の少ない 1 つまたは複数のエントリーが、次回に Evictor がエントリーを除去する必要があるかどうかをチェックした時点で除去されます。

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。 setMaxSize メソッドは、各ヒープで許容されるエントリーの数を示します。ヒープがその最大エントリー数に達すると、ヒープ内の最も使用頻度の低い 1 つまたは複数のエントリーが、次回に Evictor がエントリーを除去する必要があるかどうかをチェックした時点で除去されます。

プラグ可能 Evictor のプラグインの XML 構成アプローチ

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。

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

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

メモリー・ベースの除去

組み込み Evictor はすべて、BackingMap の evictionTriggers 属性を「MEMORY_USAGE_THRESHOLD」に設定して、BackingMap インターフェースで使用可能にできるメモリー・ベースの除去をサポートします。BackingMap での evictionTriggers 属性の設定方法について詳しくは、BackingMap インターフェースおよび eXtreme Scale 構成の解説書を参照してください。

メモリー・ベースの除去は、ヒープ使用量のしきい値に基づいています。BackingMap でメモリー・ベースの除去が使用可能になっていて、BackingMap に組み込み Evictor がある場合、使用量のしきい値は、まだ設定されていなければ、合計メモリーのデフォルトのパーセンテージに設定されます。

デフォルトの使用量しきい値のパーセンテージを変更するには、eXtreme Scale サーバー・プロセスのコンテナーおよびサーバーのプロパティー・ファイルで memoryThresholdPercentage プロパティーを設定します。 eXtreme Scale クライアント・プロセスでターゲットの使用量しきい値を設定する場合は、MemoryPoolMXBean を使用できます。containerServer.props ファイルおよび eXtreme Scale サーバー・プロセスの開始も参照してください。

実行時に、メモリー使用量がターゲットの使用量しきい値を超えると、メモリー・ベースの Evictor はエントリーの除去を開始して、メモリー使用量がターゲットの使用量しきい値を下回るようにします。ただし、そのままシステム・ランタイムによるメモリー消費が速い状態が続くと、除去速度が十分であっても、メモリー不足エラーの可能性がなくなるという保証はありません。

ロック・ストラテジーの構成

WebSphere eXtreme Scale 構成の各 BackingMap に対するオプティミスティック、ペシミスティック、あるいはロックなしのストラテジーを定義できます。

このタスクについて

ロック・ストラテジーは、プログラムで指定するか、あるいは XML を使用して指定できます。ロックについて詳しくは、*製品概要* にあるロック・ストラテジーに関する情報を参照してください。

手順

- **オプティミスティック・ロック・ストラテジーの構成**
  - プログラムによる構成

    プログラムによるオプティミスティック・ストラテジーの指定

    ```
    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("optimisticMap");
bm.setLockStrategy( LockStrategy.OPTIMISTIC );
```

– XML の使用

XML を使用したオプティミスティック・ストラテジーの指定

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

- **ペシミスティック・ロック・ストラテジーの構成**

  – プログラムによる構成

  プログラムでのペシミスティック・ストラテジーの指定

```
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("pessimisticMap");
bm.setLockStrategy( LockStrategy.PESSIMISTIC);
```

  – XML の使用

  XML を使用したペシミスティック・ストラテジーの指定

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

    <objectGrids>
        <objectGrid name="test">
            <backingMap name="pessimisticMap"
                lockStrategy="PESSIMISTIC"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

- **ロックなしストラテジーの構成**

  – プログラムによる構成

  プログラムによるロックなしストラテジーの指定

```
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("noLockingMap");
bm.setLockStrategy( LockStrategy.NONE);
```

  – XML の使用

XML を使用したロックなしストラテジーの指定

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

    <objectGrids>
        <objectGrid name="test">
            <backingMap name="noLockingMap"
                lockStrategy="NONE"/>
        </objectGrid>
    </objectGrids>
</objectGridConfig>
```

次のタスク

java.lang.IllegalStateException 例外を避けるために、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。

ローダーの構成

ローダーを実装するには、いくつかの属性を構成する必要があります。

プリロードの考慮事項

ローダーは、変更がバックアップ・マップに対して行われた場合、または、バックアップ・マップがデータ要求を満足できない (キャッシュ・ミス) 場合に呼び出されるバックアップ・マップ・プラグインです。eXtreme Scale がローダーとどのように対話するのかについての概要は、 *製品概要*のインライン・キャッシングのシナリオに関する説明を参照してください。

各バックアップ・マップには、マップのプリロードが非同期的に実行されるかどうかを示すために設定できるブール値の preloadMode 属性があります。 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クアップ・マップの初期化が完了しないことを示します。例えば、preloadMap メソッドが戻されるまで、バックアップ・マップの初期化は完了しません。 preloadMap メソッドによりバックエンドから大量のデータが読み取られて、それがマップにロードされる場合は、完了するまでに比較的長い時間を要する場合があります。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クアップ・マップを構成できます。この設定により、バックアップ・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開始し、マップのプリロードの進行中に、バックアップ・マップの初期化を完了できるようになります。

分散 eXtreme Scale のシナリオでは、プリロード・パターンの 1 つがクライアントのプリロードです。 クライアントのプリロード・パターンでは、DataGrid エージェントを使用したバックエンドからのデータの取得および分散 eXtreme Scale サーバーへのデータの挿入という役割を、eXtreme Scale クライアントが担います。 さらに、クライアントのプリロードは 1 つの特定の区画のみの Loader.preloadMap メソッドで実行される可能性があります。 この場合、グリッドに非同期でデータをロードすることがとても重要になります。 クライアントのプリロードが同じスレッドで実行されると、バックアップ・マップは決して初期化されないため、クライアント

のプリロードが常駐する区画は一度も ONLINE になりません。 このため、eXtreme Scale クライアントは要求を区画に送信することができず、最終的にそれが例外の原因となります。

eXtreme Scale クライアントが preloadMap メソッドで使用されている場合は、preloadMode 属性を true に設定してください。 代替案は、クライアントのプリロード・コードでスレッドを開始することです。

以下のコード・スニペットは、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。ローダーが TxID および TransactionCallback インターフェースをどのように使用するのかを示すため、Java Database Connectivity (JDBC) ローダーが使用されます。

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。最適な方法は、TransactionCallback インターフェースが拡張されて、ローダーにデータベースが接続され、ローダーが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。この方法論の理由は、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査す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、ローダーの get メソッドおよび batchUpdate メソッドにより、以下のようにして接続が取得されます。

```
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において、ivTcb および ivOcb はプリロードの考慮事項のセクションで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、バックアップ・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対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実装を使用している場合は、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をデータベースからフェッチします。この場合、ローダーは以下のように実装される getAutoCommitConnection メソッドを備えている場合があります。

```
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て、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・スニペットです。

```
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、以下のように実装されます。

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

アプリケーションによりセッションの commit メソッドが呼び出されると、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。すべてのローダーにより、必要なすべての接続と準備済みステートメントを取得するために MyTransactionCallback メソッドが使用されたため、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実行されることを保証します。
- TransactionCallback は、WebSphere Application Server、または他の Java 2 Platform, Enterprise Edition (J2EE) 準拠のアプリケーション・サーバーなどの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は、隠蔽されます。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、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、並行性制御にオプティミスティック・アプローチを使用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、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、鍵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、バックアップ・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。

modified batch-update method code example

```
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 に保存されます。この処理が実行されるのは、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初期 Employee オブジェクトの状態を変更する何らかのメソッドを呼び出す時の前です。

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。Loader インターフェースの batchUpdate メソッドを呼び出す前に、eXtreme Scale は OptimisticCallback インターフェースの updateVersionedObjectForValue メソッドを呼び出して、更新された Employee オブジェクトに対する新しいバージョン・オブジェクトが生成されるようにします。batchUpdate メソッドが ObjectGrid に戻された後、LogElement は新規の初期バージョン・オブジェクトになるように、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。その理由のため、eXtreme Scale は、従業員テーブル内の行が更新されるたびに LogElement が新しいバージョン・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用意されているので、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値を使用する 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により、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り後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により行が変更され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は、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。データがまれにしか変更されないか、または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行されない場合は、オプティミスティック・アプローチは実際に適切に機能します。

ローダーが OptimisticCollisionException コンストラクターのキー・パラメーターを使用して、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キーまたはキーのセット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。 eXtreme Scale 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・エントリーに失効データが含まれていて、例外の発生原因となったのかを判別します。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。この方法では、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 文節に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が含まれています。変更されたフィールドを検出する方法の 1 つに、バックアップ・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があります。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、ローダーは属性名の値インターフェースで get メソッドを呼び出して、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されます。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。

後書きローダー・サポートの構成

後書きサポートを使用可能にするには、ObjectGrid 記述子 XML ファイルを使用するか、BackingMap インターフェースでプログラマチックに行います。

後書きサポートを使用可能にするには、ObjectGrid 記述子 XML ファイルを使用するか、BackingMap インターフェースでプログラマチックに行います。

ObjectGrid 記述子 XML ファイル

ObjectGrid 記述子 XML ファイルを使用して ObjectGrid を構成する場合、backingMap タグで writeBehind 属性を設定すると、後書きローダーが使用可能になります。以下に例を示します。

```
<objectGrid name="library" >
     <backingMap name="book" writeBehind="T300;C900" pluginCollectionRef="bookPlugins"/>
```

この例では、book バックアップ・マップの後書きサポートがパラメーター T300;C900 で使用可能になります。後書き属性は、最大更新時間または最大キー更新数、あるいはその両方を指定します。後書きパラメーターの形式は以下のとおりです。

```
::= <defaults> | <update time> | <update key count> | <update time> ";"
<update key count>::= "T" <positive integer>::= "C" <positive integer>::= ""
```

- 後書き属性
- 更新時間
- 更新キー数
- デフォルト

ローダーに対する更新は、以下のいずれかのイベントが発生すると実行されます。

1. 最終更新以降、最大更新時間 (秒数) を経過する
2. キュー・マップ内の更新キーの数が最大更新キー数に達する。

これらのパラメーターは、ヒントにすぎません。実際の更新数および更新時間は、これらのパラメーターの近似値になります。ただし、実際の更新数または更新時間がパラメーターで定義されたものと同じであることを保証するものではありません。また、更新時間の範囲内で、最大 2 回まで更新が発生した後、最初の後書き更新が発生することがあります。これは、すべての区画で同時にデータベースにアクセスしないように ObjectGrid が更新開始時間をランダム化するためです。

前記の例の T300;C900 では、最終更新以降 300 秒が経過するか、900 個のキーが更新保留状態になると、ローダーはデータをバックエンドに書き込みます。デフォルトの更新時間は 300 秒で、デフォルトの更新キー数は 1000 です。

後書きキャッシング

後書きキャッシングを使用して、バックエンドとして使用しているデータベースを更新する際に発生するオーバーヘッドを減らすことができます。

概要

後書きキャッシングは、ローダー・プラグインへの更新情報を非同期でキューに入れます。eXtreme Scale トランザクションをデータベース・トランザクションから分離することにより、マップの更新、挿入、および除去の、パフォーマンスを改善できます。非同期更新は、時間ベースの遅延 (例えば、5 分)、またはエントリー・ベースの遅延 (例えば、1000 エントリー) 後に実行されます。

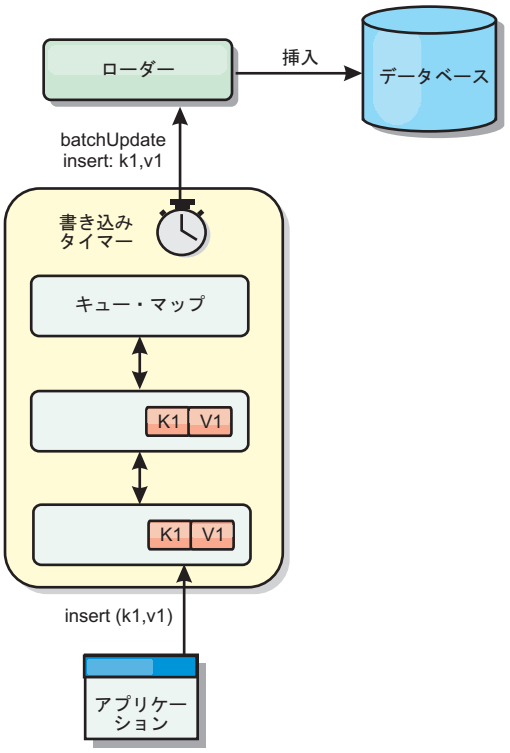


図 7. 後書きキャッシング

BackingMap の後書き構成により、ローダーとマップとの間にスレッドが作成されます。次に、ローダーは、BackingMap.setWriteBehind メソッド内の構成設定に従って、そのスレッドを通してデータ要求を委任します。eXtreme Scale トランザクションが、マップのエントリーを挿入、更新、または削除すると、これらの各レコードごとに 1 つずつ LogElement オブジェクトが作成されます。これらのエレメントは後書きローダーに送信され、キュー・マップと呼ばれる特別な ObjectMap 内でキューに入れられます。後書き設定が有効になっているバックアップ・マップは、それぞれ独自のキュー・マップを持っています。後書きスレッドは、キューに入れられたデータをキュー・マップから定期的に除去して、実際のバックエンド・ローダーにプッシュします。

後書きローダーは、挿入、更新、および削除タイプの LogElement オブジェクトのみを実際のローダーに送信します。それ以外のタイプの LogElement オブジェクト (例えば、EVICT タイプ) はすべて無視されます。

利点

後書きサポートを使用可能にすると、以下のような利点があります。

- **バックエンド障害の分離:** 後書きキャッシングは、バックエンド障害からの分離層を提供します。バックエンドのデータベースで障害が発生すると、更新はキュー・マップ内でキューに入れられます。アプリケーションは、トランザクションを eXtreme Scale に送り続けることができます。バックエンドが復旧すると、キュー・マップ内のデータはバックエンドにプッシュされます。

- **バックエンドの負荷の削減:** 後書きローダーは更新をキー単位でマージします。その結果、キュー・マップ内には、キーごとにマージされた更新が 1 つのみ存在します。このマージにより、バックエンド・データベースに対する更新の数が減ります。
- トランザクション・パフォーマンスの改善: データがバックエンドと同期されるのをトランザクションが待機する必要がないので、個別の eXtreme Scale トランザクション時間が削減されます。

アプリケーション設計に関する考慮事項

後書きサポートを使用可能にすることは簡単ですが、後書きサポートを扱うアプリケーションを設計する際には、注意すべき考慮事項があります。後書きサポートがない場合、ObjectGrid トランザクションにバックエンド・トランザクションが包含されます。 ObjectGrid トランザクションはバックエンド・トランザクションの開始前に開始し、バックエンド・トランザクションの終了後に終了します。

後書きサポートが有効な場合、ObjectGrid トランザクションは、バックエンド・トランザクションが開始する前に終了します。ObjectGrid トランザクションとバックエンド・トランザクションは切り離されます。

参照保全性の制約

後書きサポートで構成されているそれぞれのバックアップ・マップは、データをバックエンドにプッシュするための独自の後書きスレッドを持ちます。したがって、1 つの ObjectGrid トランザクションにさまざまなマップを更新するデータが含まれていても、バックエンドでは、それぞれ異なるバックエンド・トランザクションでデータの更新が行われます。例えば、トランザクション T1 はマップ Map1 のキー key1 とマップ Map2 のキー key2 を更新するとします。マップ Map1 に対する key1 更新は、1 つのバックエンド・トランザクションでバックエンドに対して更新され、マップ Map2 に対する key2 更新は、異なる後書きスレッドにより別のバックエンド・トランザクションでバックエンドに対して更新されます。Map1 に保管されたデータと Map2 に保管されたデータがバックエンドでの外部キー制約などの関係を持つ場合、更新が失敗する可能性があります。

バックエンド・データベースの参照保全性制約を設計するときは、順不同の更新に必ず対応できるようにしてください。

キュー・マップのロックの振る舞い

トランザクションの動作で他に大きく異なる点は、ロックの振る舞いです。ObjectGrid は、PESSIMISTIC、OPTIMISITIC、および NONE の 3 つの異なるロック・ストラテジーをサポートします。後書きキュー・マップは、 *バックアップ・マップに構成されているロック・ストラテジーに関係なく、ペシミスティック・ロック・ストラテジーを使用します。キュー・マップのロックを取得する操作には 2 つの異なるタイプがあります。

- ObjectGrid トランザクションのコミット時、またはフラッシュ (マップ・フラッシュまたはセッション・フラッシュ) の発生時、トランザクションはキュー・マップ内のキーを読み取り、キーに S ロックをかけます。
- ObjectGrid トランザクションのコミット時、トランザクションは、キーの S ロックを X ロックにアップグレードしようとします。

キュー・マップのこの余分な動作のため、ロックの動作に少々違いがあります。

- ユーザー・マップがペシミスティック・ロック・ストラテジーで構成されている場合、ロックの動作にほとんど違いはありません。フラッシュまたはコミットが呼び出されるたび、キュー・マップ内の同じキーに S ロックがかけられます。コミット時間中、ユーザー・マップ内のキーに X ロックが取得されるだけでなく、キュー・マップ内のキーに対しても X ロックが取得されます。
- ユーザー・マップが OPTIMISTIC または NONE ロック・ストラテジーで構成されている場合、ユーザー・トランザクションは PESSIMISTIC ロック・ストラテジーのパターンに従います。フラッシュまたはコミットが呼び出されるたびに、キュー・マップ内の同じキーに対して S ロックが取得されます。コミット時間の間、同じトランザクションを使用するキュー・マップ内のキーに対して X ロックが設定されます。

ローダー・トランザクションの再試行

ObjectGrid は、2 フェーズ・トランザクションまたは XA トランザクションをサポートしません。後書きスレッドは、キュー・マップからレコードを除去して、バックエンドに対してそのレコードを更新します。トランザクションの最中にサーバーに障害が起こると、一部のバックエンドの更新が失われる可能性があります。

後書きローダーは、失敗したトランザクションの書き込みを自動的に再試行し、データ損失を防ぐために未確定 LogSequence をバックエンドに送信します。このアクションを行うには、ローダーがべき等である必要があります。この意味は、Loader.batchUpdate(TxId, LogSequence) が同じ値で 2 回呼び出されたとき、それは適用された回数があたかも 1 回だったかのように、同じ結果を返すということです。ローダー実装は、この機能を使用可能にするため、RetryableLoader インターフェースを実装しなければなりません。詳しくは、API 資料を参照してください。

ローダーの障害

ローダー・プラグインは、バックエンド・データベースと通信できない場合、失敗することがあります。これは、データベース・サーバーまたはネットワーク接続がダウンしている場合に発生することがあります。後書きローダーは、更新をキューに入れ、データ変更を定期的にローダーにプッシュしようと試みます。ローダーは、LoaderNotAvailableException 例外をスローして、データベース接続の問題があることを ObjectGrid ランタイムに通知しなければなりません。

したがって、ローダー実装で、データ障害または物理的ローダー障害を識別できるようになっている必要があります。データ障害は LoaderException または OptimisticCollisionException としてスローまたは再スローされる必要がありますが、物理的なローダーの障害は LoaderNotAvailableException としてスローまたは再スローされる必要があります。 ObjectGrid は、これら 2 つの例外を異なる方法で処理します。

- LoaderException が後書きローダーによってキャッチされると、重複キー障害などのある種のデータ障害のため、後書きローダーはそれを障害とみなします。後書きローダーは、更新のバッチ処理を解除し、データ障害を分離するため、1 度に 1 レコードずつ更新しようとします。1 レコードの更新時に再度 {{LoaderException}} がキャッチされると、失敗した更新レコードが作成され、失敗した更新マップのログに記録されます。

- LoaderNotAvailableException が後書きローダーによってキャッチされると、データベース・エンドに接続できない (例えば、データベース・バックエンドがダウンしている、データベース接続が使用可能でない、ネットワークがダウンしているなど) ため、後書きローダーはそれを障害とみなします。後書きローダーは 15 秒待ってから、データベースへのバッチ更新を再試行します。

一般的な間違いは、LoaderNotAvailableException がスローされるべきなのに、LoaderException がスローされることです。後書きローダーでキューに入れられたすべてのレコードは、失敗更新レコードとなります。このような場合、バックエンド障害分離の目的が果たせなくなります。

パフォーマンスの考慮事項

後書きキャッシング・サポートの場合、ローダー更新をトランザクションから除去することで、応答時間が増加します。また、データベース更新が結合されるため、データベース・スループットも増加します。データをキュー・マップからプルし、ローダーにプッシュされる後書きスレッドの導入によって生じるオーバーヘッドを理解しておく必要があります。

予想される使用パターンおよび環境に基づいて、最大更新数または最大更新時間を調整する必要があります。最大更新カウントまたは最大更新時間の値が小さすぎると、後書きスレッドのオーバーヘッドが、その利点を帳消しにするおそれがあります。これら 2 つのパラメーターに大きな値を設定する場合も、データのキューイングに必要なメモリー使用が増え、データベース・レコードが不整合になる時間が増加するおそれがあります。

最善のパフォーマンスを得るために、後書き関係のパラメーターは、以下の要因を考慮に入れて調整してください。

- 読み取りトランザクションと書き込みトランザクションの比率
- 同一レコード更新の頻度
- データベース更新の待ち時間

後書きキャッシング・サポート

後書きキャッシングを使用して、バックエンド・データベースを更新する際に発生するオーバーヘッドを減らすことができます。後書きキャッシングは、ローダー・プラグインへの更新情報をキューに入れます。

概要

後書きキャッシングでは、ローダー・プラグインの更新が非同期にキューに入れられます。eXtreme Scale トランザクションをデータベース・トランザクションから分離することにより、マップの更新、挿入、および除去の、パフォーマンスを改善できます。非同期更新は、時間ベースの遅延 (例えば 5 分) またはエントリー・ベースの遅延 (例えば 1000 エントリー) 後に実行されます。

BackingMap での後書き構成により、ローダーとマップ間のスレッドが作成されます。その後は、ローダーが、BackingMap.setWriteBehind() メソッドの構成の設定値に従い、スレッド経由でデータ要求を委任します。 eXtreme Scale トランザクションが eXtreme Scale マップでエントリーの挿入、更新、または除去を行うと、これらのレコードのそれぞれに LogElement オブジェクトが作成されます。これらのエ

レメントは後書きローダーに送信され、キュー・マップと呼ばれる特別な ObjectMap 内でキューに入れられます。後書き設定が有効になっているバックアップ・マップは、それぞれ独自のキュー・マップを持っています。後書きスレッドは、キューに入れられたデータをキュー・マップから定期的に除去して、実際のバックエンド・ローダーにプッシュします。

後書きローダーは、LogElement オブジェクトの挿入、更新、および削除タイプのみを実際のローダーに送信します。それ以外のタイプの LogElement オブジェクト (例えば、EVICT タイプ) はすべて無視されます。

後書きサポートは、eXtreme Scale をデータベースに組み込む際に使用するローダー・プラグインの *拡張機能* です。例えば、JPA ローダーの構成については 246 ページの『JPA ローダーの構成』 の情報を参照してください。

利点

後書きサポートを使用可能にすると、以下のような利点があります。

- バックエンドの障害の分離: 後書きキャッシングは、バックエンドの障害からの分離層を提供します。バックエンドのデータベースで障害が発生すると、更新はキュー・マップ内でキューに入れられます。アプリケーションは、引き続きトランザクションを eXtreme Scale に送ることができます。バックエンドが回復すると、キュー・マップ内のデータがバックエンドにプッシュされます。
- バックエンド負荷の軽減: 後書きローダーは、更新をキー単位でマージするため、キュー・マップ内にはキーごとにマージされた更新が 1 つしか存在しません。このマージにより、バックエンドに対する更新の数が削減されます。
- トランザクション・パフォーマンスの改善: データがバックエンドと同期されるのをトランザクションが待機する必要がないので、個別の eXtreme Scale トランザクション時間が削減されます。

ObjectGrid 記述子 XML

eXtreme Scale 記述子 XML ファイルを使用して eXtreme Scale を構成する場合、backingMap タグで writeBehind 属性を設定すると、後書きローダーが使用可能になります。以下に例を示します。

```
<objectGrid name="library" >
     <backingMap name="book" writeBehind="T300;C900" pluginCollectionRef="bookPlugins"/>
```

前記の例では、「book」バックアップ・マップの後書きサポートがパラメーター「T300;C900」で使用可能になります。

後書き属性は、最大更新時間または最大キー更新数、あるいはその両方を指定します。後書きパラメーターの形式は以下のとおりです。

```
write-behind attribute  ::= <defaults> | <update time> | <update key count> | <update time> ";" <update key count>
update time  ::= "T" <positive integer>
update key count  ::= "C" <positive integer>
defaults  ::= "" {table}
```

ローダーに対する更新は、以下のいずれかのイベントが発生すると実行されます。

1. 最終更新以降、最大更新時間 (秒数) を経過する
2. キュー・マップ内の更新キーの数が最大更新キー数に達する。

これらのパラメーターはヒントにすぎません。実際の更新数および更新時間は、これらのパラメーターの近似値になります。ただし、実際の更新数または更新時間がパラメーターで定義されたものと同じであることを保証するものではありません。また、更新時間の範囲内で、最大 2 回まで更新が発生した後、最初の後書き更新が発生することがあります。これは、すべての区画でデータベースに同時にヒットしないよう、eXtreme Scale が更新開始時間をランダム化するためです。

前記の例の T300;C900 では、最終更新以降 300 秒が経過するか、900 個のキーが更新保留状態になると、ローダーはデータをバックエンドに書き込みます。

デフォルトの更新時間は 300 秒で、デフォルトの更新キー数は 1000 です。

下の表に、後書き属性の例がいくつか示されています。

注: 後書きローダーを空ストリング `writeBehind=""` として構成すると、後書きローダーはデフォルト値を使用して使用可能になります。したがって、後書きサポートを使用可能にしたくない場合、writeBehind 属性を指定しないでください。

表 8. いくつかの後書きオプション

属性値	時間
T100	更新時間は 100 秒で、更新キー数は 1000 (デフォルト値) です。
C2000	更新時間は 300 秒 (デフォルト値) で、更新キー数は 2000 です。
T300;C900	更新時間は 300 秒で、更新キー数は 900 です。
""	更新時間は 300 秒 (デフォルト値) で、更新キー数は 1000 (デフォルト値) です。

後書きサポートをプログラマチックに使用可能化

ローカルのメモリー内の eXtreme Scale 用のバックアップ・マップをプログラムで作成する場合、以下のメソッドを BackingMap インターフェースで使用すると、後書きサポートを使用可能または使用不可にできます。

```
public void setWriteBehind(String writeBehindParam);
```

setWriteBehind メソッドの使用方法について詳しくは、「*プログラミング・ガイド*」の BackingMap インターフェースに関する情報を参照してください。

アプリケーション設計に関する考慮事項

後書きサポートを使用可能にすることは簡単ですが、後書きサポートを扱うアプリケーションを設計する際には、注意すべき考慮事項があります。後書きサポートがないと、eXtreme Scale トランザクションにバックエンド・トランザクションが組み込まれます。eXtreme Scale トランザクションは、バックエンド・トランザクションの開始前に開始し、バックエンド・トランザクションの終了後に終了します。

後書きサポートが使用可能になっていると、eXtreme Scale トランザクションは、バックエンド・トランザクションの開始前に終了します。 eXtreme Scale トランザクションとバックエンド・トランザクションは、互いに切り離されます。

参照保全性の制約

後書きサポートで構成されているそれぞれのバックアップ・マップは、データをバックエンドにプッシュするための独自の後書きスレッドを持ちます。したがって、1 つの eXtreme Scale トランザクションで異なるマップに対して更新されたデータ

は、異なるバックエンド・トランザクションでバックエンドに対して更新されます。例えば、トランザクション T1 はマップ Map1 のキー key1 とマップ Map2 のキー key2 を更新するとします。マップ Map1 に対する key1 更新は、1 つのバックエンド・トランザクションでバックエンドに対して更新され、マップ Map2 に対する key2 更新は、異なる後書きスレッドにより別のバックエンド・トランザクションでバックエンドに対して更新されます。Map1 に保管されたデータと Map2 に保管されたデータがバックエンドでの外部キー制約などの関係を持つ場合、更新が失敗する可能性があります。

バックエンド・データベースの参照保全性制約を設計するときは、順不同の更新に必ず対応できるようにしてください。

更新の失敗

eXtreme Scale トランザクションが、バックエンド・トランザクションの開始前に終了するため、トランザクションの実行が誤った形の正常実行になる可能性があります。例えば、バックアップ・マップには存在しないが、バックエンドに存在するエントリーを eXtreme Scale トランザクションに挿入すると、重複キーになることになりますが、eXtreme Scale トランザクションは成功します。ただし、後書きスレッドがそのオブジェクトをバックエンドに挿入するトランザクションは失敗し、重複キーの例外が発生します。

このような障害の処理方法については、 139 ページの『失敗した後書き更新の処理』を参照してください。

キュー・マップのロックの振る舞い

トランザクションでのもう 1 つの振る舞い上の大きな相違は、ロックの振る舞いです。eXtreme Scale は、PESSIMISTIC、OPTIMISITIC、および NONE という 3 つの異なるロック戦略をサポートします。後書きキュー・マップは、バックアップ・マップに構成されているロック・ストラテジーに関係なく、PESSIMISTIC ロック・ストラテジーを使用します。キュー・マップでのロックを取得するために以下の 2 つの異なるタイプの操作が存在します。

- eXtreme Scale トランザクションがコミットするか、フラッシュ (マップ・フラッシュまたはセッション・フラッシュ) が発生すると、トランザクションは、キュー・マップ内のキーを読み取り、キーに対して S ロックを設定します。
- eXtreme Scale トランザクションがコミットすると、トランザクションはキーに対する S ロックを X ロックにアップグレードしようとします。

キュー・マップのこの余分な動作のため、ロックの動作に少々違いがあります。

- ユーザー・マップがペシミスティック・ロック・ストラテジーで構成されている場合、ロックの動作にほとんど違いはありません。フラッシュまたはコミットが呼び出されるたび、キュー・マップ内の同じキーに S ロックがかけられます。コミット時間中、ユーザー・マップ内のキーに X ロックが取得されるだけでなく、キュー・マップ内のキーに対しても X ロックが取得されます。
- ユーザー・マップが OPTIMISTIC または NONE ロック・ストラテジーで構成されている場合、ユーザー・トランザクションは PESSIMISTIC ロック・ストラテジーのパターンに従います。フラッシュまたはコミットが呼び出されるたびに、

キュー・マップ内の同じキーに対して S ロックが取得されます。コミット時間の間、同じトランザクションを使用するキュー・マップ内のキーに対して X ロックが設定されます。

ローダー・トランザクションの再試行

WebSphere eXtreme Scale は 2 フェーズ・トランザクションまたは XA トランザクションをサポートしません。後書きスレッドは、キュー・マップからレコードを除去して、バックエンドに対してそのレコードを更新します。トランザクションの最中にサーバーに障害が起こると、一部のバックエンドの更新が失われる可能性があります。

後書きローダーは、失敗したトランザクションの書き込みを自動的に再試行し、データ損失を防ぐために未確定 LogSequence をバックエンドに送信します。このアクションを行うには、ローダーがべき等である必要があります。この意味は、Loader.batchUpdate(TxId, LogSequence) が同じ値で 2 回呼び出されたとき、それは適用された回数があたかも 1 回だったかのように、同じ結果を返すということです。ローダー実装は、この機能を使用可能にするため、RetryableLoader インターフェースを実装しなければなりません。詳細は、 API 資料を参照してください。

ローダーの障害

ローダー・プラグインは、バックエンド・データベースと通信できない場合、失敗することがあります。これは、データベース・サーバーまたはネットワーク接続がダウンしている場合に発生することがあります。後書きローダーは、更新をキューに入れ、データ変更を定期的にローダーにプッシュしようと試みます。ローダーは、LoaderNotAvailableException 例外をスローして、データベース接続の問題が存在することを WebSphere eXtreme Scale ランタイムに通知しなければなりません。

したがって、ローダー実装で、データ障害または物理的ローダー障害を識別できるようになっている必要があります。データ障害は LoaderException 例外または OptimisticCollisionException 例外としてスローまたは再スローされる必要がありますが、物理的なローダーの障害は LoaderNotAvailableException 例外としてスローまたは再スローされる必要があります。 WebSphere eXtreme Scale は、これら 2 つの例外を異なる方法で処理します。

- LoaderException が後書きローダーによってキャッチされると、重複キー障害などのある種のデータ障害のため、後書きローダーはそれを障害とみなします。後書きローダーは、更新のバッチ処理を解除し、データ障害を分離するため、1 度に 1 レコードずつ更新しようとします。1 レコードの更新時に再度 LoaderException がキャッチされると、失敗した更新レコードが作成され、失敗した更新マップのログに記録されます。
- LoaderNotAvailableException が後書きローダーによってキャッチされると、データベース・エンドに接続できない (例えば、データベース・バックエンドがダウンしている、データベース接続が使用可能でない、ネットワークがダウンしているなど) ため、後書きローダーはそれを障害とみなします。後書きローダーは 15 秒待ってから、データベースへのバッチ更新を再試行します。

一般的な間違いは、LoaderNotAvailableException がスローされるべきなのに、LoaderException がスローされることです。後書きローダーでキューに入れられたすべてのレコードは、失敗更新レコードとなります。このような場合、バックエンド

障害分離の目的が果たせなくなります。この誤りは、データベースと対話する汎用ローダーを作成した場合に起こる可能性があります。

eXtreme Scale 提供の JPALoader はその 1 例です。 JPALoader は JPA API を使用してデータベース・バックエンドと対話します。ネットワークで障害が発生すると、JPALoader は javax.persistence.PersitenceException を取得しますが、チェーニングされた SQLException の SQL 状態および SQL エラー・コードが確認されない限り、障害の本質を認識しません。 JPALoader があらゆるタイプのデータベースを処理するように設計されているという事実は、ネットワーク・ダウン問題の場合は SQL 状態およびエラー・コードが異なるため、問題をいっそう複雑にします。これを解決するため、WebSphere eXtreme Scale は ExceptionMapper API を提供して、ユーザーによる実装のプラグインで Exception をより使いやすい例外にマップできるようにします。 例えば、ユーザーは、SQL 状態またはエラー・コードがネットワーク・ダウンを示している場合であれば、汎用 javax.persistence.PersitenceException を LoaderNotAvailableException にマップすることができます。

パフォーマンスの考慮事項

後書きキャッシング・サポートの場合、ローダー更新をトランザクションから除去することで、応答時間が増加します。また、データベース更新が結合されるため、データベース・スループットも増加します。データをキュー・マップからプルし、ローダーにプッシュされる後書きスレッドの導入によって生じるオーバーヘッドを理解しておく必要があります。

予想される使用パターンおよび環境に基づいて、最大更新数または最大更新時間を調整する必要があります。最大更新カウントまたは最大更新時間の値が小さすぎると、後書きスレッドのオーバーヘッドが、その利点を帳消しにするおそれがあります。これら 2 つのパラメーターに大きな値を設定する場合も、データのキューイングに必要なメモリー使用が増え、データベース・レコードが不整合になる時間が増加するおそれがあります。

最善のパフォーマンスを得るために、後書き関係のパラメーターは、以下の要因を考慮に入れて調整してください。

- 読み取りトランザクションと書き込みトランザクションの比率
- 同一レコード更新の頻度
- データベース更新の待ち時間

失敗した後書き更新の処理

WebSphere eXtreme Scale トランザクションが、バックエンド・トランザクションの開始前に終了するため、トランザクションの実行が誤った形の正常実行になる可能性があります。

例えば、バックアップ・マップには存在しないが、バックエンドに存在するエントリーを eXtreme Scale トランザクションに挿入すると、重複キーになることになりますが、eXtreme Scale トランザクションは成功します。ただし、後書きスレッドがそのオブジェクトをバックエンドに挿入するトランザクションは失敗し、重複キーの例外が発生します。

失敗した後書き更新の処理: クライアント・サイド

このような更新、あるいはその他失敗したバックエンド更新は、失敗した後書き更新となります。失敗した後書き更新は、失敗した後書き更新マップに保管されます。このマップは、失敗した更新のイベント・キューとして機能します。更新のキーは、固有の Integer オブジェクトで、値は、FailedUpdateElement のインスタンスになります。失敗した後書き更新マップは、エビクターによって構成されます。エビクターは、レコードを挿入後 1 時間経過すると除去します。このため、失敗した更新レコードは、1 時間以内に取得されないと失われます。

失敗した後書き更新マップのエントリーを取り出すには、ObjectMap API を使用できます。失敗した後書き更新マップの名前は IBM_WB_FAILED_UPDATES_<map name> です。各後書きシステム・マップの接頭部名については、WriteBehindLoaderConstants API の資料を参照してください。以下に例を示します。

process failed - example code

```
ObjectMap failedMap = session.getMap(
    WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + "Employee");
Object key = null;

session.begin();
while(key = failedMap.getNextKey(ObjectMap.QUEUE_TIMEOUT_NONE)) {
    FailedUpdateElement element = (FailedUpdateElement) failedMap.get(key);
    Throwable throwable = element.getThrowable();
    Object failedKey = element.getKey();
    Object failedValue = element.getAfterImage();
    failedMap.remove(key);
    // Do something interesting with the key, value, or exception.

}
session.commit();
```

getNextKey 呼び出しは、各 eXtreme Scale トランザクションごとに特定の 1 つの区画について作業します。分散環境では、すべての区画からキーを取得するため、以下の例に示すように複数のトランザクションを開始する必要があります。

getting keys from all partitions - example code

```
ObjectMap failedMap = session.getMap(
    WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + "Employee");
while (true) {
    session.begin();
    Object key = null;
    while(( key = failedMap.getNextKey(5000) )!= null ) {
        FailedUpdateElement element = (FailedUpdateElement) failedMap.get(key);
    Throwable throwable = element.getThrowable();
        Object failedKey = element.getKey();
        Object failedValue = element.getAfterImage();
        failedMap.remove(key);
        // Do something interesting with the key, value, or exception.

    }
    Session.commit();
}
```

注: 失敗した更新マップは、アプリケーションのヘルスをモニターする 1 つの手段です。システムが失敗した更新マップに数多くのレコードを作成した場合、それは、後書きサポートを使用するように、アプリケーションまたはアーキテクチャーを再評価または修正する必要があるというサインです。6.1.0.5 以降、xsadmin スクリプトを使用して、失敗した更新マップのエントリー・サイズを確認することができます。

失敗した後書き更新の処理: 断片リスナー

後書きトランザクションが失敗した場合、それを検出し、ログに記録することが重要です。後書きを使用するアプリケーションはすべて、失敗した後書き更新を処理するウォッチャーを実装する必要があります。これによって、アプリケーションが正しくない更新マップ内のレコードを処理することが期待されるため、それらが除去されずに潜在的なメモリー不足になることを防ぐことができます。

以下のコードは、そのようなウォッチャー (ダンパー) の接続方法を示しています。これは、ObjectGrid 記述子 XML にスニペットとして追加する必要があります。

```
<objectGrid name="Grid">
 <bean id="ObjectGridEventListener" className="utils.WriteBehindDumper"/>
```

ObjectGridEventListener Bean が追加されていることがわかります。これは、上記で取り上げた後書きウォッチャーです。このウォッチャーは、JVM 内のすべてのプライマリー断片のマップと対話し、後書きが使用可能になったものを検索します。後書きが使用可能になったものを検出すると、ウォッチャーは最大 100 の不適切な更新をログに記録しようとします。ウォッチャーは、プライマリー断片が別の JVM に移動されるまで、その断片を監視します。後書きを使用するすべてのアプリケーションは、これと似たウォッチャーを使用する必要があります。使用しないと、このエラー・マップが除去されないため、Java 仮想マシン がメモリー不足になります。

詳しくは、後書きダンパー・クラスのサンプル・コードを参照してください。

後書きダンパー・クラス・サンプル・コード

このサンプル・ソース・コードは、失敗した後書き更新を扱うウォッチャー (ダンパー) の作成方法を示しています。

```
//
//This sample program is provided AS IS and may be used, executed, copied and
//modified without royalty payment by customer (a) for its own instruction and
//study, (b) in order to develop applications designed to run with an IBM
//WebSphere product, either for customer's own internal use or for redistribution
//by customer, as part of such an application, in customer's own products. "
//
//5724-J34 (C) COPYRIGHT International Business Machines Corp. 2009
//All Rights Reserved * Licensed Materials - Property of IBM
//
package utils;

import java.util.Collection;
import java.util.Iterator;
import java.util.concurrent.Callable;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.ScheduledThreadPoolExecutor;
import java.util.concurrent.TimeUnit;
import java.util.logging.Logger;

import com.ibm.websphere.objectgrid.BackingMap;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridRuntimeException;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.UndefinedMapException;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventGroup;
import com.ibm.websphere.objectgrid.plugins.ObjectGridEventListener;
import com.ibm.websphere.objectgrid.writebehind.FailedUpdateElement;
import com.ibm.websphere.objectgrid.writebehind.WriteBehindLoaderConstants;

/**
 * Write behind expects transactions to the Loader to succeed. If a transaction for a key fails then
 * it inserts an entry in a Map called PREFIX + mapName. The application should be checking this
 * map for entries to dump out write behind transaction failures. The application is responsible for
 * analyzing and then removing these entries. These entries can be large as they include the key, before
 * and after images of the value and the exception itself. Exceptions can easily be 20k on their own.
 *
 * The class is registered with the grid and an instance is created per primary shard in a JVM. It creates
 * a single thread
 * and that thread then checks each write behind error map for the shard, prints out the problem and
```

```
 * then removes the entry.
 *
 * This means there will be one thread per shard. If the shard is moved to another JVM then the deactivate
 * method stops the thread.
 * @author bnewport
 *
 */
public class WriteBehindDumper implements ObjectGridEventListener, ObjectGridEventGroup.ShardEvents,
 Callable<Boolean>
{
 static Logger logger = Logger.getLogger(WriteBehindDumper.class.getName());

 ObjectGrid grid;

 /**
  * Thread pool to handle table checkers. If the application has it's own pool
  * then change this to reuse the existing pool
  */
 static ScheduledExecutorService pool = new ScheduledThreadPoolExecutor(2); // two threads to dump records

 // the future for this shard
 ScheduledFuture<Boolean> future;

 // true if this shard is active
 volatile boolean isShardActive;

 /**
  * Normal time between checking Maps for write behind errors
  */
 final long BLOCKTIME_SECS = 20L;

 /**
  * An allocated session for this shard. No point in allocating them again and again
  */
 Session session;
 /**
  * When a primary shard is activated then schedule the checks to periodically check
  * the write behind error maps and print out any problems
  */
 public void shardActivated(ObjectGrid grid)
 {
  try
  {
   this.grid = grid;
   session = grid.getSession();

   isShardActive = true;
   future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS); // check every BLOCKTIME_SECS seconds initially
  }
  catch(ObjectGridException e)
  {
   throw new ObjectGridRuntimeException("Exception activating write dumper", e);
  }
 }

 /**
  * Mark shard as inactive and then cancel the checker
  */
 public void shardDeactivate(ObjectGrid arg0)
 {
  isShardActive = false;
  // if it's cancelled then cancel returns true
  if(future.cancel(false) == false)
  {
   // otherwise just block until the checker completes
   while(future.isDone() == false) // wait for the task to finish one way or the other
   {
    try
    {
     Thread.sleep(1000L); // check every second
    }
    catch(InterruptedException e)
    {
    }
   }
  }
 }

 /**
  * Simple test to see if the map has write behind enabled and if so then return
  * the name of the error map for it.
  * @param mapName The map to test
  * @return The name of the write behind error map if it exists otherwise null
  */
 static public String getWriteBehindNameIfPossible(ObjectGrid grid, String mapName)
 {
  BackingMap map = grid.getMap(mapName);
  if(map != null && map.getWriteBehind() != null)
  {
   return WriteBehindLoaderConstants.WRITE_BEHIND_FAILED_UPDATES_MAP_PREFIX + mapName;
  }
```

```
 else
  return null;
}

/**
 * This runs for each shard. It checks if each map has write behind enabled and if it does
  * then it prints out any write behind
 * transaction errors and then removes the record.
 */
public Boolean call()
{
 logger.fine("Called for " + grid.toString());
 try
 {
  // while the primary shard is present in this JVM
  // only user defined maps are returned here, no system maps like write behind maps are in
  // this list.
  Iterator<String> iter = grid.getListOfMapNames().iterator();
  boolean foundErrors = false;
  // iterate over all the current Maps
  while(iter.hasNext() && isShardActive)
  {
   String origName = iter.next();

   // if it's a write behind error map
   String name = getWriteBehindNameIfPossible(grid, origName);
   if(name != null)
   {
    // try to remove blocks of N errors at a time
    ObjectMap errorMap = null;
    try
    {
     errorMap = session.getMap(name);
    }
    catch(UndefinedMapException e)
    {
     // at startup, the error maps may not exist yet, patience...
     continue;
    }
    // try to dump out up to N records at once
    session.begin();
    for(int counter = 0; counter < 100; ++counter)
    {
     Integer seqKey = (Integer)errorMap.getNextKey(1L);
     if(seqKey != null)
     {
      foundErrors = true;
      FailedUpdateElement elem = (FailedUpdateElement)errorMap.get(seqKey);
      //
      // Your application should log the problem here
      logger.info("WriteBehindDumper ( " + origName + ") for key (" + elem.getKey() + ") Exception: " +
       elem.getThrowable().toString());
      //
      //
      errorMap.remove(seqKey);
     }
     else
      break;
    }
    session.commit();
   }
  } // do next map
  // loop faster if there are errors
  if(isShardActive)
  {
   // reschedule after one second if there were bad records
   // otherwise, wait 20 seconds.
   if(foundErrors)
    future = pool.schedule(this, 1L, TimeUnit.SECONDS);
   else
    future = pool.schedule(this, BLOCKTIME_SECS, TimeUnit.SECONDS);
  }
 }
 catch(ObjectGridException e)
 {
  logger.fine("Exception in WriteBehindDumper" + e.toString());
  e.printStackTrace();

  //don't leave a transaction on the session.
  if(session.isTransactionActive())
  {
   try { session.rollback(); } catch(Exception e2) {}
  }
 }
 return true;
}

public void destroy() {
 // TODO Auto-generated method stub

}
```

```
 public void initialize(Session arg0) {
  // TODO Auto-generated method stub

 }

 public void transactionBegin(String arg0, boolean arg1) {
  // TODO Auto-generated method stub

 }

 public void transactionEnd(String arg0, boolean arg1, boolean arg2,
   Collection arg3) {
  // TODO Auto-generated method stub

 }
}
```

JMS を使用したピアツーピア複製の構成

Java Message Service (JMS) を使用したピアツーピア複製メカニズムは、分散およびローカルの両方の WebSphere eXtreme Scale 環境で使用されます。 JMS は、コアツーコア複製プロセスであり、これにより、ローカル ObjectGrid と分散 ObjectGrid の間でデータ更新の流れが可能になります。 例えば、このメカニズムを使用すると分散 eXtreme Scale グリッドからローカル eXtreme Scale グリッドに、あるいは、あるグリッドから別のシステム・ドメインにある別のグリッドに、データ更新を移動させることができます。

始める前に

JMS ベースのピアツーピア複製メカニズムは、組み込まれた JMS ベースの ObjectGridEventListener (com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener) に基づいています。ピアツーピア複製メカニズムの使用可能化に関する詳細については、 148 ページの『JMS イベント・リスナー』を参照してください。

詳しくは、 224 ページの『クライアント無効化メカニズムの使用可能化』を参照してください。

以下は、eXtreme Scale 構成上でピアツーピア複製メカニズムを使用可能にする XML 構成の例です。

ピアツーピア複製構成 - XML の例

```
<bean id="ObjectGridEventListener"
className="com.ibm.websphere.objectgrid.plugins.JMSObjectGridEventListener">
 <property name="replicationRole" type="java.lang.String" value="DUAL_ROLES" description="" />
  <property name="replicationStrategy" type="java.lang.String" value="PUSH" description="" />
  <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String"
  value="defaultTCF" description="" />
  <property name="jms_topicJndiName" type="java.lang.String" value="defaultTopic" description="" />
  <property name="jms_topicName" type="java.lang.String" value="defaultTopic" description="" />
  <property name="jms_userid" type="java.lang.String" value="" description="" />
  <property name="jms_password" type="java.lang.String" value="" description="" />
  <property name="jndi_properties" type="java.lang.String"
 value="java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory;
 java.naming.provider.url=tcp://localhost:61616;connectionFactoryNames=defaultTCF;
 topic.defaultTopic=defaultTopic"
  description="jndi properties" />
     </bean>
```

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 オブジェクトはピア JVM 間で変更を配布し、ObjectGridEventListener プラグインを使用して、eXtreme Scale 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詳しくは、「*製品概要*」に記載されているトランザクションの変更を配布する Java Message Service の使用に関する説明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、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。

このメカニズムを実装する必要はありません。この機能を使用するためのピアツーピア複製メカニズムが組み込まれています。「*管理ガイド*」で JMS でのピアツーピア複製の構成に関する説明を参照してください。

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、ObjectGrid で発生した変更をリモート Java 仮想マシン 内のピア ObjectGrids に簡単にパブリッシュし、それらの変更をその 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。ここでは、メッセージを伝搬するために Java Message Service (JMS) メッセージング・システムを使用するリスナーをどのように作成すればいいのかを詳しく見ていきます。eXtreme Scale トランザクションのコミットが WebSphere Application Server クラスターの複数メンバーにわたって実行された結果として生じる LogSequence の伝送を、eXtreme Scale は IBM 提供のプラグインによってサポートします。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・システムが必要となる可能性があります。

メカニズムの実装

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、配布するマップをフィルターに掛けることができます。このトピックのスニペットは、これらの API を JMS とともに使用して、ピアツーピア 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。

プラグインの初期化

ObjectGrid が開始するときに、ObjectGrid は、 ObjectGridEventListener インターフェース契約の一部である、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。

以下の例は初期化メソッドを示しています。

```
initialize method example
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 (Java SE) 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実装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。

変更の送信

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。このサンプルでは JMS を使用していますが、信頼でき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することもできます。

```
transactionEnd method example
// 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マップの名前が含まれている集合です。この変数が空の場合、すべてのマップがパブリッシュされます。
- publisher 変数: プラグインの initialize メソッド中に作成される TopicPublisher オブジェクトです。

更新メッセージの受信および適用

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。

JMS message run method example

```
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");
    }
```

JMS イベント・リスナー

JMSObjectGridEventListener は、クライアント・サイド・ニア・キャッシュの無効化およびピアツーピア複製メカニズムをサポートするように設計されています。これは、ObjectGridEventListener インターフェースの Java Message Service (JMS) 実装です。

クライアント無効化メカニズムは、クライアントのニア・キャッシュ・データがサーバーまたは他のクライアントと同期するように、分散 eXtreme Scale 環境で使用できます。この機能がないと、クライアントのニア・キャッシュに失効データが保持される可能性があります。ただし、この JMS ベースのクライアント無効化メカニズムを使用しても、クライアント・ニア・キャッシュを更新する場合の時間帯を考慮に入れる必要があります。実行時に更新の公開に遅延があるためです。

ピアツーピア複製メカニズムは、分散とローカルの両方の eXtreme Scale環境で使用できます。これは、ObjectGrid コアツーコア複製プロセスであり、これにより、ローカル ObjectGrid と分散 ObjectGrid の間で流れるデータ更新が可能になります。例えば、このメカニズムを使用すると、分散グリッドからローカル ObjectGrid に、あるいはあるグリッドから別のシステム・ドメインにある別のグリッドに、データ更新を移動させることができます。

JMSObjectGridEventListener の場合、ユーザーは、必要な JMS リソースを取得するために、JMS および Java Naming and Directory Interface (JNDI) 情報を構成する必要があります。 さらに、複製関連のプロパティーを正しく設定する必要があります。JEE 環境では、Web と Enterprise JavaBean (EJB) の両方のコンテナーで JNDI が使用可能になっている必要があります。この場合、外部 JMS リソースを取得したい場合を除いて JNDI プロパティーはオプションです。

このイベント・リスナーには、XML を使用するか、プログラマチックな方法を使用して構成できるプロパティーがあります。これは、クライアント無効化のみ、ピアツーピア複製のみ、またはその両方に使用できます。必要な機能を実現するための振る舞いをカスタマイズする場合は、ほとんどのプロパティーはオプションです。

詳しくは、JMSObjectGridEventListener API を参照してください。

JMSObjectGridEventListener プラグインの拡張

JMSObjectGridEventListener プラグインを使用すると、グリッド内のデータが変更または除去されたときに、ピア ObjectGrid インスタンスが更新を受信できます。また、eXtreme Scale グリッドからエントリーが更新または除去されたときに、クライアントが通知を受信することも可能です。このトピックでは、JMS メッセージが受信されたときに、アプリケーションが通知を受け取れるように、JMSObjectGridEventListener プラグインを拡張する方法について説明します。これは、クライアント無効化に CLIENT_SERVER_MODEL 設定を使用する場合に最も役立ちます。

receiver ロールで実行中の場合、JMSObjectGridEventListener インスタンスがグリッドから JMS メッセージ更新を受信すると、オーバーライドされた JMSObjectGridEventListener.onMessage メソッドが eXtreme Scale ランタイムによって自動的に呼び出されます。 これらのメッセージは、LogSequence オブジェクトの集まりを折り返します。LogSequence オブジェクトは onMessage メソッドに送られ、アプリケーションはこの LogSequence を使用して挿入、削除、更新、または無効化されたキャッシュ・エントリーを判別します。

onMessage 拡張ポイントを使用するために、アプリケーションは以下のステップを実行します。

1. JMSObjectGridEventListener クラスが拡張し、onMessage メソッドがオーバーライドする新規クラスを作成します。
2. ObjectGrid の ObjectGridEventListener と同様に拡張 JMSObjectGridEventListener を構成します。

拡張 JMSObjectGridEventListener クラスは、JMSObjectGridEventListener クラスの子クラスであり、initialize (オプション) と onMessage という 2 つのメソッドのみをオーバーライドできます。 JMSObjectGridEventListener クラスの子クラスが onMessage メソッド内の ObjectGrid や Session などの ObjectGrid 成果物を使用する必要がある場合、その子クラスは、initialize メソッドでその成果物を取得して、それをインスタンス変数としてキャッシュできます。 また、onMessage メソッドでは、キャッシュされた ObjectGrid 成果物は、渡された LogSequences の集まりを処理する場合に使用できます。

注: オーバーライドされた initialize メソッドは、親 JMSObjectGridEventListener を適切に初期化するために、super.initialize メソッドを呼び出す必要があります。

以下は、拡張 JMSObjectGridEventListener クラスの例です。

```
package com.ibm.websphere.samples.objectgrid.jms.price;

import java.util.*;
import com.ibm.websphere.objectgrid.*;
import com.ibm.websphere.objectgrid.plugins.LogElement;
import com.ibm.websphere.objectgrid.plugins.LogSequence;
import com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener;

public class ExtendedJMSObjectGridEventListener extends JMSObjectGridEventListener{
 protected static boolean debug = true;

    /**
     * This is the grid associated with this listener.
     */
    ObjectGrid grid;

    /**
     * This is the session associated with this listener.
     */
    Session session;

    String objectGridType;

    public List receivedLogSequenceList = new ArrayList();


 /* (non-Javadoc)
  * @see com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener
    #initialize(com.ibm.websphere.objectgrid.Session)
  */
 public void initialize(Session session) {
  // Note: if need to use any ObjectGrid artifact, this class need to get ObjectGrid
  //  from the passed Session instance and get ObjectMap from session instance
  // for any transactional ObjectGrid map operation.

  super.initialize(session);  // must invoke super's initialize method.
  this.session = session; // cache the session instance, in case need to
  //   use it to perform map operation.
  this.grid = session.getObjectGrid(); // get ObjectGrid, in case need
  //   to get ObjectGrid information.

  if (grid.getObjectGridType() == ObjectGrid.CLIENT)
   objectGridType = "CLIENT";
  else if (grid.getObjectGridType() == ObjectGrid.SERVER)
   objectGridType = "Server";

  if (debug)
   System.out.println("ExtendedJMSObjectGridEventListener[" +
     objectGridType + "].initialize() : grid = " + this.grid);
 }

 /* (non-Javadoc)
  * @see com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener
    #onMessage(java.util.Collection)
  */
 protected void onMessage(Collection logSequences) {
  System.out.println("ExtendedJMSObjectGridEventListener[" +
    objectGridType + "].onMessage(): ");

  Iterator iter = logSequences.iterator();

  while (iter.hasNext()) {
            LogSequence seq = (LogSequence) iter.next();

      StringBuffer buffer = new StringBuffer();
      String mapName = seq.getMapName();
      int size = seq.size();
      buffer.append("\nLogSequence[mapName=" + mapName + ", size=" + size + ",
     objectGridType=" + objectGridType
        + "]: ");

      Iterator logElementIter = seq.getAllChanges();
      for (int i = seq.size() - 1; i >= 0; --i) {
```

```
       LogElement le = (LogElement) logElementIter.next();
       buffer.append(le.getType() + " -> key=" + le.getCacheEntry().getKey() + ", ");
      }
      buffer.append("\n");

      receivedLogSequenceList.add(buffer.toString());

      if (debug) {
       System.out.println("ExtendedJMSObjectGridEventListener["
      + objectGridType + "].onMessage(): " + buffer.toString());
      }
  }
 }

 public String dumpReceivedLogSequenceList() {
  String result = "";
  int size = receivedLogSequenceList.size();
  result = result + "\nExtendedJMSObjectGridEventListener[" + objectGridType
    + "]: receivedLogSequenceList size = " + size + "\n";
  for (int i = 0; i < size; i++) {
   result = result + receivedLogSequenceList.get(i) + "\n";
  }
  return result;
 }

 public String toString() {
  return "ExtendedJMSObjectGridEventListener["
    + objectGridType + " - " + this.grid + "]";
 }
}
```

構成

拡張 JMSObjectGridEventListener クラスは、クライアント無効化の場合にも、ピアツーピア複製メカニズムの場合にも同様に構成する必要があります。以下は XML 構成の例です。

```
<objectGrid name="PRICEGRID">
   <bean id="ObjectGridEventListener"
    className="com.ibm.websphere.samples.objectgrid.jms.
      price.ExtendedJMSObjectGridEventListener">
    <property name="invalidationModel" type="java.lang.String"
     value="CLIENT_SERVER_MODEL" description="" />
    <property name="invalidationStrategy" type="java.lang.String"
     value="INVALIDATE" description="" />
    <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String"
     value="jms/TCF" description="" />
    <property name="jms_topicJndiName" type="java.lang.String"
     value="GRID.PRICEGRID" description="" />
    <property name="jms_topicName" type="java.lang.String"
     value="GRID.PRICEGRID" description="" />
    <property name="jms_userid" type="java.lang.String" value=""
     description="" />
    <property name="jms_password" type="java.lang.String" value=""
     description="" />
   </bean>
   <backingMap name="PRICE" pluginCollectionRef="PRICE"></backingMap>
      </objectGrid>
```

注: ObjectGridEventListener Bean の className は、一般 JMSObjectGridEventListener と同じプロパティーを持つ拡張 JMSObjectGridEventListener クラスによって構成されます。

ObjectGrid 記述子 XML ファイル

WebSphere eXtreme Scale を構成するには、ObjectGrid ディスクリプター XML ファイルおよび ObjectGrid API を使用します。

以下のセクションでは、さまざまな構成を解説するサンプル XML ファイルが提供されています。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。 ObjectGrid 記述子 XML スキーマを使用して、記述 XML ファイルを作成します。ObjectGrid 記述子 XML スキーマの例については、 169 ページの『objectGrid.xsd ファイル』 を参照してください。

オリジナルの companyGrid.xml ファイルを変更したバージョンが使用されます。次の companyGridSingleMap.xml ファイルは、companyGrid.xml ファイルと似ています。 companyGridSingleMap.xml ファイルにはマップが 1 つあり、companyGrid.xml ファイルにはマップが 4 つあります。ファイルのエレメントと属性については、この例に続いて詳しく説明します。

```
<?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="CompanyGrid">
      <backingMap name="Customer"/>
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

objectGridConfig 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、このエレメントを eXtreme Scale XML 文書に記述します。このエレメントは、ファイルの名前空間とスキーマ・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。

- 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

objectGrids 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。companyGridSingleMap.xml ファイルでは、objectGrids エレメントに CompanyGrid という objectGrid が 1 つ含まれています。

- 出現回数: 1 回以上
- 子エレメント: objectGrid エレメント

objectGrid エレメント

objectGrid エレメントを使用して 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。

- 出現回数: 1 回から複数回
- 子エレメント: Bean エレメント、backingMap エレメント、querySchema エレメント、および streamQuerySet エレメント

属性

name

ObjectGrid に割り当てられてい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。(必須)

securityEnabled
属性を true に設定したとき、ObjectGrid レベルのセキュリティーを使用可能にします。これにより、マップ内のデータに対するアクセス許可が使用可能になります。デフォルトは true です。 (オプション)

authorizationMechanism
エレメントの許可メカニズムを設定します。この属性は 2 つの値 (AUTHORIZATION_MECHANISM_JAAS または AUTHORIZATION_MECHANISM_CUSTOM) のいずれかに設定できます。デフォルトは AUTHORIZATION_MECHANISM_JAAS です。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。authorizationMechanism 属性を有効にするためには、securityEnabled 属性を true に設定する必要があります。(オプション)

permissionCheckPeriod
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。デフォルトは 0 です。属性値 0 を設定すると、すべての get、put、update、remove または evict メソッド呼び出しが許可メカニズム (Java Authentication and Authorization Service (JAAS) 権限またはカスタム権限のいずれか) に問い合わせ、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。permissionCheckPeriod 属性を有効にするためには、securityEnabled 属性を true に設定する必要があります。(オプション)

txTimeout
トランザクションを完了するのに許されている時間を秒で指定します。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。値 0 を設定すると、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。(オプション)

entityMetadataXMLFile
エンティティー・ディスクリプター XML ファイルへの相対パスを指定します。このパスは、Objectgrid® ディスクリプター・ファイルのロケーションを相対的に示します。この属性を XML ファイルで使用してエンティティー・スキーマを定義します。eXtreme Scale は、エンティティーを定義してから開始する必要があります。こうすることで、各エンティティーを BackingMap にバインドできます。(オプション)

```
<objectGrid
(1) name="objectGridName"
(2) securityEnabled="true" | "false"
(3) authorizationMechanism="AUTHORIZATION_MECHANISM_JASS" | "AUTHORIZATION_MECHANISM_CUSTOM"
(4)  permissionCheckPeriod="permission_check_period"
(5) txTimeout="seconds"
(6) entityMetadataXMLFile="URL"
/>
```

以下の例 (companyGridObjectGridAttr.xml ファイル) は、objectGrid エレメント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。セキュリティーは使用可能にされ、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。また、このファイルでは entityMetadataXMLFile 属性を指定してエンティティーを登録しています。

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

 <objectGrids>
  <objectGrid name="CompanyGrid" securityEnabled="true"
   authorizationMechanism="AUTHORIZATION_MECHANISM_JASS"
   permissionCheckPeriod="45"
   entityMetadataXMLFile="companyGridEntities.xml">
   <backingMap name="Customer"/>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridObjectGridAttr.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 companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);

companyGrid.setSecurityEnabled();
companyGrid.setAuthorizationMechanism(SecurityConstants.AUTHORIZATION_MECHANISM_JAAS);
companyGrid.setPermissionCheckPeriod(45);
companyGrid.registerEntities(new URL("file:companyGridEntities.xml"));
```

backingMap エレメント

backingMap エレメントは、ObjectGrid で BackingMap インスタンス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。詳しくは、「*プログラミング・ガイド*」の BackingMap インターフェースに関する情報を参照してください。

- 出現回数: 0 回から複数回
- 子エレメント: timeBasedDBUpdate エレメント

属性

name

backingMap インスタンスに割り当てられてい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。 (必須)

readOnly

この属性を false として指定すると、BackingMap インスタンスを読み取り/書き込みとして設定します。この属性を true として設定すると、BackingMap インスタンスは読み取り専用となります。 Session.getMap(String) を呼び出すと、メソッドに渡された名前が、この backingMap の name 属性に指定された正規表現に一致する場合、動的マップ作成が行われます。デフォルト値は false です。(オプション)

template

動的マップが使用できるかどうかを指定します。BackingMap マップがテンプレート・マップである場合、この値を true に設定します。テンプレート・マップを使用して、 ObjectGrid の開始後に動的にマップを作成できます。 Session.getMap(String) を呼び出すと、メソッドに渡された名前が、backingMap の name 属性に指定された正規表現に一致する場合、動的マップが作成されます。デフォルト値は false です。(オプション)

pluginCollectionRef

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。

一致する ID が存在しない場合、妥当性検査は失敗します。BackingMap プラグインを再利用するように、この属性を設定してください。(オプション)

numberOfBuckets

BackingMap インスタンスが使用するバケットの数を指定します。 BackingMap インスタンスは実装でハッシュ・マップを使用します。 BackingMap の中に複数のエントリーが存在する場合は、バケット数が増えると衝突のリスクが低減するため、バケットの数が多くなるほどパフォーマンスが向上します。また、バケットが多いと並行性も高くなります。値 `0` を設定すると、eXtreme Scale とのリモート通信中クライアントのニア・キャッシュが使用不可になります。クライアントでこの値を `0` に設定するときは、この値をクライアント・オーバーライド ObjectGrid XML ディスクリプター・ファイルでのみ設定してください。(オプション)

preloadMode

ローダー・プラグインがこの BackingMap インスタンス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。デフォルト値は `false` です。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、メソッドの実行がブロックされます。プリロードは初期化中に発生します。 (オプション)

lockStrategy

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ロック・マネージャーを使用するかどうかを指定します。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。デフォルト値は `OPTIMISTIC` です。 (オプション)

オプティミスティック・ロック・ストラテジーは、通常、ローダー・プラグインを持たないマップで使用します。このようなマップは大部分が既読で、書き込みや更新は頻繁には行われません。また、eXtreme Scale をサイド・キャッシュとして使用するパーシスタンス・マネージャーからも、アプリケーションからもロックは提供されません。コミット時に挿入、更新、または除去されるマップ・エントリー上では排他ロックが取得されます。コミット中のトランザクションがオプティミスティック・バージョン・チェックを実行している間、別のトランザクションによってバージョン情報が変更されないことが、ロックによって保証されます。

ペシミスティック・ロック・ストラテジーは、通常、ローダー・プラグインを持たないマップで使用します。また、eXtreme Scale をサイド・キャッシュとして使用するパーシスタンス・マネージャーからも、ローダー・プラグインからも、アプリケーションからもロックは提供されません。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・ロック・ストラテジーの失敗回数が多すぎる場合に使用されます。

ロックなしストラテジーは、内部ロック・マネージャーが不要であることを示します。 eXtreme Scale の外部では、eXtreme Scale 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、またはアプリケーション、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ローダー・プラグインのいずれかによって並行性制御が提供されます。

詳しくは、「プログラミング・ガイド」のマップ・エントリーのロックに関する説明を参照してください。

numberOfLockBuckets
ロック・マネージャーが BackingMap インスタンスのために使用するロック・バケットの数を設定します。 lockStrategy 属性を OPTIMISTIC または PESSIMISTIC に設定すると、BackingMap インスタンス用のロック・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、バケット数が増えるにつれて衝突のリスクが低減するため、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。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 属性を NONE に設定すると、BackingMap インスタンスがロック・マネージャーを使用しないことが指定されます。(オプション)

lockTimeout
ロック・マネージャーが BackingMap インスタンスのために使用するロック・タイムアウト設定します。 lockStrategy 属性を OPTIMISTIC または PESSIMISTIC に設定すると、BackingMap インスタンス用のロック・マネージャーが作成されます。デッドロックの発生を回避するために、ロック・マネージャーにはデフォルトのタイムアウト値 (15 秒) 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。 lockTimeout 属性を使用してデフォルトより大きい値を設定すれば、誤ったタイムアウト例外が発生しないようにすることができます。 lockStrategy 属性を NONE に設定すると、BackingMap インスタンスがロック・マネージャーを使用しないことが指定されます。(オプション)

CopyMode
BackingMap インスタンス内のエントリーの 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指定します。 CopyMode 属性を次の 5 つの値のいずれかに設定します。

COPY_ON_READ_AND_COMMIT
デフォルト値は COPY_ON_READ_AND_COMMIT です。値を COPY_ON_READ_AND_COMMIT に設定すると、アプリケーションが BackingMap インスタンス内にある値オブジェクトへの参照を持たないようにすることができます。代わりに、アプリケーションは BackingMap インスタンス内にある値のコピーを常に使用します。(オプション)

COPY_ON_READ
値を COPY_ON_READ に設定すると、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 値以上にパフォーマンスを向上させることができます。 BackingMap データの整合性を保持するため、アプリケーションは、トランザクションがコミットされた後、エントリーに対するすべての参照を削除します。この値を設定すると、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 エレメントに影響を与えないことを保証します。

COPY_ON_WRITE
値を COPY_ON_WRITE に設定すると、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たときに発生するコピーを除去することにより、COPY_ON_READ_AND_COMMIT 値以上にパフォーマンスを向上させることができます。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
値を NO_COPY に設定すると、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ようにすることができます。 EntityManager API エンティティーに関連付けられているマップの場合は、値を NO_COPY に設定してください。

COPY_TO_BYTES
値を COPY_TO_BYTES に設定すると、オブジェクトのコピー処理がコピー作成のシリアライゼーションに依存している場合に、複雑なオブジェクト・タイプのメモリー・フットプリントを改善し、パフォーマンスを改善します。オブジェクトが Cloneable でないか、あるいは、効率的な copyValue メソッドを使用するカスタム ObjectTransformer が指定されていない場合、デフォルトのコピー・メカニズムは、コピー作成のためにオブジェクトをシリアライズしてインフレーションします。
COPY_TO_BYTES 設定を使用すると、読み取り時にのみインフレーションが実行されて、コミット時にのみシリアライズが実行されます。

これらの設定について詳しくは、「*プログラミング・ガイド*」で CopyMode のベスト・プラクティスの情報を参照してください。

valueInterfaceClassName
CopyMode 属性を COPY_ON_WRITE に設定する際に必要なクラスを指定します。この属性は、それ以外のモードでは無視されます。ObjectMap.get メソッド呼び出しが行われると、COPY_ON_WRITE 値は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。 (オプション)

copyKey
マップ・エントリーの作成時にキーのコピーが必要であるかどうかを指定します。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。値を true に設定すると、マップ・エントリーの作成時にキー・オブジェクトがコピーされます。デフォルト値は false です。(オプション)

nullValuesSupported
値を true に設定すると、ObjectMap でヌル値がサポートされます。ヌル値がサポートされていると、NULL を戻す get 操作は、値が NULL であること、ま

たはメソッドに渡されるキーがマップに含まれていないことを意味する場合があります。デフォルト値は `true` です。(オプション)

ttlEvictorType
BackingMap エントリーの有効期限の時間を算出する方法を指定します。この属性は、`CREATION_TIME`、`LAST_ACCESS_TIME`、`LAST_UPDATE_TIME`、または `NONE` のいずれかの値に設定できます。`CREATION_TIME` 値の場合、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` 値の場合、エントリーの有効期限の時間は、エントリーの最終アクセス時間 (エントリーが更新された場合でも、読み取られただけの場合でも可) と timeToLive 属性値の合計になります。`LAST_UPDATE_TIME` 値の場合、エントリーの有効期限の時間は、このエントリーの最終更新時間と timeToLive 属性値の合計になります。`NONE` 値 (デフォルト) の場合、エントリーには有効期限がなく、アプリケーションによって明示的に除去または無効化されるまで、BackingMap 内に存続できることを示します。(オプション)

timeToLive
各マップ・エントリーの存続可能時間 (秒数) を指定します。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。その他の場合は、TTL Evictor がこの値に基づいてマップ・エントリーを除去します。(オプション)

streamRef
backingMap がストリーム・ソース・マップであることを指定します。backingMap に対するすべての挿入または更新は、ストリーム照会エンジンに対するストリーミング・イベントに変換されます。この属性は、streamQuerySet 内の有効なストリーム名を参照する必要があります。(オプション)

viewRef
backingMap がビュー・マップであることを指定します。ストリーム照会エンジンからのビュー出力は、eXtreme Scale タプル・フォーマットに変換され、マップに追加されます。(オプション)

writeBehind
後書きパラメーター (オプション) によって後書きサポートが使用可能になることを指定します。後書きパラメーターは、最大更新時間と最大キー更新カウントから成ります。後書きパラメーターの形式は `"[T(time)][;][C(count)]"` です。次のいずれかのイベントが発生すると、データベースが更新されます。

- 前回の更新が終わってから最大更新時間 (秒数で指定) が経過した。
- キュー・マップ内の使用可能な更新の数が最大更新カウントに達した。

詳しくは、 134 ページの『後書きキャッシング・サポート』を参照してください。

後書きサポートは、eXtreme Scale をデータベースに組み込む際に使用するローダー・プラグインの拡張機能です。例えば、JPA ローダーの構成については 246 ページの『JPA ローダーの構成』 の情報を参照してください。

evictionTriggers
追加使用する除去トリガーのタイプを設定します。バックアップ・マップ用の Evictor は、すべてこの追加トリガーのリストを使用します。IllegalStateException を回避するためには、ObjectGrid.initialize() メソッドを呼び

出す前にこの属性を呼び出す必要があります。また、ObjectGrid.initialize() メソッドがアプリケーションによってまだ呼び出されていない場合は、ObjectGrid.getSession() メソッドがこのメソッドを暗黙的に呼び出すので、注意してください。トリガー・リスト内のエントリーはセミコロンで区切られます。現在の除去トリガーには MEMORY_USAGE_THRESHOLD などがあります。(オプション)

```
<backingMap
(1)   name="objectGridName"
(2)   readOnly="true" | "false"
(3)   template="true" | "false"
(4)   pluginCollectionRef="reference to backingMapPluginCollection"
(5)   numberOfBuckets="number of buckets"
(6)   preloadMode="true" | "false"
(7)   lockStrategy="OPTIMISTIC" | "PESSIMISTIC" | "NONE"
(8)   numberOfLockBuckets="number of lock buckets"
(9)   lockTimeout="lock timeout"
(10)   copyMode="COPY_ON_READ_AND_COMMIT" | "COPY_ON_READ" | "COPY_ON_WRITE"
       | "NO_COPY" | "COPY_TO_BYTES"
(11)   valueInterfaceClassName="value interface class name"
(12)   copyKey="true" | "false"
(13)   nullValuesSupported="true" | "false"
(14)   ttlEvictorType="CREATION_TIME" | "LAST_ACCESS_TIME" | "LAST_UPDATE_TIME" | NONE"
(15)   timeToLive="time to live"
(16)   streamRef="reference to a stream"
(17)   viewRef="reference to a view"
(18)   writeBehind="write-behind parameters"
(19)   evictionTriggers="MEMORY_USAGE_THRESHOLD"
/>
```

次の例では、サンプルの backingMap 構成を例示するために companyGridBackingMapAttr.xml ファイルが使用されています。

```
<?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="CompanyGrid">
    <backingMap name="Customer" 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、前述の companyGridBackingMapAttr.xml ファイルの例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 companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);

BackingMap customerMap = companyGrid.defineMap("Customer");
customerMap.setReadOnly(true);
customerMap.setNumberOfBuckets(641);
customerMap.setPreloadMode(false);
customerMap.setLockStrategy(LockStrategy.OPTIMISTIC);
customerMap.setNumberOfLockBuckets(409);
customerMap.setLockTimeout(30);

// when setting copy mode to COPY_ON_WRITE, a valueInterface class is required
customerMap.setCopyMode(CopyMode.COPY_ON_WRITE,
 com.ibm.websphere.samples.objectgrid.CounterValueInterface.class);
customerMap.setCopyKey(true);
customerMap.setNullValuesSupported(false);
customerMap.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);
customerMap.setTimeToLive(3000); // set time to live to 50 minutes
```

Bean 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。プラグインを objectGrid エレメントおよび BackingMap エレメントに接続することができます。

- objectGrid エレメント内の出現回数: 0 回から複数回
- backingMapPluginCollection エレメント内の出現回数: 0 回から複数回
- 子エレメント: property エレメント

属性

id 作成するプラグインのタイプを指定します。(必須)

objectGrid エレメントの子エレメントである bean に有効なプラグインは次のリストのとおりです。

- 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

backingMapPluginCollection エレメントの子エレメントである bean に有効なプラグインは次のリストのとおりです。

- ローダー・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

className
プラグインを作成するためにインスタンス化するクラスまたは Spring Bean の名前を指定します。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。例えば、id 属性の値として `ObjectGridEventListener` を指定する場合は、className 属性値が ObjectGridEventListener インターフェースを実装するクラスを参照する必要があります。 (必須)

```
<bean
(1) id="TransactionCallback" | "ObjectGridEventListener" |"SubjectSource" |
    "MapAuthorization" | "SubjectValidation" | "Loader" | "ObjectTransformer" |
    "OptimisticCallback" | "Evictor" | "MapEventListener" | "MapIndexPlugin"
(2) className="class name" | "(spring)bean name"
/>
```

次の例では、Bean エレメントを使用するプラグインを構成する方法を示すために `companyGridBean.xml` ファイルが使用されています。 ObjectGridEventListener プラグインが CompanyGrid 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。

`companyGridBean.xml` ファイルでは BackingMap プラグインも定義されています。evictor プラグインが Customer BackingMap インスタンスに追加されています。bean

の ID が Evictor であるため、className 属性には 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する必要があります。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。

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

 <objectGrids>
  <objectGrid name="CompanyGrid">
   bean id="ObjectGridEventListener"
    className="com.ibm.websphere.objectgrid.plugins.builtins.TranPropListener"/>
   <backingMap name="Customer"
    pluginCollectionRef="customerPlugins"/>
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="customerPlugins">
   <bean id="Evictor"
     className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor/>
  </backingMapPluginCollection>
 </backingMapPluginCollections>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridBean.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 companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
TranPropListener tranPropListener = new TranPropListener();
companyGrid.addEventListener(tranPropListener);

BackingMap customerMap = companyGrid.defineMap("Customer");
Evictor lruEvictor = new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
customerMap.setEvictor(lruEvictor);
```

property エレメント

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。このプロパティーの名前は、このプロパティーを含む Bean が参照するクラスの set メソッドに対応している必要があります。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

name

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、このプロパティーを含む Bean の className 属性で指定されたクラスの 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 属性を com.ibm.MyPlugin に設定し、指定したプロパティーの名前が size である場合、com.ibm.MyPlugin クラスに setSize メソッドが必要です。(必須)

type

プロパティーのタイプを指定します。このタイプは、name 属性により識別される set メソッドに受け渡されます。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、名前を size と設定し、タイプを int と設定す

る場合は、Bean の className 属性として指定されたクラスに setSize(int) メソッドが存在している必要があります。 (必須)

value
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。 (必須)

description
プロパティーの説明です。(オプション)

```
<bean
(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"
/>
```

以下の例では、companyGridProperty.xml ファイルを使用して、Bean に property エレメントを追加する方法を示しています。この例では、maxSize という名前で int 型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。

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

 <objectGrids>
  <objectGrid name="CompanyGrid">
   <backingMap name="Customer"
    pluginCollectionRef="customerPlugins"/>
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="customerPlugins">
   <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・サンプルは、前の例の companyGridProperty.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);

BackingMap customerMap = companyGrid.defineMap("Customer");

LRUEvictor lruEvictor = new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
// if the XML file is used instead,
// the property that was added would cause the following call to occur
lruEvictor.setMaxSize(449);
customerMap.setEvictor(lruEvictor);
```

backingMapPluginsCollections エレメント

backingMapPluginsCollections エレメントは、すべての backingMapPluginCollection エレメントのコンテナーです。前のセクションにある companyGridProperty.xml ファイルでは、backingMapPluginCollections エレメントに customerPlugins という ID の backingMapPluginCollection エレメントが 1 つ含まれています。

- 出現回数: 0 回から 1 回
- 子エレメント: backingMapPluginCollection エレメント

backingMapPluginCollection エレメント

backingMapPluginCollection エレメントは、BackingMap プラグインを定義し、**id** 属性によって識別されます。 pluginCollectionRef 属性を指定して、このプラグインを参照します。いくつかの BackingMaps プラグインを同様の方法で構成すると、各 BackingMap は同じ backingMapPluginCollection エレメントを参照できます。

- 出現回数: 0 回から複数回
- 子エレメント: Bean エレメント

属性

id backingMapPluginCollection を識別し、backingMap エレメントの pluginCollectionRef 属性によって参照されます。各 ID は固有である必要があります。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。 (必須)

```
<backingMapPluginCollection
(1)   id="id"
/>
```

以下の例では、companyGridCollection.xml ファイルを使用して、backingMapPluginCollection エレメントの使用法を示しています。このファイルでは、Customer BackingMap が customerPlugins backingMapPluginCollection を使用して、LRUEvictor を使用する Customer BackingMap を構成します。 Item および OrderLine の BackingMap は、collection2 backingMapPluginCollection を参照します。これらの BackingMap には、それぞれ LFUEvictor のセットが含まれています。

```
<?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="CompanyGrid">
   <backingMap name="Customer"
    pluginCollectionRef="customerPlugins"/>
   <backingMap name="Item" pluginCollectionRef="collection2"/>
   <backingMap name="OrderLine"
    pluginCollectionRef="collection2"/>
   <backingMap name="Order"/>
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="customerPlugins">
   <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・サンプルは、前の例の companyGridCollection.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();

ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
BackingMap customerMap = companyGrid .defineMap("Customer");
LRUEvictor customerEvictor = new LRUEvictor();
customerMap.setEvictor(customerEvictor);

BackingMap itemMap = companyGrid.defineMap("Item");
LFUEvictor itemEvictor = new LFUEvictor();
itemMap.setEvictor(itemEvictor);

BackingMap orderLineMap = companyGrid.defineMap("OrderLine");
LFUEvictor orderLineEvictor = new LFUEvictor();
orderLineMap.setEvictor(orderLineEvictor);

BackingMap orderMap = companyGrid.defineMap("Order");
```

querySchema エレメント

querySchema エレメントは、BackingMap 間のリレーションシップを定義し、各マップ内にあるオブジェクトのタイプを識別します。この情報は、照会言語ストリングをマップ・アクセス呼び出しに変換するために ObjectQuery によって使用されます。

- 出現回数: 0 回から 1 回
- 子エレメント: mapSchemas エレメント、relationships エレメント

mapSchemas エレメント

各 querySchema エレメントは、1 つ以上の mapSchema エレメントを含む mapSchemas エレメントを 1 つ持ちます。

- 出現回数: 1 回
- 子エレメント: mapSchema エレメント

mapSchema エレメント

mapSchema エレメントは、BackingMap に保管されるオブジェクトのタイプ、およびそのデータにアクセスする方法を定義します。

- 出現回数: 1 回以上
- 子エレメント: なし

属性

mapName
スキーマに追加する BackingMap の名前を指定します。(必須)

valueClass
BackingMap の値部分に保管されるオブジェクトのタイプを指定します。(必須)

primaryKeyField

valueClass 属性内の 1 次キー属性の名前を指定します。1 次キーも BackingMap のキー部分に保管する必要があります。(オプション)

accessType

照会エンジンが valueClass オブジェクト・インスタンス内の永続データをイントロスペクトしてそのデータにアクセスする方法を示します。値を FIELD に設定すると、クラスのフィールドがイントロスペクトされ、スキーマに追加されます。値が PROPERTY の場合、get メソッドおよび is メソッドに関連付けられた属性が使用されます。デフォルト値は PROPERTY です。 (オプション)

```
<mapSchema
(1)   mapName="backingMapName"
(2)   valueClass="com.mycompany.OrderBean"
(3)   primaryKeyField="orderId"
(4)   accessType="PROPERTY" | "FIELD"
/>
```

以下の例では、companyGridQuerySchemaAttr.xml ファイルを使用して、サンプル mapSchema 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="CompanyGrid">
   <backingMap name="Order"/>
   <backingMap name="Customer"/>

   <querySchema>
    <mapSchemas>
     <mapSchema mapName="Order"
      valueClass="com.mycompany.OrderBean"
      primaryKeyField="orderNumber"
      accessType="FIELD"/>
     <mapSchema mapName="Customer"
      valueClass="com.mycompany.CustomerBean"
      primaryKeyField="id"
      accessType="FIELD"/>
    </mapSchemas>
   </querySchema>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridQuerySchemaAttr.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
companyGrid.defineMap("Order");
companyGrid.defineMap("Customer");

// Define the schema
QueryConfig queryCfg = new QueryConfig();
queryCfg.addQueryMapping(new QueryMapping(
    "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryMapping(new QueryMapping(
    "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
companyGrid.setQueryConfig(queryCfg);
```

relationships エレメント

各 querySchema エレメントは、relationships エレメントをまったく持たないか (ゼロ)、または 1 つ以上の relationship エレメントを含む relationships エレメントを 1 つ持ちます。

- 出現回数: 0 回または 1 回
- 子エレメント: relationship エレメント

relationship エレメント

relationship エレメントは、2 つの BackingMap 間のリレーションシップ、およびそのリレーションシップをバインドする valueClass 属性の属性を定義します。

- 出現回数: 1 回以上
- 子エレメント: なし

属性

source
リレーションシップのソース側 valueClass の名前を指定します。 (必須)

target
リレーションシップのターゲット側 valueClass の名前を指定します。 (必須)

relationField
ソース valueClass 内でターゲットを参照している属性の名前を指定します。(必須)

invRelationField
ターゲット valueClass 内でソースを参照している属性の名前を指定します。この属性が指定されていないと、リレーションシップは単一方向となります。(オプション)

```
<mapSchema
(1)   source="com.mycompany.OrderBean"
(2)   target="com.mycompany.CustomerBean"
(3)   relationField="customer"
(4)   invRelationField="orders"
/>
```

以下の例では、companyGridQuerySchemaWithRelationshipAttr.xml ファイルを使用して、双方向リレーションシップを含むサンプル mapSchema 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="CompanyGrid">
   <backingMap name="Order"/>
   <backingMap name="Customer"/>

   <querySchema>
    <mapSchemas>
     <mapSchema mapName="Order"
      valueClass="com.mycompany.OrderBean"
      primaryKeyField="orderNumber"
      accessType="FIELD"/>
     <mapSchema mapName="Customer"
      valueClass="com.mycompany.CustomerBean"
      primaryKeyField="id"
      accessType="FIELD"/>
    </mapSchemas>
    <relationships>
     <relationship
      source="com.mycompany.OrderBean"
      target="com.mycompany.CustomerBean"
      relationField="customer"/>
      invRelationField="orders"/>
    </relationships>
```

```
   </querySchema>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridQuerySchemaWithRelationshipAttr.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
companyGrid.defineMap("Order");
companyGrid.defineMap("Customer");

// Define the schema
QueryConfig queryCfg = new QueryConfig();
queryCfg.addQueryMapping(new QueryMapping(
    "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryMapping(new QueryMapping(
    "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryRelationship(new QueryRelationship(
     OrderBean.class.getName(), CustomerBean.class.getName(), "customer", "orders"));
companyGrid.setQueryConfig(queryCfg);
```

streamQuerySet エレメント

streamQuerySet エレメントは、ストリーム照会セットを定義するための最上位レベルのエレメントです。

- 出現回数: 0 回から複数回
- 子エレメント: stream エレメント、view エレメント

stream エレメント

stream エレメントは、ストリーム照会エンジンへのストリームを表します。stream エレメントの各属性は、StreamMetadata インターフェースのメソッドに対応します。

- 出現回数: 1 回から複数回
- 子エレメント: basic エレメント

属性

name
ストリームの名前を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

valueClass
このストリーム ObjectMap に保管される値のクラス・タイプを指定します。このクラス・タイプは、オブジェクトをストリーム・イベントに変換し、SQL ステートメントが指定されていない場合は SQL ステートメントを生成するために使用されます。 (必須)

sql ストリームの SQL ステートメントを指定します。このプロパティーを指定しなかった場合、valueClass 属性で属性または accessor メソッドのリフレクション、またはエンティティー・メタデータの tuple 属性を使用してストリーム SQL が生成されます。(オプション)

access
値クラスの属性にアクセスするためのタイプを指定します。この値を FIELD に設定すると、属性は Java のリフレクションを使用してフィールドから直接取り

出されます。そうでない場合、属性は accessor メソッドを使用して読み取られます。デフォルト値は PROPERTY です。 (オプション)

```
<stream
(1) name="streamName"
(2) valueClass="streamMapClassType"
(3) sql="streamSQL create stream stockQuote
     keyed by t ( transactionvolume INTEGER, price DECIMAL (9,2), issue VARCHAR(100) );"
(4) access="PROPERTY" | "FIELD"
/>
```

view エレメント

view エレメントはストリーム照会ビューを表します。各 stream エレメントが ViewMetadata インターフェースのメソッドに対応します。

- 出現回数: 1 回から複数回
- 子エレメント: basic エレメント、id エレメント

属性

name
ビューの名前を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

sql ビューの変換を定義する、ストリームの SQL を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

valueClass
ObjectMap のこのビューに保管される値のクラス・タイプを指定します。このクラス・タイプは、ビュー・イベントをこのクラス・タイプと互換性のある適切なタプル・フォーマットに変換するのに使用されます。クラス・タイプを指定しなかった場合、Stream Processing Technology Structured Query Language (SPTSQL) の列定義に従ってデフォルトのフォーマットが使用されます。このビュー・マップにエンティティー・メタデータが定義されている場合には、 valueClass 属性を使用しないでください。(オプション)

access
値クラスの属性にアクセスするためのタイプを指定します。アクセス・タイプを FIELD に設定すると、列値は、Java のリフレクションを使用して直接にフィールドに設定されます。そうでない場合、accessor メソッドを使用して属性が設定されます。デフォルト値は PROPERTY です。 (オプション)

```
<view
(1)   name="viewName"
(2)   valueClass="viewMapValueClass"
(3)   sql="viewSQL CREATE VIEW last5MinuteAvgPrice AS
      SELECT issue, avg(price) as totalVolume
        FROM (SELECT * FROM stockQuote FETCH LATEST 5 MINUTES) group by issue;"/>
(4)   access="PROPERTY" | "FIELD"
/>
```

basic エレメント

basic エレメントは、値クラスまたはエンティティー・メタデータの属性名から SPTSQL で定義された列へのマッピングを定義するために使用します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

```
<basic
(1)   name="attributeName"
(2)   column="columnName"
/>
```

id エレメント

id エレメントは、キー属性のマッピングに使用されます。

- 出現回数: 0 回から複数回
- 子エレメント: なし

```
<id
(1)   name="idName"
(2)   column="columnName"
/>
```

以下の例では、StreamQueryApp2.xml ファイルを使用して、streamQuerySet の属性を構成する方法を示しています。ストリーム照会セット _stockQuoteSQS_ にはストリームおよびビューが 1 つずつあります。ストリームおよびビューの両方で、名前、valueClass、sql、およびアクセス・タイプが定義されています。ストリームは basic エレメントも定義します。これにより、StockQuote クラスの volume 属性が、SQL ステートメントで定義される SQL 列トランザクション・ボリュームにマップされることが指定されます。

```
<?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="og1">
   <backingMap name="stockQuote" readOnly="false" copyKey="true" streamRef="stockQuote"/>
   <backingMap name="last5MinuteAvgPrice" readOnly="false" copyKey="false"
    viewRef="last5MinuteAvgPrice"/>

   <streamQuerySet name="stockQuoteSQS">
    <stream
     name="stockQuote"
     valueClass="com.ibm.ws.objectgrid.streamquery.sample.guide.StockQuote"
     sql="create stream stockQuote
         keyed by t ( transactionvolume INTEGER, price DECIMAL (9,2), issue VARCHAR(100) );"
     access="FIELD">
     <basic name="volume" column="transactionvolume"/>
    </stream>

    <view
     name="last5MinuteAvgPrice"
     valueClass="com.ibm.ws.objectgrid.streamquery.sample.guide.AveragePrice"
     sql="CREATE VIEW last5MinuteAvgPrice AS SELECT issue, avg(price) as avgPrice
       FROM (SELECT * FROM stockQuote FETCH LATEST 5 MINUTES) group by issue;"
     access="FIELD"
    </view>
   </streamQuerySet>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

objectGrid.xsd ファイル

ObjectGrid 記述子 XML スキーマを使用して、WebSphere eXtreme Scale を構成します。

objectGrid.xsd ファイルに定義されるエレメントおよび属性の説明は、151 ページの『ObjectGrid 記述子 XML ファイル』を参照してください。 Spring の objectgrid.xsd ファイルについては、293 ページの『Spring 記述子 XML ファイル』を参照してください。

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

 <xsd:element name="objectGridConfig">
  <xsd:complexType>
   <xsd:sequence>
    <xsd:element maxOccurs="1" minOccurs="1" name="objectGrids"
       type="dgc:objectGrids">
     <xsd:unique name="objectGridNameUnique">
      <xsd:selector xpath="dgc:objectGrid"/>
      <xsd:field xpath="@name"/>
     </xsd:unique>
    </xsd:element>
    <xsd:element maxOccurs="1" minOccurs="0" name="backingMapPluginCollections"
       type="dgc:backingMapPluginCollections"/>
   </xsd:sequence>
  </xsd:complexType>

  <xsd:key name="backingMapPluginCollectionId">
   <xsd:selector xpath="dgc:backingMapPluginCollections/dgc:
     backingMapPluginCollection"/>
   <xsd:field xpath="@id"/>
  </xsd:key>

  <xsd:keyref name="pluginCollectionRef" refer="dgc:backingMapPluginCollectionId">
   <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:backingMap"/>
   <xsd:field xpath="@pluginCollectionRef"/>
  </xsd:keyref>

  <xsd:key name="streamName">
   <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:
     streamQuerySet/dgc:stream"/>
   <xsd:field xpath="@name"/>
  </xsd:key>

  <xsd:keyref name="streamRef" refer="dgc:streamName">
   <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:backingMap"/>
   <xsd:field xpath="@streamRef"/>
  </xsd:keyref>

  <xsd:key name="viewName">
   <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:streamQuerySet/dgc:view"/>
   <xsd:field xpath="@name"/>
  </xsd:key>

  <xsd:keyref name="viewRef" refer="dgc:viewName">
   <xsd:selector xpath="dgc:objectGrids/dgc:objectGrid/dgc:backingMap"/>
   <xsd:field xpath="@viewRef"/>
  </xsd:keyref>
 </xsd:element>

 <xsd:complexType name="objectGrids">
  <xsd:sequence>
   <xsd:element maxOccurs="unbounded" minOccurs="1" name="objectGrid"
      type="dgc:objectGrid">
    <xsd:unique name="backingMapNameUnique">
     <xsd:selector xpath="dgc:backingMap"/>
     <xsd:field xpath="@name"/>
    </xsd:unique>
    <xsd:unique name="streamQuerySetNameUnique">
     <xsd:selector xpath="dgc:streamQuerySet"/>
     <xsd:field xpath="@name"/>
    </xsd:unique>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="backingMapPluginCollections">
  <xsd:sequence>
   <xsd:element maxOccurs="unbounded" minOccurs="0" name="backingMapPluginCollection"
     type="dgc:backingMapPluginCollection"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="objectGrid">
  <xsd:sequence>
```

```
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="bean" type="dgc:bean"/>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="backingMap"
     type="dgc:backingMap"/>
  <xsd:element maxOccurs="1" minOccurs="0" name="querySchema" type="dgc:querySchema"/>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="streamQuerySet"
     type="dgc:streamQuerySet">
   <xsd:unique name="stream">
    <xsd:selector xpath="dgc:stream"/>
    <xsd:field xpath="@name"/>
   </xsd:unique>
   <xsd:unique name="view">
    <xsd:selector xpath="dgc:view"/>
    <xsd:field xpath="@name"/>
   </xsd:unique>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="authorizationMechanism" type="dgc:authorizationMechanism"
   use="optional"/>
 <xsd:attribute name="accessByCreatorOnlyMode" type="dgc:accessByCreatorOnlyMode"
   use="optional"/>
 <xsd:attribute name="securityEnabled" type="xsd:boolean" use="optional"/>
 <xsd:attribute name="txTimeout" type="xsd:int" use="optional"/>
 <xsd:attribute name="permissionCheckPeriod" type="xsd:int" use="optional"/>
 <xsd:attribute name="entityMetadataXMLFile" type="xsd:string" use="optional"/>
 <xsd:attribute name="initialState" type="dgc:initialState" use="optional"/>
</xsd:complexType>


<xsd:complexType name="backingMap">
 <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="0" name="timeBasedDBUpdate" type="dgc:
    timeBasedDBUpdate"/>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="readOnly" type="xsd:boolean" use="optional"/>
 <xsd:attribute name="pluginCollectionRef" type="xsd:string" use="optional"/>
 <xsd:attribute name="preloadMode" type="xsd:boolean" use="optional"/>
 <xsd:attribute name="lockStrategy" type="dgc:lockStrategy" use="optional"/>
 <xsd:attribute name="copyMode" type="dgc:copyMode" use="optional"/>
 <xsd:attribute name="valueInterfaceClassName" type="xsd:string" use="optional"/>
 <xsd:attribute name="numberOfBuckets" type="xsd:int" use="optional"/>
 <xsd:attribute name="nullValuesSupported" type="xsd:boolean" use="optional"/>
 <xsd:attribute name="lockTimeout" type="xsd:int" use="optional"/>
 <xsd:attribute name="numberOfLockBuckets" type="xsd:int" use="optional"/>
 <xsd:attribute name="copyKey" type="xsd:boolean" use="optional"/>
 <xsd:attribute name="timeToLive" type="xsd:int" use="optional"/>
 <xsd:attribute name="ttlEvictoryType" type="dgc:ttlEvictorType" use="optional"/>
 <xsd:attribute name="streamRef" type="xsd:string" use="optional"/>
 <xsd:attribute name="viewRef" type="xsd:string" use="optional"/>
 <xsd:attribute name="writeBehind" type="xsd:string" use="optional"/>
 <xsd:attribute name="evictionTriggers" type="xsd:string" use="optional"/>
 <xsd:attribute name="template" type="xsd:boolean" use="optional"/>
</xsd:complexType>

<xsd:complexType name="bean">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="property" type="dgc:property"/>
 </xsd:sequence>
 <xsd:attribute name="className" type="xsd:string" use="required"/>
 <xsd:attribute name="id" type="dgc:beanId" use="required"/>
</xsd:complexType>

<xsd:complexType name="backingMapPluginCollection">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="bean" type="dgc:bean"/>
 </xsd:sequence>
 <xsd:attribute name="id" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="property">
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="value" type="xsd:string" use="required"/>
 <xsd:attribute name="type" type="dgc:propertyType" use="required"/>
 <xsd:attribute name="description" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:simpleType name="propertyType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="java.lang.Boolean"/>
```

```
  <xsd:enumeration value="boolean"/>
  <xsd:enumeration value="java.lang.String"/>
  <xsd:enumeration value="java.lang.Integer"/>
  <xsd:enumeration value="int"/>
  <xsd:enumeration value="java.lang.Double"/>
  <xsd:enumeration value="double"/>
  <xsd:enumeration value="java.lang.Byte"/>
  <xsd:enumeration value="byte"/>
  <xsd:enumeration value="java.lang.Short"/>
  <xsd:enumeration value="short"/>
  <xsd:enumeration value="java.lang.Long"/>
  <xsd:enumeration value="long"/>
  <xsd:enumeration value="java.lang.Float"/>
  <xsd:enumeration value="float"/>
  <xsd:enumeration value="java.lang.Character"/>
  <xsd:enumeration value="char"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="beanId">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="TransactionCallBack"/>
  <xsd:enumeration value="ObjectGridEventListener"/>
  <xsd:enumeration value="SubjectSource"/>
  <xsd:enumeration value="MapAuthorization"/>
  <xsd:enumeration value="SubjectValidation"/>
  <xsd:enumeration value="ObjectGridAuthorization"/>

  <xsd:enumeration value="Loader"/>
  <xsd:enumeration value="ObjectTransformer"/>
  <xsd:enumeration value="OptimisticCallback"/>
  <xsd:enumeration value="Evictor"/>
  <xsd:enumeration value="MapEventListener"/>
  <xsd:enumeration value="MapIndexPlugin"/>

 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="copyMode">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="COPY_ON_READ_AND_COMMIT"/>
  <xsd:enumeration value="COPY_ON_READ"/>
  <xsd:enumeration value="COPY_ON_WRITE"/>
  <xsd:enumeration value="NO_COPY"/>
  <xsd:enumeration value="COPY_TO_BYTES"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="lockStrategy">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="OPTIMISTIC"/>
  <xsd:enumeration value="PESSIMISTIC"/>
  <xsd:enumeration value="NONE"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ttlEvictorType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="CREATION_TIME"/>
  <xsd:enumeration value="LAST_ACCESS_TIME"/>
      <xsd:enumeration value="LAST_UPDATE_TIME"/>
  <xsd:enumeration value="NONE"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="authorizationMechanism">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="AUTHORIZATION_MECHANISM_JAAS"/>
  <xsd:enumeration value="AUTHORIZATION_MECHANISM_CUSTOM"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="accessByCreatorOnlyMode">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="disabled"/>
  <xsd:enumeration value="complement"/>
  <xsd:enumeration value="supersede"/>
 </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:complexType name="streamQuerySet">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="stream" type="dgc:stream">
   <xsd:unique name="streamBasicColumnUnique">
    <xsd:selector xpath="dgc:basic"/>
    <xsd:field xpath="@column"/>
   </xsd:unique>
  </xsd:element>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="view" type="dgc:view">
   <xsd:unique name="viewBasicColumnUnique">
    <xsd:selector xpath="dgc:basic"/>
    <xsd:field xpath="@column"/>
   </xsd:unique>
  </xsd:element>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="viewResultsToListenersOnly" type="xsd:boolean"
    default="false" use="optional"/>
 <xsd:attribute name="deployInPrimaryOnly" type="xsd:boolean" default="true"
    use="optional"/>
</xsd:complexType>

<xsd:complexType name="stream">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="basic" type="dgc:basic"/>
 </xsd:sequence>
 <xsd:attribute name="valueClass" type="xsd:string" use="required"/>
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="sql" type="xsd:string" use="optional"/>
 <xsd:attribute name="access" type="cc:accessType" use="optional"/>
</xsd:complexType>

<xsd:complexType name="view">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="0" name="id" type="dgc:basic"/>
  <xsd:element element maxOccurs="unbounded" minOccurs="0" name="basic"
    type="dgc:basic"/>
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="sql" type="xsd:string" use="optional"/>
 <xsd:attribute name="valueClass" type="xsd:string" use="optional"/>
 <xsd:attribute name="access" type="cc:accessType" use="optional"/>
</xsd:complexType>

<xsd:complexType name="basic">
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="column" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="id">
 <xsd:attribute name="name" type="xsd:string" use="required"/>
 <xsd:attribute name="column" type="xsd:string" use="required"/>
</xsd:complexType>

<xsd:complexType name="timeBasedDBUpdate">
 <xsd:attribute name="persistenceUnitName" type="xsd:string" use="optional"/>
 <xsd:attribute name="mode" type="cc:dbUpdateMode" use="optional"/>
 <xsd:attribute name="timestampField" type="xsd:string" use="optional"/>
 <xsd:attribute name="entityClass" type="xsd:string" use="required"/>
 <xsd:attribute name="jpaPropertyFactory" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:simpleType name="dbUpdateMode">
 <xsd:restriction base="xsd:string"/>
  <xsd:enumeration value="INVALIDATE_ONLY"/>
  <xsd:enumeration value="UPDATE_ONLY"/>
  <xsd:enumeration value="INSERT_UPDATE"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="querySchema">
 <xsd:sequence>
  <xsd:element maxOccurs="1" minOccurs="1" name="mapSchemas" type="dgc:mapSchemas">
   <xsd:unique name="mapNameUnique">
    <xsd:selector xpath="dgc:mapSchema"/>
    <xsd:field xpath="@mapName"/>
   </xsd:unique>
  </xsd:element>
  <xsd:element maxOccurs="1" minOccurs="0" name="relationships"
    type="dgc:relationships"/>
```

```
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="mapSchemas">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="1" name="mapSchema"
    type="dgc:mapSchema"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="relationships">
 <xsd:sequence>
  <xsd:element maxOccurs="unbounded" minOccurs="1" name="relationship"
    type="dgc:relationship"/>
 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="mapSchema">
 <xsd:attribute name="mapName" type="xsd:string" use="required"/>
 <xsd:attribute name="valueClass" type="xsd:string" use="required"/>
 <xsd:attribute name="primaryKeyField" type="xsd:string" use="optional"/>
 <xsd:attribute name="accessType" type="cc:accessType" use="optional"/>
</xsd:complexType>

<xsd:complexType name="relationship">
 <xsd:attribute name="source" type="xsd:string" use="required"/>
 <xsd:attribute name="target" type="xsd:string" use="required"/>
 <xsd:attribute name="relationField" type="xsd:string" use="required"/>
 <xsd:attribute name="invRelationField" type="xsd:string" use="optional"/>
</xsd:complexType>

<xsd:simpleType name="accessType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="PROPERTY"/>
  <xsd:enumeration value="FIELD"/>
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="initialState">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="OFFLINE"/>
  <xsd:enumeration value="PRELOAD"/>
  <xsd:enumeration value="ONLINE"/>
 </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

デプロイメント・ポリシーの構成

デプロイメント・ポリシー記述子 XML ファイルおよび objectgrid 記述子 XML ファイルを使用して、分散トポロジーを管理します。デプロイメント・ポリシーは、eXtreme Scale コンテナーに提供される XML ファイルとしてエンコードされます。デプロイメント・ポリシーは、マップ、マップ・セット、区画、レプリカなどの情報を提供します。また、共有配置の動作も制御します。

分散デプロイメントの構成

デプロイメント・ポリシー記述子 XML ファイルおよび objectgrid 記述子 XML ファイルを使用して、トポロジーを管理します。

デプロイメント・ポリシーは、eXtreme Scale コンテナーに提供される XML ファイルとしてエンコードされます。XML ファイルでは、以下の情報が指定されます。

- 各マップ・セットに属するマップ
- 区画の数
- 同期複製および非同期複製の数

コンテナー・サーバーの始動については、 eXtreme Scale コンテナーの自動始動または コンテナー・プロセスの開始を参照してください。

デプロイメント・ポリシーでは、以下の配置の振る舞いも制御されます。

- 配置を実行する前のアクティブ・コンテナーの最小数
- 破損した断片の自動置き換え
- 単一区画の各断片の別のマシンへの配置

ポリシー構成の詳細については、デプロイメント・ポリシー記述子 XML ファイルを参照してください。

エンドポイント情報は、動的環境では事前構成されません。デプロイメント・ポリシーには、サーバー名も物理トポロジー情報もありません。グリッド内のすべての断片は、カタログ・サービスによって自動的にコンテナーに配置されます。カタログ・サービスは、デプロイメント・ポリシーで定義されている制約を使用して、断片配置を自動的に管理します。この自動断片配置により、大きなグリッドの構成が容易になります。また必要に応じて、使用している環境にサーバーを追加することもできます。

制約事項: WebSphere Application Server 環境では、50 を超えるメンバーが入っているコア・グループ・サイズはサポートされません。

デプロイメント・ポリシー XML ファイルは、始動時に eXtreme Scale コンテナーに渡されます。デプロイメント・ポリシーは、ObjectGrid XML ファイルと一緒に使用する必要があります。デプロイメント・ポリシーはコンテナーの始動には不要ですが、使用されることをお勧めします。デプロイメント・ポリシーは、それと一緒に使用される ObjectGrid XML ファイルと互換性がある必要があります。デプロイメント・ポリシー内の各 objectgridDeployment エレメントごとに、対応する 1 つの objectGrid エレメントが ObjectGrid XML ファイル内に必要です。objectgridDeployment 内のマップは、ObjectGrid XML 内の backingMap エレメントと整合している必要があります。すべての backingMap は、1 つの mapSet エレメント内のみで参照する必要があります。

以下の例では、companyGridDpReplication.xml ファイルは、対応する companyGrid.xml ファイルとペアになっていることが想定されています。

companyGridDpReplication.xml

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

 <objectgridDeployment objectgridName="CompanyGrid">
  <mapSet name="mapSet1" numberOfPartitions="11"
   minSyncReplicas="1" maxSyncReplicas="1"
   maxAsyncReplicas="0" numInitialContainers="4">
   <map ref="Customer" />
   <map ref="Item" />
   <map ref="OrderLine" />
   <map ref="Order" />
  </mapSet>
 </objectgridDeployment>

</deploymentPolicy>
```

companyGrid.xml

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

 <objectGrids>
  <objectGrid name="CompanyGrid">
```

```
    <backingMap name="Customer" />
    <backingMap name="Item" />
    <backingMap name="OrderLine" />
    <backingMap name="Order" />
   </objectGrid>
  </objectGrids>

</objectGridConfig>
```

companyGridDpReplication.xml ファイルには、11 個の区画に分割されている mapSet エレメントが 1 つあります。各区画に含めることができる同期複製は 1 つだけです。同期レプリカの数は、minSyncReplicas 属性および maxSyncReplicas 属性によって指定します。 minSyncReplicas 属性が 1 に設定されているため、書き込みトランザクションを処理するためには、 mapSet エレメント内の各区画では、少なくとも 1 つの同期レプリカが使用可能になっている必要があります。maxSyncReplicas が 1 に設定されているため、各区画の同期レプリカの数は 1 を超えることはできません。この mapSet エレメント内の区画には、非同期レプリカは含まれていません。

カタログ・サービスは、この ObjectGrid インスタンスをサポートするために、numInitialContainers 属性に従って、4 つのコンテナーが使用可能になるまで配置を据え置きます。 numInitialContainers 属性は、コンテナーが指定数に到達した後は、無視されます。

companyGridDpReplication.xml ファイルは基本的な例ですが、デプロイメント・ポリシーによって、 eXtreme Scale 環境を完全に制御できます。 デプロイメント・ポリシー記述子 XML ファイル を参照してください。

分散トポロジー

分散コヒーレント・キャッシュを使用すると、構成できるパフォーマンス、可用性、およびスケーラビリティーが改善されます。

WebSphere eXtreme Scale は自動的にサーバーのバランスを取ります。 WebSphere eXtreme Scale を再始動しなくても、追加サーバーを組み込むことができます。eXtreme Scale を再始動せずにサーバーを追加できることで、単純なデプロイメントだけでなく、数千ものサーバーが必要になる大規模なテラバイト・サイズのデプロイメントが可能になります。

このデプロイメント・トポロジーは柔軟です。カタログ・サービスを使用すると、キャッシュ全体を除去することなく、サーバーを追加および除去して、リソースを効率的に使用できるようになります。 startOgServer コマンドおよび stopOgServer コマンドを使用して、コンテナー・サーバーを始動および停止できます。これらのいずれのコマンドを使用した場合でも、-catalogServiceEndPoints オプションを指定する必要があります。分散トポロジーのすべてのクライアントは、Internet Interoperability Object Protocol (IIOP) を介してカタログ・サービスと通信します。すべてのクライアントは、ObjectGrid インターフェースを使用してサーバーと通信できます。

WebSphere eXtreme Scale の動的構成機能を使用すると、簡単にシステムにリソースを追加することができます。コンテナーはデータをホストします。また、カタログ・サービスによって、クライアントはコンピューター・グリッドと通信できます。カタログ・サービスは、要求を転送し、ホスト・コンテナー内でスペースを割り振り、システム全体のヘルスと可用性を管理します。クライアントは、カタロ

グ・サービスに接続して、コンテナー/サーバー・トポロジーの説明を取得してから、必要に応じて各サーバーと直接通信します。新規サーバーの追加または他のサーバーの障害によってサーバー・トポロジーが変更されると、カタログ・サービスは、データをホスティングする適切なサーバーにクライアント要求を自動的に経路指定します。

通常、カタログ・サービスは、自身の Java 仮想マシン グリッド内に存在します。単一のカタログ・サーバーで複数のサーバーを管理できます。コンテナーは、JVM 内で単独で開始することも、別のサーバーの他のコンテナーとともに任意の JVM にロードすることもできます。クライアントはどの JVM 内にも存在でき、1 つ以上のサーバーと通信できます。また、コンテナーと同じ JVM 内に存在することも可能です。

既存の Java プロセスまたはアプリケーションにコンテナーを組み込む際には、プログラムでデプロイメント・ポリシーを作成することもできます。詳細については、eXtreme Scale DeploymentPolicy API の資料を参照してください。

レプリカ配置のためのゾーンの構成

ゾーン・サポートは、データ・センター間でのレプリカ配置のための高度な構成を可能にします。 この機能により、少数のオプションの配置ルールを使用して、何千という区画のグリッドを簡単に管理することができます。データ・センターは、ゾーン・ルールによる構成に従って、ビルの別フロア、別ビル、または異なる都市にさえ配置することも、またはその他の区分に配置することができます。

ゾーンの柔軟性

ゾーンに断片を配置することができます。この機能により、eXtreme Scale がグリッド上に断片をどのように配置するかをさらに制御できるようになります。eXtreme Scale サーバーをホストする Java 仮想マシンは、ゾーン ID によってタグ付けすることができます。現在、デプロイメント・ファイルには 1 つ以上のゾーン・ルールを含めることができます。これらのゾーン・ルールは、断片タイプに関連付けられます。このことを説明する最善の方法として、いくつか例を挙げ、詳しく説明します。

配置ゾーンは、高度なトポロジーを構成するために、eXtreme Scale がプライマリーとレプリカの割り当てをどのように達成するかを制御します。

Java 仮想マシンは、複数のコンテナーを持つことができますが、サーバーは 1 つだけしか持てません。コンテナーは、単一の ObjectGrid の複数の断片をホストできます。

この機能を利用すると、レプリカとプライマリーを異なるロケーションまたはゾーンに配置して、より優れた高可用性を実現することができます。通常、eXtreme Scale は、同じ IP アドレスでプライマリー断片と複製断片を Java 仮想マシンに配置することはありません。この単純なルールにより、一般的には、2 つの eXtreme Scale サーバーが同じ物理コンピューターに配置されることがなくなります。ただし、より柔軟なメカニズムが必要になる場合もあります。例えば、2 つのブレード・シャーシを使用していて、プライマリーを両方のシャーシ間でストライプ し、それぞれのプライマリーのレプリカがそのプライマリーの他方のシャーシに配置されるようにしたい場合があります。

ストライプ・プライマリーは、プライマリーが各ゾーンに配置され、その各プライマリーの複製が対向ゾーンに配置されることを意味します。例えば、プライマリー 0 は zoneA に入り、同期複製 0 は zoneB に入ります。プライマリー 1 は zoneB に入り、同期複製 1 は zoneA に入ります。

この場合、シャーシ名がゾーン名となります。代替方法として、ビルのフロアの後にゾーンを命名し、ゾーンを使用して、同じデータのプライマリーとレプリカが別フロアに配置されるようにすることができます。ビルおよびデータ・センターでも可能です。データ・センター間でデータが適切に複製されるようにするためのメカニズムとしてゾーンを使用し、データ・センター全体に渡るテストが実行されています。eXtreme Scale の HTTP セッション・マネージャーを使用している場合も、ゾーンを使用することができます。このフィーチャーを使用すると、1 つの Web アプリケーションを 3 つのデータ・センターに渡ってデプロイできます。これにより、ユーザーの HTTP セッションがデータ・センター間で複製され、1 つのデータ・センター全体が障害を起こした場合にも、セッションを回復できるようになります。

WebSphere eXtreme Scale は、複数のデータ・センターに渡る大きなグリッドを管理する必要性を配慮されています。これにより、同一区画のバックアップとプライマリーを必要に応じて異なるデータ・センターに配置することが可能です。すべてのプライマリーをデータ・センター 1 に、すべてのレプリカをデータ・センター 2 に配置したり、両方のデータ・センター間でプライマリーとレプリカをラウンドロビンしたりすることができます。ルールは柔軟であり、さまざまなシナリオが考えられます。また eXtreme Scale は数千ものサーバーを管理することができます。これにより、データ・センター認識による完全な自動配置を同時に使用することによって、管理の観点から手ごろな大規模グリッドの作成が可能です。管理者は、簡単かつ効果的に実行するものを指定できます。

管理者の場合、配置ゾーンを使用して、プライマリー断片と複製断片の配置場所を制御できます。これにより、高性能でかつ高可用性のトポロジーのセットアップが可能になります。前述したように、eXtreme Scale プロセスのいずれの論理グループに対してもゾーンを定義できます。これらのゾーンは、データ・センター、データ・センターのフロア、ブレード・シャーシなど、物理ワークステーション・ロケーションに対応付けることができます。ゾーン間でデータをストライプすることができます。これにより、可用性が増します。またホット・スタンバイが必要な場合に、プライマリーとレプリカを別々のゾーンに分割することができます。

eXtreme Scale サーバーの WebSphere Extended Deployment を使用しないゾーンとの関連付け

eXtreme Scale が Java Standard Edition で使用されるか、あるいは、WebSphere Extended Deployment バージョン 6.1 をベースにしていないアプリケーション・サーバーで使用される場合、断片コンテナーである JVM は、以下の手法を使用して、ゾーンに関連付けられます。

startOgServer スクリプトを使用するアプリケーション

startOgServer スクリプトは、既存のサーバーに組み込まれない eXtreme Scale アプリケーションを開始するために使用されます。**-zone** パラメーターを使用すると、サーバー内のすべてのコンテナーに使用するゾーンを指定できます。

API を使用するコンテナーを開始する場合のゾーンの指定

WebSphere Extended Deployment ノードのゾーンとの関連付け

eXtreme Scale を WebSphere Extended Deployment JEE アプリケーションで使用している場合、WebSphere Extended Deploymentノード・グループを活用して、サーバーを特定のゾーンに配置できます。

eXtreme Scale では、JVM は、単一ゾーンのメンバーになることだけを許可されています。 ただし、WebSphere は、ノードが複数ノード・グループの一部になることを許可しています。 各ノードが 1 つのゾーンのノード・グループ内のみにあると確認できれば、eXtreme Scale のゾーンのこの機能を使用できます。

ノード・グループがゾーンであると宣言するために、次の構文を使用して、ノード・グループに名前を付けてください。`ReplicationZone<UniqueSuffix>` そのようなノード・グループの一部であるノード上で稼働しているサーバーは、ノード・グループ名で指定されるゾーンに組み込まれます。 以下に、トポロジーの例を説明します。

まず、4 つのノードを構成します。node1、node2、node3、および node4 で、各ノードには 2 台のサーバーがあります。 その後、ReplicationZoneA という名前のノード・グループと ReplicationZoneB という名前のノード・グループを作成します。 次に、node1 と node2 を ReplicationZoneA に追加し、node3 と node4 を ReplicationZoneB に追加します。

node1 と node2 のサーバーが始動すると、それらのサーバーは ReplicationZoneA の一部になり、同様に、node3 と node4 のサーバーは ReplicationZoneB の一部になります。

グリッド・メンバー JVM は、始動時にのみゾーン・メンバーシップをチェックします。 新規ノード・グループを追加するか、メンバーシップを変更した場合、それは新たに始動されるか、再始動される JVM のみに影響します。

ゾーン・ルール

eXtreme Scale 区画には、1 つのプライマリー断片とゼロ個以上の複製断片があります。この例の場合、これらの断片について以下の命名規則を考慮してください。`P` はプライマリー断片で、`S` は同期複製で、`A` は非同期複製です。ゾーン・ルールは以下の 3 つの部分からなります。

- ルール名
- ゾーンのリスト
- 包含または排他フラグ

コンテナーのゾーン名は、組み込みサーバー API の資料に記述されているとおりに指定できます。ゾーン・ルールは、断片を配置できるゾーンのセットを指定します。包含フラグは、1 つの断片がリストからゾーンに配置されると、他のすべての断片もそのゾーンに配置されることを示します。排他設定は、区画の各断片がゾーン・リストの異なるゾーンに配置されることを示します。例えば、排他設定を使用する場合、3 つの断片 (プライマリーと 2 つの同期複製) がある場合は、ゾーン・リストに 3 つのゾーンがなければならないということです。

各断片は、1 つのゾーン・ルールに関連付けることができます。ゾーン・ルールは、2 つの断片間で共有できます。ルールが共有される場合、包含または排他フラグは、1 つのルールを共有するすべてのタイプの断片に拡張されます。

例

さまざまなシナリオおよびそのシナリオを実装するためのデプロイメント構成を示す一連の例は、以下のとおりです。

ゾーン間でプライマリーとレプリカをストライピングする

3 つのブレード・シャーシがあり、3 つすべてにプライマリーを分散し、1 つの同期複製をプライマリー以外のシャーシに配置するものとします。各シャーシをシャーシ名 ALPHA、BETA、および GAMMA を持つゾーンとして定義します。デプロイメント XML の例は以下のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xsi:schemaLocation=
 "http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
    xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="library">
   <mapSet name="ms1" numberOfPartitions="37" minSyncReplicas="1"
    maxSyncReplicas="1" maxAsyncReplicas="0">
   <map ref="book" />
   <zoneMetadata>
    <shardMapping shard="P" zoneRuleRef="stripeZone"/>
    <shardMapping shard="S" zoneRuleRef="stripeZone"/>
    <zoneRule name ="stripeZone" exclusivePlacement="true" >
     <zone name="ALPHA" />
     <zone name="BETA" />
     <zone name="GAMMA" />
    </zoneRule>
   </zoneMetadata>
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

このデプロイメント XML には、「book」という名前の 1 つのマップを持つ「library」という名前のグリッドが含まれます。さらに、1 つの同期複製を持つ 4 つの区画を使用します。zone metadata 文節は、1 つのゾーン・ルールの定義およびゾーン・ルールと断片の関連付けを示します。プライマリー断片と同期断片は、ともにゾーン・ルール「stripeZone」に関連付けられます。このゾーン・ルールでは 3 つのゾーンがすべて含まれ、排他配置が使用されるようになっています。このルールは、区画 0 のプライマリーが ALPHA に配置されると、区画 0 のレプリカは BETA か GAMMA のいずれかに配置されることになります。他の区画のプライマリーは他のゾーンに配置され、レプリカが同様に配置されます。

非同期複製がプライマリーや同期複製と異なるゾーンにある

この例では、2 つのビルがあり、その間の接続は待ち時間が長いものとします。すべてのシナリオでデータ損失のない高可用性が保たれるようにします。ただし、ビル間の同期複製によるパフォーマンス・インパクトのためトレードオフが生じます。このため、一方のビルに同期複製があり、他方のビルに非同期複製があるようなプライマリーが必要になります。通常では、障害は、大規模な問題ではなく、JVM の破損やコンピューター障害です。このトポロジーを使用すると、データ損失なしに通常の障害を切り抜けることができます。ビルがなくなることは非常にまれなことであるため、その場合でもある程度のデータ損失は許容範囲内に収まります。それぞれのビルに 1 つずつ、合計 2 つのゾーンを作成できます。デプロイメント XML ファイルは以下のようになります。

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

 <objectgridDeployment objectgridName="library">
  <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="1"
   maxSyncReplicas="1" maxAsyncReplicas="1">
   <map ref="book" />
   <zoneMetadata>
    <shardMapping shard="P" zoneRuleRef="primarySync"/>
    <shardMapping shard="S" zoneRuleRef="primarySync"/>
    <shardMapping shard="A" zoneRuleRef="aysnc"/>
    <zoneRule name ="primarySync" exclusivePlacement="false" >
      <zone name="BldA" />
      <zone name="BldB" />
    </zoneRule>
    <zoneRule name="aysnc" exclusivePlacement="true">
      <zone name="BldA" />
      <zone name="BldB" />
    </zoneRule>
   </zoneMetadata>
  </mapSet>
 </objectgridDeployment>
</deploymentPolicy>
```

プライマリーと同期複製は、排他フラグ設定 false で primaySync ゾーン・ルールを共有します。このため、プライマリーか同期のいずれかがゾーンに配置されると、他方も同じゾーンに配置されます。非同期複製は、primarySync ゾーン・ルールと同じゾーンで第 2 のゾーン・ルールを使用しますが、true に設定された **exclusivePlacement** 属性を使用します。この属性は、断片を同じ区画の別の断片があるゾーンには配置できないことを示します。結果的に、非同期複製は、プライマリーまたは同期複製と同じゾーンに配置されません。

すべてのプライマリーを 1 つのゾーンに配置し、すべてのレプリカを別のゾーンに配置する

ここでは、すべてのプライマリーが特定のゾーンに配置され、すべてのレプリカが別のゾーンに配置されます。これにより、1 つのプライマリーと 1 つの非同期複製を持つことになります。すべてのレプリカはゾーン A に、プライマリーはゾーン B に配置されます。

```
<?xml version="1.0" encoding="UTF-8"?>

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

 <objectgridDeployment objectgridName="library">
  <mapSet name="ms1" numberOfPartitions="13" minSyncReplicas="0"
   maxSyncReplicas="0" maxAsyncReplicas="1">
   <map ref="book" />
   <zoneMetadata>
    <shardMapping shard="P" zoneRuleRef="primaryRule"/>
    <shardMapping shard="A" zoneRuleRef="replicaRule"/>
    <zoneRule name ="primaryRule">
     <zone name="A" />
    </zoneRule>
    <zoneRule name="replicaRule">
     <zone name="B" />
      </zoneRule>
     </zoneMetadata>
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

ここには、プライマリー用に 1 つ (P)、レプリカ用に 1 つ (A) という 2 つのルールがあります。

広域ネットワーク (WAN) 上のゾーン

低速のネットワーク相互接続を使用する複数のビルまたはデータ・センターにまたがって、1 つの eXtreme Scale をデプロイしたい場合があります。ネットワーク接続が低速になれば、それだけ処理能力が低下し、接続待ち時間が長くなります。このモードでは、ネットワーク輻輳やその他の要因のために、ネットワーク分割の可能性も増します。eXtreme Scale は、以下の方法でこの厳しい環境に対処します。

ゾーン間のハートビート処理の制限

コア・グループにグループ化された Java 仮想マシンは、互いにハートビートを実行します。カタログ・サービスが Java 仮想マシンをグループに編成した場合、こうしたグループはゾーンをまたぎません。そのグループ内のリーダーがメンバーシップ情報をカタログ・サービスにプッシュします。カタログ・サービスは報告された障害を検証してから、アクションを実行します。問題のある Java 仮想マシンとの接続を試みて、この処理を実行します。カタログ・サービスは、誤障害検出を認めた場合、何のアクションも実行しません。コア・グループ区画が短時間で回復するためです。

またカタログ・サービスは、定期的にコア・グループ・リーダーに低速でハートビートを行い、コア・グループ分離の症状を処理します。

フェイルオーバー検出の構成

ハートビート間隔設定で、障害の起きたサーバーがないかを調べるシステム・チェックの間の時間を構成できます。

このタスクについて

フェイルオーバーの構成は、使用している環境のタイプによって異なります。スタンドアロン環境を使用している場合は、コマンド行でフェイルオーバーを構成できます。 WebSphere Application Server Network Deployment 環境を使用している場合は、WebSphere Application Server Network Deployment 管理コンソールでフェイルオーバーを構成する必要があります。

手順

- スタンドアロン環境のフェイルオーバーを構成します。

  ハートビート間隔は、startOgServer.bat | startOgServer.sh スクリプト・ファイルに **-heartbeat** パラメーターを使用してコマンド行で構成できます。このパラメーターは以下のいずれかの値に設定します。

表 9. ハートビート間隔

値	アクション	説明
0	標準 (デフォルト)	通常、30 秒以内にフェイルオーバーが検出されます。
-1	高速	通常、5 秒以内にフェイルオーバーが検出されます。
1	低速	通常、180 秒以内にフェイルオーバーが検出されます。

  高速のハートビート間隔は、プロセスおよびネットワークが安定している場合に役立ちます。ネットワークまたはプロセスが最適に構成されていないと、ハート

ビートを見逃す可能性があり、そうなった場合は誤って障害検出が示されることがあります。

- WebSphere Application Server 環境のフェイルオーバーを構成します。

  WebSphere Application Server Network Deployment バージョン 6.0.2 以降は、WebSphere eXtreme Scale のフェイルオーバーを高速で行えるように構成できます。ハード障害の場合のデフォルトのフェイルオーバー時間は、約 200 秒です。ハード障害は、物理的なコンピューターまたはサーバーの破損、ネットワーク・ケーブルの切断、オペレーティング・システム・エラーのことです。プロセスの異常終了やソフト障害による障害は、一般的に 1 秒未満でフェイルオーバーされます。ソフト障害の障害検出は、デッド・プロセスのネットワーク・ソケットがそのプロセスをホスティングするサーバーのオペレーティング・システムにより自動的にクローズされるときに発生します。

  コア・グループのハートビート構成

  WebSphere Application Server プロセスで実行されているWebSphere eXtreme Scale は、アプリケーション・サーバーのコア・グループ設定のフェイルオーバー特性を継承します。以下のセクションでは、以下のようなさまざまなバージョンの WebSphere Application Server Network Deployment のコア・グループ・ハートビート設定を構成する方法について説明します。

  – **WebSphere Application Server Network Deployment バージョン 6.x または 7.x のコア・グループ設定を更新します。**

    ハートビート間隔は、WebSphere Application Server のバージョン 6.0 からバージョン 6.1.0.12 までは秒単位、バージョン 6.1.0.13 からはミリ秒単位で指定します。また、欠落ハートビートの数も指定する必要があります。この値は、ピア Java 仮想マシン (JVM) に障害が起きたと見なされるまでに、容認される欠落ハートビートの数を示します。ハード障害の検出時間は、ほぼハートビート間隔と欠落ハートビート数の積です。

    これらのプロパティーは、WebSphere 管理コンソールで、コア・グループに対してカスタム・プロパティーを使用して指定します。構成について詳しくは、コア・グループ・カスタム・プロパティーを参照してください。アプリケーションによって使用されるすべてのコア・グループに対して、以下のプロパティーを指定する必要があります。

    - ハートビート間隔は、IBM_CS_FD_PERIOD_SEC カスタム・プロパティー (秒単位) または IBM_CS_FD_PERIOD_MILLIS カスタム・プロパティー (ミリ秒単位、V6.1.0.13 以降が必要) を使用して指定します。
    - 欠落ハートビート数は、IBM_CS_FD_CONSECUTIVE_MISSED カスタム・プロパティーを使用して指定します。

    IBM_CS_FD_PERIOD_SEC プロパティーのデフォルト値は 20 で、IBM_CS_FD_CONSECUTIVE_MISSED プロパティーのデフォルト値は 10 です。IBM_CS_FD_PERIOD_MILLIS プロパティーを指定すると、設定されている IBM_CS_FD_PERIOD_SEC カスタム・プロパティーがオーバーライドされます。これらのプロパティーの値は、正の整数値です。

WebSphere Application Server Network Deployment 6.x サーバーで 1500 ミリ秒の障害検出時間を実現するには、以下の設定を使用します。

- IBM_CS_FD_PERIOD_MILLIS = 750 を設定 (WebSphere Application Server Network Deployment バージョン 6.1.0.13 以降)
- IBM_CS_FD_CONSECUTIVE_MISSED = 2 を設定

– **WebSphere Application Server Network Deployment バージョン 7.0 でのコア・グループ設定を更新します。**

バージョン 7.0 の WebSphere Application Server Network Deployment は、フェイルオーバー検出を増減するために調整できる以下の 2 つのコア・グループ設定を提供します。

- **ハートビート伝送期間。** デフォルト値は 30000 ミリ秒です。
- **ハートビート・タイムアウト期間。** デフォルト値は 180000 ミリ秒です。

これらの設定を変更する方法について詳しくは、WebSphere Application Server Network Deployment インフォメーション・センター: ディスカバリーおよび障害検出の設定を参照してください。

WebSphere Application Server Network Deployment バージョン 7 サーバーで 1500 ミリ秒の障害検出時間を実現するには、以下の設定を使用します。

- ハートビート伝送期間を 750 ミリ秒に設定します。
- ハートビート・タイムアウト期間を 1500 ミリ秒に設定します。

次のタスク

短いフェイルオーバー時間を指定するようにこれらの設定を変更すると、注意すべきシステム調整上の問題が生じます。まず Java はリアルタイム環境ではありません。JVM に長期のガーベッジ・コレクション時間が発生すると、スレッドが遅延する可能性があります。JVM をホスティングするマシンの負荷が大きくなった (JVM 自身またはマシンで実行中の他のプロセスが原因) 場合にも、スレッドが遅延する可能性があります。スレッドが遅延された場合、ハートビートが正確な時間で送信されない可能性があります。最悪の場合、必要なフェイルオーバー時間で遅延が生じる可能性があります。 スレッドが遅延すると、誤障害検出が発生します。実動環境で誤障害検出が発生しないように、システムを調整し、サイズ設定する必要があります。これを確実にするには、適切な負荷テストが最善の策です。

注: eXtreme Scale の現行バージョンは、WebSphere Real Time をサポートします。

デプロイメント・ポリシー記述子 XML ファイル

デプロイメント・ポリシーを構成するには、デプロイメント・ポリシー記述子 XML ファイルを使用します。

以下のセクションでは、デプロイメント・ポリシー記述子 XML ファイルのエレメントおよび属性が定義されます。対応するデプロイメント・ポリシー XML スキーマについては、190 ページの『deploymentPolicy.xsd ファイル』 を参照してください。

deploymentPolicy.xml ファイルのエレメント

```
(1) <deploymentPolicy>
(2)  <objectgridDeployment objectGridName="blah">
(3)   <mapSet
(4)    name="mapSetName"
(5)    numberOfPartitions="numberOfPartitions"
(6)    minSyncReplicas="minimumNumber"
(7)     maxSyncReplicas="maximumNumber"
(8)    maxAsyncReplicas="maximumNumber"
(9)    replicaReadEnable="true|false"
(10)    numInitialContainers="numberOfInitialContainersBeforePlacement"
(11)    autoReplaceLostShards="true|false"
(12)    developmentMode="true|false"
(13)    placementStrategy="FIXED_PARTITION|PER_CONTAINER">
(14)    <map ref="backingMapReference" />
(15)   </mapSet>
(16)   <zoneMetadata>
(17)    <shardMapping
(18)      shard="shardName"
(19)      zoneRuleRef="zoneRuleRefName" />
(20)    <zoneRule
(21)      name="zoneRuleName"
(22)      exclusivePlacement="true|false" >
(23)      <zone name="ALPHA" />
(24)      <zone name="BETA" />
(25)      <zone name="GAMMA" />
(26)    </zoneRule>
(27)   </zoneMetadata>
(28)  </objectgridDeployment>
</deploymentPolicy>
```

deploymentPolicy エレメント (行 1)

deploymentPolicy エレメントは、デプロイメント・ポリシー XML ファイルの最上位エレメントです。このエレメントは、ファイルの名前空間とスキーマ・ロケーションをセットアップします。スキーマは `deploymentPolicy.xsd` ファイルで定義されます。

- **出現回数:** 1 回
- **子エレメント:** objectgridDeployment

objectgridDeployment エレメント (行 2)

objectgridDeployment エレメントは、ObjectGrid XML ファイルの ObjectGrid インスタンスを参照する場合に使用します。objectgridDeployment エレメント内では、マップをマップ・セットに分割できます。

- **出現回数:** 1 回以上
- **子エレメント:** mapSet

属性

objectgridName
デプロイする ObjectGrid インスタンスの名前を指定します。この属性は、ObjectGrid XML ファイルで定義されている objectGrid エレメントを参照します。(必須)

例えば、objectgridName 属性は、 companyGridDpReplication.xml ファイル内で `CompanyGrid` として設定されています。objectgridName 属性は、companyGrid.xml ファイルに定義されている `CompanyGrid` を参照します。 ObjectGrid インスタンスごとにデプロイメント・ポリシー・ファイルと組み合わせる必要がある 151 ページの『ObjectGrid 記述子 XML ファイル』を参照してください。

mapSet エレメント (行 3)

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 エレメント内のマップは同じように区画に分割され、複製されます。各マップは、単一の mapSet エレメントのみに属している必要があります。

- **出現回数:** 1 回以上
- **子エレメント:** map

属性

name
MapSet の名前を指定します。この属性は、objectgridDeployment エレメント内では固有である必要があります。(必須)

numberOfPartitions
MapSet エレメント用の区画の数を指定します。デフォルト値は 1 です。この数は、区画をホストしているコンテナーの数に適した値である必要があります。(オプション)

minSyncReplicas
同期複製の最小数を MapSet 内の区画ごとに指定します。デフォルト値は 0 です。断片は、ドメインが最小数の同期レプリカをサポートできるようになるまで配置されません。 minSyncReplicas 値をサポートするには、minSyncReplicas の値よりも 1 つだけ多いコンテナーが必要です。同期複製の数が minSyncReplicas の値よりも小さくなると、その区画に対しては書き込みトランザクションを行えなくなります。(オプション)

maxSyncReplicas
同期複製の最大数を MapSet 内の区画ごとに指定します。デフォルト値は 0 です。ドメインがある特定の区画でこの同期レプリカ数に達すると、その区画に対しては他の同期レプリカは配置されません。まだ maxSyncReplicas 値を満たしていない場合には、この ObjectGrid をサポートできるコンテナーを追加すると、同期複製の数を増やすことができます。(オプション)

maxAsyncReplicas
非同期複製の最大数を MapSet 内の区画ごとに指定します。デフォルト値は 0 です。ある区画に対してプライマリーおよびすべての同期レプリカが配置された後は、maxAsyncReplicas 値になるまで、非同期レプリカが配置されます。(オプション)

replicaReadEnabled
この属性が `true` に設定されている場合、プライマリー区画とそのレプリカに読み取り要求が配布されます。 replicaReadEnabled 属性が `false` の場合は、読み取り要求はプライマリーにのみ送付されます。デフォルト値は false です。(オプション)

numInitialContainers
この mapSet エレメント内の断片に対して初期配置が行われる前に必要となる eXtreme Scale コンテナーの数を指定します。デフォルト値は 1 です。この属性は、ObjectGrid インスタンスをオンラインにする際にプロセスとネットワーク帯域幅を節約するのに役立ちます。(オプション)

eXtreme Scale コンテナーを開始すると、カタログ・サービスにイベントが送信されます。アクティブ・コンテナーの数が初めて mapSet エレメントの numInitialContainers 値と等しくなり、さらに minSyncReplicas も満たすことができる場合には、カタログ・サービスは mapSet の断片を配置します。numInitialContainers 値を満たすと、コンテナーによって始動された各イベントは、未配置の断片と既に配置されている断片の再バランシングを引き起こす場合があります。この mapSet エレメントに対して開始する予定のコンテナーのおおよその数が分かっている場合は、その数に近い numInitialContainers 値を設定して、すべてのコンテナーが開始した後の再バランシングを回避することができます。配置は、mapSet エレメントに指定した numInitialContainers 値に到達した場合にのみ行われます。

autoReplaceLostShards
失われた断片がその他のコンテナーに配置されるかどうかを指定します。デフォルト値は true です。コンテナーが停止するか、障害を発生すると、コンテナーで実行中の断片が失われます。プライマリー断片が失われると、対応する区画のレプリカ断片のいずれかがプライマリー断片にプロモートされます。このプロモーションのため、レプリカの 1 つが失われます。失われた断片を未配置のままにしておく場合は、autoReplaceLostShards 属性を `false` に設定します。この設定はプロモーション・チェーンには影響せず、そのチェーン内の最後の断片の置き換えにのみ影響します。(オプション)

developmentMode
この属性を使用すると、ある断片をそのピア断片との関係でどこに配置するかを制御できます。デフォルト値は true です。 developmentMode 属性が `false` に設定されている場合は、同じ区画の 2 つの断片は同じコンピューターには配置されません。 developmentMode 属性が `true` に設定されている場合は、同じ区画の断片を同じマシンに配置することができます。いずれの場合も、同一の区画の 2 つの断片は、同一のコンテナーには配置されません。(オプション)

placementStrategy
配置ストラテジーには 2 つあります。デフォルトのストラテジーは `FIXED_PARTITION` です。この場合、使用可能なコンテナー全体で配置されるプライマリー断片の数は、定義されている区画数にレプリカ数を加えたものになります。代替のストラテジーは `PER_CONTAINER` です。この場合、各コンテナーに配置されるプライマリー断片の数は、定義されている区画数になり、同じ数のレプリカが他のコンテナーに配置されます。 (オプション)

map エレメント (行 14)

mapSet エレメント内の各 map エレメントは、対応する 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 eXtreme Scale 環境内のすべてのマップは、単一の mapSet エレメントにのみ属することができます。

- **出現回数:** 1 回以上
- **子エレメント:** なし

属性

ref ObjectGrid XML ファイル内の backingMap エレメントを参照できるようにします。mapSet エレメント内の各マップは、ObjectGrid XML ファイルの

backingMap エレメントを参照する必要があります。以下のコード・スニペットのように、ref 属性に割り当てられている値は、ObjectGrid XML ファイル内の backingMap エレメントの 1 つの name 属性に一致している必要があります。(必須)

zoneMetadata エレメント (行 16)

ゾーンに断片を配置することができます。この機能により、eXtreme Scale がグリッド上に断片をどのように配置するかをさらに制御できるようになります。eXtreme Scale サーバーをホストする Java™ 仮想マシンは、ゾーン ID によってタグ付けすることができます。デプロイメント・ファイルには 1 つ以上のゾーン・ルールを含めることができます。これらのゾーン・ルールは、断片タイプに関連付けられます。追加のバックグラウンドについては、177 ページの『レプリカ配置のためのゾーンの構成』 を参照してください。

- **出現回数:** 0 回または 1 回
- **子エレメント:**
  - shardMapping
  - zoneRule

属性: なし

shardMapping エレメント (行 17)

各断片は、1 つのゾーン・ルールに関連付けることができます。ゾーン・ルールは、2 つの断片間で共有できます。ルールが共有される場合、包含または排他フラグは、1 つのルールを共有するすべてのタイプの断片に拡張されます。

- **出現回数:** 0 回または 1 回
- **子エレメント:** なし

属性

断片
zoneRule と関連付ける断片の名前を指定します。 (必須)

zoneRuleRef
断片と関連付ける zoneRule の名前を指定します。 (オプション)

zoneRule エレメント (行 20)

ゾーン・ルールは、断片を配置できるゾーンのセットを指定します。

- **出現回数:** 1 回以上
- **子エレメント:** zone

属性

name
以前に shardMapping エレメント内の zoneRuleRef として定義したゾーン・ルールの名前を指定します。(必須)

exclusivePlacement
排他設定は、区画の各断片がゾーン・リストの異なるゾーンに配置されることを

示します。包含設定は、1 つの断片がリストからゾーンに配置されると、他のすべての断片もそのゾーンに配置されることを示します。例えば、排他設定を使用する場合、3 つの断片 (プライマリーと 2 つの同期複製) がある場合は、ゾーン・リストに 3 つのゾーンがなければならないということです。(オプション)

zone エレメント (行 23 から 25)

3 つのブレード・シャーシがあり、3 つすべてにプライマリー断片を分散し、1 つの同期複製をプライマリー断片以外のシャーシに配置するものとします。各シャーシをゾーンとして定義し、ゾーン名をシャーシ名に対応して
ALPHA、BETA、GAMMA とすることができます。

- **出現回数:** 1 回以上
- **子エレメント:** なし

属性

name
指定されたゾーン・ルールを適用するゾーンの名前を指定します。(必須)

例

以下の例では、mapSet エレメントがデプロイメント・ポリシーを構成するために使用されています。値は、mapSet1 に設定され、10 区画に分割されています。これらの分割のそれぞれにおいて、少なくとも 1 つの同期複製と、2 つ以下の同期複製が使用可能になっている必要があります。環境が非同期複製をサポートできる場合は、各区画には非同期複製も含まれています。すべての同期複製は、非同期複製が配置される前に配置されます。さらに、ドメインが minSyncReplicas 値をサポートできるようになるまでは、カタログ・サービスは mapSet1 エレメントの断片を配置しません。 minSyncReplicas 値をサポートするには、複数のコンテナー、すなわちプライマリー用に 1 つ、同期複製用に 2 つのコンテナーが必要です。

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

 <objectgridDeployment objectgridName="CompanyGrid">
  <mapSet name="mapSet1" numberOfPartitions="10"
   minSyncReplicas="1" maxSyncReplicas="2" maxAsyncReplicas="1"
    numInitialContainers="10" autoReplaceLostShards="true"
   developmentMode="false" replicaReadEnabled="true">
   <map ref="Customer"/>
   <map ref="Item"/>
   <map ref="OrderLine"/>
   <map ref="Order"/>
  </mapSet>
 </objectgridDeployment>

</deploymentPolicy>
```

複製の設定を満たすのにコンテナーが 2 つしか必要ない場合でも、
numInitialContainers 属性は、カタログ・サービスがこの mapSet エレメントに断片を配置する前に、使用可能なコンテナーを 10 個要求します。ドメインが、
CompanyGrid ObjectGrid をサポートできるコンテナーを 10 個持つと、mapSet1 エレメント内のすべての断片が配置されます。

autoReplaceLostShards 属性が true に設定されているため、この mapSet エレメント内の、コンテナー障害のために失われた断片は、失われた断片をコンテナーがホストできる場合には、自動的に別のコンテナーに置き換えられます。 mapSet1 エレメ

ントでは、同じ区画内の断片は、同じマシンには配置できません。developmentMode 属性が false に設定されているためです。読み取り専用要求は、区画ごとにプライマリー断片とそのレプリカ全体に配布されます。これは、replicaReadEnabled 値が true だからです。

companyGridDpMapSetAttr.xml ファイルは、この map の ref 属性を使用して、companyGrid.xml ファイルの各 backingMap エレメントを参照します。

deploymentPolicy.xsd ファイル

デプロイメント・ポリシー XML スキーマを使用すると、デプロイメント記述子 XML ファイルを作成できます。

deploymentPolicy.xsd ファイルに定義されるエレメントおよび属性の説明は、 184 ページの『デプロイメント・ポリシー記述子 XML ファイル』を参照してください。

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:dp="http://www.ibm.com/ws/objectgrid/deploymentPolicy"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNameSpace="http://ibm.com/ws/objectgrid/deploymentPolicy"
 elementFormDefault="qualified">

 <xsd:element name="deploymentPolicy">
  <xsd:complexType>
   <xsd:choice>
    <xsd:element name="objectgridDeployment"
     type="dp:objectgridDeployment" minOccurs="1"
     maxOccurs="unbounded">
     <xsd:unique name="mapSetNameUnique">
      <xsd:selector xpath="dp:mapset" />
      <xsd:field xpath="@name" />
     </xsd:unique>
    </xsd:element>
   </xsd:choice>
  </xsd:complexType>
 </xsd:element>

 <xsd:complexType name="objectgridDeployment">
  <xsd:sequence>
   <xsd:element name="mapSet" type="dp:mapSet"
    maxOccurs="unbounded" minOccurs="1">
    <xsd:unique name="mapNameUnique">
     <xsd:selector xpath="dp:map" />
     <xsd:field xpath="@ref" />
    </xsd:unique>
   </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="objectgridName" type="xsd:string"
   use="required" />
 </xsd:complexType>

 <xsd:complexType name="mapSet">
  <xsd:sequence>
   <xsd:element name="map" type="dp:map" maxOccurs="unbounded"
    minOccurs="1" />
   <xsd:element name="zoneMetadata" type="dp:zoneMetadata"
    maxOccurs="1" minOccurs="0">

    <xsd:key name="zoneRuleName">
     <xsd:selector xpath="dp:zoneRule" />
     <xsd:field xpath="@name" />
    </xsd:key>

    <xsd:keyref name="zoneRuleRef"
     refer="dp:zoneRuleName">
     <xsd:selector xpath="dp:shardMapping" />
     <xsd:field xpath="@zoneRuleRef" />
    </xsd:keyref>

   </xsd:element>
  </xsd:sequence>
  <xsd:attribute name=="name" type="xsd:string" use="required" />
  <xsd:attribute name="numberOfPartitions" type="xsd:int"
   use="optional" />
  <xsd:attribute name="minSyncReplicas" type="xsd:int"
   use="optional" />
  <xsd:attribute name="maxSyncReplicas" type="xsd:int"
   use="optional" />
  <xsd:attribute name="maxAsyncReplicas" type="xsd:int"
```

```
  use="optional" />
 <xsd:attribute name="replicaReadEnabled" type="xsd:boolean"
  use="optional" />
 <xsd:attribute name="numInitialContainers" type="xsd:int"
  use="optional" />
 <xsd:attribute name="autoReplaceLostShards" type="xsd:boolean"
  use="optional" />
 <xsd:attribute name="developmentMode" type="xsd:boolean"
  use="optional" />
 <xsd:attribute name="placementStrategy"
  type="dp:placementStrategy" use="optional" />
</xsd:complexType>

<xsd:simpleType name="placementStrategy">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="FIXED_PARTITIONS" />
  <xsd:enumeration value="PER_CONTAINER" />
 </xsd:restriction>
</xsd:simpleType>


<xsd:complexType name="map">
 <xsd:attribute name="ref" use="required" />
</xsd:complexType>

<xsd:complexType name="zoneMetadata">
 <xsd:sequence>
  <xsd:element name="shardMapping" type="dp:shardMapping"
   maxOccurs="unbounded" minOccurs="1" />
  <xsd:element name="zoneRule" type="dp:zoneRule"
   maxOccurs="unbounded" minOccurs="1">

  </xsd:element>

 </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="shardMapping">
 <xsd:attribute name="shard" use="required">
  <xsd:simpleType>
   <xsd:restriction base="xsd:string">
    <xsd:enumeration value="P"></xsd:enumeration>
    <xsd:enumeration value="S"></xsd:enumeration>
    <xsd:enumeration value="A"></xsd:enumeration>
   </xsd:restriction>
  </xsd:simpleType>
 </xsd:attribute>
 <xsd:attribute name="zoneRuleRef" type="xsd:string"
  use="required" />
</xsd:complexType>

<xsd:complexType name="zoneRule">
 <xsd:sequence>
  <xsd:element name="zone" type="dp:zone"
   maxOccurs="unbounded" minOccurs="1" />
 </xsd:sequence>
 <xsd:attribute name="name" type="xsd:string" use="required" />
 <xsd:attribute name="exclusivePlacement" type="xsd:boolean" />
  use="optional" />
</xsd:complexType>

<xsd:complexType name="zone">
 <xsd:attribute name="name" type="xsd:string" use="required" />
<xsd:complextType>

</xsd:schema>
```

カタログ・サーバーおよびコンテナー・サーバーの構成

プロパティー・ファイルを使用して、カタログ・サーバーおよびコンテナー・サーバーを構成します。 WebSphere eXtreme Scale には、カタログ・サーバーとコンテナー・サーバーの 2 つのタイプのサーバーがあります。カタログ・サーバーは、断片の配置を制御し、コンテナー・サーバーの検出とモニターをします。複数のカタログ・サーバーがまとまってカタログ・サービスを構成します。コンテナー・サーバーは、グリッドのアプリケーション・データを保管する Java™ 仮想マシンです。

マルチ・マスター複製トポロジーの構成

マルチ・マスター非同期複製フィーチャーでは、リンクを使用してドメイン・セットを相互接続します。その後、eXtreme Scale はそれらのリンクを介して複製を使用

し、ドメインを同期します。複数のドメイン間にリンクを定義するため、ほぼすべてのトポロジーを構成できます。各ドメインのカタログ・サーバーのプロパティー・ファイルでリンクを定義するか、 JMX プログラムまたは xsadmin コマンド行ユーティリティーを使用して実行時にリンクを定義します。どのようにリンクを作成しても、ドメインの現在のリンク・セットが、カタログ・サービス内に保管されます。これにより、ドメイン (グリッド) を再始動せずにリンクを追加および削除できます。

始める前に

マルチ・マスター・グリッド複製トポロジー (AP) では、さまざまなマルチ・マスター複製トポロジーの特性について説明しています。以下の手順では、複数のドメイン間にさまざまなリンクを構成して、トポロジー (任意のトポロジー) を形成する方法について説明します。手順の後に、ハブおよびスポーク・フォーメーションなどの特定のトポロジーをセットアップする方法を説明する例をいくつか示します。

手順

1. ブートストラップのために、トポロジー内の各ドメインのカタログ・サーバーのプロパティー・ファイル内でリンクを定義します。

   `objectGridServer.properties` (一部のシステムでは大/小文字が区別されます) という名前にして、カタログ・サービス・インスタンスの始動時に使用されるクラスパス内に配置すると、プロパティー・ファイルは自動的に検出されます。また、`startOgServer.bat|sh` スクリプトを呼び出すコマンド行で *-serverProps* パラメーターを使用して、場所を指定することもできます。

   プロパティー名では大/小文字が区別されるため、プロパティー・ファイルの更新時には大/小文字に注意してください。

   ローカル・ドメイン・ネーム

   「この」ドメインの名前を指定します (ドメイン A など)。

   例:

   ```
   domainName=A
   ```

   外部ドメイン・ネームのオプション・リスト

   マルチ・マスター複製トポロジー内の「その他の」ドメインの名前を指定します (ドメイン B など)。

   ```
   foreignDomains=B
   ```

   外部ドメイン・ネームのエンドポイントのオプション・リスト

   外部ドメイン (ドメイン B など) のカタログ・サービスの接続情報を指定します。

   例:

   ```
   B.endPoints=hostB1:2809, hostB2:2809
   ```

   外部ドメインに複数のカタログ・サービスがある場合には、すべてを指定します。

2. xsadmin コマンド・ユーティリティーまたは JMX プログラミングを使用して、実行時にリンクを追加または削除します。

ドメインのリンクは、複製メモリー内のカタログ・サービスで保持されます。このリンク・セットは、このドメインまたはその他のドメインを再始動することなく、いつでも管理者が変更できます。 xsadmin コマンド行ユーティリティーには、リンクを処理するためのいくつかのオプションが用意されています。

xsadmin ユーティリティーは 1 つのカタログ・サービス (単一ドメイン) に接続します。そのため、xsadmin を使用して、接続先のドメインと任意のその他のドメインとの間のリンクを作成および破棄できます。

例えば以下のように、コマンド行を使用して、新規リンクを作成します。

```
xsadmin –ch host –p 1099 –establishLink dname fdHostA:2809,fdHostB:2809
```

このコマンドは、「ローカル」ドメインと「dname」という外部ドメインの間に新規リンクを確立します。「dname」のカタログ・サービスは、fdHostA:2809 および fdHostB:2809 で実行されています。ローカル・ドメインには、JMX アドレスが host:1099 のカタログ・サービス JVM があります。外部ドメインからのすべてのカタログ・エンドポイントを指定して、ドメインへのフォールト・トレランス接続を可能にします。外部ドメインのカタログ・サービスで、単一の「ホスト:ポート」ペアを使用することはお勧めできません。

xsadmin で –ch および –p を使用して指定するローカル・カタログ・サービス JVM はどのようなものでも構いません。任意のカタログ JVM が機能します。カタログが WebSphere Application Server デプロイメント・マネージャー内にホストされている場合は通常、ポートは 9809 です。

外部ドメインに指定するポートは、非 JMX ポートです。 eXtreme Scale クライアントで使用する通常のポートです。

新規リンクを追加するコマンドの発行後に、カタログ・サービスは外部ドメインへの複製を開始するように、管理下のすべてのコンテナーに指示します。リンクは、両側には必要ありません。リンクは片側で作成するだけで十分です。

例えば以下のように、コマンド行を使用して、リンクを削除します。

```
xsadmin -ch host -p 1099 -dismissLink dname
```

このコマンドはドメインのカタログ・サービスに接続して、特定のドメインへの複製を停止するように指示します。リンクの解除は片側からのみ行う必要があります。

例

ドメイン A とドメイン B という 2 つドメインがあるセットアップを構成するものとします。

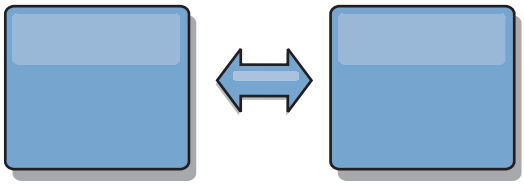


図 8. ドメイン間のリンク

ドメイン A 内のカタログ・サーバーのプロパティー・ファイルを以下のようにします。

```
domainName=A
foreignDomains=B
B.endPoints=hostB1:2809, hostB2:2809
```

ドメイン B 内のカタログ・サーバーのプロパティー・ファイルを以下のようにします。 2 つのプロパティー・ファイルが似ていることに着目してください。

```
domainName=B
foreignDomains=A
B.endPoints=hostB1:2809, hostB2:2809
```

2 つのドメインを始動すると、以下の特性を備えたすべてのグリッドがドメイン間で複製されます。

- 固有のドメイン・ネームを持つ専用カタログ・サービスがある
- ドメイン内の他のグリッドと同じグリッド名である
- ドメイン内の他のグリッドと同じ数の区画がある
- FIXED_PARTITION グリッドである (PER_CONTAINER グリッドは複製不可)
-
- ドメイン内の他のグリッドと同じデータ・タイプが複製される
- ドメイン内の他のグリッドと同じマップ・セット名、マップ名、および動的マップ・テンプレートがある

なお、ドメインの複製ポリシーは無視されます。

前の例は、他のドメインへのリンクを持つように各ドメインを構成する方法を示していますが、リンクは一方向で定義するだけで十分です。これは、ハブおよびスポーク・トポロジーでは特に便利であり、構成を大幅に単純化できます。スポークの追加時にハブ・プロパティー・ファイルを更新する必要はありません。各スポーク・ファイルにハブの情報を含めるだけで十分です。同様に、リング・トポロジーでは、各ドメインに、リング内の前と次のドメインへのリンクを含めるだけで十分です。

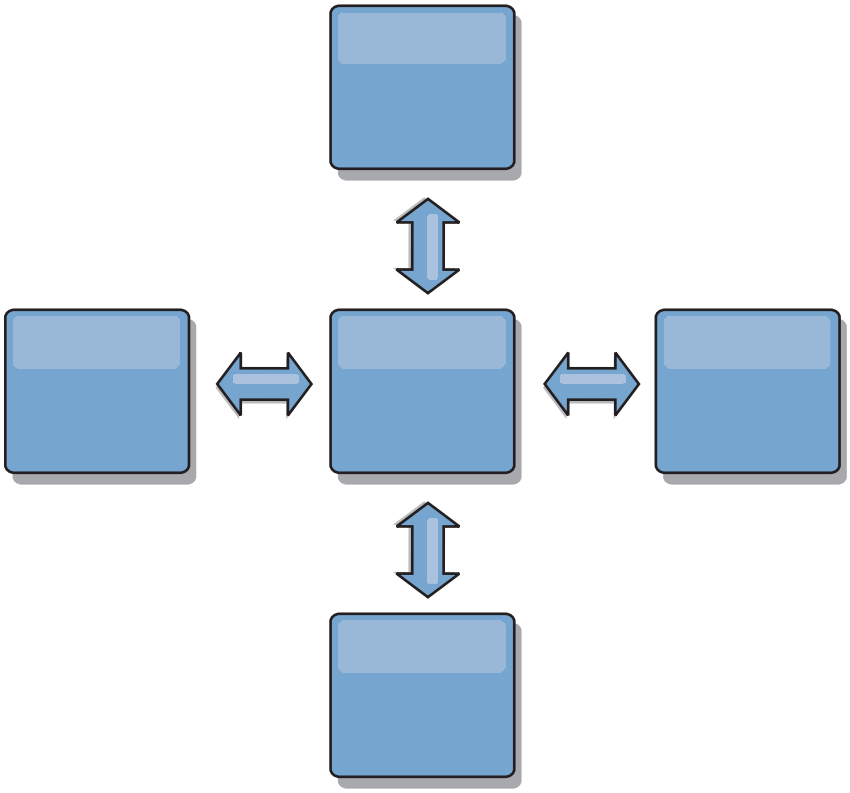


図 9. ハブおよびスポーク・トポロジー

ハブと 4 つのスポーク (ドメイン A、B、C、D) の場合、以下の例のようなカタログ・サーバー・プロパティー・ファイルになります。

```
domainName=Hub
```

最初のスポークのプロパティーは以下のようなものになります。

```
domainName=A
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

2 番目のスポークのプロパティーは以下のようなものになります。

```
domainName=B
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

3 番目のスポークのプロパティーは以下のようなものになります。

```
domainName=C
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

4 番目のスポークのプロパティーは以下のようなものになります。

```
domainName=D
foreignDomains=Hub
Hub.endPoints=hostH1:2809, hostH2:2809
```

サーバー・プロパティー・ファイル

サーバー・プロパティー・ファイルには、サーバーのさまざまな設定 (例えば、トレース設定、ロギング、およびセキュリティー構成など) を定義する複数のプロパティーが含まれます。サーバー・プロパティー・ファイルは、カタログ・サービスとコンテナー・サービスによって使用されます。

サンプル・サーバー・プロパティー・ファイル

extremescale_root/properties ディレクトリーにある sampleServer.properties ファイルを使用して、プロパティー・ファイルを作成することができます。

サーバー・プロパティー・ファイルの指定

サーバー・プロパティー・ファイルは、以下のいずれかの方法で指定できます。このリスト内の項目のいずれかを使用して後で設定を指定すると、前の設定はオーバーライドされます。例えば、サーバー・プロパティー・ファイルのシステム・プロパティー値を指定すると、そのファイルのプロパティーにより、クラスパスにある objectGridServer.properties ファイルの値が指定変更されます。

1. クラスパス内のわかりやすい名前のファイルで指定。このわかりやすい名前のファイルを現行ディレクトリーに置いても、その現行ディレクトリーがクラスパスにない限りそのファイルは検出されません。使用される名前は次のようになります。

   ```
   objectGridServer.properties
   ```

2. システム現行ディレクトリー内のファイルを指定する、スタンドアロンまたは WebSphere Application Server 構成のいずれかのシステム・プロパティーとして指定。このファイルをクラスパスに入れることはできません。

   ```
   -Dobjectgrid.server.props=file_name
   ```

3. startOgServer コマンドを実行する際のパラメーターとして指定。次のように、これらのプロパティーを手動で指定変更して、システム現行ディレクトリーにファイルを指定することができます。

   ```
   -serverProps file_name
   ```

4. ServerFactory.getServerProperties メソッドおよび ServerFactory.getCatalogServerProperties メソッドを使用したプログラマチックな指定変更。オブジェクトのデータに、プロパティー・ファイルからのデータが取り込まれます。

サーバー・プロパティー

一般プロパティー

workingDirectory

コンテナー・サーバー出力が書き込まれるロケーションを指定します。この値が指定されない場合、出力は、現行ディレクトリー内の log ディレクトリーに書き込まれます。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

デフォルト: 値なし

minThreads

組み込み evictor および DataGrid の操作で、実行時に内部スレッド・プールによって使用されるスレッドの最小数。

デフォルト: 10

maxThreads

組み込み evictor および DataGrid の操作で、実行時に内部スレッド・プールによって使用されるスレッドの最大数。

デフォルト: 50

traceSpec

コンテナー・サーバーのトレースおよびトレース仕様ストリングを使用可能にします。トレースは、デフォルトで使用不可に設定されています。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

デフォルト: `*=all=disabled`

traceFile

トレース情報を書き込むファイル名を指定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

systemStreamToFileEnabled

コンテナーから、ファイルに SystemOut、SystemErr、およびトレース出力を書き込めるように設定します。このプロパティーが `false` に設定されていると、出力はファイルに書き込まれず、代わりにコンソールに書き込まれます。

デフォルト: `true`

enableMBeans

ObjectGrid コンテナーの Managed Beans (MBean) を使用可能にします。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

デフォルト: `true`

serverName

サーバーを識別するために使用されるサーバーの名前を指定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

zoneName

サーバーが含まれるゾーンの名前を設定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

haManagerPort

peerPort と同義。HA マネージャーが使用するポート番号を指定します。このプロパティーが設定されていない場合は、カタログ・サービスは使用可能なポートを自動的に生成します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

listenerHost

オブジェクト・リクエスト・ブローカー (ORB) のバインド先のホスト名を指定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

ご使用の構成に複数のネットワーク・カードが含まれている場合、リスナー・ホストとリスナー・ポートを設定し、バインドする IP アドレスを JVM 内のオブジェクト・リクエスト・ブローカーに通知します。カタログ・サーバーとコンテナー・サーバーについては、サーバー・プロパティー・ファイル内で、リスナー・ホストとリスナー・ポートを指定します。使用する IP アドレスの指定を怠ると、コネクション・タイムアウトや異常な API 障害、クライアントがハングしたような状態になる、などの症状が生じます。

listenerPort

オブジェクト・リクエスト・ブローカー (ORB) のバインド先のポート番号を指定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

JMXServicePort

MBean サーバーが listen するポート番号を指定します。このプロパティーは、コンテナー・サーバーとカタログ・サービスの両方に適用されます。

コンテナー・サーバー・プロパティー

statsSpec

コンテナー・サーバーの統計仕様を指定します。

例:

```
all=disabled
```

memoryThresholdPercentage

メモリー・ベース除去のメモリーしきい値を設定します。このパーセンテージは、Java 仮想マシン (JVM) で使用される最大ヒープを指定し、これを超えると除去が行われます。デフォルト値は -1 で、メモリーしきい値が設定されていないことを示します。 memoryThresholdPercentage プロパティーが設定されていると、MemoryPoolMXBean 値は指定された値に設定されます。詳しくは、Java API 仕様の MemoryPoolMXBean インターフェースを参照してください。ただし、除去は Evictor で使用可能に設定されている場合にのみ行われます。メモリー・ベースの除去を使用可能に設定するには、*製品概要*を参照してください。このプロパティーは、コンテナー・サーバーにのみ適用されます。

catalogServiceEndPoints

カタログ・サービス・ドメインに接続するエンドポイントを指定します。この値は、形式 `host:port<,host:port>` で指定します。ここで、ホスト値は、 listenerHost 値で、ポート値は、カタログ・サーバーの listenerPort 値です。このプロパティーは、コンテナー・サーバーにのみ適用されます。

カタログ・サービス・プロパティー

domainName

複数ドメインに経路指定する際にクライアントに対してこのカタログ・サー

ビス・ドメインを固有に識別するために使用するドメイン・ネームを指定します。このプロパティーは、カタログ・サービスにのみ適用されます。

enableQuorum
カタログ・サービスのクォーラムを使用可能にします。使用可能なコンテナー・サーバー上の区画の配置を変更できるようにする前に、クォーラムを使用して、大半のカタログ・サービス・ドメインを確実に使用可能にします。クォーラムを使用可能にするには、この値を `true` または `enabled` に設定します。デフォルト値は `disabled` です。このプロパティーは、カタログ・サービスにのみ適用されます。

catalogClusterEndpoints
カタログ・サービスのカタログ・サービス・ドメイン・エンドポイントを指定します。このプロパティーで、カタログ・サービス・エンドポイントを指定して、カタログ・サービス・ドメインを開始します。以下のフォーマット設定を使用します。

```
serverName:hostName:clientPort:peerPort<serverName:hostName:clientPort:peerPort>
```

このプロパティーは、カタログ・サービスにのみ適用されます。

heartBeatFrequencyLevel
ハートビートを発生させる頻度を指定します。 ハートビートの頻度レベルは、リソースの使用量と障害発見時間のトレードオフです。頻繁にハートビートが発生するほど、より多くのリソースが使用されますが、より迅速に障害は発見されます。このプロパティーは、カタログ・サービスにのみ適用されます。以下のいずれかの値を使用します。

- `0`: 標準的な速度でのハートビート・レベルを指定します。この値を使用すると、リソースを過剰使用することなく適正な速度でフェイルオーバーの検出が行われます。 (デフォルト)
- `-1`: 積極的なハートビート・レベルを指定します。この値を指定すると、障害はより迅速に検出されますが、プロセッサーおよびネットワークのリソースもより多く使用します。このレベルは、サーバーが混んでいる場合に、欠落ハートビートの検出精度が高くなります。
- `1`: 緩やかなハートビート・レベルを指定します。この値を使用すると、ハートビート頻度の減少により障害を検出するまでの時間は増加しますが、プロセッサーおよびネットワークの使用も減少します。

セキュリティー・サーバー・プロパティー

サーバー・プロパティー・ファイルは、 eXtreme Scale サーバー・セキュリティーの構成にも使用されます。単一サーバー・プロパティー・ファイルを使用して、基本的なプロパティーおよびセキュリティー・プロパティーの両方を指定できます。

一般セキュリティー・プロパティー

securityEnabled
`true` に設定すると、コンテナー・サーバー・セキュリティーが使用可能になります。デフォルト値は `false` です。このプロパティーは、カタログ・サーバーに提供される `objectGridSecurity.xml` ファイルに指定されている securityEnabled property と一致する必要があります。

credentialAuthentication

このサーバーが、クレデンシャル認証をサポートするかどうかを示します。次のいずれかの値を選択します。

- Never: サーバーは、クレデンシャル認証をサポートしません。
- Supported: クライアントがクレデンシャル認証をサポートする場合に、このサーバーもクレデンシャル認証をサポートします。
- Required: クライアントはクレデンシャル認証を必要とします。

クレデンシャル認証に関して詳しくは、 386 ページの『アプリケーション・クライアントの認証』を参照してください。

Transport Layer Security の設定

transportType

サーバーのトランスポート・タイプを指定します。以下のいずれかの値を使用します。

- TCP/IP: サーバーが、TCP/IP 接続のみをサポートすることを示します。
- SSL-Supported: サーバーが TCP/IP 接続と Secure Sockets Layer (SSL) 接続の両方をサポートすることを示します。 (デフォルト)
- SSL-Required: サーバーが SSL 接続を必要とすることを示します。

SSL 構成プロパティー

別名 (alias)

鍵ストア内の別名を指定します。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。

デフォルト: 値なし

contextProvider

トラスト・サービスのコンテキスト・プロバイダーの名前を指定します。有効でない値を指定すると、コンテキスト・プロバイダー・タイプが正しくないことを示すセキュリティー例外が発生します。

有効値: IBMJSSE2、IBMJSSE、IBMJSSEFIPS など。

プロトコル

クライアントに使用するセキュリティー・プロトコルのタイプを指定します。このプロトコルの値は、使用する Java Secure Socket Extension (JSSE) プロバイダーに基づいて設定します。有効でない値を指定すると、プロトコル値が正しくないことを示すセキュリティー例外が発生します。

有効値: SSL、SSLv2、SSLv3、TLS、TLSv1 など。

keyStoreType

鍵ストアのタイプを示します。有効でない値を指定すると、ランタイム・セキュリティー例外が発生します。

有効値: JKS、JCEK、PKCS12 など。

trustStoreType

トラストストアのタイプを指定します。有効でない値を指定すると、ランタイム・セキュリティー例外が発生します。

有効値: JKS、JCEK、PKCS12 など。

keyStore
鍵ストア・ファイルへの完全修飾パスを指定します。

例:

```
etc/test/security/client.private
```

trustStore
トラストストア・ファイルへの完全修飾パスを指定します。

例:

```
etc/test/security/server.public
```

keyStorePassword
鍵ストアのストリング・パスワードを指定します。この値はエンコードすることも、実際の値を使用することもできます。

trustStorePassword
トラストストアのストリング・パスワードを指定します。この値はエンコードすることも、実際の値を使用することもできます。

clientAuthentication
このプロパティーを true に設定した場合は、SSL クライアントを認証する必要があります。SSL クライアントの認証は、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とは、証明書チェーンに基づくユーザー・レジストリーと照合してクライアントを認証することです。このプロパティーは、サーバーが正しいクライアントに接続していることを保証します。

SecureTokenManager の設定

SecureTokenManager の設定は、サーバー相互認証の秘密ストリングの保護とシングル・サインオン・トークンの保護に使用されます。 384 ページの『グリッド・セキュリティー』を参照してください。

secureTokenManagerType
SecureTokenManager 設定のタイプを指定します。次の設定のいずれかを使用できます。

- none: セキュア・トークン・マネージャーが使用されないことを示します。
- default: WebSphere eXtreme Scale 製品で提供されるトークン・マネージャーが使用されることを示します。 SecureToken 鍵ストア構成を指定する必要があります。
- custom: SecureTokenManager 実装クラスを使用して指定した独自のトークン・マネージャーがあることを示しています。

customTokenManagerClass
SecureTokenManagerType プロパティー値を custom と指定した場合に、SecureTokenManager 実装クラスの名前を指定します。この実装クラスでは、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
カスタム SecureTokenManager 実装クラス・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) メソッドを使用して SecureTokenManager オブジェクトに設定されます。

セキュア・トークンの鍵ストア構成

secureTokenKeyStore
公開鍵と秘密鍵のペアおよび秘密鍵が保管されている鍵ストアのファイル・パス名を指定します。

secureTokenKeyStoreType
鍵ストアのタイプ (例、JCKES など) 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、鍵ストアが秘密鍵をサポートしている必要があります。

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
暗号の指定に使用されるアルゴリズムを指定します。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。

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。

認証ストリング

authenticationSecret
サーバーのチャレンジに使用される秘密ストリングを指定します。サーバーは始動すると、このストリングをプレジデント・サーバーあるいはカタログ・サーバーに提示する必要があります。秘密ストリングが、プレジデント・サーバーにあるものと一致した場合に、このサーバーは参加できます。

ポートの構成

WebSphere eXtreme Scale は、分散キャッシュであり、オブジェクト・リクエスト・ブローカー (ORB) および伝送制御プロトコル (TCP) スタックを使用して Java™ 仮想マシンと他のマシン間で通信するためにオープン・ポートを必要とします。

ネットワーク・ポートの計画

WebSphere eXtreme Scale は、分散キャッシュであり、オブジェクト・リクエスト・ブローカー (ORB) および伝送制御プロトコル (TCP) スタックを使用して Java 仮想マシンと他のマシン間で通信するためにオープン・ポートを必要とします。特

に、カタログ・サービスやコンテナーを複数ポートで使用している場合などのファイアウォール環境では、ポートを計画し、制御することが必要です。

カタログ・サービス・ドメイン

カタログ・サービス・ドメインでは、以下のポートを定義する必要があります。

peerPort
高可用性 (HA) マネージャーがピア・カタログ・サーバー間で TCP スタックを介して通信するためのポートを指定します。

clientPort
カタログ・サーバーがカタログ・サービス・データにアクセスするためのポートを指定します。

JMXServicePort
Java Management Extensions (JMX) サービスが使用することになるポートを指定します。

listenerPort
コンテナーおよびクライアントが ORB を介してカタログ・サービスと通信するための ORB リスナー・ポートを定義します。

上記のポートを定義する方法は、スタンドアロン・モードを使用しているか、あるいは eXtreme Scale カタログ・サーバーを WebSphere Application Server 環境で開始しているかによって異なります。

- **スタンドアロン・モードの場合:**

  上記のポートを指定するには、以下の例のように、スタンドアロン・モードでオプションを指定した startOgServer コマンドを使用します。

  ```
  –catalogServiceEndPoints cs1:host1:clientPort:peerPort, cs2:host2:clientPort:peerPort,
  cs3:host3:clientPort:peerPort -listenerPort <orbPort> -JMXServicePort <jmxPort>
  ```

  スタンドアロン・モードでのカタログ・サービスの開始について詳しくは、354 ページの『スタンドアロン環境でのカタログ・サービスの開始』を参照してください。

- **WebSphere Application Server 環境の場合:**

  カタログ・サービス・ドメインは管理コンソールで定義できます。詳しくは、371 ページの『WebSphere Application Server でのカタログ・サービス・ドメインの作成』を参照してください。

コンテナー・サーバー

WebSphere eXtreme Scale コンテナー・サーバーも、いくつかのポートが作動することを必要とします。デフォルトでは、eXtreme Scale コンテナー・サーバーは、HA マネージャー・ポートおよび ORB リスナー・ポートを、動的ポートと共に自動的に生成します。ファイアウォール環境では、ポートを計画し、制御することはユーザーにとって有益なので、eXtreme Scale コンテナー・サーバーを開始する際のオプションが用意されています。このコンテナー・サーバーは、以下の例に示すように、startOgServer コマンド中のオプションで HAManager ポートおよび ORB リスナー・ポートを指定して開始することができます。

```
-HaManagerPort <peerPort>
-listenerPort <orbPort>
```

ポート制御の適切な計画は有益ですが、数百の Java 仮想マシンを 1 台のマシンで開始する場合は、これらのポートの計画と管理には特有の難しさがあります。ポート競合があると、サーバーの始動が失敗します。

セキュリティーが有効になっている場合、上記のリストに示されたポートに加えて、Secure Socket Layer (SSL) ポートが必要です。
`Dcom.ibm.CSI.SSLPort=<sslPort>` を **-jvmArgs** 引数として使用すると、SSL ポートが `<sslPort>` に設定されます。eXtreme Scale を使用したポートの計画については、セキュリティー設定を参照してください。

スタンドアロン・モードでのポートの構成

カタログ・サービス・インスタンスをホストする Java 仮想マシンには 4 つのポートが必要です。内部使用のために 2 つのポートが使用され、クライアントおよびコンテナー断片が Internet Inter-ORB Protocol (IIOP) を使用して通信するために 3 番目のポートが使用され、 Java Management Extensions (JMX) の通信のために 4 番目のポートが使用されます。

このタスクについて

カタログ・サービス Java 仮想マシン (JVM) エンドポイント

eXtreme Scale は、主に Java 仮想マシン 間の通信のために IIOP を使用します。カタログ・サービス Java 仮想マシン は、IIOP サービスおよびグループ・サービス・ポートのためにポートの明示的構成を必要とする唯一の Java 仮想マシン です。内部ポートは **-catalogServiceEndPoints** コマンド行オプションを使用して指定されます。

```
-catalogServiceEndPoints <server:host:port:port,server:host:port:port>
```

-catalogServiceEndPoints コマンド行オプションで、サーバーあたり 2 つのポートを構成できます。 IIOP ポートは次のコマンド行オプションを使用して構成されます。

```
-listenerHost <host_name>
-listenerPort <port>
```

各カタログ・サービス JVM の始動時に、その JVM 用の単一リスナー・ポートとともに、すべて揃った カタログ・サービス・エンドポイントのセットを指定します。

コンテナー JVM エンドポイント

コンテナー Java 仮想マシン は 2 つのポートを使用します。 1 つのポートは内部使用で、もう 1 つのポートは IIOP 通信用です。コンテナー Java 仮想マシン は通常、未使用ポートを自動的に検出し、次いで 2 つの動的に作成されたポートを使用するように自己構成します。この自動処理により構成は最小化されます。ただし、ファイアウォールが使用されていたり、明示的にポートを構成する場合は、コマンド行オプションにより、使用するオブジェクト・リクエスト・ブローカー (ORB) ポートを指定することができます。

```
-listenerHost <host_name>
-listenerPort <port>
```

これらのコマンド行オプションを使用して、ホスト名 (正しいネットワーク・カードにバインドするために重要) とその JVM 用に指定されるポートを指定できます。 **-haManagerPort** コマンド行引数を使用して内部ポートを指定することができます。ただし、最も単純な構成では、実行時にポートを選択させることができます。

手順

1. hostA で最初のカタログ・サーバーを始動します。 コマンドの例は以下のとおりです。

   ```
   ./startOgServer.sh cs1 -listenerHost hostA -listenerPort 2809
   -catalogServiceEndPoints cs1:hostA:6601:6611,cs2:hostB:6601:6611
   ```

2. hostB で 2 番目のカタログ・サーバーを始動します。 コマンドの例は以下のとおりです。

   ```
   ./startOgServer.sh cs2 -listenerHost hostB -listenerPort 2809
   -catalogServiceEndPoints cs1:hostA:6601:6611,cs2:hostB:6601:6611
   ```

   クライアントは、カタログ・サービスのリスナー・エンドポイントを知ってさえいれば十分です。クライアントは、カタログ・サービスから自動的にコンテナー Java 仮想マシン のエンドポイント (すなわち、データを保管する Java 仮想マシン) を取得します。前述の例でカタログ・サービスに接続するには、クライアントは、以下の `host:port` のペアのリストを接続 API に渡す必要があります。

   ```
   hostA:2809,hostB:2809
   ```

   コンテナー JVM を始動して、カタログ・サービス例を使用するには、次のコマンドを使用します。

   ```
   ./startOgServer.sh c0 -catalogServiceEndPoints hostA:2809,hostB:2809
   ```

WebSphere Application Server 環境でのポートの構成

カタログ・サービスは、デフォルトではデプロイメント・マネージャー内で単一インスタンスを実行し、デプロイメント・マネージャー Java 仮想マシンの Internet Inter-ORB Protocol (IIOP) ブートストラップ・ポートを使用します。

このタスクについて

セル内の Web アプリケーションまたは Enterprise JavaBeans™ (EJB) アプリケーションは、カタログ・サービス・ブートストラップ・ポートを指定するのではなく、`null,null` 接続呼び出しを使用して、同じセル内のグリッドに接続できます。

WebSphere Application Server のカタログ・サービス・ドメインがデプロイメント・マネージャーでホストされている場合、セル外部のクライアント (Java Platform, Standard Edition クライアントを含む) は、デプロイメント・マネージャー・ホスト名および IIOP ブートストラップ・ポートを使用してカタログ・サービスに接続する必要があります。カタログ・サービスが WebSphere Application Server セルで実行され、クライアントがそのセル外で実行されているとき、クライアントをカタログ・サービスに指定するために必要な情報については、WebSphere Application Server 管理コンソールの eXtreme Scale ドメイン構成ページを参照してください。

クライアントが WebSphere Application Server セル内にある場合は、CatalogServerProperties インターフェースから直接ポートを取得できます。

7.1+ `catalog.services.cluster` プロパティーを使用してクライアント接続ポートを見つけることは引き続きできますが、この手法は推奨されません。この手法を使っても `catalog.services.cluster` エントリーが見つからない場合は、デプロイメント・マネージャーの IIOP ポートをクライアント接続に使用してください。

eXtreme Scale は、グループ・メンバーシップ用の HA マネージャーの分散および整合性サービス (DCS) ポートを再利用します。 Java Management Extensions (JMX) ポートも再利用されます。

オブジェクト要求ブローカーの構成

orb.properties ファイルを使用して、オブジェクト・リクエスト・ブローカー (ORB) がグリッドのトランスポート動作を変更するために使用するプロパティーを渡します。 WebSphere® eXtreme Scale は、オブジェクト・リクエスト・ブローカー (ORB) を使用して、プロセス間の通信を使用可能にします。 WebSphere eXtreme Scale または WebSphere Application Server で WebSphere eXtreme Scale サーバー用に提供されるオブジェクト・リクエスト・ブローカー (ORB) を使用するために、アクションは必要ありません。このセクションでは、実行することができる、あるいは実行する必要がある、いくつかのチューニング考慮事項とその他の ORB 関連の考慮事項のタスクの概要を説明します。

ORB プロパティー・ファイル

Object Request Broker (ORB) がグリッドのトランスポート動作を変更するために使用するプロパティーが、`orb.properties` ファイルを使用して受け渡されます。

ロケーション

`orb.properties` ファイルは、`java/jre/lib` ディレクトリーにあります。WebSphere Application Server `java/jre/lib` ディレクトリー内のファイルを変更すると、そのインストール済み環境に構成されているアプリケーション・サーバーも、そのファイルの設定を使用します。

ベースラインの設定

以下の設定は、適切なベースラインですが、必ずしもすべての環境に最適な設定とは限りません。ご使用の環境においてどの値が適切であるか、正しい決定をできるようにこれらの設定をよく理解してください。

```
com.ibm.CORBA.RequestTimeout=30
com.ibm.CORBA.ConnectTimeout=10
com.ibm.CORBA.FragmentTimeout=30
com.ibm.CORBA.ThreadPool.MinimumSize=256
com.ibm.CORBA.ThreadPool.MaximumSize=256
com.ibm.CORBA.ThreadPool.IsGrowable=false
com.ibm.CORBA.ConnectionMultiplicity=1
com.ibm.CORBA.MinOpenConnections=1024
com.ibm.CORBA.MaxOpenConnections=1024
```

```
com.ibm.CORBA.ServerSocketQueueDepth=1024
com.ibm.CORBA.FragmentSize=0
com.ibm.CORBA.iiop.NoLocalCopies=true
com.ibm.CORBA.NoLocalInterceptors=true
```

プロパティーの説明

タイムアウト設定

以下の設定は、ORB が要求の操作に見切りをつけるまで待機する時間に関係しています。

要求タイムアウト

プロパティー名: com.ibm.CORBA.RequestTimeout

値: 秒数を表す整数値。

説明: 要求 (任意) が応答を待機する秒数。その秒数を過ぎると待機を止めます。このプロパティーは、ネットワーク停止の障害が発生した場合にクライアントがフェイルオーバーするまでに要する時間に影響します。このプロパティーの値を極端に低く設定すると、要求が誤ってタイムアウトになる可能性があります。不用意なタイムアウトを回避するためにこのプロパティーの値は慎重に考慮してください。

接続タイムアウト

プロパティー名: com.ibm.CORBA.ConnectTimeout

値: 秒数を表す整数値。

説明: ソケット接続試行で待機する秒数。その秒数を過ぎると待機を止めます。このプロパティーは、要求タイムアウトと同様に、ネットワーク停止の障害が発生した場合にクライアントがフェイルオーバーするまでに要する時間に影響します。一般に、このプロパティーは要求タイムアウト値よりも小さい値に設定します。接続の確立に要する時間は比較的一定であるためです。

フラグメント・タイムアウト

プロパティー名: com.ibm.CORBA.FragmentTimeout

値: 秒数を表す整数値。

説明: フラグメント要求が待機する秒数。その秒数を過ぎると待機を止めます。このプロパティーは、要求タイムアウト・プロパティーと類似しています。

スレッド・プールの設定

このプロパティーは、スレッド・プール・サイズを特定のスレッド数に制約します。サーバー要求がソケットで受信されると、そのサーバー要求をスピンオフさせるために、ORB によってスレッドが使用されます。このプロパティー値を低い値に設定すると、ソケットのキュー項目数が増加して、タイムアウトになる可能性もあります。

接続多重度

プロパティー名: com.ibm.CORBA.ConnectionMultiplicity

値: クライアントとサーバーの間の接続数を表す整数値。デフォルト値は 1 です。これより大きい値に設定すると、複数接続にまたがる多重化の設定になります。**説明:** ORB が任意のサーバーとの複数の接続を使用できるように許可します。理論的に、この値の設定は、複数接続にまたがる並列性を促進します。実際には、接続多重度の設定によるパフォーマンス上の利点はありません。このパラメーターは設定しないでください。

オープン接続

プロパティー名:
com.ibm.CORBA.MinOpenConnections、com.ibm.CORBA.MaxOpenConnections

値: 接続数を表す整数値。**説明:** オープン接続の最小数と最大数。 ORB は、クライアントとの間に確立された接続のキャッシュを保持します。 com.ibm.CORBA.MaxOpenConnections 値を超過すると、その接続は消去されます。接続の消去は、グリッド内の動作の低下の原因になる可能性があります。

成長可能

プロパティー名: com.ibm.CORBA.ThreadPool.IsGrowable

値: ブール値。 `true` または `false` に設定します。**説明:** これを使用可能に設定すると、 ORB が着信要求用に使用するスレッド・プールは、そのプールで現在サポートするものよりもさらに大きく成長します。プール・サイズを上回ると、要求の処理のために新規スレッドが作成されますが、そのスレッドはプールされません。

サーバー・ソケットのキュー項目数

プロパティー名: com.ibm.CORBA.ServerSocketQueueDepth

値: 接続数を表す整数値。**説明:** クライアントからの着呼接続のキューの長さを指定します。 ORB は、クライアントからの着呼接続をキューに入れます。キューがフルになると、接続は拒否されます。接続の拒否は、グリッド内の動作の低下の原因になる可能性があります。

フラグメント・サイズ

プロパティー名: com.ibm.CORBA.FragmentSize

値: バイト数を指定する整数。デフォルトは 1024 です。**説明:** ORB が要求の送信時に使用する最大パケット・サイズを指定します。要求がフラグメント・サイズ制限より大きい場合、その要求は要求フラグメントに分割されて、それぞれ別々に送信されて、サーバー上で再組み立てされます。要求のフラグメント化は、パケットの再送が必要になる可能性のある不安定なネットワークで有効です。ただし、ネットワークの信頼性が高い場合、要求をフラグメントに分割すると、オーバーヘッドの原因になる可能性があります。

ローカル・コピーなし

プロパティー名: com.ibm.CORBA.iiop.NoLocalCopies

値: ブール値。 `true` または `false` に設定します。 **説明:** ORB が参照による受け渡しをするかどうかを指定します。 ORB は、デフォルトで、値の呼び出しによる受け渡しを使用します。インターフェースがローカルで呼び出される際、値の呼び出しによる受け渡しは、パスに余分なガーベッジやシ

リアライゼーションのコストをもたらす原因になります。この値を true に設定すると、ORB は、値の呼び出しによる受け渡しよりも効率的な参照による受け渡し方式を使用します。

ローカル・インターセプターなし

プロパティー名: com.ibm.CORBA.NoLocalInterceptors

値: ブール値。 `true` または `false` に設定します。 **説明:** ローカル要求 (内部プロセス) の場合にも、ORB が要求インターセプターを呼び出すかどうかを指定します。 WebSphere eXtreme Scale が使用するインターセプターは、セキュリティーと経路処理を目的とし、要求がプロセス内で処理される場合には必須ではありません。プロセス間を仲介するインターセプターは、リモート・プロシージャー・コール (RPC) 操作の場合にのみ必要です。ローカル・インターセプターなしを設定すると、ローカル・インターセプターを使用することにより生じる余分なオーバーヘッドを回避できます。

重要: WebSphere eXtreme Scale セキュリティーが使用可能になっている場合は、com.ibm.CORBA.NoLocalInterceptors プロパティー値を `FALSE` に設定してください。セキュリティー・インフラストラクチャーは、認証のためにインターセプターを使用します。

ObjectGrid のクライアントとサーバー間でトランスポート・セキュリティーを実現するには、 `orb.properties` ファイルにさらにプロパティーを追加する必要があります。これらのプロパティーについて詳しくは、 392 ページの『トランスポート層セキュリティーおよび Secure Sockets Layer』にあるトランスポート・セキュリティー・サポートの `orb.properties` ファイルに関するセクションを参照してください。

ORB プロパティーおよびファイル記述子の設定

チューニング考慮事項には、Object Request Broker (ORB) プロパティーおよびファイル記述子の設定などが含まれます。

ORB プロパティー

ORB は WebSphere eXtreme Scale によって TCP スタックでの通信のために使用されます。必要な `orb.properties` ファイルは、`java/jre/lib` ディレクトリーにあります。大容量オブジェクトの重い負荷に備えるため、以下の設定値を指定して ORB フラグメント化を有効にしてください。

```
com.ibm.CORBA.FragmentSize=<right size>
```

スレッド・プールの増大を防止するため、次のように設定してください。

```
com.ibm.CORBA.ThreadPool.IsGrowable=false
```

異常なシチュエーションでの過剰スレッドを回避するため、以下を設定して適切なタイムアウトを設定してください。

- com.ibm.CORBA.RequestTimeout
- com.ibm.CORBA.ConnectTimeout
- com.ibm.CORBA.FragmentTimeout

ファイル記述子

UNIX® システムおよび Linux システムでは、プロセス当たりに許容されるオープン・ファイルの数に制限があります。オペレーティング・システムが、許容されるオープン・ファイルの数を指定します。この値が小さすぎる場合、AIX ではメモリー割り振りエラーが発生し、多すぎるオープン・ファイルはログに記録されます。

UNIX システム端末ウィンドウで、この値をデフォルトのシステム値よりも大きく設定してください。クローンを持つ大容量 SMP マシンの場合、無限に設定してください。

AIX 構成では、コマンド `ulimit -n -1` を使用して、この値を `-1` (無限) に設定してください。

Solaris 構成の場合、コマンド `ulimit -n 16384` を使用して、この値を `16384` に設定してください。

コマンド `ulimit –a` を使用すれば、現行値を表示できます。

ORB での NIO または ChannelFramework の使用可能化

WebSphere eXtreme Scale は、ORB でノンブロッキング I/O (NIO) を使用可能にするために、TransportMode を ChannelFramework に設定するサーバー・プロパティーを用意しています。

始める前に

既存のサーバー・プロパティー・ファイルを検索するか、サーバー・プロパティー・ファイルを作成します。カタログ・サービスおよびコンテナー・サーバーで ChannelFramework を使用可能にすることができます。詳しくは、サーバー・プロパティー・ファイルを参照してください。

このタスクについて

現在、WebSphere eXtreme Scale スタンドアロン・シナリオで NIO または ChannelFramework で実行できます。 IBM ORB でノンブロッキング I/O (NIO) を使用して実行するには、 IBM ORB の TransportMode を ChannelFramework に設定する必要があります。デフォルトでは、IBM ORB は Pluggable モードで実行されます。 WebSphere eXtreme Scale にはサーバー・プロパティーがあり、TransportMode を ChannelFramework に設定します。

重要:

WebSphere Application Server の現行リリースでは、Pluggable モードのみがサポートされます。 WebSphere eXtreme Scale は WebSphere Application Server と統合して実行する場合には、Pluggable モードに従う必要があります。 WebSphere eXtreme Scale は WebSphere Application Server SSL/Transport Security も使用するため、現在は Pluggable 限定モードもサポートします。

手順

1. enableChannelFramework=true プロパティーをサーバー・プロパティー・ファイルに追加します。

2. サーバー・プロパティー・ファイルが ORB プロパティー・ファイルと矛盾していないか、確認します。

   サーバー・プロパティー・ファイルで ChannelFramework TransportMode を使用可能にしたのにもかかわらず、 orb.properties ファイルで TransportMode を Pluggable に設定している場合には、サーバーは orb.properties の設定をオーバーライドしません。 2 つの設定が存在するという警告メッセージがログに表示されます。 enableChannelFramework=true プロパティーを有効にするには、TransportMode を Pluggable に設定することを指示するプロパティーを調整します (com.ibm.CORBA.TransportMode=Pluggable を ChannelFramework に変更するか、プロパティーを削除します)。
3. カタログ・サービスまたはコンテナー・サーバーの始動時に更新したサーバー・プロパティー・ファイルを提供します。 サーバー・プロパティー・ファイルを使用したサーバーの始動の詳細については、サーバー・プロパティー・ファイルを参照してください。

タスクの結果

カタログ・サービスまたはコンテナー・サーバーで channelFramework TransportMode を使用する場合には、以下のメッセージがログに出力されます。

```
CWOBJ0052I: The IBM ORB TransportMode property was set to ChannelFramework
(IBM ORB TransportMode プロパティーが ChannelFramework に設定されました)
```

以下のメッセージがログに表示された場合には、前述のとおり ORB プロパティーを検査してください。

```
CWOBJ0055W: The IBM ORB TransportMode property was set to ChannelFramework
in the server properties file, but the existing orb.properties file already
had a TransportMode set. The TransportMode will not be overridden.
(サーバー・プロパティー・ファイルで IBM ORB TransportMode プロパティーが
ChannelFramework に設定されましたが、既存の orb.properties ファイルには既に
TransportMode が設定されています。TransportMode はオーバーライドされません。)
```

なお、ChannelFramework を使用可能にすると、ServerSocketQueueDepth の最大値は 512 になります。 orb.properties の ServerSocketQueueDepth 設定が 512 より大きい場合は、サーバーは orb.properties の ServerSocketQueueDepth を 512 に自動的に設定して、情報メッセージをログに出力することでユーザーに通知します。アクションは不要です。

```
CWOBJ0053I: The IBM ORB ServerSocketQueueDepth property was set to 512 to run
with correctly with the ChannelFramework TransportMode.
(ChannelFramework TransportMode で正常に実行されるように、
IBM ORB の ServerSocketQueueDepth が 512 に設定されました。)
```

スタンドアロン WebSphere eXtreme Scale プロセスでのオブジェクト・リクエスト・ブローカーの使用

WebSphere Application Server または WebSphere Application Server Network Deployment を含まない環境で、オブジェクト・リクエスト・ブローカー (ORB) を直接使用するアプリケーションを使用して WebSphere eXtreme Scale を使用することができます。

始める前に

eXtreme Scale に組み込まれていないアプリケーション、あるいは他のコンポーネントやフレームワークを実行中に、eXtreme Scale と同じプロセス内で ORB を使用する場合、追加のタスクを実行して、ご使用の環境で eXtreme Scale が正しく実行されていることを確認する必要がある場合があります。

このタスクについて

ご使用の環境で ORB の使用を初期化するには、`orb.properties` ファイルに **ObjectGridInitializer** プロパティーを追加します。 ORB を使用して、ご使用の環境にある eXtreme Scale プロセスと別のプロセスの間の通信を使用可能にします。`orb.properties` ファイルは、`java/jre/lib` ディレクトリーにあります。プロパティーおよび設定の説明については、 206 ページの『ORB プロパティー・ファイル』を参照してください。

手順

次の行を入力してから、変更を保存します。

```
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer
```

タスクの結果

eXtreme Scale は ORB を正しく初期化し、その ORB が使用可能に設定されている他のアプリケーションと共存できます。

eXtreme Scale でカスタム・バージョンの ORB を使用する場合は、『カスタム・オブジェクト・リクエスト・ブローカーの構成』を参照してください。

カスタム・オブジェクト・リクエスト・ブローカーの構成

WebSphere eXtreme Scale は、プロセス間の通信を使用可能にするために Object Request Broker (ORB) を使用します。 WebSphere eXtreme Scale または WebSphere Application Server によって、ご使用の WebSphere eXtreme Scale サーバー用に提供される Object Request Broker (ORB) を使用する際は、何のアクションも必要ありません。必要な作業はほとんどなく、ご使用の WebSphere eXtreme Scale クライアント用に同じ ORB を使用することができます。「カスタム」ORB を使用する必要がある場合、IBM SDK と一緒に提供される ORB が良い選択です。ただし、ここに説明するとおり、いくらか構成を行う必要があります。他のベンダーから提供される ORB を使用することも可能です。この場合も構成が必要になります。

始める前に

WebSphere eXtreme Scale または WebSphere Application Server で提供される ORB、IBM SDK で提供される ORB、サード・パーティーの ORB のうち、いずれを使用するかを決定します。

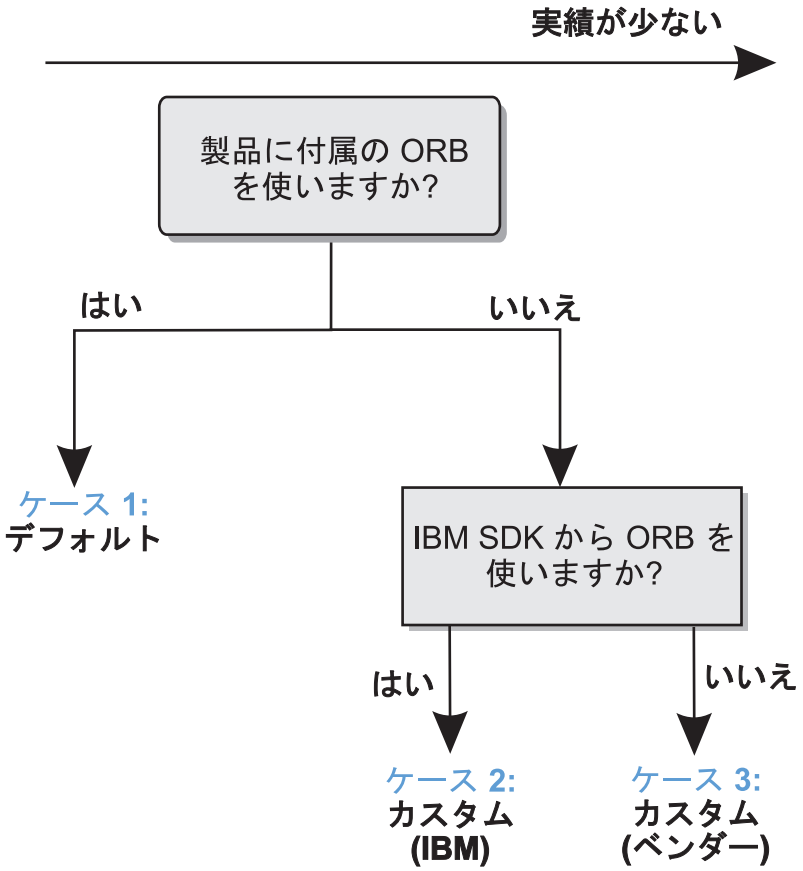


図 10. ORB の選択

WebSphere eXtreme Scale サーバー・プロセス用と WebSphere eXtreme Scale クライアント・プロセス用に、別々に決定することができます。 eXtreme Scale は、ほとんどのベンダーの開発キットをサポートしていますが、サーバー・プロセスとクライアント・プロセスの両方において、eXtreme Scale で提供される ORB を使用することを推奨します。eXtreme Scale は、Sun Microsystems Java Development Kit (JDK) で提供される ORB はサポートしません。

このタスクについて

選択した ORB の使用に必要な構成を、よく理解してください。

ケース 1: デフォルトの ORB

- WebSphere eXtreme Scale サーバー・プロセスでは、WebSphere eXtreme Scale または WebSphere Application Server で提供される ORB を使用するために必要とされる構成はありません。
- WebSphere eXtreme Scale クライアント・プロセスでは、WebSphere eXtreme Scale または WebSphere Application Server で提供される ORB を使用するために、最小限のクラスパス構成が必要となります。

ケース 2: カスタム ORB (IBM)

IBM SDK で提供される ORB を使用するために WebSphere eXtreme Scale のクライアント・プロセスを構成するには、このトピックで後述する説明を

参照してください。IBM SDK を使用する場合も、他の開発キットを使用する場合も、IBM ORB を使用できます。

IBM SDK バージョン 5 (以降) を使用する方が、IBM SDK バージョン 1.4.2 を使用する場合より必要とする構成作業が少なくて済みます。

ケース 3: カスタム ORB (サード・パーティー・ベンダー)

WebSphere eXtreme Scale のクライアント・プロセスにサード・パーティーの ORB を使用するケースは、最もテスト回数の少ないオプションです。独立系ソフトウェア・ベンダーの ORB を使用して発生した問題は、IBM ORB で再現可能で、JRE との互換性がないとサポートに問い合わせることはできません。

Sun Microsystems Java Development Kit (JDK) で提供される ORB は、サポート対象外です。

手順

- デフォルト ORB の 1 つを使用するようクライアント・プロセスを構成します (**ケース 1**)。

  `-jvmArgs –Djava.endorsed.dirs=`*`default_ORB_directory`*

- IBM SDK バージョン 5 を使用するようクライアント・プロセスまたはサーバー・プロセスを構成します (**ケース 2**)。

  1. ORB JAR ファイルを空のディレクトリーにコピーします。これ以降このディレクトリーは *custom_ORB_directory* と呼ばれます。
     - `ibmorb.jar`
     - `ibmorbapi.jar`

     ヒント: サード・パーティー・ベンダーのカスタム ORB を使用する場合 (**ケース 3**)、これらの追加 JAR ファイルが必要となるかもしれません。

     - `ibmext.jar`
     - `ibmcfw.jar` (NIO ORB を使用する場合)
  2. *custom_ORB_directory* を、Java コマンドを開始するスクリプト内の承認済みディレクトリーとして指定します。

     ヒント: ご使用の Java コマンドが既に承認済みディレクトリーを参照している場合、もう 1 つのオプションは、*custom_ORB_directory* を既存の承認済みディレクトリー内に配置することです。こうすることで、スクリプトを更新する必要はなくなります。いずれにしてもあえてこのスクリプトを更新する場合は、既存の引数を完全に置き換えるのではなく、必ず既存の `–Djava.endorsed.dirs=` 引数の前に *custom_ORB_directory* を付加するようにしてください。

     - スタンドアロンの eXtreme Scale 環境用にスクリプトを更新します。

       `setupCmdLine.bat|sh`ファイルの *OBJECTGRID_ENDORSED_DIRS* 変数に対するパスを、*custom_ORB_directory* を参照するように編集します。変更内容を保存します。

     - eXtreme Scale が WebSphere Application Server 環境に組み込まれている場合、スクリプトを更新します。

startOgServer スクリプトに以下のシステム・プロパティーとパラメーターを追加します。

```
-jvmArgs –Djava.endorsed.dirs=custom_ORB_directory
```

– クライアント・アプリケーション・プロセスまたはサーバー・プロセスの開始に使用するカスタム・スクリプトを更新します。

```
-Djava.endorsed.dirs=custom_ORB_directory
```

- IBM SDK バージョン 1.4.2 を使用するようクライアント・プロセスまたはサーバー・プロセスを構成します **(ケース 2)**。 ご使用の環境にバージョン 1.4.2 の SDK が含まれている場合は、IBM ORB を指定された SDK に組み込みます。
  1. IBM SDK バージョン 1.4.2 から ORB をダウンロードし、解凍します。

     ご使用のプラットフォームで使用可能な IBM SDK がない場合は、 IBM Developer Kit for Linux、Java Technology Edition をダウンロードして解凍します。IBM developer kit を参照してください。
  2. ORB JAR ファイルをターゲット SDK にコピーします。 java/jre/lib/ibmorb.jar ファイルと java/jre/lib/ibmorbapi.jar ファイルをターゲット SDK 上の java/jre/lib/ext ディレクトリーにコピーします。
  3. ORB プロパティーを更新します。 orb.properties ファイルを作成または編集します。このファイルは、SDK の java/jre/lib ディレクトリーに入ります。以下のプロパティーを追加するか、または以下のプロパティーがファイルに存在することを確認します。

     ```
     org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
     org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton
     ```

     プロパティーおよび設定の説明については、206 ページの『ORB プロパティー・ファイル』を参照してください。
  4. XML パーサーが使用可能であることを確認してください。
     – The Apache Xerces Project - Downloads から Xerces2 Java 2.9 をダウンロードしてください。
     – xercesImpl.jar ファイルと xml-apis.jar ファイルを見つけます。
     – それらのファイルを lib/ext ディレクトリーにコピーしてください。

クライアントの構成

スタンドアロン環境で実行するための WebSphere® eXtreme Scale を構成します。あるいは、WebSphere Application Server または WebSphere Application Server Network Deployment を使用した環境で実行するための eXtreme Scale を構成します。eXtreme Scale のデプロイメントでサーバー・グリッド・サイドの構成変更を反映させるには、動的な適用ではなく、プロセスを再始動して変更を有効にする必要があります。一方、クライアント・サイドでは、既存のクライアント・インスタンスの構成設定は変更できませんが、 XML ファイルを使用するか、またはプログラムによって、新しいクライアントを必要な設定で作成できます。クライアントの作成時には、現行のサーバー構成からのデフォルト設定をオーバーライド可能です。

次の方法で、eXtreme Scale クライアントを構成することができます。いずれの方法も、クライアント・オーバーライド XML ファイルを使用して、もしくはプログラムで実行することができます。

- XML 構成
- プログラマチック構成
- Spring Framework 構成
- ニア・キャッシュの使用不可化

以下のプラグインをクライアントにオーバーライドすることができます。

- **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
  - numberOfBuckets 属性
  - ttlEvictorType 属性
  - timeToLive 属性

クライアント・プロパティー・ファイル

eXtreme Scale クライアント・プロセスの要件に基づいて、プロパティー・ファイルを作成できます。

サンプル・クライアント・プロパティー・ファイル

extremescale_root¥properties ディレクトリーにある sampleClient.properties ファイルを使用して、プロパティー・ファイルを作成することができます。

クライアント・プロパティー・ファイルの指定

クライアント・プロパティー・ファイルは、以下のいずれかの方法で指定できます。このリスト内の項目のいずれかを使用して後で設定を指定すると、前の設定はオーバーライドされます。例えば、クライアント・プロパティー・ファイルのシステム・プロパティー値を指定すると、そのファイルのプロパティーにより、クラスパスにある objectGridClient.properties ファイルの値が指定変更されます。

1. クラスパス内の任意の場所にあるわかりやすい名前のファイルで指定。このファイルをシステム現行ディレクトリーに置くことはサポートされていません。

   ```
   objectGridClient.properties
   ```

2. スタンドアロンまたは WebSphere Application Server 構成のいずれかでシステム・プロパティーとして指定。この値に、システム現行ディレクトリー内のファイルを指定することはできますが、クラスパス内のファイルは指定できません。

   ```
   -Dobjectgrid.client.props=file_name
   ```

3. ClientClusterContext.getClientProperties メソッドを使用したプログラマチックな指定変更。オブジェクトのデータに、プロパティー・ファイルからのデータが取り込まれます。セキュリティー・プロパティーは、このメソッドで構成できません。

クライアント・プロパティー

7.1+ **listenerHost**
オブジェクト・リクエスト・ブローカー (ORB) のバインド先のホスト名を指定します。

ご使用の構成に複数のネットワーク・カードが含まれている場合、リスナー・ホストとリスナー・ポートを設定し、バインドする IP アドレスを JVM 内のオブジェクト・リクエスト・ブローカーに通知します。クライアントについては、クライアント・プロパティー・ファイルを使用してください。使用する IP アドレスの指定を怠ると、コネクション・タイムアウトや異常な API 障害、クライアントがハングしたような状態になる、などの症状が生じます。

7.1+ **listenerPort**
オブジェクト・リクエスト・ブローカー (ORB) のバインド先のポート番号を指定します。

preferLocalProcess
ルーティングについてローカル・プロセスが優先されているかどうかを指定します。 true に設定すると、要求は、それが適切な場合は、クライアントと同じプロセスに配置されている断片に経路指定されます。

デフォルト: `true`

preferLocalHost
ルーティングについてローカル・ホストが優先されているかどうかを指定します。 true に設定すると、要求は、それが適切な場合はクライアントと同じホストに配置されている断片に経路指定されます。

デフォルト: `true`

preferZones
優先ルーティング・ゾーンのリストを指定します。各ゾーンは、`preferZones=ZoneA,ZoneB,ZoneC` の形式でコンマで区切って指定します。

デフォルト: 値なし

requestRetryTimeout
要求を再試行する時間をミリ秒で指定します。以下の有効値のいずれかを使用します。

- 値が `0` の場合、要求は、フェイル・ファーストになり、内部の再試行ロジックは飛ばされます。
- 値が `-1` の場合、要求再試行のタイムアウトは設定されません。すなわち、要求の所要時間は、トランザクション・タイムアウトによって制御されます。 (デフォルト)
- `0` より大きい値は、要求再試行のタイムアウト値をミリ秒で示しています。再試行しても正常に処理できない例外 (DuplicateException 例外など) は、即時に戻されます。トランザクション・タイムアウトは、最大待ち時間として引き続き使用されます。

セキュリティー・クライアント・プロパティー

一般セキュリティー・プロパティー

securityEnabled

WebSphere eXtreme Scale クライアント・セキュリティーを使用可能にします。このセキュリティー使用可能化設定は、WebSphere eXtreme Scale サーバー・プロパティー・ファイルの securityEnabled 設定と一致している必要があります。この設定が一致しない場合、例外が発生します。

デフォルト: `false`

クレデンシャル認証の構成プロパティー

credentialAuthentication

クライアントのクレデンシャル認証のサポートを指定します。以下の有効値のいずれかを使用します。

- `Never`: クライアントは、クレデンシャル認証をサポートしません。
- `Supported`: サーバーもクレデンシャル認証をサポートする場合に、クライアントはクレデンシャル認証をサポートします。 (デフォルト)
- `Required`: クライアントはクレデンシャル認証を必要とします。

authenticationRetryCount

クレデンシャルの有効期限が切れている場合に認証を再試行できる回数を指定します。この値が 0 に設定されていると、認証の再試行はできません。

デフォルト: `3`

credentialGeneratorClass

com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator インターフェースを実装するクラスの名前を指定します。このクラスを使用して、クライアントのクレデンシャルが取得されます。

デフォルト: 値なし

credentialGeneratorProps

CredentialGenerator 実装クラスのプロパティーを指定します。このプロパティーが、setProperties(String) メソッドを使用してオブジェクトに設定されます。 credentialGeneratorprops 値は、credentialGeneratorClass プロパティーの値が非ヌルの場合にのみ使用されます。

Transport Layer Security の構成プロパティー

transportType

クライアントのトランスポート・タイプを指定します。可能な値は以下のとおりです。

- `TCP/IP`: クライアントが、TCP/IP 接続のみをサポートすることを示します。
- `SSL-Supported`: クライアントが TCP/IP 接続と Secure Sockets Layer (SSL) 接続の両方をサポートすることを示します。(デフォルト)
- `SSL-Required`: クライアントが SSL 接続を必要とすることを示します。

SSL 構成プロパティー

別名 (alias)
鍵ストア内の別名を指定します。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。

デフォルト: 値なし

contextProvider
トラスト・サービスのコンテキスト・プロバイダーの名前を指定します。有効でない値を指定すると、コンテキスト・プロバイダー・タイプが正しくないことを示すセキュリティー例外が発生します。

有効値: `IBMJSSE2`、`IBMJSSE`、`IBMJSSEFIPS` など。

プロトコル
クライアントに使用するセキュリティー・プロトコルのタイプを指定します。このプロトコルの値は、使用する Java Secure Socket Extension (JSSE) プロバイダーに基づいて設定します。有効でない値を指定すると、プロトコル値が正しくないことを示すセキュリティー例外が発生します。

有効値: `SSL`、`SSLv2`、`SSLv3`、`TLS`、`TLSv1` など。

keyStoreType
鍵ストアのタイプを示します。有効でない値を指定すると、ランタイム・セキュリティー例外が発生します。

有効値: `JKS`、`JCEK`、`PKCS12` など。

trustStoreType
トラストストアのタイプを指定します。有効でない値を指定すると、ランタイム・セキュリティー例外が発生します。

有効値: `JKS`、`JCEK`、`PKCS12` など。

keyStore
鍵ストア・ファイルへの完全修飾パスを指定します。

例:

```
etc/test/security/client.private
```

trustStore
トラストストア・ファイルへの完全修飾パスを指定します。

例:

```
etc/test/security/server.public
```

keyStorePassword
鍵ストアのストリング・パスワードを指定します。この値はエンコードすることも、実際の値を使用することもできます。

trustStorePassword
トラストストアのストリング・パスワードを指定します。この値はエンコードすることも、実際の値を使用することもできます。

WebSphere eXtreme Scale のクライアントの構成

設定値をオーバーライドしなければならないなどの、ユーザーの要件に基づいて eXtreme Scale クライアントを構成することができます。

XML を使用したクライアントの構成

ObjectGrid XML ファイルを使用して、クライアント・サイドで設定を変更できます。eXtreme Scale クライアントの設定を変更するには、eXtreme Scale サーバーに使用されたファイルに構造が類似している ObjectGrid XML ファイルを作成する必要があります。

以下の XML ファイルがデプロイメント・ポリシー XML ファイルと対になっており、これらのファイルを使用して eXtreme Scale サーバーが始動されたものと想定します。

companyGridServerSide.xml

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

    <objectGrids>
        <objectGrid name="CompanyGrid">
            <bean id="TransactionCallback"
                className="com.company.MyTxCallback" />
            <bean id="ObjectGridEventListener"
                className="com.company.MyOgEventListener" />
            <backingMap name="Customer"
                pluginCollectionRef="customerPlugins" />
            <backingMap name="Item" />
            <backingMap name="OrderLine" numberOfBuckets="1049"
                timeToLive="1600" ttlEvictorType="LAST_ACCESS_TIME" />
            <backingMap name="Order" lockStrategy="PESSIMISTIC"
                pluginCollectionRef="orderPlugins" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="customerPlugins">
            <bean id="Evictor"
                className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
            <bean id="MapEventListener"
                className="com.company.MyMapEventListener" />
        </backingMapPluginCollection>
        <backingMapPluginCollection id="orderPlugins">
            <bean id="MapIndexPlugin"
                className="com.company.MyMapIndexPlugin" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

eXtreme Scale サーバーでは、CompanyGrid という名前の ObjectGrid インスタンスは companyGridServerSide.xml ファイルによる定義に従って動作します。デフォルトでは CompanyGrid クライアントの設定は、サーバーで実行している CompanyGrid インスタンスの設定と同じです。ただし、設定のいくつかはクライアント上で次のとおりオーバーライドできます。

1. クライアント固有の ObjectGrid インスタンスを作成します。
2. サーバーの開始に使用された ObjectGrid XML ファイルをコピーします。
3. その新規ファイルを編集してクライアント・サイドでカスタマイズします。
   - クライアントの属性を設定または更新するには、新しい値を指定するか、あるいは既存の値を変更します。
   - クライアントからプラグインを除去するには、className 属性の値として空ストリングを使用します。
   - 既存のプラグインを変更するには、className 属性に新しい値を指定します。

- クライアント・オーバーライドでサポートされるプラグイン (TRANSACTION_CALLBACK、 OBJECTGRID_EVENT_LISTENER、 EVICTOR、MAP_EVENT_LISTENER) を追加することもできます。

4. 新規に作成されたクライアント・オーバーライド XML ファイルを使用して、クライアントを作成します。

以下の ObjectGrid XML ファイルを使用すると、CompanyGrid クライアントの属性およびプラグインのいくつかを指定できます。

companyGridClientSide.xml

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

    <objectGrids>
        <objectGrid name="CompanyGrid">
            <bean id="TransactionCallback"
                className="com.company.MyClientTxCallback" />
            <bean id="ObjectGridEventListener" className="" />
            <backingMap name="Customer" numberOfBuckets="1429"
                pluginCollectionRef="customerPlugins" />
            <backingMap name="Item" />
            <backingMap name="OrderLine" numberOfBuckets="701"
                timeToLive="800" ttlEvictorType="LAST_ACCESS_TIME" />
            <backingMap name="Order" lockStrategy="PESSIMISTIC"
                pluginCollectionRef="orderPlugins" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="customerPlugins">
            <bean id="Evictor"
                className="com.ibm.websphere.objectGrid.plugins.builtins.LRUEvictor" />
            <bean id="MapEventListener" className="" />
        </backingMapPluginCollection>
        <backingMapPluginCollection id="orderPlugins">
            <bean id="MapIndexPlugin"
                className="com.company.MyMapIndexPlugin" />
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

- クライアントの TransactionCallback は、サーバー・サイドで設定されている com.company.MyTxCallback ではなく、com.company.MyClientTxCallback になります。
- className 値が空ストリングであるため、クライアントには ObjectGridEventListener プラグインがありません。
- クライアントは Customer backingMap に対して numberOfBuckets を 1429 に設定し、その Evictor プラグインを保持して、MapEventListener プラグインを除去します。
- OrderLine backingMap の numberOfBuckets 属性および timeToLive 属性は変更されます。
- 異なる lockStrategy 属性が指定されていても、lockStrategy 属性はクライアント・オーバーライドでサポートされていないため、影響はありません。

companyGridClientSide.xml ファイルを使用して CompanyGrid クライアントを作成するには、ObjectGrid XML ファイルを URL として、ObjectGridManager の接続メソッドの 1 つに渡します。

```
Creating the client for XML
ObjectGridManager ogManager =
 ObjectGridManagerFactory.ObjectGridManager();
```

```
ClientClusterContext clientClusterContext =
 ogManager.connect("MyServer1.company.com:2809", null, new URL(
                "file:xml/companyGridClientSide.xml"));
```

クライアントのプログラマチック構成

クライアント・サイドの ObjectGrid 設定をプログラマチックにオーバーライドすることもできます。サーバー・サイド ObjectGrid インスタンスと同様の構造を持つ ObjectGridConfiguration オブジェクトを作成します。以下のコードで、XML ファイルを使用する上記セクションのクライアント・オーバーライドと機能的に同等な、クライアント・サイド ObjectGrid インスタンスが作成されます。

```
client-side override programmatically
ObjectGridConfiguration companyGridConfig = ObjectGridConfigFactory
    .createObjectGridConfiguration("CompanyGrid");
Plugin txCallbackPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.TRANSACTION_CALLBACK, "com.company.MyClientTxCallback");
companyGridConfig.addPlugin(txCallbackPlugin);

Plugin ogEventListenerPlugin = ObjectGridConfigFactory.createPlugin(
    PluginType.OBJECTGRID_EVENT_LISTENER, "");
companyGridConfig.addPlugin(ogEventListenerPlugin);

BackingMapConfiguration customerMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("Customer");
customerMapConfig.setNumberOfBuckets(1429);
Plugin evictorPlugin = ObjectGridConfigFactory.createPlugin(PluginType.EVICTOR,
    "com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor");
customerMapConfig.addPlugin(evictorPlugin);

companyGridConfig.addBackingMapConfiguration(customerMapConfig);

BackingMapConfiguration orderLineMapConfig = ObjectGridConfigFactory
    .createBackingMapConfiguration("OrderLine");
orderLineMapConfig.setNumberOfBuckets(701);
orderLineMapConfig.setTimeToLive(800);
orderLineMapConfig.setTtlEvictorType(TTLType.LAST_ACCESS_TIME);

companyGridConfig.addBackingMapConfiguration(orderLineMapConfig);

List ogConfigs = new ArrayList();
ogConfigs.add(companyGridConfig);

Map overrideMap = new HashMap();
overrideMap.put(CatalogServerProperties.DEFAULT_DOMAIN, ogConfigs);

ogManager.setOverrideObjectGridConfigurations(overrideMap);
ClientClusterContext client = ogManager.connect(catalogServerAddresses, null, null);
ObjectGrid companyGrid = ogManager.getObjectGrid(client, objectGridName);
```

ObjectGridManager の ogManager インスタンスは、overrideMap マップに組み込まれている ObjectGridConfiguration オブジェクトおよび BackingMapConfiguration オブジェクトのオーバーライドのみをチェックします。例えば、上記のコードは、OrderLine マップ上のバケットの数をオーバーライドします。ただし、そのマップに対する構成が組み込まれていないため、クライアント・サイドの Order マップは変更されないままです。

Spring Framework でのクライアントの構成

クライアント・サイドの ObjectGrid 設定は、Spring Framework を使用してオーバーライドすることもできます。以下の例の XML ファイルは、ObjectGridConfiguration エレメントをビルドし、それをクライアント・サイド設定をオーバーライドするために使用する方法を示しています。 この例では、プログラマチック構成で示したのと同じ API が呼び出されます。またこの例は、ObjectGrid XML 構成の例と機能的に同等です。

client configuration with Spring

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
  "http://www.springframework.org/dtd/spring-beans.dtd">
<beans>
  <bean id="companyGrid" factory-bean="manager" factory-method="getObjectGrid"
    singleton="true">
    <constructor-arg type="com.ibm.websphere.objectgrid.ClientClusterContext">
      <ref bean="client" />
    </constructor-arg>
    <constructor-arg type="java.lang.String" value="CompanyGrid" />
</bean>

  <bean id="manager" class="com.ibm.websphere.objectgrid.ObjectGridManagerFactory"
    factory-method="getObjectGridManager" singleton="true">
    <property name="overrideObjectGridConfigurations">
      <map>
        <entry key="DefaultDomain">
          <list>
            <ref bean="ogConfig" />
          </list>
        </entry>
      </map>
    </property>
  </bean>

  <bean id="ogConfig"
    class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
    factory-method="createObjectGridConfiguration">
    <constructor-arg type="java.lang.String">
      <value>CompanyGrid</value>
    </constructor-arg>
    <property name="plugins">
      <list>
   <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
          factory-method="createPlugin">
          <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
            value="TRANSACTION_CALLBACK" />
          <constructor-arg type="java.lang.String"
            value="com.company.MyClientTxCallback" />
        </bean>
        <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
          factory-method="createPlugin">
          <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
            value="OBJECTGRID_EVENT_LISTENER" />
          <constructor-arg type="java.lang.String" value="" />
        </bean>
      </list>
 </property>
      <property name="backingMapConfigurations">
        <list>
     <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
            factory-method="createBackingMapConfiguration">
            <constructor-arg type="java.lang.String" value="Customer" />
              <property name="plugins">
      <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
                  factory-method="createPlugin">
                  <constructor-arg type="com.ibm.websphere.objectgrid.config.PluginType"
     value="EVICTOR" />
   <constructor-arg type="java.lang.String"
                    value="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" />
                </bean>
              </property>
              <property name="numberOfBuckets" value="1429" />
            </bean>
            <bean class="com.ibm.websphere.objectgrid.config.ObjectGridConfigFactory"
              factory-method="createBackingMapConfiguration">
                <constructor-arg type="java.lang.String" value="OrderLine" />
                  <property name="numberOfBuckets" value="701" />
   <property name="timeToLive" value="800" />
   <property name="ttlEvictorType">
                    <value type="com.ibm.websphere.objectgrid.
          TTLType">LAST_ACCESS_TIME</value>
   </property>
            </bean>
          </list>
        </property>
      </bean>

      <bean id="client" factory-bean="manager" factory-method="connect"
```

```
        singleton="true">
        <constructor-arg type="java.lang.String">
     <value>localhost:2809</value>
        </constructor-arg>
   <constructor-arg
          type="com.ibm.websphere.objectgrid.security.
      config.ClientSecurityConfiguration">
          <null />
        </constructor-arg>
   <constructor-arg type="java.net.URL">
     <null />
        </constructor-arg>
      </bean>
</beans>
```

XML ファイルの作成後、そのファイルをロードし、以下のコード・スニペットで ObjectGrid をビルドします。

```
BeanFactory beanFactory = new XmlBeanFactory(new

UrlResource("file:test/companyGridSpring.xml"));

ObjectGrid companyGrid = (ObjectGrid) beanFactory.getBean("companyGrid");
```

XML 記述子ファイルの作成について詳しくは、Spring Framework の統合の概要を参照してください。

クライアントのニア・キャッシュの使用不可化

ニア・キャッシュは、ロックがオプティミスティックまたはロックなしで構成されている場合、デフォルトで使用可能になっています。 クライアントは、ロック設定がペシミスティックで構成されている場合はニア・キャッシュを保持しません。ニア・キャッシュを使用不可にするには、クライアント・オーバーライド ObjectGrid 記述子ファイルで numberOfBuckets 属性を 0 に設定します。

クライアント無効化メカニズムの使用可能化

分散 WebSphere eXtreme Scale 環境では、optimistic ロック・ストラテジーが使用されているか、ロックが無効にされている場合、クライアント・サイドにはデフォルトでニア・キャッシュがあります。ニア・キャッシュにはそれ独自のローカル・キャッシュ・データがあります。eXtreme Scale クライアントが更新をコミットすると、この更新はクライアントのニア・キャッシュとサーバーに送られます。 しかし、他の eXtreme Scale クライアントはこの更新情報を受け取らないので、それらのデータは古くなっていることがあります。

ニア・キャッシュ

アプリケーションは、eXtreme Scale クライアントのこの失効データの問題を認識しておく必要があります。Java Message Service (JMS) ベースの組み込み ObjectGridEventListener クラス com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener を使用して、分散 eXtreme Scale 環境 (eXtreme Scale グリッドと呼ばれます) 内で、クライアント無効化メカニズムを使用可能にすることができます。

クライアント無効化メカニズムは、分散 eXtreme Scale 環境におけるクライアントのニア・キャッシュ内の失効データの問題に対する解決方法です。このメカニズムにより、クライアントのニア・キャッシュはサーバーまたは他のクライアントと同

期します。ただし、この JMS ベースのクライアント無効化メカニズムを使用しても、クライアントのニア・キャッシュが即時に更新されるわけではありません。eXtreme Scale ランタイムが更新を公開するときに遅延が発生します。

分散 eXtreme Scale 環境におけるクライアント無効化メカニズムには、次の 2 つのモデルがあります。

- クライアント/サーバー・モデル: このモデルでは、すべてのサーバー・プロセスは、指定された JMS の宛先にすべてのトランザクション変更を公開する publisher ロールにあります。すべてのクライアント・プロセスは受信側ロールにあり、指定された JMS の宛先からすべてのトランザクション変更を受け取ります。
- 二重ロール・モデルとしてのクライアント: このモデルでは、すべてのサーバー・プロセスは JMS の宛先とは無関係です。すべてのクライアント・プロセスは JMS publisher ロールであり、かつ receiver ロールです。クライアントで発生するトランザクション変更は JMS の宛先に公開され、すべてのクライアントがこれらのトランザクション変更を受け取ります。

詳しくは、 148 ページの『JMS イベント・リスナー』 を参照してください。

クライアント/サーバー・モデル

クライアント/サーバー・モデルでは、サーバーは JMS publisher ロールにあり、クライアントは JMS receiver ロールにあります。

client-server model XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<?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="AgentObjectGrid">
      <bean id="ObjectGridEventListener"
        className="com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener">
        <property name="invalidationModel" type="java.lang.String" value="CLIENT_SERVER_MODEL" description="" />
        <property name="invalidationStrategy" type="java.lang.String" value="PUSH" description="" />
        <property name="mapsToPublish" type="java.lang.String" value="agent;profile;pessimisticMap" description="" />
        <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String" value="defaultTCF" description="" />
        <property name="jms_topicJndiName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_topicName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_userid" type="java.lang.String" value="" description="" />
        <property name="jms_password" type="java.lang.String" value="" description="" />
        <property name="jndi_properties" type="java.lang.String"
          value="java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory;
     java.naming.provider.url=
    tcp://localhost:61616;connectionFactoryNames=defaultTCF;topic.defaultTopic=defaultTopic"
          description="jndi properties" />
      </bean>

      <backingMap name="agent" readOnly="false" pluginCollectionRef="agent" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="28800" />
      <backingMap name="profile" readOnly="false" pluginCollectionRef="profile" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="pessimisticMap" readOnly="false" pluginCollectionRef="pessimisticMap" preloadMode="false"
        lockStrategy="PESSIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap1" readOnly="false" pluginCollectionRef="excludedMap1" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap2" readOnly="false" pluginCollectionRef="excludedMap2" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="agent">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.AgentObjectTransformer" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="profile">
```

```
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.ProfileObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
        <property name="maxSize" type="int" value="2000" 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="50" description="set number of LRU queues" />
      </bean>
    </backingMapPluginCollection>

    <backingMapPluginCollection id="pessimisticMap" />
    <backingMapPluginCollection id="excludedMap1" />
    <backingMapPluginCollection id="excludedMap2" />
  </backingMapPluginCollections>

</objectGridConfig>
```

二重ロール・モデルとしてのクライアント

二重ロール・モデルとしてのクライアントでは、各クライアントは JMS publisher ロールと receiver ロールの両方を持っています。クライアントは、コミットされたすべてのトランザクション変更を、指定された JMS 宛先に公開し、コミットされたすべてのトランザクション変更を他のクライアントから受け取ります。このモデルでは、サーバーは JMS とは無関係です。

dual-roles model XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="AgentObjectGrid">
      <bean id="ObjectGridEventListener"
        className="com.ibm.websphere.objectgrid.plugins.builtins.JMSObjectGridEventListener">
        <property name="invalidationModel" type="java.lang.String" value="CLIENT_AS_DUAL_ROLES_MODEL" description="" />
        <property name="invalidationStrategy" type="java.lang.String" value="PUSH" description="" />
        <property name="mapsToPublish" type="java.lang.String" value="agent;profile;pessimisticMap" description="" />
        <property name="jms_topicConnectionFactoryJndiName" type="java.lang.String" value="defaultTCF" description="" />
        <property name="jms_topicJndiName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_topicName" type="java.lang.String" value="defaultTopic" description="" />
        <property name="jms_userid" type="java.lang.String" value="" description="" />
        <property name="jms_password" type="java.lang.String" value="" description="" />
        <property name="jndi_properties" type="java.lang.String"
          value="java.naming.factory.initial=org.apache.activemq.jndi.ActiveMQInitialContextFactory;java.naming.provider.url=
     tcp://localhost:61616;connectionFactoryNames=defaultTCF;topic.defaultTopic=defaultTopic"
          description="jndi properties" />
      </bean>

      <backingMap name="agent" readOnly="false" pluginCollectionRef="agent" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="28800" />
      <backingMap name="profile" readOnly="false" pluginCollectionRef="profile" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="pessimisticMap" readOnly="false" pluginCollectionRef="pessimisticMap" preloadMode="false"
        lockStrategy="PESSIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap1" readOnly="false" pluginCollectionRef="excludedMap1" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
      <backingMap name="excludedMap2" readOnly="false" pluginCollectionRef="excludedMap2" preloadMode="false"
        lockStrategy="OPTIMISTIC" copyMode="COPY_ON_READ_AND_COMMIT" ttlEvictorType="LAST_ACCESS_TIME"
        timeToLive="2700" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="agent">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.AgentObjectTransformer" />
    </backingMapPluginCollection>
    <backingMapPluginCollection id="profile">
      <bean id="ObjectTransformer" className="com.ibm.ws.objectgrid.test.scenario.ProfileObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor">
        <property name="maxSize" type="int" value="2000" 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="50" description="set number of LRU queues" />
      </bean>
    </backingMapPluginCollection>

    <backingMapPluginCollection id="pessimisticMap" />
    <backingMapPluginCollection id="excludedMap1" />
    <backingMapPluginCollection id="excludedMap2" />
  </backingMapPluginCollections>

</objectGridConfig>
```

要求再試行タイムアウトの構成

信頼できるマップがあれば、WebSphere eXtreme Scale にトランザクション要求の再試行タイムアウトを指定できます。

信頼できるマップを構成するには、2 つの方法があります。ゼロより大きい値に設定された場合、タイムアウト条件が満たされるか、永続的な障害 (DuplicateKeyException 例外など) が起こるまで、要求は再試行されます。値ゼロはフェイル・ファースト・モード設定を表し、eXtreme Scale は再試行を行いません。

クライアント・プロパティー・ファイルまたはセッションで、タイムアウト値をミリ秒単位で指定します。 セッションは常にクライアント・プロパティー設定に優先します。ランタイム中、トランザクション・タイムアウトが再試行タイムアウトと一緒に使用されます。これは、再試行タイムアウトがトランザクション・タイムアウトを超えないようにするためです。

自動コミット・トランザクション、および非自動コミット・トランザクション (明示的な begin および commit メソッドを使用するトランザクション) がどのように完了するのかには多様なバリエーションがあるため、再試行が有効な例外もさまざまです。

セッション内で呼び出されるトランザクションの場合、CORBA SystemException および eXtreme Scale TargetNotAvailable 例外に対して再試行は有効です。

自動コミット・トランザクションでは、再試行は CORBA SystemException、eXtreme Scale アベイラビリティー例外 (ReplicationVotedToRollbackTransactionException、TargetNotAvailable、AvailabilityException など) の場合に有効です。

詳しくは、「*プログラミング・ガイド*」に記載されているセッションを使用したグリッド内データへのアクセスに関するトピックを参照してください。

アプリケーション障害やその他の永続障害はすぐに再発するので、クライアントはトランザクションを再試行しません。 このような永続障害には DuplicateKeyException や KeyNotFoundException 例外があります。

フェイル・ファースト設定は、いかなる例外についても、再試行なしですべての例外を戻します。

以下のリストは例外を詳細に示しています。

クライアントが再試行を実施する例外

- ReplicationVotedToRollbackTransactionException (自動コミットの場合のみ)
- TargetNotAvailable
- org.omg.CORBA.SystemException
- AvailabilityException (自動コミットの場合のみ)
- LockTimeoutException (自動コミットの場合のみ)
- UnavailableServiceException (自動コミットの場合のみ)

他の例外

- DuplicateKeyException
- KeyNotFoundException
- LoaderException
- TransactionAffinityException
- LockDeadlockException
- OptimisticCollisionException

クライアント・プロパティー・ファイル内での requestRetryTimeout プロパティーの設定

クライアントの requestRetryTimeout 値を設定するには、 216 ページの『クライアント・プロパティー・ファイル』内の requestRetryTimeout プロパティーを追加または変更します。クライアント・プロパティーは、デフォルトでは objectGridClient.properties ファイルです。requestRetryTimeout プロパティーはミリ秒単位で指定されます。 ゼロより大きい値に設定すると、再試行可能な例外が起こったときに要求は再試行されます。値を 0 に設定すると、例外が起こったときに再試行なしで失敗します。デフォルトの動作を使用するには、このプロパティーを除去するか、値を -1 に設定します。

objectGridClient.properties

```
# eXtreme Scale client config
preferLocalProcess = false
preferLocalhost = false
requestRetryTimeout = 30000
```

requestRetryTimeout 値はミリ秒で指定されます。上の例で、設定した値が ObjectGrid インスタンスで使用されると、requestRetryTimeout 値は 30 秒です。

ObjectGrid 接続を取得することによる、プログラムでのクライアント・プロパティーの設定

クライアント・プロパティーをプログラマチックに設定するには、まず最初にアプリケーションの適切な <ロケーション>にクライアント・プロパティー・ファイルを作成します。 以下の例では、クライアント・プロパティー・ファイルは、前のセクションの objectGridClient.properties スニペットを参照します。ObjectGridManager と接続を確立した後、前述のようにクライアント・プロパティーを設定します。 その後、ObjectGrid インスタンスを取得すると、ファイルで定義したクライアント・プロパティーがインスタンスに入っています。 クライアント・プロパティー・ファイルを変更する場合は、そのたびに新しい ObjectGrid インスタンスを明示的に取得してください。

```
ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
String objectGridName = "testObjectGrid";
URL clientXML = null;
ClientClusterContext ccc = manager.connect("localhost:2809", null, clientXML);
File file = new File("<location>/objectGridClient.properties");
URL url = file.toURI().toURL();
ccc.setClientProperties(objectGridName, url);
ObjectGrid objectGrid = ogManager.getObjectGrid(ccc, objectGridName);
```

自動コミットでのセッションのオーバーライド例

要求再試行タイムアウトをセッションに設定するか、requestRetryTimeout クライアント・プロパティーをオーバーライドするには、Session インターフェースの setRequestRetryTimeout(long) メソッドを呼び出します。

```
Session sessionA = objectGrid.getSession();
sessionA.setRequestRetryTimeout(30000);
ObjectMap mapA = sessionA.getMap("payroll");
String key = "key:" + j;
mapA.insert(key, "valueA");
```

クライアント・プロパティー・ファイルに設定されている値に関係なく、このセッションでは現在 30000 ミリ秒または 30 秒という requestRetryTimeout 値が使用されています。セッション・インターフェースについて詳しくは、セッションを使用したグリッド内データへのアクセスを参照してください。

エンティティーの構成

ObjectGrid は、任意の数の論理エンティティー・スキーマを持つことができます。エンティティーは、アノテーション付き Java™ クラス、XML、または XML と Java クラスの組み合わせを使用して定義されます。定義されたエンティティーは、eXtreme Scale サーバーに登録され、BackingMap、索引、およびその他のプラグインにバインドされます。

始める前に

エンティティー・スキーマとは、1 組のエンティティーとそれらエンティティーの間のリレーションシップのことです。スキーマ定義とエンティティーの構成について詳しくは、エンティティー・スキーマの定義 を参照してください。

リレーションシップ管理

Java などのオブジェクト指向言語、およびリレーショナル・データベースは、リレーションシップまたは関連をサポートします。リレーションシップは、オブジェクト参照または外部キーの使用を通してストレージ量を削減します。

グリッド内でリレーションシップを使用するときには、データはコンストレインド・ツリーに編成されている必要があります。そのツリーには 1 つのルート・タイプがなければならず、すべての子は 1 つのルートのみに関連付けられていなければなりません。例: 部門には多数の従業員が属し、1 人の従業員が多くのプロジェクトを持つことができます。しかし、1 つのプロジェクトが異なる部門に属している多くの従業員を持つことはできません。ルートが定義されると、ルート・オブジェクトとその子孫へのすべてのアクセスはルートを通して管理されるようになります。WebSphere eXtreme Scale は、ルート・オブジェクトのキーのハッシュ・コードを使用して区画を選択します。

例: partition = (hashCode MOD numPartitions)

1 つのリレーションシップに関係するすべてのデータが単一のオブジェクト・インスタンスに結びついている場合、ツリー全体を 1 つの区画内に配列し、1 つのトランザクションを使用して非常に効率的にアクセスすることができます。データが複

数のリレーションシップにまたがっている場合は、複数の区画についての処理が必要になるため、追加のリモート呼び出しが必要になり、結果的にパフォーマンス上のボトルネックとなることがあります。

参照データ

一部のリレーションシップは、ルックアップまたは参照データを含んでいます。例: CountryName。これは、データがどの区画にも存在しているという特殊なケースです。このような場合は、データはどのルート・キーでもアクセスでき、同じ結果が戻ります。 このような参照データは、すべての区画での更新が必要でコストがかかるため、データが相当に静的なケースに限って使用するべきです。参照データを最新に保つ手法として、DataGrid API がよく使われます。

正規化のコストと利点

リレーションシップを使用してデータを正規化することは、データの重複が減るため、グリッドによるメモリー使用量を削減するのに寄与します。ただし、一般的には、追加される関係データが多いほど、スケールアウトは少なくなります。データがグループ化されている場合、リレーションシップを維持し、管理できる程度のサイズに保つために、より多くのコストがかかるようになります。グリッドは、ツリーのルートのキーに基づいてデータを区画に分けるので、ツリーのサイズは考慮には入れられません。したがって、1 つのツリー・インスタンスに対して多数のリレーションシップがある場合、グリッドは不平衡になり、結果的に 1 つの区画に他の区画よりも多くのデータが入ってしまうことが起こりえます。

データが非正規化またはフラット化されている場合、通常であれば 2 つのオブジェクト間で共有されるデータが、そうされずに複写され、各表は別々に区画化されるので、より平衡したグリッドになります。 これは使用されるメモリー量を増やしますが、必要なデータがすべて入っている単一のデータ行にアクセスできるため、アプリケーションはスケーラブルになります。データ保守にかかるコストは最近ますます高くなっているため、読み取り主体のグリッドにはこれは理想的です。

詳しくは、 XTP システムの分類およびスケーリングを参照してください。

データ・アクセス API を使用したリレーションシップ管理

ObjectMap API は、最も高速かつ柔軟で粒度の細かいデータ・アクセス API であり、マップのグリッド内のデータにアクセスする手段として、トランザクションを使用する、セッション・ベースの方法を提供します。ObjectMap API によって、クライアントは、標準 CRUD (作成、読み取り、更新、および削除) 操作を使用して分散グリッド内のオブジェクトのキーと値のペアを管理することができます。

ObjectMap API を使用するときには、オブジェクトのリレーションシップが、すべてのリレーションシップの外部キーを親オブジェクトに埋め込むことによって表される必要があります。

以下に例を示します。

```
public class Department {
 Collection<String> employeeIds;
}
```

EntityManager API は、外部キーを含んでいるオブジェクトから永続データを抽出することにより、リレーションシップ管理を単純にします。以下の例に示すように、オブジェクトが後でグリッドから取り出されるとき、リレーションシップ・グラフは再構築されます。

```
@Entity
public class Department {
 Collection<String> employees;
}
```

EntityManager API は、JPA や Hibernate といった他の Java オブジェクト・パーシスタンス・テクノロジー (管理対象 Java オブジェクト・インスタンスのグラフはパーシスタント・ストアと同期化されます) にとてもよく似ています。このケースでは、パーシスタント・ストアは eXtreme Scale グリッドであり、そこでは、各エンティティーがマップとして表され、マップはオブジェクト・インスタンスではなくエンティティー・データを含みます。

エンティティー・メタデータ記述子 XML ファイル

エンティティー・メタデータ記述子ファイルは、WebSphere eXtreme Scale のエンティティー・スキーマを定義するために使用される XML ファイルです。すべてのエンティティー・メタデータを XML ファイルで定義するか、エンティティー・メタデータをエンティティー Java クラス・ファイルでアノテーションとして定義します。主に Java アノテーションを使用できないエンティティーに使用されます。

XML ファイルに基づくエンティティー・メタデータを作成するには、XML 構成を使用します。XML 構成で定義されている属性の一部をアノテーションとともに使用すると、対応するアノテーションがオーバーライドされます。あるエレメントをオーバーライドできる場合は、そのオーバーライドが明示的に以降のセクションに出現します。エンティティー・メタデータ記述子 XML ファイルの例については、242 ページの『emd.xsd ファイル』を参照してください。

id エレメント

id エレメントは、属性がキーであることを暗黙に示しています。最低限、少なくとも 1 つの id エレメントを指定する必要があります。複数の id キーを指定して、1 つの複合キーとして使用することができます。

属性

name
属性の名前を指定します。この属性は Java ファイルに存在している必要があります。

alias
エレメントの別名を指定します。alias 値は、アノテーション付きエンティティーとともに使用されるとオーバーライドされます。

basic エレメント

basic エレメントは、属性が以下のようなプリミティブ型またはプリミティブ型のラッパーであることを暗黙に示しています。

- java.lang.String
- java.math.BigInteger

- java.math.BigDecimal
- java.util.Date
- java.util.Calendar
- java.sql.Date
- java.sql.Time
- java.sql.Timestamp
- byte[]
- Byte[]
- char[]
- Character[]
- Java Platform, Standard Edition バージョン 5 の列挙型

属性を basic として指定する必要はありません。basic エレメント属性は、リフレクションを使用して自動的に構成されます。

id-class エレメント

id_class エレメントは、複合キーを使用してエンティティーを見つけるときに役立つ複合キー・クラスを指定します。

属性

class-name
id-class エレメントで使用するクラス名 (すなわち id-class) を指定します。

transient

transient エレメントは、このエレメントが無視され、処理されないことを暗黙に示しています。アノテーション付きエンティティーと一緒に使用された場合は、オーバーライドすることもできます。

属性

name
無視される属性の名前を指定します。

version

transient エレメントは、このエレメントが無視され、処理されないことを暗黙に示しています。アノテーション付きエンティティーと一緒に使用された場合は、オーバーライドすることもできます。

属性

name
無視される属性の名前を指定します。

property エレメント

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。このプロパティーの名前は、このプロパティーを含む Bean が参照するクラスの set メソッドに対応している必要があります。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

name
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、このプロパティーを含む Bean の className 属性で指定されたクラスの 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 属性を `com.ibm.MyPlugin` に設定し、指定したプロパティーの名前が `size` である場合、`com.ibm.MyPlugin` クラスに setSize メソッドが必要です。(必須)

type
プロパティーのタイプを指定します。このタイプは、name 属性により識別される set メソッドに受け渡されます。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、名前を `size` と設定し、タイプを `int` と設定する場合は、Bean の className 属性として指定されたクラスに setSize(int) メソッドが存在している必要があります。 (必須)

value
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。 (必須)

description
プロパティーの説明です。(オプション)

```
<bean
(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"
/>
```

以下の例では、`companyGridProperty.xml` ファイルを使用して、Bean に property エレメントを追加する方法を示しています。この例では、maxSize という名前で int 型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。

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

 <objectGrids>
  <objectGrid name="CompanyGrid">
   <backingMap name="Customer"
    pluginCollectionRef="customerPlugins"/>
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="customerPlugins">
   <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・サンプルは、前の例の companyGridProperty.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);

BackingMap customerMap = companyGrid.defineMap("Customer");

LRUEvictor lruEvictor = new com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor();
// if the XML file is used instead,
// the property that was added would cause the following call to occur
lruEvictor.setMaxSize(449);
customerMap.setEvictor(lruEvictor);
```

backingMapPluginsCollections エレメント

backingMapPluginsCollections エレメントは、すべての backingMapPluginCollection エレメントのコンテナーです。前のセクションにある companyGridProperty.xml ファイルでは、backingMapPluginCollections エレメントに customerPlugins という ID の backingMapPluginCollection エレメントが 1 つ含まれています。

- 出現回数: 0 回から 1 回
- 子エレメント: backingMapPluginCollection エレメント

backingMapPluginCollection エレメント

backingMapPluginCollection エレメントは、BackingMap プラグインを定義し、**id** 属性によって識別されます。 pluginCollectionRef 属性を指定して、このプラグインを参照します。いくつかの BackingMaps プラグインを同様の方法で構成すると、各 BackingMap は同じ backingMapPluginCollection エレメントを参照できます。

- 出現回数: 0 回から複数回
- 子エレメント: Bean エレメント

属性

id backingMapPluginCollection を識別し、backingMap エレメントの pluginCollectionRef 属性によって参照されます。各 ID は固有である必要があります。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。 (必須)

```
<backingMapPluginCollection
(1)   id="id"
/>
```

以下の例では、companyGridCollection.xml ファイルを使用して、backingMapPluginCollection エレメントの使用法を示しています。このファイルでは、Customer BackingMap が customerPlugins backingMapPluginCollection を使用して、LRUEvictor を使用する Customer BackingMap を構成します。 Item および OrderLine の BackingMap は、collection2 backingMapPluginCollection を参照します。これらの BackingMap には、それぞれ LFUEvictor のセットが含まれています。

```
<?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="CompanyGrid">
   <backingMap name="Customer"
    pluginCollectionRef="customerPlugins"/>
   <backingMap name="Item" pluginCollectionRef="collection2"/>
   <backingMap name="OrderLine"
    pluginCollectionRef="collection2"/>
   <backingMap name="Order"/>
  </objectGrid>
 </objectGrids>
 <backingMapPluginCollections>
  <backingMapPluginCollection id="customerPlugins">
   <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・サンプルは、前の例の companyGridCollection.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();

ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
BackingMap customerMap = companyGrid .defineMap("Customer");
LRUEvictor customerEvictor = new LRUEvictor();
customerMap.setEvictor(customerEvictor);

BackingMap itemMap = companyGrid.defineMap("Item");
LFUEvictor itemEvictor = new LFUEvictor();
itemMap.setEvictor(itemEvictor);

BackingMap orderLineMap = companyGrid.defineMap("OrderLine");
LFUEvictor orderLineEvictor = new LFUEvictor();
orderLineMap.setEvictor(orderLineEvictor);

BackingMap orderMap = companyGrid.defineMap("Order");
```

querySchema エレメント

querySchema エレメントは、BackingMap 間のリレーションシップを定義し、各マップ内にあるオブジェクトのタイプを識別します。この情報は、照会言語ストリングをマップ・アクセス呼び出しに変換するために ObjectQuery によって使用されます。詳しくは、*プログラミング・ガイド* で ObjectQuery スキーマの定義の詳細を参照してください。

- 出現回数: 0 回から 1 回
- 子エレメント: mapSchemas エレメント、relationships エレメント

mapSchemas エレメント

各 querySchema エレメントは、1 つ以上の mapSchema エレメントを含む mapSchemas エレメントを 1 つ持ちます。

- 出現回数: 1 回
- 子エレメント: mapSchema エレメント

mapSchema エレメント

mapSchema エレメントは、BackingMap に保管されるオブジェクトのタイプ、およびそのデータにアクセスする方法を定義します。

- 出現回数: 1 回以上
- 子エレメント: なし

属性

mapName
スキーマに追加する BackingMap の名前を指定します。(必須)

valueClass
BackingMap の値部分に保管されるオブジェクトのタイプを指定します。(必須)

primaryKeyField
valueClass 属性内の 1 次キー属性の名前を指定します。1 次キーも BackingMap のキー部分に保管する必要があります。(オプション)

accessType
照会エンジンが valueClass オブジェクト・インスタンス内の永続データをイントロスペクトしてそのデータにアクセスする方法を示します。値を FIELD に設定すると、クラスのフィールドがイントロスペクトされ、スキーマに追加されます。値が PROPERTY の場合、get メソッドおよび is メソッドに関連付けられた属性が使用されます。デフォルト値は PROPERTY です。 (オプション)

```
<mapSchema
(1)   mapName="backingMapName"
(2)   valueClass="com.mycompany.OrderBean"
(3)   primaryKeyField="orderId"
(4)   accessType="PROPERTY" | "FIELD"
/>
```

以下の例では、companyGridQuerySchemaAttr.xml ファイルを使用して、サンプル mapSchema 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="CompanyGrid">
   <backingMap name="Order"/>
   <backingMap name="Customer"/>

   <querySchema>
    <mapSchemas>
```

```
      <mapSchema mapName="Order"
       valueClass="com.mycompany.OrderBean"
       primaryKeyField="orderNumber"
       accessType="FIELD"/>
      <mapSchema mapName="Customer"
       valueClass="com.mycompany.CustomerBean"
       primaryKeyField="id"
       accessType="FIELD"/>
     </mapSchemas>
    </querySchema>
   </objectGrid>
  </objectGrids>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridQuerySchemaAttr.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
companyGrid.defineMap("Order");
companyGrid.defineMap("Customer");

// Define the schema
QueryConfig queryCfg = new QueryConfig();
queryCfg.addQueryMapping(new QueryMapping(
    "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryMapping(new QueryMapping(
    "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
companyGrid.setQueryConfig(queryCfg);
```

relationships エレメント

各 querySchema エレメントは、relationships エレメントをまったく持たないか (ゼロ)、または 1 つ以上の relationship エレメントを含む relationships エレメントを 1 つ持ちます。

- 出現回数: 0 回または 1 回
- 子エレメント: relationship エレメント

relationship エレメント

relationship エレメントは、2 つの BackingMap 間のリレーションシップ、およびそのリレーションシップをバインドする valueClass 属性の属性を定義します。

- 出現回数: 1 回以上
- 子エレメント: なし

属性

source
リレーションシップのソース側 valueClass の名前を指定します。 (必須)

target
リレーションシップのターゲット側 valueClass の名前を指定します。 (必須)

relationField
ソース valueClass 内でターゲットを参照している属性の名前を指定します。(必須)

invRelationField

ターゲット valueClass 内でソースを参照している属性の名前を指定します。この属性が指定されていないと、リレーションシップは単一方向となります。(オプション)

```
<mapSchema
(1)   source="com.mycompany.OrderBean"
(2)   target="com.mycompany.CustomerBean"
(3)   relationField="customer"
(4)   invRelationField="orders"
/>
```

以下の例では、companyGridQuerySchemaWithRelationshipAttr.xml ファイルを使用して、双方向リレーションシップを含むサンプル mapSchema 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="CompanyGrid">
   <backingMap name="Order"/>
   <backingMap name="Customer"/>

   <querySchema>
    <mapSchemas>
     <mapSchema mapName="Order"
      valueClass="com.mycompany.OrderBean"
      primaryKeyField="orderNumber"
      accessType="FIELD"/>
     <mapSchema mapName="Customer"
      valueClass="com.mycompany.CustomerBean"
      primaryKeyField="id"
      accessType="FIELD"/>
    </mapSchemas>
    <relationships>
     <relationship
      source="com.mycompany.OrderBean"
      target="com.mycompany.CustomerBean"
      relationField="customer"/>
      invRelationField="orders"/>
    </relationships>
   </querySchema>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

以下のコード・サンプルは、前の例の companyGridQuerySchemaWithRelationshipAttr.xml ファイルと同じ構成をプログラムで実現する方法を示しています。

```
ObjectGridManager objectGridManager = ObjectGridManagerFactory.getObjectGridManager();
ObjectGrid companyGrid = objectGridManager.createObjectGrid("CompanyGrid", false);
companyGrid.defineMap("Order");
companyGrid.defineMap("Customer");

// Define the schema
QueryConfig queryCfg = new QueryConfig();
queryCfg.addQueryMapping(new QueryMapping(
    "Order", OrderBean.class.getName(), "orderNumber", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryMapping(new QueryMapping(
    "Customer", CustomerBean.class.getName(), "id", QueryMapping.FIELD_ACCESS));
queryCfg.addQueryRelationship(new QueryRelationship(
     OrderBean.class.getName(), CustomerBean.class.getName(), "customer", "orders"));
companyGrid.setQueryConfig(queryCfg);
```

timeBasedDBUpdate エレメント

timeBasedDBUpdate エレメントは、時間ベースのデータベース・アップデーターの構成を定義します。 timeBasedDBUpdate エレメントには、Java Persistence API (JPA) を使用して新しく挿入および更新されたレコードをデータベースから取り出す頻度、および対応する ObjectGrid マップ内のデータを更新する方法に関する情報が含まれています。

- 出現回数: 0 回または 1 回
- 子エレメント: なし

属性

entityClass
JPA プロバイダーと対話するために使用されるエンティティー・クラス名を指定します。このエンティティー・クラス名は、エンティティー照会を使用して JPA エンティティーを検索するために使用されます。(必須)

persistenceUnitName
JPA エンティティー・マネージャー・ファクトリーを作成するための JPA パーシスタンス・ユニット名を指定します。デフォルト値は、persistence.xml ファイル内で最初に定義されたパーシスタンス・ユニットの名前です。(オプション)

mode
時間ベースのデータベース更新モードを指定します。デフォルトでは、時間ベースのデータベース更新モードは、INVALIDATE_ONLY に設定されます。INVALIDATE_ONLY タイプは、データベース内の対応するレコードが変更された場合に、ObjectGrid マップ内のエントリーを無効にすることを示します。UPDATE_ONLY タイプは、ObjectGrid マップ内の既存のエントリーをデータベースの最新の値で更新することを示します。ただし、データベースに新たに挿入されたレコードは、すべて無視されます。INSERT_UPDATE タイプは、ObjectGrid マップ内の既存のエントリーをデータベースの最新の値で更新することを示します。また、データベースに新たに挿入されたレコードが、すべて ObjectGrid マップに挿入されます。(オプション)

timestampField
タイム・スタンプ・フィールドの名前を指定します。タイム・スタンプ・フィールドの値は、データベース・バックエンド・レコードが最後に更新された日時または順序を識別するために使用されます。 (オプション)

jpaPropertyFactory
JPAPropertyFactory 実装クラス名または spring Bean を示します。デフォルトの JPA プロパティーをオーバーライドするパーシスタンス・プロパティー・マップをプラグインするために、com.ibm.websphere.objectgrid.jpa.JPAPropertyFactory インターフェースが使用されます。JPAPropertyFactory インスタンス上で追加の属性を設定する必要がある場合は、Spring Framework Beans を使用してください。詳しくは、Spring Framework の統合の概要を参照してください。(オプション)

```
<timeBasedDBUpdate
(1)   persistenceUnitName="SamplePU"
(2)   mode="INVALIDATE_ONLY" | "UPDATE_ONLY" | "INSERT_UPDATE"
```

```
(3)   timestampField="TIMESTAMP"
(4)   entityClass="entity class"
(5)   jpaPropertyFactory="JPA property factory class" | "{spring}bean name"
/>
```

streamQuerySet エレメント

streamQuerySet エレメントは、ストリーム照会セットを定義するための最上位レベルのエレメントです。

- 出現回数: 0 回から複数回
- 子エレメント: stream エレメント、view エレメント

stream エレメント

stream エレメントは、ストリーム照会エンジンへのストリームを表します。stream エレメントの各属性は、StreamMetadata インターフェースのメソッドに対応します。

- 出現回数: 1 回から複数回
- 子エレメント: basic エレメント

属性

name
ストリームの名前を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

valueClass
このストリーム ObjectMap に保管される値のクラス・タイプを指定します。このクラス・タイプは、オブジェクトをストリーム・イベントに変換し、SQL ステートメントが指定されていない場合は SQL ステートメントを生成するために使用されます。 (必須)

sql ストリームの SQL ステートメントを指定します。このプロパティーを指定しなかった場合、valueClass 属性で属性または accessor メソッドのリフレクション、またはエンティティー・メタデータの tuple 属性を使用してストリーム SQL が生成されます。(オプション)

access
値クラスの属性にアクセスするためのタイプを指定します。この値を FIELD に設定すると、属性は Java のリフレクションを使用してフィールドから直接取り出されます。そうでない場合、属性は accessor メソッドを使用して読み取られます。デフォルト値は PROPERTY です。 (オプション)

```
<stream
(1)   name="streamName"
(2)   valueClass="streamMapClassType"
(3)   sql="streamSQL create stream stockQuote
       keyed by t ( transactionvolume INTEGER, price DECIMAL (9,2),
        issue VARCHAR(100) );"
(4)   access="PROPERTY" | "FIELD"
/>
```

view エレメント

view エレメントはストリーム照会ビューを表します。各 stream エレメントが ViewMetadata インターフェースのメソッドに対応します。

- 出現回数: 1 回から複数回
- 子エレメント: basic エレメント、id エレメント

属性

name
ビューの名前を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

sql ビューの変換を定義する、ストリームの SQL を指定します。この属性が指定されていないと、妥当性検査は失敗します。(必須)

valueClass
ObjectMap のこのビューに保管される値のクラス・タイプを指定します。このクラス・タイプは、ビュー・イベントをこのクラス・タイプと互換性のある適切なタプル・フォーマットに変換するのに使用されます。クラス・タイプを指定しなかった場合、Stream Processing Technology Structured Query Language (SPTSQL) の列定義に従ってデフォルトのフォーマットが使用されます。このビュー・マップにエンティティー・メタデータが定義されていると、この属性は使用されません。代わりに、エンティティー・メタデータが使用されます。 (オプション)

access
値クラスの属性にアクセスするためのタイプを指定します。アクセス・タイプを FIELD に設定すると、列値は、Java のリフレクションを使用して直接にフィールドに設定されます。そうでない場合、accessor メソッドを使用して属性が設定されます。デフォルト値は PROPERTY です。 (オプション)

```
<view
(1)   name="viewName"
(2)   valueClass="viewMapValueClass"
(3)   sql="viewSQL CREATE VIEW last5MinuteAvgPrice AS
      SELECT issue, avg(price) as totalVolume
        FROM (SELECT * FROM stockQuote FETCH LATEST 5 MINUTES) group by issue;"/>
(4)   access="PROPERTY" | "FIELD"
/>
```

basic エレメント

basic エレメントは、値クラスまたはエンティティー・メタデータの属性名から SPTSQL で定義された列へのマッピングを定義するために使用します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

```
<basic
(1)   name="attributeName"
(2)   column="columnName"
/>
```

id エレメント

id エレメントは、キー属性のマッピングに使用されます。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

```
<id
(1)   name="idName"
(2)   column="columnName"
/>
```

以下の例では、StreamQueryApp2.xml ファイルを使用して、streamQuerySet の属性を構成する方法を示しています。ストリーム照会セット _stockQuoteSQS_ にはストリームおよびビューが 1 つずつあります。ストリームおよびビューの両方で、名前、valueClass、sql、およびアクセス・タイプがそれぞれ定義されています。ストリームは basic エレメントも定義します。これにより、StockQuote クラスの volume 属性が、SQL ステートメントで定義される SQL 列 transactionvolume にマップされることが指定されます。

```
<?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="og1">
   <backingMap name="stockQuote" readOnly="false" copyKey="true"
    streamRef="stockQuote"/>
   <backingMap name="last5MinuteAvgPrice" readOnly="false" copyKey="false"
    viewRef="last5MinuteAvgPrice"/>

   <streamQuerySet name="stockQuoteSQS">
    <stream
     name="stockQuote"
     valueClass="com.ibm.ws.objectgrid.streamquery.sample.guide.StockQuote"
     sql="create stream stockQuote
         keyed by t ( transactionvolume INTEGER, price DECIMAL (9,2),
        issue VARCHAR(100) );"
     access="FIELD">
     <basic name="volume" column="transactionvolume"/>
    </stream>

    <view
     name="last5MinuteAvgPrice"
     valueClass="com.ibm.ws.objectgrid.streamquery.sample.guide.AveragePrice"
     sql="CREATE VIEW last5MinuteAvgPrice AS SELECT issue, avg(price) as avgPrice
       FROM (SELECT * FROM stockQuote FETCH LATEST 5 MINUTES) group by issue;"
     access="FIELD"
    </view>
   </streamQuerySet>
  </objectGrid>
 </objectGrids>
</objectGridConfig>
```

emd.xsd ファイル

エンティティー・メタデータ XML スキーマ定義を使用して記述子 XML ファイルを作成し、 WebSphere eXtreme Scale のエンティティー・スキーマを定義します。

emd.xsd ファイルの各エレメントおよび属性の説明は、231 ページの『エンティティー・メタデータ記述子 XML ファイル』を参照してください。

emd.xsd ファイル

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

    <!-- ************************************************* -->
    <xsd:element name="entity-mappings">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="description" type="xsd:string" minOccurs="0" />
                <xsd:element name="entity" type="emd:entity" minOccurs="1" maxOccurs="unbounded" />
```

```
        </xsd:sequence>
    </xsd:complexType>
    <xsd:unique name="uniqueEntityClassName">
        <xsd:selector xpath="emd:entity" />
        <xsd:field xpath="@class-name" />
    </xsd:unique>
</xsd:element>

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

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

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

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

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

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

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

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

<!-- ************************************************** -->
<xsd:complexType name="many-to-one">
    <xsd:sequence>
```

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

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

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

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

<!-- ************************************************** -->
<xsd:complexType name="emptyType" />

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

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

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

<!-- ************************************************** -->
<xsd:complexType name="entity-listener">
    <xsd:sequence>
        <xsd:element name="pre-persist" type="emd:pre-persist" minOccurs="0" />
        <xsd:element name="post-persist" type="emd:post-persist" minOccurs="0" />
        <xsd:element name="pre-remove" type="emd:pre-remove" minOccurs="0" />
        <xsd:element name="post-remove" type="emd:post-remove" minOccurs="0" />
        <xsd:element name="pre-invalidate" type="emd:pre-invalidate" minOccurs="0" />
        <xsd:element name="post-invalidate" type="emd:post-invalidate" minOccurs="0" />
        <xsd:element name="pre-update" type="emd:pre-update" minOccurs="0" />
```

```
            <xsd:element name="post-update" type="emd:post-update" minOccurs="0" />
            <xsd:element name="post-load" type="emd:post-load" minOccurs="0" />
        </xsd:sequence>
        <xsd:attribute name="class-name" type="xsd:string" use="required" />
    </xsd:complexType>

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

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

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

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

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

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

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

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

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

</xsd:schema>
```

キャッシュ統合の構成

WebSphere eXtreme Scale を他のキャッシュ関連製品と統合することができます。WebSphere eXtreme Scale とデータベースとの間で、変更を統合するためのローダーとして JPA を使用することができます。また、WebSphere eXtreme Scale 動的キャッシュ・プロバイダーを使用して、WebSphere eXtreme Scale を WebSphere Application Server 内の動的キャッシュ・コンポーネントに接続することもできます。 WebSphere Application Server に対する拡張としてもう 1 つ考えられるのは、HTTP セッションをキャッシュに入れる操作を支援する WebSphere eXtreme Scale HTTP セッション・マネージャーです。

キャッシュ統合の概要: JPA、セッション、および動的キャッシング

多用途性や信頼性などを持ちながら実行する能力を WebSphere eXtreme Scale に付与する重大な要素は、キャッシング概念のアプリケーションです。それは、ほとんどすべてのデプロイメント環境で、データのパーシスタンスや再収集を最適化します。

JPA ローダーの構成

Java Persistence API (JPA) ローダーは、JPA を使用してデータベースと対話するプラグイン実装です。

始める前に

- Hibernate または OpenJPA などの JPA 実装が必要です。
- データベースには、選択された JPA プロバイダーがサポートする任意のバックエンドを使用できます。
- ObjectMap API を使用してデータを保管する場合、JPALoader プラグインを使用することができます。EntityManager API を使用してデータを保管する場合、JPAEntityLoader プラグインを使用します。

このタスクについて

Java Persistence API (JPA) ローダーがどのように機能するかについて詳しくは、*製品概要*にある情報を参照してください。

手順

1. データベースと対話するために JPA が必要とする必須パラメーターを構成します。

   次のパラメーターが必要です。これらのパラメーターは、JPALoader または JPAEntityLoader Bean、および JPATxCallback Bean 内で構成されます。

   - **persistenceUnitName**: パーシスタンス・ユニット名を指定します。このパラメーターは、JPA EntityManagerFactory の作成、および `persistence.xml` ファイル内の JPA エンティティー・メタデータの検索という 2 つの目的のために必要です。この属性は、JPATxCallback Bean に設定されます。
   - **JPAPropertyFactory**: パーシスタンス・プロパティー・マップを作成し、デフォルトのパーシスタンス・プロパティーをオーバーライドするためのファクトリーを指定します。この属性は、JPATxCallback Bean に設定されます。この属性を設定するために、 Spring スタイルの構成が必要です。
   - **entityClassName**: JPA メソッド (EntityManager.persist、EntityManager.find など) を使用する場合に必要なエンティティー・クラス名を指定します。JPALoader にはこのパラメーターが必要ですが、**JPAEntityLoader** ではこのパラメーターはオプションです。 JPAEntityLoader の場合、**entityClassName** パラメーターが構成されていないと、ObjectGrid エンティティー・マップに構成されているエンティティー・クラスが使用されます。 eXtreme Scale EntityManager と JPA プロバイダーに同じクラスを使用する必要があります。この属性は、JPALoader または JPAEntityLoader Bean に設定されます。
   - **preloadPartition**: マップ・プリロードが開始される区画を指定します。プリロード区画がゼロより小さいか、または「区画総数マイナス 1」よりも大きい場合、マップ・プリロードは開始されません。デフォルト値は -1 ですから、デフォルトではプリロードは開始されません。この属性は、JPALoader または JPAEntityLoader Bean に設定されます。

   この 4 つの JPA パラメーターが eXtreme Scale に構成されるほか、JPA エンティティーからキーを取得するため、JPA メタデータが使用されます。JPA メ

タデータは、アノテーションとして構成するか、 persistence.xml ファイル内に指定される orm.xml ファイルとして構成できます。これは、eXtreme Scale 構成には含まれません。

2. JPA 構成用に XML ファイルを構成します。

   JPALoader あるいは JPAEntityLoader を構成する場合は、*プログラミング・ガイド* にあるローダー・プラグインに関する情報を参照してください。

   ローダー構成と一緒に、JPATxCallback トランザクション・コールバックを構成します。以下に、JPAEntityLoader と JPATxCallback が構成されている ObjectGrid XML 記述子ファイル (objectgrid.xml) の例を示します。

   コールバックを含むローダーの構成 - XML の例

```
<?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="JPAEM" entityMetadataXMLFile="jpaEMD.xml">
        <bean id="TransactionCallback"
           className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
           <property
              name="persistenceUnitName"
              type="java.lang.String"
              value="employeeEMPU" />
       </bean>
       <backingMap name="Employee" pluginCollectionRef="Employee" />
    </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="Employee">
        <bean id="Loader"
          className="com.ibm.websphere.objectgrid.jpa.JPAEntityLoader">
       <property
              name="entityClassName"
              type="java.lang.String"
              value="com.ibm.ws.objectgrid.jpa.test.entity.Employee"/>
        </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

   JPAPropertyFactory を構成する場合は、Spring スタイルの構成を使用する必要があります。以下に示すのは、Spring Bean が eXtreme Scale 構成に使用されるように構成している XML 構成ファイル・サンプル例 (JPAEM_spring.xml) です。

   JPA プロパティー・ファクトリーを含むローダーの構成 - XML の例

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

  <objectgrid:JPAEntityLoader id="jpaLoader"
  entityClassName="com.ibm.ws.objectgrid.jpa.test.entity.Employee"/>
  <objectgrid:JPATxCallback id="jpaTxCallback" persistenceUnitName="employeeEMPU" />
</beans>
```

次に Objectgrid.xml 構成 XML ファイルを示します。 ObjectGrid の名前は JPAEM であり、これは、JPAEM_spring.xml Spring 構成ファイルの ObjectGrid 名に一致しています。

JPAEM ローダー構成 - XML の例

```
<?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="JPAEM" entityMetadataXMLFile="jpaEMD.xml">
      <bean id="TransactionCallback"
            className="{spring}jpaTxCallback"/>
        <backingMap name="Employee" pluginCollectionRef="Employee"
                    writeBehind="T4"/>
      </objectGrid>
  </objectGrids>

  <backingMapPluginCollections>
    <backingMapPluginCollection id="Employee">
       <bean id="Loader" className="{spring}jpaLoader" />
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

エンティティーには、JPA アノテーションおよび eXtreme Scale エンティティー・マネージャー・アノテーションの両方でアノテーションを付けることができます。各アノテーションには、使用可能な等価 XML があります。そのため、eXtreme Scale は Spring 名前空間を追加しています。この Spring 名前空間サポートを使用してこれらを構成することもできます。

JPA 時間ベース・データ・アップデーターの構成

時間ベース・データベース更新は、ローカルまたは分散 eXtreme Scale 構成の場合、XML を使用して構成することができます。ローカル構成はプログラムでも構成できます。

このタスクについて

Java Persistence API (JPA) 時間ベース・データ・アップデーターがどのように機能するかについて詳しくは、 「*プログラミング・ガイド*」の情報を参照してください。

手順

timeBasedDBUpdate 構成を作成します。

- **XML ファイルを使用:**

  次に示すのは、timeBasedDBUpdate 構成を含む objectgrid.xml ファイルの例です。

  JPA 時間ベース・アップデーター - XML の例

```
<?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="changeOG"
            entityMetadataXMLFile="userEMD.xml">
            <backingMap name="user" >
                <timeBasedDBUpdate timestampField="rowChgTs"
                      persistenceUnitName="userderby"
                      entityClass="com.test.UserClass"
                      mode="INVALIDATE_ONLY"
                />
            </backingMap>
```

```
        </objectGrid>
    </objectGrids>
    <backingMapPluginCollections>
</objectGridConfig>
```

この例では、マップ "user" が、時間ベース・データベース更新で構成されています。データベースの更新モードは、INVALIDATE_ONLY、タイム・スタンプ・フィールドは、rowChgTs です。

分散 ObjectGrid "changeOG" が、コンテナー・サーバーで開始されると、時間ベース・データベース更新スレッドが、区画 0 で自動的に始動されます。

- **プログラムで:**

  ローカル ObjectGrid を作成する場合、TimeBasedDBUpdateConfig オブジェクトを作成し、これを BackingMap インスタンスに設定することもできます。

  ```
  public void setTimeBasedDBUpdateConfig(TimeBasedDBUpdateConfig dbUpdateConfig);
  ```

  BackingMap インスタンスへのオブジェクトの設定に関して詳しくは、 API 資料にある BackingMap インターフェースに関する情報を参照してください。

  また、com.ibm.websphere.objectgrid.jpa.dbupdate.annotation.Timestamp アノテーションを使用して、エンティティー・クラスのタイム・スタンプ・フィールドにアノテーションを付けることができます。このクラスの値を構成すれば、XML 構成の timestampField を構成する必要はありません。

次のタスク

JPA 時間ベース・データ・アップデーターを開始します。詳しくは、「*プログラミング・ガイド*」で JPA 時間ベース・データ・アップデーターの開始に関する情報を参照してください。

JPA キャッシュ・プラグインの構成

WebSphere eXtreme Scale には、OpenJPA と Hibernate Java Persistence API (JPA) プロバイダーの両方に対するレベル 2 のキャッシュ・プラグインが組み込まれています。

ObjectGrid JPA キャッシュ構成プロパティー

JPA キャッシュ・プラグインは、以下のプロパティーで構成することができます。プロパティーはすべてオプションです。

ObjectGridName
固有の ObjectGrid 名を指定します。デフォルト値は、定義済みのパーシスタンス・ユニット名になります。パーシスタンス・ユニット名が JPA プロバイダーから使用できない場合、生成名が使用されます。

ObjectGridType
ObjectGrid のタイプを指定します。

有効な値:

- **EMBEDDED**: デフォルトおよび推奨構成タイプ。そのデフォルト設定には、`NumberOfPartitions=1`、`ReplicaMode=SYNC`、`ReplicaReadEnabled=true` および `MaxNumberOfReplicas=47` が含まれま

す。 **ReplicaMode** パラメーターは複製モードの設定に、**MaxNumberOfReplicas** パラメーターはレプリカの最大数の設定に使用します。システムに 47 を超える Java 仮想マシンがある場合、**MaxNumberOfReplicas** 値を Java 仮想マシン の数に等しくなるように設定します。

- **EMBEDDED_PARTITION**: システムが、分散システムで大量のデータをキャッシュに入れる必要がある場合に使用するタイプ。デフォルトの区画数は 47 でレプリカ・モードは NONE です。少数の Java 仮想マシン のみを持つ小規模のシステムでは、**NumberOfPartitions** を Java 仮想マシン の数以下に設定します。システムを調整するために、**ReplicaMode**、**NumberOfPartitions**、および **ReplicaReadEnabled** 値を指定できます。
- **REMOTE**: キャッシュは、カタログ・サービスからリモートの分散 ObjectGrid に接続しようとします。

NumberOfPartitions

有効な値: 1 以上キャッシュに使用される区画の数を指定します。このプロパティーは、 ObjectGridType の値が EMBEDDED_PARTITION に設定されている場合に適用されます。デフォルト値は 47 です。EMBEDDED タイプの場合、 **NumberOfPartitions** 値は常に 1 です。

ReplicaMode

有効な値: SYNC/ASYNC/NONEキャッシュを複製にコピーする場合に使用するメソッドを指定します。このプロパティーは、ObjectGridType の値を EMBEDDED または EMBEDDED_PARTITION に設定している場合に適用されます。デフォルト値は、EMBEDDED_PARTITION タイプの場合が NONE で、EMBEDDED タイプの場合は SYNC です。**ReplicaMode** が、EMBEDDED ObjectGridType の場合に NONE に設定されても、EMBEDDED タイプではやはり SYNC の **ReplicaMode** が使用されます。

ReplicaReadEnabled

有効な値: TRUE または FALSE使用可能にする場合、クライアントはレプリカから読み込みます。このプロパティーは、EMBEDDED_PARTITION タイプに適用されます。デフォルト値は、EMBEDDED_PARTITION タイプの場合 FALSE です。 EMBEDDED タイプの場合は、常に **ReplicaReadEnabled** の値は TRUE に設定されます。

MaxUsedMemory

有効な値: TRUE または FALSEメモリーに余裕がなくなった場合にキャッシュ・エントリーの除去を使用可能にします。デフォルト値は TRUE で、JVM ヒープ使用率しきい値が 70 % を超えると、データの除去が開始されます。デフォルトの JVM ヒープ使用率しきい値の比率 (%) は、objectGridServer.properties ファイルの memoryThresholdPercentage プロパティーを設定し、このファイルをクラスパスに配置することによって変更することができます。 Evictor について詳しくは、「*製品概要*」で Evictor に関する情報を参照してください。サーバー・プロパティー・ファイルに関して詳しくは、「*管理ガイド*」を参照してください。

MaxNumberOfReplicas

有効な値: 1 以上キャッシュに使用されるレプリカの最大数を指定します。この値は、EMBEDDED タイプのみに適用されます。この数値は、システム内の Java 仮想マシン 数以上にする必要があります。デフォルト値は 47 です。

NumberOfPartitions、ReplicaMode、ReplicaReadEnabled、および MaxNumberOfReplicas プロパティーは、ObjectGrid デプロイメントの要素となります。NumberOfPartitions、ReplicaMode、および ReplicaReadEnabled は、EMBEDDED_PARTITION タイプに適用されます。
ReplicaMode、MaxNumberOfReplicas は、いずれも EMBEDDED タイプに適用されます。

EMBEDDED および EMBEDDED_PARTITION の考慮事項

組み込み ObjectGrid タイプでは、前述の構成プロパティーを使用して、必要に応じ、ObjectGrid コンテナー・サーバーのセットおよびカタログ・サービスを構成およびデプロイします。コンテナーのライフサイクルは、JPA アプリケーションにバインドされ、アプリケーションのクラスパス内に連結されます。アプリケーションが開始されると、プラグインでは、自動的にカタログ・サービスを検出または開始し、コンテナーを開始して、カタログ・サービスに接続します。それから、プラグインは、クライアント接続を使用して別のアプリケーション・サーバー・プロセスで実行されている ObjectGrid コンテナーおよびそのピアと通信します。

各 JPA エンティティーには、エンティティーのクラス名を使用して独立したバックアップ・マップが割り当てられます。各 BackingMap には、次の属性があります。

- readOnly="false"
- copyKey="false"
- lockStrategy="NONE"
- copyMode="NO_COPY"

注: EMBEDDED または EMBEDDED_PARTITION ObjectGridType を Java SEJava SE 環境で使用する場合、組み込み eXtreme Scale サーバーを停止するため、プログラムの末尾に System.exit(0) メソッドを使用してください。そうでないと、プログラムが反応しなくなったように見えます。

ObjectGridType のデフォルト値

ObjectGridType 値は、ObjectGrid キャッシュがデプロイされるトポロジーを指定します。デフォルトで、パフォーマンスが最善のタイプは EMBEDDED です。以下のセクションでは、ObjectGridType 値について、デフォルトのプロパティーを説明します。

EMBEDDED ObjectGrid JPA キャッシュ・トポロジーのデフォルト

EMBEDDED の ObjectGrid タイプを使用する場合、構成で何の値も指定されない場合、次のデフォルト・プロパティー値が使用されます。

- **ObjectGridName**: パーシスタンス・ユニット名
- **ObjectGridType**: EMBEDDED
- **NumberOfPartitions**: 1 (ObjectGrid タイプが EMBEDDED の場合、変更不可)

- **ReplicaMode**: SYNC
- **ReplicaReadEnabled**: TRUE (ObjectGrid タイプが EMBEDDED の場合、変更不可)
- **MaxUsedMemory**: TRUE
- **MaxNumberOfReplicas**: 47 (分散システムでは、Java 仮想マシン の数以下)

名前が競合しないように、固有の ObjectGridName 値を指定する必要があります。MaxNumberOfReplicas 値は、システム内の合計の Java 仮想マシン 数以上にする必要があります。

REMOTE ObjectGrid キャッシュ・トポロジー

REMOTE ObjectGrid タイプでは、ObjectGrid およびデプロイメント・ポリシーが JPA アプリケーションとは別に定義されるため、プロパティー設定は必要ありません。JPA キャッシュ・プラグインは、リモートで、既存のリモート ObjectGrid に接続します。

ObjectGrid とのすべての対話がリモートで行われるため、このトポロジーは、すべての ObjectGrid タイプの中で、パフォーマンスが最も遅くなります。

カタログ・サービスの考慮事項および構成

EMBEDDED または EMBEDDED_PARTITION トポロジーで実行する場合、JPA キャッシュ・プラグインでは、必要に応じてアプリケーション・プロセスの 1 つの内部で、単一のカタログ・サービスを自動的に開始します。実稼働環境では、カタログ・サービス・ドメインを作成する必要があります。カタログ・サービスの定義について詳しくは、 *製品概要* にある高可用性カタログ・サービスに関する情報を参照してください。

WebSphere Application Server プロセス内で実行する場合、JPA キャッシュ・プラグインでは、WebSphere Application Server セルに定義されたカタログ・サービスまたはカタログ・サービス・ドメインに自動的に接続します。カタログ・サービス・ドメインの定義について詳しくは、*管理ガイド*にあるカタログ・サービスの開始に関する情報を参照してください。

WebSphere Application Server プロセス内でサーバーを実行しない場合、カタログ・サービス・ドメインのホストおよびポートは、objectGridServer.properties というプロパティー・ファイルを使用して指定されます。このファイルは、アプリケーションのクラスパスに保管し、**catalogServiceEndPoints** プロパティーを定義しておく必要があります。カタログ・サービス・グリッドは、アプリケーション・プロセスとは別に開始され、アプリケーション・プロセスが開始される前に開始されなければなりません。

objectGridServer.properties ファイルのフォーマットは次のとおりです。

```
catalogServiceEndPoints=<hostname1>:<port1>,<hostname2>:<port2>
```

JPA キャッシュ・プラグイン

WebSphere eXtreme Scale には、OpenJPA と Hibernate Java Persistence API (JPA) プロバイダーの両方に対するレベル 2 (L2) のキャッシュ・プラグインが組み込まれています。

eXtreme Scale を L2 キャッシュ・プロバイダーとして使用することにより、データ読み取りおよび照会時のパフォーマンスが向上し、データベースに対する負荷が軽減します。WebSphere eXtreme Scale ではキャッシュがすべてのプロセスで自動的に複製されるので、組み込みキャッシュ実装をしのぐ利点があります。あるクライアントが値をキャッシュすると、他のすべてのクライアントが、そのキャッシュされた値をローカルのメモリー内で使用できるようになります。

OpenJPA および Hibernate ObjectGrid キャッシュ・プラグインを使用すると、組み込み、組み込みの区画化、およびリモートの 3 つのトポロジー・タイプが作成できます。

組み込みトポロジー

組み込みトポロジーでは、各アプリケーションのプロセス・スペース内に eXtreme Scale サーバーを作成します。OpenJPA および Hibernate が、キャッシュのメモリー内コピーで直接読み取りを行い、他のすべてのコピーに書き込みを行います。非同期複製を使用することによって、書き込みのパフォーマンスを向上させることができます。このデフォルト・トポロジーは、キャッシュ・データの量が少なく、1 つのプロセスに十分収まる場合に最も良く機能します。

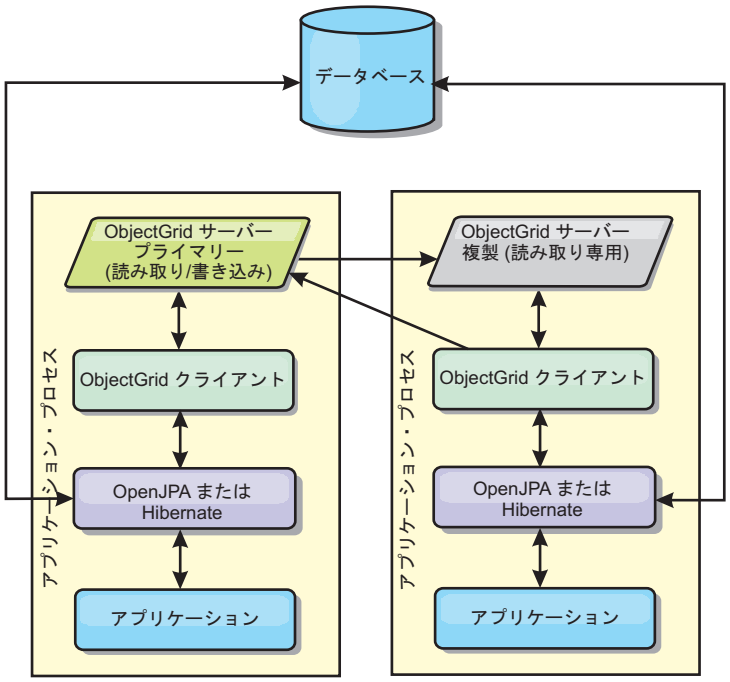


図 11. JPA 組み込みトポロジー

利点:

- すべてのキャッシュ読み取りが非常に速く、ローカル・アクセスである。
- 構成が簡単である。

制約:

- データ量が、プロセスのサイズに限られる。
- すべてのキャッシュ更新が 1 つのプロセスに送られる。

組み込みの区画化トポロジー

キャッシュ・データの量が多く、1 つのプロセスに収まらない場合、組み込みの区画化トポロジーでは、ObjectGrid 区画を使用して、データを複数のプロセスに分割します。多くのキャッシュ読み取りがリモートになるため、パフォーマンスは組み込みトポロジーほど高くありません。データベース待ち時間が大きい場合でも、このオプションを使用できます。

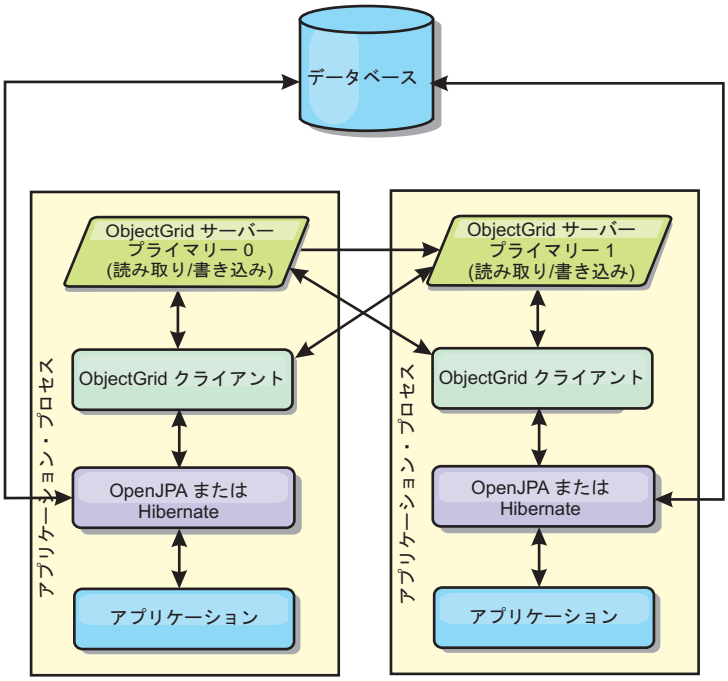


図 12. JPA 組み込みの区画化トポロジー

利点:

- 大容量のデータを保管できる。
- 構成が簡単である。
- キャッシュ更新が複数のプロセスに分散される。

制約:

- ほとんどのキャッシュ読み取りおよび更新がリモートで行われる。

例えば、JVM あたり最大 1 GB で 10 GB のデータをキャッシュに入れる場合、Java 仮想マシン が 10 必要になります。したがって、区画数は 10 以上に設定されます。理想的には、区画数は素数に設定され、各断片が適当な量のメモリーを保管することが望ましいと言えます。通常、numberOfPartitions は、Java 仮想マシン の数に等しくなるようにします。このように設定すれば、各 JVM に 1 つの区画が格納されます。複製を使用可能にする場合、システム内の Java 仮想マシンの数を増やす必要があります。 そうでないと、各 JVM ごとに 1 つのレプリカ区画も格納することになり、この区画はプライマリー区画と同量のメモリーを消費します。

選択された構成のパフォーマンスを最大化するには、「*管理ガイド*」の『メモリー・サイズ設定および区画数の計算』を参照してください。

例えば、4 つの Java 仮想マシン があるシステムで numberOfPartitions 設定値が 4 の場合、各 JVM は 1 つのプライマリー区画をホストします。読み取り操作では、25 パーセントの確率でローカルで使用可能な区画からデータを取り出すことができ、これは、リモート JVM からデータを取得する場合と比較してはるかに速くなります。照会の実行などの読み取り操作で 4 つの区画が均等に関わるデータ・コレクションを取り出す必要がある場合、呼び出しの 75 パーセントがリモートで、呼び出しの 25 パーセントがローカルです。ReplicaMode が SYNC または ASYNC に設定され、ReplicaReadEnabled が true に設定されている場合、4 つのレプリカ区画が作成され、これが 4 つの Java 仮想マシン に分散されます。各 JVM は、1 つのプライマリー区画と 1 つのレプリカ区画をホストします。読み取り操作をローカルで実行する確率は、50 パーセントに増えます。4 つの区画が均等に関わるデータ・コレクションを取り出す読み取り操作では、50 パーセントのリモート呼び出しと 50 パーセントのローカル呼び出しがあります。ローカル呼び出しは、リモート呼び出しよりはるかに高速です。リモート呼び出しが行われると、パフォーマンスは落ちます。

リモート・トポロジー

リモート・トポロジーでは、すべてのキャッシュ・データを 1 つ以上の個別のプロセスに保管し、アプリケーション・プロセスのメモリー使用を減らします。区画化され、複製された eXtreme Scale グリッドをデプロイすることによって、個別のプロセスへデータ配布するという利点が生かせます。 前のセクションで説明した組み込み構成および組み込み区画化構成とは対照的に、リモート・グリッドを管理する場合は、アプリケーションおよび JPA プロバイダーから独立して管理する必要があります。 eXtreme Scale グリッド・デプロイメントの管理について詳しくは、デプロイメント環境のモニターを参照してください。

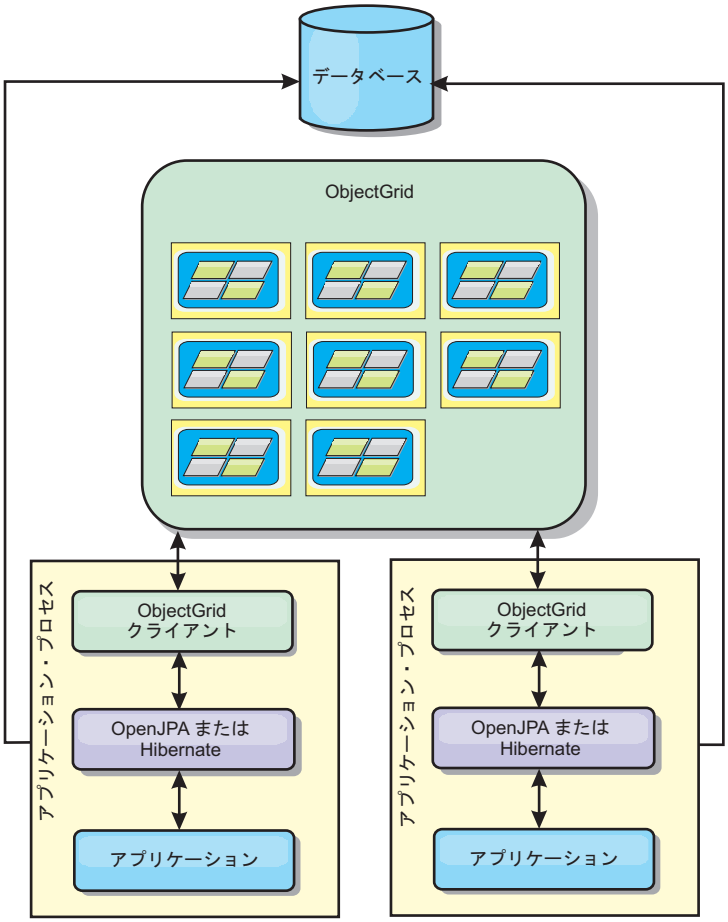


図 13. JPA リモート・トポロジー

利点:

- 大容量のデータを保管できる。
- アプリケーション・プロセスがキャッシュ・データから解放される。
- キャッシュ更新が複数のプロセスに分散される。
- 非常に柔軟な構成オプションがある。

制約:

- すべてのキャッシュ読み取りおよび更新がリモートで行われる。

Hibernate キャッシュ・プラグイン構成

構成ファイルのプロパティーを設定することにより、Hibernate に対して eXtreme Scale キャッシュを使用可能にすることができます。

7.1+ WebSphere Application Serverと統合するために、Hibernate キャッシュ・プラグインは `oghibernate-cache.jar` にパッケージされ、 `WAS_HOME/optionalLibraries/ObjectGrid` にインストールされます。 Hibernate キャッシュ・プラグインを使用するには、Hibernate ライブラリーに `oghibernate-cache.jar` が含まれていなければなりません。 例えば、使用するアプリケーションに Hibernate ライブラリーが含まれている場合、`oghibernate-cache.jar` ファイルも含まれてい

なければなりません。 Hibernate ライブラリーを含むように共有ライブラリーを定義している場合は、oghibernate-cache.jar をその共有ライブラリー・ディレクトリーに置かなければなりません。

7.1+ eXtreme Scale バージョン 7.1 は WebSphere Application Server 環境で cglib.jar をインストールしません。 Hibernate などの cglib.jar に依存する既存アプリケーションまたは共有ライブラリーがある場合、cglib.jar を見つけ、クラスパスに組み込んでください。 例えば、アプリケーションにすべての Hibernate ライブラリーの JAR ファイルが含まれていても、Hibernate で使用できる cglib.jar が除外されている場合、Hibernate からの cglib.jar をアプリケーションに組み込んでください。

設定

persistence.xml ファイルのプロパティーを設定するための構文は、以下のとおりです。

persistence.xml

```
<property name="hibernate.cache.provider_class"
        value="com.ibm.websphere.objectgrid.hibernate.cache.ObjectGridHibernateCacheProvider" />
<property name="hibernate.cache.use_query_cache" value="true"/>
<property name="objectgrid.configuration" value="<property>=<value>,..." />
<property name="objectgrid.hibernate.regionNames" value="<regionName>,.." />
```

hibernate.cfg.xml ファイルのプロパティーを設定するための構文は、以下のとおりです。

hibernate.cfg.xml

```
<property name="cache.provider_class">com.ibm.websphere.objectgrid.
        hibernate.cache.ObjectGridHibernateCacheProvider</property>
<property name="cache.use_query_cache">true</property>
<property name="objectgrid.configuration"><property>=<value>,...</property>
<property name="objectgrid.hibernate.regionNames"><regionName>,...</property>
```

provider_class プロパティーは、com.ibm.websphere.objectgrid.hibernate.cache.ObjectGridHibernateCacheProvider プロパティーです。照会キャッシュを使用可能にするには、use_query_cache プロパティーの値を「true」に設定してください。objectgrid.configuration プロパティーを使用して、eXtreme Scale キャッシュ構成プロパティーを指定します。

名前が競合する可能性をなくすため、固有の ObjectGridName プロパティー値を指定する必要があります。他の eXtreme Scale キャッシュ構成プロパティーはオプションです。

objectgrid.hibernate.regionNames プロパティーはオプションですが、eXtreme Scale キャッシュの初期化後に regionNames 値が定義される場合は指定する必要があります。regionName にマップされるエンティティー・クラスで、そのエンティティー・クラスが persistence.xml ファイルに指定されていないか、または Hibernate マッピング・ファイルに含まれていない例を考えます。さらに、これにエンティティー・アノテーションが設定されているとします。これにより、このエンティティー・クラスの regionName は、eXtreme Scale キャッシュが初期化される場合、クラス・ロード時に解決されます。別の例としては、eXtreme Scale キャッシュ初期化後に実行される Query.setCacheRegion(String regionName) メソッドがあります。こうした状態では、動的決定の可能性があるすべての regionNames を

objectgrid.hibernate.regionNames プロパティーに組み入れ、eXtreme Scale キャッシュがすべての regionNames の BackingMaps を準備できるようにします。

以下に示すのは、persistence.xml および hibernate.cfg.xml ファイルの例です。

persistence.xml

```
<persistence-unit name="testPU2">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <class>com.ibm.websphere.objectgrid.jpa.test.Department</class>
    <properties>
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.connection.url" value="jdbc:derby:DB_testPU2;create=true" />
      <property name="hibernate.connection.driver_class" value="org.apache.derby.jdbc.EmbeddedDriver" />

      <property name="hibernate.cache.provider_class" value="com.ibm.websphere.objectgrid.
              hibernate.cache.ObjectGridHibernateCacheProvider" />
      <property name="hibernate.cache.use_query_cache" value="true"/>
      <property name="objectgrid.configuration" value="ObjectGridName=myOGName,ObjectGridType=
              EMBEDDED,MaxNumberOfReplicas=4" writeBehind=true, writeBehindInterval=5000,
              writeBehindPoolSize=10, writeBehindMaxBatchSize=1000" />
      <property name="objectgrid.hibernate.regionNames" value="queryRegion1, queryRegion2" />
    </properties>
</persistence-unit>
```

7.1+ バージョン 7.1 では、標準 JPA キャッシュ・プラグイン構成オプションに加えて、Hibernate キャッシュ・プラグイン用の特定の構成オプションである、追加の後書き機能が導入されました。

writeBehind

有効な値: TRUE または FALSE

デフォルト値: FALSE

writeBehind が使用可能になっていると、writeBehindInterval 条件または writeBehindMaxBatchSize 条件のどちらかが一致するまで、更新は一時的に JVM の有効範囲データ・ストレージに保管されます。

重要: writeBehind が使用可能でない場合、他の後書き構成設定は無視されます。

writeBehindInterval

有効な値: 1 以上

デフォルト値: 5000 (5 秒)

更新をキャッシュにフラッシュする時間間隔 (ミリ秒) を指定します。

writeBehindPoolSize

有効な値: 1 以上

デフォルト値: 5

更新をキャッシュにフラッシュするときに使用するスレッド・プールの最大サイズを指定します。

writeBehindMaxBatchSize

有効な値: 1 以上

デフォルト値: 1000

更新をキャッシュにフラッシュする際の、領域キャッシュごとの最大バッチ・サイズを指定します。

前記のコードの例には、以下の後書き機能構成が表示されています。

```
writeBehind=true, writeBehindInterval=5000, writeBehindPoolSize=10, writeBehindMaxBatchSize=1000
```

ここで、

- writeBehind=TRUE は、後書き機能を使用可能にします。
- writeBehindInterval=5000 は、更新が約 5 秒ごとにキャッシュにフラッシュされることを意味します。
- writeBehindPoolSize=10 は、更新をキャッシュにフラッシュする際に、作業を実行するために使用するスレッドの最大数が 10スレッドであることを示します。
- writeBehindMaxBatchSize=1000 は、領域キャッシュの後書きストレージに保管された更新が 1000 エントリーを超過すると、指定された writeBehindInterval 条件が一致しなくても、更新はキャッシュにフラッシュされることを意味します。 すなわち、約 5 秒ごと、または各領域キャッシュの後書きストレージのサイズが 1000 エントリーを超過した場合のどちらかで、更新はキャッシュにフラッシュされます。 writeBehindMaxBatchSize 条件が一致する場合は注意してください。この条件に一致する領域キャッシュのみが後書きストレージの更新をキャッシュにフラッシュします。 領域キャッシュは、通常エンティティーまたは照会に対応しています。

重要:

後書き機能構成を使用する場合は注意してください。 JVM 全体でデータを同期する待ち時間はさらに長くなり、更新が失われる確率がより高くなります。 4 つ以上の JVM を使用して後書き構成を使用しているシステムでは、1 つの JVM で実行される更新は、約 15 秒遅れてから、その更新が他の JVM で使用可能になります。 任意の 2 つの JVM が同じエントリーを更新する場合は、最初に更新をフラッシュする 1 つの JVM はその更新を失います。

hibernate.cfg.xml

```
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">org.apache.derby.jdbc.EmbeddedDriver</property>
        <property name="connection.url">jdbc:derby:DB_testPU2;create=true</property>
        <!-- ObjectGrid cache setting-->
        <property name="cache.provider_class">com.ibm.websphere.objectgrid.hibernate.cache.
                ObjectGridHibernateCacheProvider</property>
        <property name="cache.use_query_cache">true</property>
        <property name="objectgrid.configuration">ObjectGridName=myOGName,
                ObjectGridType=EMBEDDED,MaxNumberOfReplicas=4 </property>
        <property name="objectgrid.hibernate.regionNames">queryRegion1, queryRegion2</property>

        <mapping resource="com/ibm/websphere/objectgrid/jpa/test/Employee.hbm.xml"/>

    </session-factory>
</hibernate-configuration>
```

ObjectGrid キャッシュに対するデータのプリロード

ObjectGridHibernateCacheProvider クラスの preload メソッドを使用して、特定のエンティティー・クラスの ObjectGrid キャッシュにデータをプリロードできます。

例 1

EntityManagerFactory の使用

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("testPU");
ObjectGridHibernateCacheProvider.preload("objectGridName", emf, TargetEntity.class, 100, 100);
```

例 2

SessionFactory の使用

```
org.hibernate.cfg.Configuration cfg = new Configuration();
// use addResource, addClass, and setProperty method of Configuration to prepare
// configuration required to create SessionFactor
SessionFactory sessionFactory= cfg.buildSessionFactory();
ObjectGridHibernateCacheProvider.preload("objectGridName", sessionFactory,
TargetEntity.class, 100, 100);
```

注:

1. 分散システムでは、このプリロード・メカニズムは、1 つの Java 仮想マシンからのみ呼び出すことができます。プリロード・メカニズムは、複数の Java 仮想マシン から同時に実行することはできません。
2. プリロードを実行する前に、eXtreme Scale キャッシュを初期化する必要があります。この初期化は、対応するすべての BackingMap が作成されるようにするため、EntityManagerFactory を使用して EntityManager を作成することによって行います。そうしないでプリロードを実行すると、1 つのみのデフォルト BackingMap がすべてのエンティティーをサポートするようにキャッシュが初期化されてしまいます。これは、単一の BackingMap がすべてのエンティティーで共用されることを示しています。

XML による Hibernate キャッシュ構成のカスタマイズ

ほとんどのシナリオでは、キャッシュ・プロパティーを設定すれば十分です。キャッシュによって使用される ObjectGrid をさらにカスタマイズするには、Hibernate ObjectGrid 構成 XML ファイルを、 `persistence.xml` ファイルと同様に `META-INF` ディレクトリーに提供することができます。初期化時に、キャッシュはこれらの XML ファイルを検索し、検出されるとそれらのファイルを処理します。

Hibernate ObjectGrid 構成 XML ファイルには、`hibernate-objectGrid.xml` (ObjectGrid 構成)、`hibernate-objectGridDeployment.xml` (デプロイメント・ポリシー)、および `hibernate-objectGrid-client-override.xml` (クライアント ObjectGrid オーバーライド構成) の 3 つのタイプがあります。構成された eXtreme Scale トポロジーに応じて、これらの 3 つの XML ファイルのいずれか任意のファイルを提供してそのトポロジーをカスタマイズすることができます。

EMBEDDED と EMBEDDED_PARTITION の両方のタイプの場合、3 つの XML ファイルの任意のいずれかを提供し、ObjectGrid、デプロイメント・ポリシー、およびクライアント ObjectGrid オーバーライド構成をカスタマイズできます。

REMOTE ObjectGrid の場合、キャッシュは動的 ObjectGrid を作成しません。キャッシュは、カタログ・サービスからクライアント・サイドの ObjectGrid を取得するだけです。`hibernate-objectGrid-client-override.xml` ファイルを提供して、クライアント ObjectGrid オーバーライド構成をカスタマイズできるだけです。

1. **ObjectGrid 構成**: `META-INF/hibernate-objectGrid.xml` ファイルを使用します。このファイルは、EMBEDDED および EMBEDDED_PARTITION タイプの両方に対して ObjectGrid 構成をカスタマイズする場合に使用されます。REMOTE タイプの場合、このファイルは無視されます。デフォルトでは、各エンティティー・クラスには、regionName が関連付けられ (エンティティー・クラス名にデフォルト設定)、その regionName が、ObjectGrid 構成内の regionName として名付けられた BackingMap 構成にマップされます。例えば、com.mycompany.Employee エンティティー・クラスには、

com.mycompany.Employee BackingMap とデフォルト設定される regionName が関連付けられます。デフォルト BackingMap 構成は、readOnly="false"、copyKey="false"、lockStrategy="NONE"、および copyMode="NO_COPY" です。一部の BackingMap を選択された構成でカスタマイズできます。予約キーワード「ALL_ENTITY_MAPS」を使用すれば、hibernate-objectGrid.xml ファイルにリストされた他のカスタマイズ・マップを除くすべてのマップを示すことができます。この hibernate-objectGrid.xml ファイルにリストされていない BackingMap は、デフォルト構成を使用します。

2. **ObjectGridDeployment 構成**: META-INF/hibernate-objectGridDeployment.xml ファイルを使用します。このファイルは、デプロイメント・ポリシーのカスタマイズに使用されます。デプロイメント・ポリシーをカスタマイズする場合、hibernate-objectGridDeployment.xml が提供されている場合は、デフォルトのデプロイメント・ポリシーは廃棄されます。すべてのデプロイメント・ポリシー属性値は、この提供された hibernate-objectGridDeployment.xml ファイルから得られます。
3. **クライアント・オーバーライド ObjectGrid 構成**: META-INF/hibernate-objectGrid-client-override.xml ファイルを使用します。このファイルは、クライアント・サイド ObjectGrid のカスタマイズに使用されます。デフォルトでは、ObjectGrid キャッシュは、ニア・キャッシュを使用不可にするデフォルト・クライアント・オーバーライド構成を適用します。アプリケーションがニア・キャッシュを必要とする場合、アプリケーションはこのファイルを提供し、numberOfBuckets="xxx" を指定することができます。デフォルトのクライアント・オーバーライドでは、numberOfBuckets="0" を設定し、ニア・キャッシュを使用不可にします。ニア・キャッシュは、hibernate-objectGrid-client-override.xml ファイルによって numberOfBuckets 属性の値をゼロより大きい値に再設定すれば、アクティブにすることができます。 hibernate-objectGrid-client-override.xml ファイルの機能は、 hibernate-objectGrid.xml に似ています。このファイルは、デフォルトのクライアント ObjectGrid オーバーライド構成をオーバーライドまたは拡張します。

Hibernate ObjectGrid XML ファイルの例

Hibernate ObjectGrid XML ファイルは、パーシスタンス・ユニットの構成に基づいて作成する必要があります。

パーシスタンス・ユニットの構成を表す persistence.xml ファイルの例を以下に示します。

persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0">
  <persistence-unit name="AnnuityGrid">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>

    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.FixedAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.EquityAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Person</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityHolder</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Address</class>

    <exclude-unlisted-classes>true</exclude-unlisted-classes>
```

```
    <properties>
      <property name="hibernate.show_sql" value="false" />
      <property name="hibernate.connection.url" value="jdbc:db2:Annuity" />
      <property name="hibernate.connection.driver_class" value="com.ibm.db2.jcc.DB2Driver" />
      <property name="hibernate.default_schema" value="EJB30" />

      <!-- Cache -->
      <property name="hibernate.cache.provider_class"
        value="com.ibm.websphere.objectgrid.hibernate.cache.ObjectGridHibernateCacheProvider" />
      <property name="hibernate.cache.use_query_cache" value="true" />
      <property name="objectgrid.configuration" value="ObjectGridType=EMBEDDED,
               ObjectGridName=Annuity, MaxNumberOfReplicas=4" />
    </properties>
  </persistence-unit>

</persistence>
```

以下に示すのは、この persistence.xml ファイルに適合する hibernate-objectGrid.xml ファイルです。

hibernate-objectGrid.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="Annuity">
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <backingMap name="defaultCacheMap" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="defaultCacheMap" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <backingMap name="org.hibernate.cache.UpdateTimestampsCache" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="org.hibernate.cache.UpdateTimestampsCache" />
      <backingMap name="org.hibernate.cache.StandardQueryCache" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="org.hibernate.cache.StandardQueryCache" />
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="defaultCacheMap">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="org.hibernate.cache.UpdateTimestampsCache">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
```

```
    </backingMapPluginCollection>
    <backingMapPluginCollection id="org.hibernate.cache.StandardQueryCache">
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

注: org.hibernate.cache.UpdateTimestampsCache、org.hibernate.cache.StandardQueryCache および defaultCacheMap マップが必要です。

以下に示すのは、この persistence.xml に適合する hibernate-objectGridDeployment.xml ファイルです。

hibernate-objectGridDeployment.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
  xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="Annuity">
    <mapSet name="MAPSET_Annuity" numberOfPartitions="1" numInitialContainers="1" minSyncReplicas="0"
          maxSyncReplicas="4" maxAsyncReplicas="0" replicaReadEnabled="true">
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <map ref="defaultCacheMap" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <map ref="org.hibernate.cache.UpdateTimestampsCache" />
      <map ref="org.hibernate.cache.StandardQueryCache" />
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

注: org.hibernate.cache.UpdateTimestampsCache、org.hibernate.cache.StandardQueryCache および defaultCacheMap マップが必要です。

REMOTE ObjectGrid タイプのキャッシュ用の外部システム

REMOTE ObjectGrid タイプでキャッシュを構成する場合、外部 eXtreme Scale システムをセットアップする必要があります。外部システムをセットアップするためには、persistence.xml ファイルに基づく ObjectGrid および ObjectGridDeployment 構成 XML ファイルの両方が必要になります。 Hibernate ObjectGrid XML ファイルの例のセクションに記載された Hibernate ObjectGrid および ObjectGridDeployment 構成 XML ファイルを使用しても外部 eXtreme Scale システムをセットアップできます。

外部 ObjectGrid システムには、カタログ・サービスおよび ObjectGrid サーバー・プロセスの両方があります。コンテナー・サーバーを始動する前に、カタログ・サービスを始動する必要があります。詳しくは、「*管理ガイド*」内のスタンドアロン eXtreme Scale サーバーの開始およびコンテナー・プロセスの開始に関する説明を参照してください。

トラブルシューティング

1. **CacheException: ObjectGrid サーバーを取得できません。**

   EMBEDDED または EMBEDDED_PARTITION ObjectGridType の場合、キャッシュは、eXtreme Scale ランタイムからサーバー・インスタンスを取得しようとします。Java Platform, Standard Edition 環境では、組み込みカタログ・サービスの eXtreme Scale サーバーが始動されます。組み込みカタログ・サービスでは、ポート 2809 を listen しようとします。このポートが別のプロセスによって使用

中の場合、このエラーが発生します。外部カタログ・サービス・エンドポイントが、例えば objectGridServer.properties ファイルにより指定されている場合、ホスト名またはポートの指定に誤りがあると、このエラーが発生します。

2. **CacheException: 構成済みの REMOTE ObjectGrid に対して REMOTE ObjectGrid を取得できません。objectGridName = [ObjectGridName]、PU 名 = [persistenceUnitName]**

   このエラーは、キャッシュが、指定されたカタログ・サービス・エンドポイントから ObjectGrid を取得できない場合に発生します。一般的には、このエラーは、ホスト名またはポートに誤りがあることが原因です。

3. **CacheException: 2 つの PU [persistenceUnitName_1, persistenceUnitName_2] を EMBEDDED ObjectGridType の同一の ObjectGridName [ObjectGridName] では構成できません。**

   多数のパーシスタンス・ユニットを持つ構成があり、これらのユニットのキャッシュが同じ ObjectGrid 名および EMBEDDED ObjectGridType で構成されている場合、この例外が発生します。これらのパーシスタンス・ユニット構成は、同じか、または異なる persistence.xml ファイルに入れることができます。ObjectGridType が EMBEDDED の場合、各パーシスタンス・ユニットについて ObjectGrid 名が固有であることを確認する必要があります。

4. **CacheException: REMOTE ObjectGrid [ObjectGridName] に必要な BackingMaps [mapName_1, mapName_2,...] が含まれていません。**

   REMOTE ObjectGrid タイプの場合に、取得されたクライアント・サイド ObjectGrid にパーシスタンス・ユニットのキャッシュをサポートする完全なエンティティー BackingMap がないと、この例外が発生します。例えば、パーシスタンス・ユニットの構成に 5 つのエンティティー・クラスがリストされているが、取得された ObjectGrid には 2 つの BackingMap しかない場合などです。取得された ObjectGrid に 10 の BackingMap があったとしても、この 10 の BackingMap に必要な 5 つのエンティティー BackingMap のいずれかが検出されないと、やはりこの例外が発生します。

OpenJPA キャッシュ・プラグイン構成

WebSphere eXtreme Scale では、OpenJPA に対して DataCache 実装および QueryCache 実装の両方を提供しています。OpenJPA ObjectGrid キャッシュ、あるいは省略して ObjectGrid キャッシュは、DataCache 実装および QueryCache 実装の両方に共通の用語です。

設定

eXtreme Scale キャッシュは、persistence.xml ファイルに openjpa.DataCache および openjpa.QueryCache 構成プロパティーを設定することにより、OpenJPA に対して使用可能または使用不可にします。このプロパティー設定の構文は以下のとおりです。

```
<property name="openjpa.DataCache"
          value="<object_grid_datacache_class(<property>=<value>,...)"/>
<property name="openjpa.QueryCache"
          value="<object_grid_querycache_class(<property>=<value>,...)"/>
```

DataCache、QueryCache のいずれも eXtreme Scale キャッシュ・プロパティーを利用して、パーシスタンス・ユニットで使用されるキャッシュを構成することができます。

eXtreme Scale キャッシュ設定に加え、openjpa.RemoteCommitProvider プロパティーを sjvm に設定する必要があります。

```
<property name="openjpa.RemoteCommitProvider" value="sjvm"/>
```

各エンティティー・クラスの @DataCache アノテーションで指定されたタイムアウト値は、BackingMap に渡され、そこに各エンティティーがキャッシュされます。ただし、@DataCache アノテーションで指定された名前値は、eXtreme Scale キャッシュで無視されます。完全修飾エンティティー・クラス名が、キャッシュ・マップ名になります。

OpenJPA StoreCache および QueryCache の pin および unpin メソッドはサポートされておらず、何の機能も果たしません。

ObjectGrid キャッシュ・クラスのプロパティー・リストに ObjectGridName プロパティー、ObjectGridType プロパティー、その他の単純なデプロイメント・ポリシー関連のプロパティーを指定すれば、キャッシュ構成をカスタマイズすることができます。以下に例を示します。

```
<property name="openjpa.DataCache"
          value="com.ibm.websphere.objectgrid.openjpa.ObjectGridDataCache(
          ObjectGridName=BasicTestObjectGrid,ObjectGridType=EMBEDDED,
          maxNumberOfReplicas=4)"/>
<property name="openjpa.QueryCache"
          value="com.ibm.websphere.objectgrid.openjpa.ObjectGridQueryCache()"/>
<property name="openjpa.RemoteCommitProvider" value="sjvm"/>
```

DataCache および QueryCache 構成は、互いに独立しています。いずれかの構成を使用可能にすることができます。ただし、両方の構成を使用可能にした場合、QueryCache 構成では、DataCache 構成と同じ構成が使用され、その構成プロパティーは廃棄されます。

XML による OpenJPA キャッシュ構成のカスタマイズ

ほとんどのシナリオでは、eXtreme Scale キャッシュ・プロパティーを設定すれば十分です。キャッシュによって使用される ObjectGrid をさらにカスタマイズするには、OpenJPA ObjectGrid 構成 XML ファイルを persistence.xml ファイルと同様に META-INF ディレクトリーに提供することができます。キャッシュの初期化時に、ObjectGrid キャッシュはこれらの XML ファイルを検索し、検出されるとそれらのファイルを処理しようとします。

OpenJPA ObjectGrid 構成 XML ファイルには、openjpa-objectGrid.xml (ObjectGrid 構成)、 openjpa-objectGridDeployment.xml (デプロイメント・ポリシー)、および openjpa-objectGrid-client-override.xml (クライアント ObjectGrid オーバーライド構成) の 3 つのタイプがあります。構成された ObjectGrid タイプに応じて、これらの 3 つの XML ファイルのいずれか任意のファイルを提供して ObjectGrid をカスタマイズすることができます。

EMBEDDED と EMBEDDED_PARTITION のタイプの場合、3 つの XML ファイルの任意のいずれかを提供し、ObjectGrid、デプロイメント・ポリシー、およびクライアント ObjectGrid オーバーライド構成をカスタマイズできます。

REMOTE ObjectGrid の場合、ObjectGrid キャッシュは動的 ObjectGrid を作成しません。代わりにキャッシュは、カタログ・サービスからクライアント・サイドの ObjectGrid を取得するだけです。openjpa-objectGrid-client-override.xml ファイルを提供して、クライアント ObjectGrid オーバーライド構成をカスタマイズできるだけです。

1. **ObjectGrid 構成**: META-INF/openjpa-objectGrid.xml ファイルを使用します。このファイルは、EMBEDDED および EMBEDDED_PARTITION タイプの両方に対して ObjectGrid 構成をカスタマイズする場合に使用されます。REMOTE タイプの場合、このファイルは無視されます。デフォルトでは、各エンティティー・クラスは、ObjectGrid 構成内のエンティティー・クラス名として名付けられた独自の BackingMap 構成にマップされます。例えば、com.mycompany.Employee エンティティー・クラスは、com.mycompany.Employee BackingMap にマップされます。デフォルト BackingMap 構成は、readOnly="false"、copyKey="false"、lockStrategy="NONE"、および copyMode="NO_COPY" です。一部の BackingMap を選択された構成でカスタマイズできます。ALL_ENTITY_MAPS 予約キーワードを使用すれば、openjpa-objectGrid.xml ファイルにリストされた他のカスタマイズ・マップを除くすべてのマップを示すことができます。この openjpa-objectGrid.xml ファイルにリストされていない BackingMap は、デフォルト構成を使用します。カスタマイズされた BackingMaps が BackingMaps 属性またはプロパティーを指定しておらず、それらの属性がデフォルト構成で指定されている場合は、デフォルト構成の属性値が適用されます。例えば、エンティティー・クラスが timeToLive=30 でアノテーションを付けられている場合、そのエンティティーのデフォルトの BackingMap 構成には timeToLive=30 が設定されます。カスタム openjpa-objectGrid.xml ファイルにもその BackingMap が含まれているが、timeToLive 値を指定していない場合、カスタマイズされた BackingMap には、デフォルトで、timeToLive=30 値が設定されます。openjpa-objectGrid.xml ファイルは、デフォルト構成をオーバーライドまたは拡張しようとします。
2. **ObjectGridDeployment 構成**: META-INF/openjpa-objectGridDeployment.xml ファイルを使用します。このファイルは、デプロイメント・ポリシーのカスタマイズに使用されます。デプロイメント・ポリシーをカスタマイズする場合、openjpa-objectGridDeployment.xml ファイルが提供されている場合は、デフォルトのデプロイメント・ポリシーは廃棄されます。デプロイメント・ポリシー属性値は、すべて openjpa-objectGridDeployment.xml ファイルから提供されます。
3. **クライアント・オーバーライド ObjectGrid 構成**: META-INF/openjpa-objectGrid-client-override.xml ファイルを使用します。このファイルは、クライアント・サイド ObjectGrid のカスタマイズに使用されます。デフォルトでは、ObjectGrid キャッシュは、ニア・キャッシュを使用不可にするデフォルト・クライアント・オーバーライド ObjectGrid 構成を適用します。アプリケーションがニア・キャッシュを必要とする場合、アプリケーションはこのファイルを提供し、numberOfBuckets="xxx" を指定することができます。デフォルトのクライアント・オーバーライドでは、numberOfBuckets="0" を設定し、ニア・キャッシュを使用不可にします。ニア・キャッシュは、openjpa-objectGrid-client-

override.xml ファイルによって numberOfBuckets の値をゼロより大きい値に再設定すれば、アクティブにすることができます。 openjpa-objectGrid-client-override.xml ファイルの機能は、 openjpa-objectGrid.xml ファイルに似ています。このファイルは、デフォルトのクライアント ObjectGrid オーバーライド構成をオーバーライドまたは拡張します。

OpenJPA ObjectGrid XML ファイルの例

OpenJPA ObjectGrid XML ファイルは、パーシスタンス・ユニットの構成に基づいて作成する必要があります。

パーシスタンス・ユニットの構成を表す persistence.xml ファイルの例を以下に示します。

persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="1.0">
  <persistence-unit name="AnnuityGrid">
    <provider>org.apache.openjpa.persistence.PersistenceProviderImpl</provider>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.FixedAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.EquityAnnuity</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Person</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityHolder</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact</class>
    <class>com.ibm.wssvt.acme.annuity.common.bean.jpa.Address</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>

    <properties>
    <!-- Database setting -->


    <!--  enable cache -->
      <property name="openjpa.DataCache"
        value="com.ibm.websphere.objectgrid.openjpa.ObjectGridDataCache(objectGridName=Annuity,
              objectGridType=EMBEDDED, maxNumberOfReplicas=4)" />
      <property name="openjpa.RemoteCommitProvider" value="sjvm" />
      <property name="openjpa.QueryCache"
                value="com.ibm.websphere.objectgrid.openjpa.ObjectGridQueryCache()" />
    </properties>
  </persistence-unit>
</persistence>
```

以下に示すのは、この persistence.xml ファイルに適合する openjpa-objectGrid.xml ファイルです。

openjpa-objectGrid.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?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="Annuity">
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject"
                  readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject" />
```

```
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <backingMap name="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" evictionTriggers="MEMORY_USAGE_THRESHOLD"
                  pluginCollectionRef="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <backingMap name="ObjectGridQueryCache" readOnly="false" copyKey="false"
                  lockStrategy="NONE" copyMode="NO_COPY" pluginCollectionRef="ObjectGridQueryCache"
                  evictionTriggers="MEMORY_USAGE_THRESHOLD"  />
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection
            id="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout">
      <bean id="ObjectTransformer"
            className="com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer" />
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
    <backingMapPluginCollection id="ObjectGridQueryCache">
      <bean id="MapIndexPlugin" className="com.ibm.websphere.objectgrid.plugins.index.HashIndex" >
          <property name="Name" type="java.lang.String"
                    value="QueryCacheKeyIndex" description="name of index"/>
          <property name="POJOKeyIndex" type="boolean" value="true" description="POJO Key Index "/>
      </bean>
      <bean id="Evictor" className="com.ibm.websphere.objectgrid.plugins.builtins.LRUEvictor" >
      </bean>
    </backingMapPluginCollection>
  </backingMapPluginCollections>
</objectGridConfig>
```

注:

1. 各エンティティーは、完全修飾エンティティー・クラス名として名付けられた BackingMap にマップされます。
2. エンティティー・クラスが、継承の階層にある場合、子クラスは親の BackingMap にマップされます。継承の階層は単一の BackingMap を共用します。
3. QueryCache のサポートには ObjectGridQueryCache マップが必要です。

4. 各エンティティー・マップの backingMapPluginCollection には、com.ibm.ws.objectgrid.openjpa.ObjectGridPCDataObjectTransformer クラスを使用する ObjectTransformer がなければなりません。
5. ObjectGridQueryCache マップの backingMapPluginCollection には、サンプルに示されているように QueryCacheKeyIndex と名付けられたキー索引がなければなりません。
6. 各マップで、Evictor はオプションです。

以下に示すのは、この persistence.xml ファイルに適合する openjpa-objectGridDeployment.xml ファイルです。

openjpa-objectGridDeployment.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
  xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
  <objectgridDeployment objectgridName="Annuity">
    <mapSet name="MAPSET_Annuity" numberOfPartitions="1" numInitialContainers="1"
            minSyncReplicas="0" maxSyncReplicas="4" maxAsyncReplicas="0"
            replicaReadEnabled="true">
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Annuity" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Address" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payor" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Person" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Contact" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.AnnuityPersistebleObject" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Rider" />
      <map ref="com.ibm.wssvt.acme.annuity.common.bean.jpa.Payout" />
      <map ref="ObjectGridQueryCache" />
    </mapSet>
  </objectgridDeployment>
</deploymentPolicy>
```

注: QueryCache のサポートには ObjectGridQueryCache マップが必要です。

REMOTE ObjectGrid タイプのキャッシュ用の外部システム

REMOTE ObjectGrid タイプでキャッシュを構成する場合、外部システムをセットアップする必要があります。外部システムをセットアップするためには、persistence.xml ファイルに基づく ObjectGrid および ObjectGridDeployment 構成 XML ファイルの両方が必要になります。OpenJPA ObjectGrid XML ファイルの例のセクションに記載された OpenJPA ObjectGrid および ObjectGridDeployment 構成 XML ファイルを使用しても外部 eXtreme Scale システムをセットアップできます。

外部 eXtreme Scale システムには、カタログ・サービスおよびコンテナー・サーバー・プロセスの両方があります。コンテナー・サーバーを始動する前に、カタログ・サーバーを始動する必要があります。

トラブルシューティング

1. **CacheException: ObjectGrid サーバーを取得できません。**

   EMBEDDED または EMBEDDED_PARTITION ObjectGridType の場合、eXtreme Scale キャッシュは、ランタイムからサーバー・インスタンスを取得しようとします。Java Platform, Standard Edition 環境では、組み込みカタログ・サービスを持つ eXtreme Scale サーバーが始動されます。組み込みカタログ・サービスでは、ポート 2809 を listen しようとします。このポートが別のプロセスによって使用中の場合、このエラーが発生します。外部カタログ・サービス・エンドポイ

ントが、例えば objectGridServer.properties ファイルにより指定されている場合、ホスト名またはポートの指定に誤りがあると、このエラーが発生します。

2. **CacheException: 構成済みの REMOTE ObjectGrid に対して REMOTE ObjectGrid を取得できません。objectGridName = [ObjectGridName]、PU 名 = [persistenceUnitName]**

   このエラーは、キャッシュが、指定されたカタログ・サービス・エンドポイントから ObjectGrid を取得できない場合に発生します。一般的には、このエラーは、ホスト名またはポートに誤りがあることが原因です。

3. **CacheException: 2 つの PU [persistenceUnitName_1, persistenceUnitName_2] を EMBEDDED ObjectGridType の同一の ObjectGridName [ObjectGridName] では構成できません。**

   多数のパーシスタンス・ユニット構成があり、これらのユニットの eXtreme Scale キャッシュが同じ ObjectGrid 名および EMBEDDED ObjectGridType で構成されている場合、この例外が発生します。これらのパーシスタンス・ユニット構成は、同じまたは異なる persistence.xml ファイルに入れることができます。ObjectGridType が EMBEDDED の場合、各パーシスタンス・ユニットについて ObjectGrid 名が固有であることを確認する必要があります。

4. **CacheException: REMOTE ObjectGrid [ObjectGridName] に必要な BackingMaps [mapName_1, mapName_2,...] が含まれていません。**

   REMOTE ObjectGrid タイプの場合に、取得されたクライアント・サイド ObjectGrid に、パーシスタンス・ユニットのキャッシュをサポートする完全なエンティティー BackingMap がないと、この例外が発生します。例えば、パーシスタンス・ユニット構成に 5 つのエンティティー・クラスがリストされているが、取得された ObjectGrid には 2 つの BackingMap しかない場合などです。取得された ObjectGrid に 10 の BackingMap があったとしても、この 10 の BackingMap に必要な 5 つのエンティティー BackingMap のいずれかが検出されないと、やはりこの例外が発生します。

注: OpenJPA eXtreme Scale キャッシュは、パフォーマンス強化のためにデータ・フォーマットを変更しました。eXtreme Scale を L2 キャッシュとして使用するよう構成された OpenJPA アプリケーションをホスティングするシステムは、WebSphere eXtreme Scale バージョン 7.0 へのマイグレーションの前に停止される必要があります。

HTTP セッション・マネージャーの構成

HTTP セッション・マネージャーは、関連するアプリケーションのセッション複製機能を提供します。 セッション複製マネージャーは、Web コンテナーのセッション・マネージャーと連動して HTTP セッションを作成し、そのアプリケーションに関連付けられた HTTP セッションのライフサイクルを管理します。

HTTP セッション・マネージャー構成のための XML ファイル

HTTP セッション・データを格納するコンテナー・サーバーを始動する際は、デフォルトの XML ファイルを使用することもできるし、特定の ObjectGrid 名、レプリカ数などを作成する、カスタマイズされた XML ファイルを指定することもできます。

サンプル・ファイルの場所

これらの XMLファイルは、スタンドアロン・インストール用として <*extremescale*>/ObjectGrid/session/samples に、または WebSphere Application Server セル内にインストールされる WebSphere eXtreme Scale 用として <*WAS_install_root*>/optionalLibraries/ObjectGrid/session/samples にパッケージされています。

組み込み XML パッケージ

組み込みシナリオを構成する場合、すなわちコンテナー・サーバーが Web コンテナー層で始動される場合は、objectGrid.xml ファイルと objectGridDeployment.xml ファイルがデフォルトで提供されます。これらのファイルを更新して HTTP セッション・マネージャーの振る舞いをカスタマイズ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="session">
   <bean id="ObjectGridEventListener" className="com.ibm.ws.xs.sessionmanager.SessionHandleManager"/>
   <backingMap name="objectgridSessionMetadata" pluginCollectionRef="objectgridSessionMetadata" readOnly="false"
     lockStrategy="PESSIMISTIC" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600" copyMode="NO_COPY"/>
   <backingMap name="objectgridSessionAttribute.*" template="true" readOnly="false" lockStrategy="PESSIMISTIC"
     ttlEvictorType="NONE" copyMode="NO_COPY"/>
       <backingMap name="objectgridSessionTTL.*" template="true" readOnly="false" lockStrategy="PESSIMISTIC"
     ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600" copyMode="NO_COPY"/>
  </objectGrid>
 </objectGrids>
    <backingMapPluginCollections>
        <backingMapPluginCollection id="objectgridSessionMetadata">
            <bean id="MapEventListener" className="com.ibm.ws.xs.sessionmanager.MetadataMapListener"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

変更可能な値:

- **ObjectGrid name 属性:** Web アプリケーションを接続するのに使用される `splicer.properties` ファイル内の objectGridName プロパティー、および `objectGridDeployment.xml` ファイル内の objectgridName 属性と同じである必要があります。複数のアプリケーションを持っていて、セッション・データを異なるグリッドに格納したい場合は、それらのアプリケーションが異なる ObjectGrid name 属性値を持つ必要があります。このファイル内で変更可能なのは、ObjectGrid の名前だけです。

変更不可の値:

- **ObjectGridEventListener:** ObjectGridEventListener 行は変更不可で、内部で使用されます。
- **objectgridSessionMetadata**: objectgridSessionMetadata 行は HTTP セッション・メタデータが格納されるマップを参照します。 このマップのグリッドに格納される HTTP セッションにはそれぞれ 1 つのエントリーがあります。
- **objectgridSessionAttribute.*** objectgridSessionAttribute.* 行は、Web アプリケーションの接続に使用する `splicer.properties` ファイル内で **fragmentedSession** パラメーターが `TRUE` に設定されている場合に、HTTP セッションの属性が格納されるマップを作製するのに使用する動的マップを定義します。この動的マップは objectgridSessionAttribute と呼ばれます。objectgridSessionAttributeEvicted と呼ばれるこのテンプレートをベースにしてもう 1 つのマップが作成されます。このマップには、タイムアウトになったけれど Web コンテナーは無効になっていないセッションが格納されます。
- **MetadataMapListener** 行は内部用で、変更はできません。

図 14. objectGrid.xml ファイル

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

 <objectgridDeployment objectgridName="session">
  <mapSet name="sessionMapSet" numberOfPartitions="5" minSyncReplicas="0" maxSyncReplicas="0"
    maxAsyncReplicas="1" developmentMode="false" placementStrategy="PER_CONTAINER">
             <map ref="objectgridSessionMetadata"/>
             <map ref="objectgridSessionAttribute.*"/>
             <map ref="objectgridSessionTTL.*"/>
    </mapSet>
   </objectgridDeployment>
</deploymentPolicy>
```

変更可能な値:

- **objectgridName 属性**: Web アプリケーションを接続するのに使用される `splicer.properties` ファイル内の objectGridName プロパティー、および `objectGrid.xml` ファイル内の ObjectGrid name 属性と同じ名前である必要があります。
- **mapSet エレメント属性**: placementStrategy 属性を除き、すべての mapSet プロパティーは変更可能です。

  Name　任意の値に更新できます。

  numberOfPartitions
  Web アプリケーションをホスティングしている各サーバーで開始されるプライマリー区画の数を指定します。区画を追加すると、フェイルオーバー時にデータがより散らばった状態になります。デフォルト値は 5 区画となっており、これはほとんどのアプリケーションで問題のない値です。

  minSyncReplicas、maxSyncReplicas、および maxAsyncReplicas
  HTTP セッション・データを格納するレプリカの数とタイプを指定します。デフォルトは 1 つの非同期レプリカとなっており、これはほとんどのアプリケーションで問題のない値です。同期複製は要求パス中に行われますが、これによってご使用の Web アプリケーションの応答時間が長くなる場合があります。

  developmentMode
  区画のレプリカ断片をそのプライマリー断片と同じノードに配置することができるかどうかを、eXtreme Scale 配置サービスに通知します。開発環境ではこの値を TRUE に設定できますが、ノード障害がセッション・データの損失を引き起こす場合があるため、実稼働環境ではこの機能を使用不可に設定してください。

  placementStrategy
  この属性の値は変更しないでください。
- ファイルの残りの部分は objectGrid.xml 内と同じマップ名を参照します。 これらの名前は変更できません。

変更不可の値:

- mapSet エレメントの placementStrategy 属性

図 15. objectGridDeployment.xml ファイル

リモート XML パッケージ

コンテナーがスタンドアロン・プロセスとして実行されるリモート・モードを使用する場合、`objectGridStandAlone.xml` ファイルおよび `objectGridDeploymentStandAlone.xml` ファイルを使用してプロセスを開始する必要があります。これらのファイルを更新することで構成を変更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="session">
   <bean id="ObjectGridEventListener" className="com.ibm.ws.xs.sessionmanager.SessionHandleManager"/>
           <backingMap name="objectgridSessionMetadata" pluginCollectionRef="objectgridSessionMetadata"
       readOnly="false" lockStrategy="PESSIMISTIC" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600"
       copyMode="COPY_TO_BYTES"/>
           <backingMap name="objectgridSessionAttribute.*" template="true" readOnly="false" lockStrategy="PESSIMISTIC"
       ttlEvictorType="NONE" copyMode="COPY_TO_BYTES"/>
           <backingMap name="objectgridSessionTTL.*" template="true" readOnly="false" lockStrategy="PESSIMISTIC"
       ttlEvictorType="LAST_ACCESS_TIME" timeToLive="3600" copyMode="COPY_TO_BYTES"/>
  </objectGrid>
 </objectGrids>
    <backingMapPluginCollections>
        <backingMapPluginCollection id="objectgridSessionMetadata">
            <bean id="MapEventListener" className="com.ibm.ws.xs.sessionmanager.MetadataMapListener"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

変更可能な値:

- **objectgridName attribute**: 変更できます。ただし、Web アプリケーションを接続するのに使用される `splicer.properties` ファイル内の objectGridName プロパティー、および `objectGridDeploymentStandAlone.xml` ファイル内の objectgridName 属性と同じである必要があります。 複数のアプリケーションを持っていて、セッション・データを異なるグリッドに格納したい場合は、それらのアプリケーションが異なるグリッド名を持つ必要があります。このファイル内で変更可能なのは、グリッドの名前だけです。

変更不可の値:

- **ObjectGridEventListener:** ObjectGridEventListener 行は変更不可で、内部で使用されます。
- **objectgridSessionMetadata**: objectgridSessionMetadata 行は HTTP セッション・メタデータが格納されるマップを参照します。 このマップのグリッドに格納される HTTP セッションにはそれぞれ 1 つのエントリーがあります。
- **objectgridSessionAttribute.*** objectgridSessionAttribute.* 行は、Web アプリケーションの接続に使用する `splicer.properties` ファイル内で **fragmentedSession** パラメーターが `true` に設定されている場合に、HTTP セッションの属性が格納されるマップを作製するのに使用する動的マップを定義します。この動的マップは objectgridSessionAttribute と呼ばれます。objectgridSessionAttributeEvicted と呼ばれるこのテンプレートをベースにしてもう 1 つのマップが作成されます。このマップには、タイムアウトになったけれど Web コンテナーは無効になっていないセッションが格納されます。
- **MetadataMapListener** 行は内部用で、変更はできません。

図 16. objectGridStandAlone.xml

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

 <objectgridDeployment objectgridName="session">
  <mapSet name="sessionMapSet" numberOfPartitions="5" minSyncReplicas="0" maxSyncReplicas="0"
   maxAsyncReplicas="1" developmentMode="false" placementStrategy="PER_CONTAINER">
            <map ref="objectgridSessionMetadata"/>
            <map ref="objectgridSessionAttribute.*"/>
            <map ref="objectgridSessionTTL.*"/>
     </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

変更可能な値:

- **objectgridName 属性**: Web アプリケーションを接続するのに使用される `splicer.properties` ファイル内の objectGridName プロパティー、および `objectGrid.xml` ファイル内の ObjectGrid name 属性と同じ名前である必要があります。
- **mapSet エレメント属性**: placementStrategy 属性を除き、すべての mapSet プロパティーは変更可能です。

  Name 任意の値に更新できます。

  numberOfPartitions
  Web アプリケーションをホスティングしている各サーバーで開始されるプライマリー区画の数を指定します。区画を追加すると、フェイルオーバー時にデータがより散らばった状態になります。デフォルト値は 5 区画となっており、これはほとんどのアプリケーションで問題のない値です。

  minSyncReplicas、maxSyncReplicas、および maxAsyncReplicas
  HTTP セッション・データを格納するレプリカの数とタイプを指定します。デフォルトは 1 つの非同期レプリカとなっており、これはほとんどのアプリケーションで問題のない値です。同期複製は要求パス中に行われますが、これによってご使用の Web アプリケーションの応答時間が長くなる場合があります。

  developmentMode
  区画のレプリカ断片をそのプライマリー断片と同じノードに配置することができるかどうかを、eXtreme Scale 配置サービスに通知します。開発環境ではこの値を TRUE に設定できますが、ノード障害がセッション・データの損失を引き起こす場合があるため、実稼働環境ではこの機能を使用不可に設定してください。

  placementStrategy
  この属性の値は変更しないでください。
- ファイルの残りの部分は `objectGrid.xml` ファイル内と同じマップ名を参照します。 これらの名前は変更できません。

変更不可の値:

- mapSet エレメントの placementStrategy 属性

図 17. objectGridDeploymentStandAlone.xml:

WebSphere Application Server での HTTP セッション・マネージャーの構成

WebSphere Application Server はセッション管理機能を備えていますが、このサポートは極度の要求ロード下では拡張しません。 WebSphere eXtreme Scale には、セッション複製、高可用性、優れたスケーラビリティー、および堅固な構成オプションを備えたセッション管理実装がバンドルされています。

始める前に

- eXtreme Scale セッション・マネージャーを使用するには、 WebSphere eXtreme Scale は、WebSphere Application Server または WebSphere Application Server Network Deployment セルにインストールする必要があります。詳しくは、 24 ページの『WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントの WebSphere Application Server との統合』を参照してください。

このタスクについて

WebSphere eXtreme Scale HTTP セッション・マネージャーは、キャッシング用に、組み込みサーバーとリモート・サーバーの両方をサポートします。

組み込みシナリオ

組み込みシナリオでは、サーブレットが実行される同じプロセス内で WebSphere eXtreme Scale サーバーが相互に連結されています。セッション・マネージャーはローカルの ObjectGrid インスタンスと直接通信できるため、コストのかかるネットワーク遅延を回避することができます。

WebSphere Application Server を使用している場合は、提供された `objectgridRoot/session/samples/objectGrid.xml` および `objectgridRoot/session/samples/objectGridDeployment.xml` ファイルを、ご使用の Web アーカイブ (WAR) ファイルの `META-INF` ディレクトリーに配置してください。アプリケーションが始動して、セッション・マネージャーと同じプロセス内の eXtreme Scale コンテナーを自動的に始動すると、eXtreme Scale がこれらのファイルを自動的に検出します。

使用する複製が同期複製か非同期複製かということ、および構成するレプリカの数に応じて、`objectGridDeployment.xml` ファイルを変更できます。

リモート・サーバー・シナリオ

リモート・サーバー・シナリオでは、サーブレットとは異なるプロセスでコンテナー・サーバーが実行されます。セッション・マネージャーはリモートのコンテナー・サーバーと通信します。リモートのネットワーク接続のコンテナー・サーバーを使用するためには、カタログ・サービス・ドメインのホスト名およびポート番号によってセッション・マネージャーを構成する必要があります。そうすると、セッション・マネージャーは、eXtreme Scale クライアント接続を使用して、カタログ・サーバーおよびコンテナー・サーバーと通信します。

コンテナー・サーバーを独立したスタンドアロン・プロセスで始動する場合は、セッション・マネージャーのサンプル・ディレクトリーで提供される `objectGridStandAlone.xml` および `objectGridDeploymentStandAlone.xml` ファイルを使用して、 eXtreme Scale コンテナーを始動します。

手順

1. アプリケーションを接合することで、アプリケーションがセッション・マネージャーを使用できるようにします。 セッション・マネージャーを使用するためには、適切なフィルター宣言をアプリケーションの Web デプロイメント記述子に追加する必要があります。さらに、セッション・マネージャー構成パラメーター

が、デプロイメント記述子内のサーブレット・コンテキスト初期化パラメーターという形式でセッション・マネージャーに渡されます。この情報は、以下に示す複数の方法でアプリケーションに導入することができます。

- 7.1+ **WebSphere Application Server 管理コンソールを使用する方法**

  アプリケーションのインストール時に、WebSphere eXtreme Scale HTTP セッション・マネージャーを使用するようにアプリケーションを構成できます。また、アプリケーションまたはサーバーの構成を編集して、WebSphere eXtreme Scale HTTP セッション・マネージャーを使用することもできます。詳しくは、 278 ページの『データ・グリッド (data grid)へのセッション・パーシスタンスの作成』を参照してください。

  注: このオプションは、アプリケーションを実行しているすべてのノードの同じパスに `splicer.properties` ファイルが存在する場合にのみ機能します。Windows ノードと UNIX ノードがともに存在する混合環境では、このオプションは使用できないため、アプリケーションを手動で接合する必要があります。

- **自動接合オプション**

  WebSphere Application Server または WebSphere Application Server Network Deployment でアプリケーションを実行している場合には、アプリケーションを手動で接合する必要はありません。そのスコープのすべての Web アプリケーションに影響を与えるセルまたはサーバーにカスタム・プロパティーを追加できます。プロパティー名は `com.ibm.websphere.xs.sessionFilterProps` であり、アプリケーションで必要な `splicer.properties` ファイルの場所に値を設定する必要があります。

  ファイルの場所のパスは、例えば `/opt/splicer.properties` などです。

  – **セル内のすべての Web アプリケーションを接合するように指定するには、管理コンソールを使用して以下のようにします。**

    「システム管理」 → **「セル」** → **「カスタム・プロパティー」**に移動し、プロパティーを追加します。

  – **特定のアプリケーション・サーバー内のすべての Web アプリケーションを接合するように指定するには、管理コンソールを使用して以下のようにします。**

    「アプリケーション・サーバー」 → **<「*server_name*」>** → **「管理」** → **「カスタム・プロパティー」**に移動して、プロパティーを追加します。

  使用可能なスコープはセルおよびサーバーのスコープのみであり、デプロイメント・マネージャーで実行している場合にのみ使用可能です。別のスコープが必要な場合は、Web アプリケーションを手動で接合する必要があります。

  注: 自動接合オプションは、アプリケーションを実行しているすべてのノードの同じパスに `splicer.properties` ファイルが存在する場合にのみ機能します。 Windows ノードと UNIX ノードがともに存在する混合環境では、このオプションは使用できないため、アプリケーションを手動で接合する必要があります。

- **addObjectGridFilter スクリプト**

  eXtreme Scale とともに提供されるコマンド行スクリプトを使用して、フィルター宣言と構成によってアプリケーションをサーブレット・コンテキスト初期化パラメーターの形式で接合します。このスクリプト (objectgridRoot/bin/addObjectGridFilter.sh または objectgridRoot/session/bin/addObjectGridFilter.bat) は、接合する必要があるアプリケーション (エンタープライズ・アーカイブ・ファイルへの絶対パス) および各種構成プロパティーを含むプロパティー・ファイルへの絶対パスという 2 つのパラメーターを取ります。このスクリプトの使用形式は以下の通りです。

  Windows

  ```
  addObjectGridFilter.bat [EAR ファイルのロケーション]
  [プロパティー・ファイルのロケーション]
  ```

  UNIX

  ```
  addObjectGridFilter.sh [EAR ファイルのロケーション]
  [プロパティー・ファイルのロケーション]
  ```

  UNIX **Unix 上の WebSphere Application Server にインストールされている eXtreme Scale の使用例:**

  a. `cd objectgridRoot/optionalLibraries/ObjectGrid/session/bin`

  b. `addObjectGridFilter.sh /tmp/mySessionTest.ear wasRoot/optionalLibraries/ObjectGrid/session/samples/splicer.properties`

  UNIX **Unix 上のスタンドアロン・ディレクトリーにインストールされている eXtreme Scale の使用例:**

  a. `cd objectgridRoot/session/bin`

  b. `addObjectGridFilter.sh /tmp/mySessionTest.ear objectgridRoot/session/samples/splicer.properties`

  接合されるサーブレット・フィルターは構成値のデフォルトを保持します。これらのデフォルト値は、2 番目の引数にあるプロパティー・ファイルで指定する構成オプションでオーバーライドできます。使用できるパラメーターのリストについては、286 ページの『サーブレット・コンテキスト初期化パラメーター』を参照してください。

  eXtreme Scale インストールとともに提供されるサンプルの splicer.properties ファイルを変更して使用することができます。また、各サーブレットを拡張することによってセッション・マネージャーを挿入する、addObjectGridServlets スクリプトも使用できます。ただし、推奨スクリプトは addObjectGridFilter スクリプトです。

- **Ant ビルド・スクリプト**

  WebSphere eXtreme Scale には Apache Ant で使用できる build.xml ファイルが同梱されています。このファイルは WebSphere Application Server インストールの wasRoot/bin フォルダーに含まれています。 build.xml を修正してセッション・マネージャー構成プロパティーを変更することができます。構成

プロパティーは splicer.properties ファイル内のプロパティー名と同一です。 build.xml の変更後に、以下を実行して Ant プロセスを呼び出します。

– UNIX ant.sh、ws_ant.sh

– Windows ant.bat、ws_ant.bat

(UNIX) または (Windows)

- **手動による Web 記述子の更新**

  Web アプリケーションに同梱されている web.xml ファイルを編集して、フィルター宣言、そのサーブレット・マッピング、およびサーブレット・コンテキスト初期化パラメーターが組み込まれるようにします。この方法はエラーを起こしやすいため、使用しないようにしてください。

使用できるパラメーターのリストについては、 286 ページの『サーブレット・コンテキスト初期化パラメーター』を参照してください。

2. アプリケーションをデプロイします。 サーバーやクラスターに対して通常使用する手順に従ってアプリケーションをデプロイしてください。アプリケーションをデプロイした後、アプリケーションを始動することができます。
3. アプリケーションにアクセスします。 これで、セッション・マネージャーおよび WebSphere eXtreme Scale と対話するアプリケーションにアクセスすることができます。

次のタスク

アプリケーションの装備時にセッション・マネージャーの構成属性の大多数を変更して、セッション・マネージャーを使用するようにすることができます。これらの属性には、同期または非同期の複製、セッション ID の長さ、メモリー内セッション・テーブル・サイズなどがあります。アプリケーションの装備時に変更できる属性を別にすれば、アプリケーションのデプロイメント後に変更できるその他の構成属性は、WebSphere eXtreme Scale サーバー・クラスター・トポロジーと、それらのクラスターのクライアント (セッション・マネージャー) がそれらのクラスターに接続する方法に関係する属性のみです。

関連タスク

284 ページの『各種アプリケーション・サーバー用の HTTP セッション・マネージャーの構成』

WebSphere eXtreme Scale には、Web コンテナーのデフォルト・セッション・マネージャーをオーバーライドし、セッション複製、高可用性、優れたスケーラビリティー、および堅固な構成オプションを備えたセッション管理実装がバンドルされています。 eXtreme Scale セッション複製マネージャーおよび汎用組み込み ObjectGrid コンテナーの開始を有効にします。

データ・グリッド (data grid)へのセッション・パーシスタンスの作成:

データ・グリッドへのセッションをパーシストするように WebSphere Application Server アプリケーションを構成できます。このデータ・グリッドは、WebSphere Application Server 内で実行される組み込みコンテナー・サーバー内またはリモート・データ・グリッド内に配置できます。

始める前に

WebSphere Application Server で構成を変更する前に、以下の情報が必要です。

- 使用するセッション・データ・グリッドの名前。セッション・データ・グリッドの作成については、 274 ページの『WebSphere Application Server での HTTP セッション・マネージャーの構成』を参照してください。
- セッションを管理するために使用するカタログ・サービスが、セッション・アプリケーションをインストールするセルの外側にある場合には、カタログ・サービス・ドメインを作成する必要があります。詳しくは、 371 ページの『WebSphere Application Server でのカタログ・サービス・ドメインの作成』を参照してください。

手順

- **アプリケーションのインストール時にセッション管理を構成するには、以下の手順を完了します。**
  1. WebSphere Application Server の管理コンソールで、「**アプリケーション**」 → 「**新規アプリケーション**」 → 「**新規エンタープライズ・アプリケーション**」をクリックします。「**詳細**」パスを選択してアプリケーションを作成し、初期のウィザード・ステップを完了します。
  2. ウィザードの「**eXtreme Scale セッション管理設定**」ステップで、使用するデータ・グリッドを構成します。 「**リモート eXtreme Scale データ・グリッド**」または「**組み込み eXtreme Scale データ・グリッド**」のいずれかを選択します。
     - 「**リモート eXtreme Scale データ・グリッド**」オプションの場合、セッション・データ・グリッドを管理するカタログ・サービス・ドメインを選択して、アクティブ・セッション・データ・グリッドのリストからデータ・グリッドを選択します。
     - 「**組み込み eXtreme Scale データ・グリッド**」オプションの場合、デフォルト ObjectGrid 構成を選択するか、ObjectGrid 構成ファイルの特定の場所を指定します。
  3. ウィザードのステップを完了して、アプリケーションのインストールを終了します。

  wsadmin スクリプトを付加してアプリケーションをインストールすることもできます。次の例では、**-SessionManagement** パラメーターにより、管理コンソールで作成するものと同じ構成を作成できます。

リモート eXtreme Scale データ・グリッド構成の場合:

```
AdminApp.install('C:/A.ear', '[  -nopreCompileJSPs -distributeApp
-nouseMetaDataFromBinary -nodeployejb -appname A -edition 8.0
-createMBeansForResources -noreloadEnabled -nodeployws -validateinstall
off -noprocessEmbeddedConfig -filepermission .*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
-buildVersion Unknown -noallowDispatchRemoteInclude -noallowServiceRemoteInclude
-asyncRequestDispatchType DISABLED -nouseAutoLink -SessionManagement [[true
XSRemoteSessionManagement cs0:!:grid0]]
-MapWebModToVH [[MicroWebApp microwebapp.war,WEB-INF/web.xml default_host] [MicroSipApp
microsipapp.war,WEB-INF/web.xml default_host] [MicroDG1App microdg1app.war,WEB-INF/web.xml
default_host] [MicroDG2App microdg2app.war,WEB-INF/web.xml default_host] [MicroSip2App
microsip2app.war,WEB-INF/web.xml default_host]]]')
```

デフォルト構成を使用した eXtreme Scale 組み込みシナリオの場合:

```
AdminApp.install('C:/A.ear', '[  -nopreCompileJSPs -distributeApp
-nouseMetaDataFromBinary -nodeployejb -appname A -edition 8.0
-createMBeansForResources -noreloadEnabled -nodeployws -validateinstall
off -noprocessEmbeddedConfig -filepermission .*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
-buildVersion Unknown -noallowDispatchRemoteInclude -noallowServiceRemoteInclude
-asyncRequestDispatchType DISABLED -nouseAutoLink -SessionManagement [[true
XSRemoteSessionManagement  :!: :!:default]]  -MapWebModToVH [[MicroWebApp microwebapp.war,
WEB-INF/web.xml default_host] [MicroSipApp
microsipapp.war,WEB-INF/web.xml default_host] [MicroDG1App microdg1app.war,WEB-INF/web.xml
default_host] [MicroDG2App microdg2app.war,WEB-INF/web.xml default_host] [MicroSip2App
microsip2app.war,WEB-INF/web.xml default_host]]]')
```

カスタム構成を使用した eXtreme Scale 組み込みシナリオの場合:

```
AdminApp.install('C:/A.ear', '[  -nopreCompileJSPs -distributeApp
-nouseMetaDataFromBinary -nodeployejb -appname A -edition 8.0
-createMBeansForResources -noreloadEnabled -nodeployws -validateinstall
off -noprocessEmbeddedConfig -filepermission .*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
-buildVersion Unknown -noallowDispatchRemoteInclude -noallowServiceRemoteInclude
-asyncRequestDispatchType DISABLED -nouseAutoLink -SessionManagement [[true
XSRemoteSessionManagement  :!: :!:custom:!:c:\XS\objectgrid.xml:!:c:\XS\objectgriddeployment.xml]]
-MapWebModToVH [[MicroWebApp microwebapp.war,WEB-INF/web.xml default_host] [MicroSipApp
microsipapp.war,WEB-INF/web.xml default_host] [MicroDG1App microdg1app.war,WEB-INF/web.xml
default_host] [MicroDG2App microdg2app.war,WEB-INF/web.xml default_host] [MicroSip2App
microsip2app.war,WEB-INF/web.xml default_host]]]')
```

- **WebSphere Application Server の管理コンソールで既存のアプリケーション上にセッション管理を構成する場合は以下を行います。**

  1. WebSphere Application Server の管理コンソールで、「**アプリケーション**」 → 「**アプリケーション・タイプ**」 → 「**WebSphere エンタープライズ・アプリケーション**」 → 「*application_name*」 → 「**Web モジュール・プロパティー**」 → 「**セッション管理**」 → 「**セッション管理設定**」をクリックします。
  2. フィールドを更新して、データ・グリッドへのセッション・パーシスタンスを使用可能にします。

  wsadmin スクリプトを使用して、アプリケーションを更新することもできます。次の例では、**-SessionManagement** パラメーターにより、管理コンソールで作成するものと同じ構成を作成できます。

  リモート eXtreme Scale データ・グリッド構成の場合:

  ```
  AdminApp.edit('DefaultApplication','[-SessionManagement[[[true
  XSRemoteSessionManagement cs0:!:grid0]]]')
  ```

  デフォルト構成を使用した eXtreme Scale 組み込みシナリオの場合:

  ```
  AdminApp.edit('DefaultApplication','[-SessionManagement[[[true
  XSEmbeddedSessionManagement :!: :!:default]]]')
  ```

  カスタム構成を使用した eXtreme Scale 組み込みシナリオの場合:

  ```
  AdminApp.edit('DefaultApplication','[-SessionManagement[[[true
  XSEmbeddedSessionManagement :!: :!:
  custom:!:c:\XS\objectgrid.xml:!:c:\XS\objectgriddeployment.xml]]]')
  ```

  変更を保存する場合、アプリケーションはアプライアンス上のセッション・パーシスタンスに構成済みのデータ・グリッド (data grid)を使用します。

- **既存のサーバー上でセッション管理を構成するには、以下を行います。**

1. WebSphere Application Server の管理コンソールで、「**サーバー**」 → 「**サーバー・タイプ**」 → 「**WebSphere アプリケーション・サーバー**」 → 「*server_name*」 → 「**コンテナー設定**」 → 「**eXtreme Scale セッション管理設定**」をクリックします。
2. フィールドを更新して、セッション・パーシスタンスを使用可能にします。

また、以下の wsadmin ツール・コマンドで、既存のサーバーでのセッション管理を構成できます。

リモート eXtreme Scale データ・グリッド構成の場合:

```
AdminTask.configureServerSessionManagement('[-nodeName IBM-C77EE220EB6Node01 -serverName server1
-enableSessionManagement true -sessionManagementType XSRemoteSessionManagement -XSRemoteSessionManagement
[-catalogService cs0 -csGridName grid0]]')
```

eXtreme Scale 組み込み構成の場合:

– デフォルト構成 (デフォルト XML ファイルを使用する場合):

```
AdminTask.configureServerSessionManagement('[-nodeName IBM-C77EE220EB6Node01 -serverName server1
-enableSessionManagement true -sessionManagementType XSEmbeddedSessionManagement
-XSEmbeddedSessionManagement [-embeddedGridType default -objectGridXML -objectGridDeploymentXML ]]')
```

– カスタム構成 (カスタマイズした XML ファイルを使用する場合):

```
AdminTask.configureServerSessionManagement('[-nodeName IBM-C77EE220EB6Node01 -serverName server1
-enableSessionManagement true -sessionManagementType XSEmbeddedSessionManagement
-XSEmbeddedSessionManagement
[-embeddedGridType custom -objectGridXML c:¥XS¥objectgrid.xml -objectGridDeploymentXML
c:¥XS¥objectgriddeployment.xml]]')
```

変更を保存すると、今後サーバーはサーバー上で稼働するすべてのアプリケーションでセッション・パーシスタンスの構成済み データ・グリッド (data grid)を使用します。

- HTTP セッション構成の他の局面を編集する場合は、splicer.properties ファイルを編集します。
  1. WebSphere Application Server 管理コンソールで、「**システム管理**」 → 「**セル**」 → 「**カスタム・プロパティー**」をクリックします。
  2. <*app_name*>,com.ibm.websphere.xs.sessionFilterProps カスタム・プロパティーを編集します。 このカスタム・プロパティーの値には、編集する splicer.properties ファイルのロケーションが指定されています。 このファイルは、デプロイメント・マネージャーに存在します。
  3. デプロイメント・マネージャー・プロファイル上のカスタム・プロパティーへのパス内に存在する splicer.properties ファイルを編集します。
  4. 更新された splicer.properties ファイルが複数のノードに伝搬するように、ノードを同期化します。 「**システム管理**」 → 「**ノード**」をクリックします。アプリケーションがインストールされている複数のノードを選択し、「**同期化**」をクリックします。

タスクの結果

データ・グリッド (data grid)へのセッションをパーシストするように HTTP セッション・マネージャーが構成されました。

splicer.properties ファイル:

splicer.properties ファイルには、サーブレット・フィルター・ベース ObjectGrid セッション・マネージャーのすべての構成オプションが含まれます。

サンプル・スプライサー・プロパティー

```
# サーブレット・フィルター・ベース ObjectGrid
# セッション・マネージャーが使用するように構成できる、
# すべての構成オプションが含まれたプロパティー・ファイル。
#
# このプロパティー・ファイルで、これらの構成設定に割り当てるすべての
# デフォルト値を保持できます。また、このプロパティー・ファイルを
# filtersplicer ANT タスクと組み合わせて使用する場合には、ANT タスク・
# プロパティーを使用して、個々の設定をオーバーライドできます。

# ストリング値「REMOTE」または「EMBEDDED」。デフォルトは REMOTE です。

# これにより、WebSphere、WebLogic、JBoss、Tomcat などの
# アプリケーション・サーバー・プロセス内の組み込み WebSphere eXtreme
# Scale コンテナーを始動するのかどうかを指定します。

objectGridType= REMOTE

# 特定の Web アプリケーションで使用する Object Grid インスタンスの
# 名前を定義するストリング値。デフォルト名は、sessionmanager です。
# このパラメーターを設定すると、その値がオーバーライドされます。

# objectGridName =

# カタログ・サーバーに接続して、クライアント・サイドの ObjectGrid
# インスタンスを取得できます。値の形式は、「host:port<,host:port>」
# にする必要があります。

# このリストは任意の長さにすることができ、ブートストラッピングにのみ使用
# されます。最初の実行可能なアドレスが使用されます。catalog.services.cluster
# が構成されている場合は、WebSphere 内でオプションです。

catalogHostPort = host:port<,host:port>

# 更新されたセッションを Object Grid に書き込む時間間隔を秒数で定義
# する整数値。デフォルトは 2 秒です。

replicationInterval = 1

# メモリー内に保持するセッション参照数を定義する整数値。
# デフォルトは 2000 です。

sessionTableSize =

# ストリング値「true」または「false」。デフォルトは false です。
# これは、セッション・データを項目全体として保管するのか、
# 各属性を別個に保管するのかを制御します。

fragmentedSession = false

# (Cookie ではなく) URL エンコードの使用を考慮するかどうかを
# セッション・マネージャーに指示します。設定しない場合のデフォルトは、
# false です。

useURLEncoding = false

# objectgrid XML ファイルのファイルの場所を示すストリング。
# これが指定されない場合、および objectGridType=EMBEDDED の場合、
# 組み込み XML ファイルが自動的にロードされます。

objectGridXML =

# objectGrid デプロイメント・ポリシー XML ファイルのファイルの場所を示すストリング。
```

```
# これが指定されない場合、および objectGridType=EMBEDDED の場合、
# 組み込み XML ファイルが自動的にロードされます。

objectGridDeploymentXML =

# IBM WebShere トレース仕様を示すストリング。
# WebLogic などの他のすべてのアプリケーション・サーバーで有用です。

traceSpec=

# トレース・ファイルの場所を示すストリング。
# WebLogic などの他のすべてのアプリケーション・サーバーで有用です。

traceFile=
```

関連概念

HTTP セッション管理

関連タスク

284 ページの『各種アプリケーション・サーバー用の HTTP セッション・マネージャーの構成』

WebSphere eXtreme Scale には、Web コンテナーのデフォルト・セッション・マネージャーをオーバーライドし、セッション複製、高可用性、優れたスケーラビリティー、および堅固な構成オプションを備えたセッション管理実装がバンドルされています。 eXtreme Scale セッション複製マネージャーおよび汎用組み込み ObjectGrid コンテナーの開始を有効にします。

WebSphere eXtreme Scale を使用した SIP セッション管理:

Session Initiation Protocol (SIP) セッション複製用のデータ複製サービス (DRS) の代わりに、WebSphere eXtreme Scale を、信頼できる SIP 複製メカニズムとして使用できます。

SIP セッション管理の構成

WebSphere eXtreme Scale を SIP 複製メカニズムとして使用するには、com.ibm.sip.ha.replicator.type カスタム・プロパティーを設定します。このカスタム・プロパティーを追加する各サーバーごとに、管理コンソールで、「**アプリケーション・サーバー**」 → ***my_application_server*** → 「**SIP コンテナー**」 → 「**カスタム・プロパティー**」を選択します。「名前」には `com.ibm.sip.ha.replicator.type` と入力し、「値」には `OBJECTGRID` と入力します。

以下のプロパティーを使用して、SIP セッションの保管に使用する ObjectGrid の振る舞いをカスタマイズします。このカスタム・プロパティーを追加する各サーバーごとに、管理コンソールで、「**アプリケーション・サーバー**」 → ***my_application_server*** → 「**SIP コンテナー**」 → 「**カスタム・プロパティー**」をクリックします。「**名前**」および「**値**」を入力します。各サーバーは、機能のプロパティーに設定されているものと同じプロパティーを所有する必要があります。

表 10. ObjectGrid を使用した SIP セッション管理のためのカスタム・プロパティー

property	値	デフォルト
com.ibm.sip.ha.replicator.type	`OBJECTGRID`: SIP セッション・ストアとして ObjectGrid を使用	
min.synchronous.replicas	同期複製の最小数	0
max.synchronous.replicas	同期複製の最大数	0

表 10. *ObjectGrid* を使用した *SIP* セッション管理のためのカスタム・プロパティー *(続き)*

property	値	デフォルト
max.asynchronous.replicas	非同期複製の最大数	1
auto.replace.lost.shards	詳しくは、 174 ページの『分散デプロイメントの構成』を参照してください。	true
development.mode	• `true` - プライマリーと同じノード上で複製をアクティブにできる • `false` - 複製はプライマリーと異なるノード上になければならない	false

各種アプリケーション・サーバー用の HTTP セッション・マネージャーの構成

WebSphere eXtreme Scale には、Web コンテナーのデフォルト・セッション・マネージャーをオーバーライドし、セッション複製、高可用性、優れたスケーラビリティー、および堅固な構成オプションを備えたセッション管理実装がバンドルされています。 eXtreme Scale セッション複製マネージャーおよび汎用組み込み ObjectGrid コンテナーの開始を有効にします。

このタスクについて

WebSphere Application Server Community Edition などの WebSphere Application Server を実行していない他のアプリケーション・サーバーで HTTP セッション・マネージャーを使用できます。データ・グリッドを使用するように他のアプリケーション・サーバーを構成するには、アプリケーションを接合して、WebSphere eXtreme Scale JAR ファイルをアプリケーションに取り込む必要があります。

手順

1. アプリケーションを接合することで、アプリケーションがセッション・マネージャーを使用できるようにします。 セッション・マネージャーを使用するためには、適切なフィルター宣言をアプリケーションの Web デプロイメント記述子に追加する必要があります。さらに、セッション・マネージャー構成パラメーターが、デプロイメント記述子内のサーブレット・コンテキスト初期化パラメーターという形式でセッション・マネージャーに渡されます。この情報は、以下に示す 3 とおりの方法でアプリケーションに導入することができます。
   - **addObjectGridFilter スクリプト**

     eXtreme Scale とともに提供されるコマンド行スクリプトを使用して、フィルター宣言と構成によってアプリケーションをサーブレット・コンテキスト初期化パラメーターの形式で接合します。このスクリプト `objectgridRoot/session/bin/addObjectGridFilter.sh` または `objectgridRoot/session/bin/addObjectGridFilter.bat` は 2 つのパラメーターを取ります。 1 つは接合するアプリケーション・エンタープライズ・アーカイブ (EAR) ファイルへの絶対パスで、もう 1 つは各種構成プロパティーが含まれたスプライサー・プロパティー・ファイルへの絶対パスです。このスクリプトの使用形式は以下の通りです。

     Windows

```
addObjectGridFilter.bat [EAR ファイルのロケーション]
[プロパティー・ファイルのロケーション]
```

UNIX

```
addObjectGridFilter.sh [EAR ファイルのロケーション]
[プロパティー・ファイルのロケーション]
```

UNIX **UNIX 上のスタンドアロン・ディレクトリーにインストールされている eXtreme Scale の使用例:**

a. `cd objectgridRoot/session/bin`

b. `addObjectGridFilter.sh /tmp/mySessionTest.ear objectgridRoot/session/samples/splicer.properties`

接合されるサーブレット・フィルターは構成値のデフォルトを保持します。これらのデフォルト値は、2 番目の引数にあるプロパティー・ファイルで指定する構成オプションでオーバーライドできます。使用できるパラメーターのリストについては、286 ページの『サーブレット・コンテキスト初期化パラメーター』を参照してください。

eXtreme Scale インストールとともに提供されるサンプルの `splicer.properties` ファイルを変更して使用することができます。また、各サーブレットを拡張することによってセッション・マネージャーを挿入する、addObjectGridServlets スクリプトも使用できます。ただし、推奨スクリプトは addObjectGridFilter スクリプトです。

- **Ant ビルド・スクリプト**

  WebSphere eXtreme Scale には Apache Ant で使用できる `build.xml` ファイルが同梱されています。このファイルは WebSphere Application Server インストールの `wasRoot/bin` フォルダーに含まれています。 `build.xml` を修正してセッション・マネージャー構成プロパティーを変更することができます。構成プロパティーは `splicer.properties` ファイル内のプロパティー名と同一です。 `build.xml` ファイルの変更後に、 `ant.sh`、`ws_ant.sh` (UNIX) または `ant.bat`、 `ws_ant.bat` (Windows) を実行することで、Ant プロセスを呼び出します。

- **手動による Web 記述子の更新**

  Web アプリケーションに同梱されている `web.xml` ファイルを編集して、フィルター宣言、そのサーブレット・マッピング、およびサーブレット・コンテキスト初期化パラメーターが組み込まれるようにします。この方法はエラーを起こしやすいため、使用しないようにしてください。

使用できるパラメーターのリストについては、286 ページの『サーブレット・コンテキスト初期化パラメーター』を参照してください。

2. WebSphere eXtreme Scale セッション複製マネージャーの JAR ファイルをアプリケーションに取り込みます。 ファイルは、アプリケーション・モジュールの `WEB-INF/lib` ディレクトリーまたはアプリケーション・サーバーのクラスパスに組み込むことができます。必要な JAR ファイルは、以下のように、使用しているコンテナーのタイプによって異なります。

- リモート eXtreme Scale コンテナー: `ogclient.jar` および `sessionobjectgrid.jar`
- 組み込み eXtreme Scale コンテナー: `objectgrid.jar` および `sessionobjectgrid.jar`

3. オプション: リモート eXtreme Scale コンテナーを使用する場合には、コンテナーを始動します。 詳しくは、 357 ページの『コンテナー・プロセスの開始』を参照してください。
4. アプリケーションをデプロイします。 サーバーやクラスターに対して通常使用する手順に従ってアプリケーションをデプロイしてください。アプリケーションをデプロイした後、アプリケーションを始動することができます。
5. アプリケーションにアクセスします。 これで、セッション・マネージャーおよび WebSphere eXtreme Scale と対話するアプリケーションにアクセスすることができます。

次のタスク

アプリケーションの装備時にセッション・マネージャーの構成属性の大多数を変更して、セッション・マネージャーを使用するようにすることができます。これらの属性には、複製タイプ (同期または非同期) のバリエーション、メモリー内セッション・テーブルのサイズなどがあります。アプリケーションの装備時に変更できる属性を別にすれば、アプリケーションのデプロイメント後に変更できるその他の構成属性は、WebSphere eXtreme Scale サーバー・クラスター・トポロジーと、それらのクラスターのクライアント (セッション・マネージャー) がそれらのクラスターに接続する方法に関係する属性のみです。

関連概念

HTTP セッション管理

関連タスク

274 ページの『WebSphere Application Server での HTTP セッション・マネージャーの構成』
WebSphere Application Server はセッション管理機能を備えていますが、このサポートは極度の要求ロード下では拡張しません。 WebSphere eXtreme Scale には、セッション複製、高可用性、優れたスケーラビリティー、および堅固な構成オプションを備えたセッション管理実装がバンドルされています。

関連資料

281 ページの『splicer.properties ファイル』
splicer.properties ファイルには、サーブレット・フィルター・ベース ObjectGrid セッション・マネージャーのすべての構成オプションが含まれます。

サーブレット・コンテキスト初期化パラメーター

以下に示すサーブレット・コンテキスト初期化パラメーターのリストは、選択した接続メソッドで必要なスプライサー・プロパティー・ファイルに指定できるものです。

パラメーター

objectGridType

ストリング値「REMOTE」または「EMBEDDED」。デフォルトは REMOTE です。

この値が「REMOTE」になっている場合、セッション・データは Web アプリケーションが実行されているサーバーの外に格納されます。

この値が「EMBEDDED」に設定されている場合は、Web アプリケーションが実行されているアプリケーション・サーバー・プロセス内で組み込みの eXtreme Scale コンテナーが開始されます。

objectGridName

特定の Web アプリケーションで使用する ObjectGrid インスタンスの名前を定義するストリング値。デフォルトの名前は「session」です。

eXtreme Scale コンテナーの開始時に使用するオブジェクト・グリッド XML ファイルとデプロイメント XML ファイルの両方で、このプロパティーは objectGridName を反映する必要があります。

catalogHostPort

カタログ・サーバーに接続して、クライアント・サイドの ObjectGrid インスタンスを取得できます。この値は、形式 `host:port<,host:port>` で指定します。ここでの「host」はカタログ・サーバーが実行されているリスナー・ホストで、「port」はそのカタログ・サーバー・プロセスのリスナー・ポートです。このリストは任意の長さにすることができ、ブートストラッピングにのみ使用されます。最初の実行可能なアドレスが使用されます。catalog.services.cluster プロパティーが構成されている場合は、WebSphere Application Server 内でオプションです。

replicationInterval

更新されたセッションを ObjectGrid に書き込む時間間隔を秒数で定義する整数値。デフォルトは 2 で、0 から 60 までの値が設定可能です。0 の場合は、更新されたセッションを ObjectGrid に書き込むのは、各要求のサーブレット・サービス・メソッド呼び出しの最後ということになります。replicationInterval 値が高いほど、パフォーマンスが改善されます。これは、グリッドに書き込まれるアップデートの数が少ないためです。ただし、値が高いとそれだけ構成のフォールト・トレラント性が低くなります。

この設定は、objectGridType が「REMOTE」に設定されている場合のみ適用されます。

sessionTableSize

メモリー内に保持するセッション参照数を定義する整数値。デフォルトは 2000 です。

組み込みトポロジーは既に Web コンテナーと同じ層にセッション・データを持っているため、この設定はリモート・トポロジーのみに関するものです。

セッションは、最長未使用時間ロジックに基づいて、メモリー内のテーブルから除去されます。メモリー内のテーブルから除去されたセッションは Web コンテナーから無効化されますが、データ自体はグリッドから削除されないので、そのセッションに対する後続の要求は引き続きデータを検索することができます。

fragmentedSession

ストリング値「true」または「false」。デフォルト値は「true」です。この設定を使用して、製品がセッション・データをエントリー全体として保管するか、それぞれの属性を個別に保管するかを制御します。

Web アプリケーションのセッションが持っている属性の数が多い場合やサイズが大きい場合は、fragmentedSession を「TRUE」に設定してください。すべての属性はグリッド内の同じキーに保管されるため、fragmentedSession を「FALSE」に設定するのは属性の数が少ない場合のみにしてください。

以前のフィルター・ベースの実装環境では、このプロパティーは「persistenceMechanism」と呼ばれており、設定可能な値は ObjectGridStore (フラグメント化されている場合) および ObjectGridAtomicSessionStore (フラグメント化されていない場合) でした。

securityEnabled

ストリング値「true」または「false」。デフォルト値は「false」です。この設定により、eXtreme Scale クライアント・セキュリティーを使用可能にすることができます。この設定は、eXtreme Scale サーバー・プロパティー・ファイルの securityEnabled 設定と一致していなければなりません。これらの設定が一致しないと、例外が発生します。

credentialGeneratorClass

com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator インターフェースを実装するクラスの名前。このクラスを使用して、クライアントの資格情報が取得されます。

credentialGeneratorProps

CredentialGenerator 実装クラスのプロパティー。このプロパティーが、**setProperties(String)** メソッドを使用してオブジェクトに設定されます。credentialGeneratorProps 値は、**credentialGeneratorClass** プロパティーの値が非ヌルの場合にのみ使用されます。

objectGridXML

`objectgrid.xml` ファイルが置かれているロケーション。objectGridType=EMBEDDED および **objectGridXML** プロパティーが指定されていない場合、eXtreme Scale ライブラリーにパッケージされた組み込み XML ファイルが自動的にロードされます。

objectGridDeploymentXML

objectGrid デプロイメント・ポリシーの XML ファイルが置かれているロケーション。 objectGridType=EMBEDDED および **objectGridDeploymentXML** プロパティーが指定されていない場合、eXtreme Scale ライブラリーにパッケージされた組み込み XML ファイルが自動的にロードされます。

traceSpec

ストリング値として設定される、IBM WebSphere のトレース仕様。この設定は、WebSphere Application Server以外のアプリケーション・サーバーに使用してください。

traceFile

ストリング値として設定される、トレース・ファイルのロケーション。この設定は、WebSphere Application Server以外のアプリケーション・サーバーに使用してください。

WebSphere eXtreme Scale の動的キャッシュ・プロバイダーの構成

eXtreme Scale の動的キャッシュ・プロバイダーのインストールと構成は、要件と、セットアップした環境によって異なります。

始める前に

動的キャッシュ・プロバイダーをインストールします。

動的キャッシュ・プロバイダーを使用するには、WebSphere Application Server ノードの各デプロイメント (デプロイメント・マネージャー・ノードを含む) 上に WebSphere eXtreme Scale をインストールしておく必要があります。 eXtreme Scale 動的キャッシュ・プロバイダーは、以下のバージョンの WebSphere Application Server でサポートされています。

- WebSphere Application Server 6.1.0.25 + PK85622 以降
- WebSphere Application Server 7.0.0.3 + PK85622 以降

インストールの説明は、バージョン 6.1 の場合のインストール (Installing for 6.1)または、バージョン 7.0 の場合のインストール (Installing for 7.0)を参照してください。

IBM WebSphere Commerce での eXtreme Scale 動的キャッシュ・プロバイダーの使用については、 IBM WebSphere Commerce 資料の以下のトピックを参照してください。

- Enabling the dynamic cache service and servlet caching
- Enabling WebSphere Commerce data cache

このタスクについて

eXtreme Scale 動的キャッシュ・プロバイダーを構成するには、以下のステップに従ってください。

1. eXtreme Scale 動的キャッシュ・プロバイダーを使用可能にします。

   WebSphere Application Server 7.0 では、動的キャッシュ・サービスを構成すると、管理コンソールまたはカスタム・プロパティーで eXtreme Scale 動的キャッシュ・プロバイダーを使用できます。

   eXtreme Scale をインストールすると、 eXtreme Scale 動的キャッシュ・プロバイダーが、管理コンソールの「キャッシュ・プロバイダー」オプションとして即時に使用可能になります。詳しくは、 動的キャッシュ・サービス・プロバイダーの選択 (Selecting a cache service provider) を参照してください。

   また、キャッシュ・インスタンスの eXtreme Scale 動的キャッシュ・プロバイダーは、以下のカスタム・プロパティーと値の組をそのインスタンス上で設定する

ことでも構成できます。これらのカスタム・プロパティーは、WebSphere Application Server 6.1 上の eXtreme Scale プロバイダーを使用可能にする唯一の方法で、次のようになります。

```
com.ibm.ws.cache.CacheConfig.cacheProviderName =
   "com.ibm.ws.objectgrid.dynacache.CacheProviderImpl"
```

定義したオブジェクト・キャッシュまたはサーブレット・キャッシュ・インスタンスにキャッシュを特別に送信しない場合には、動的キャッシュ API 呼び出しは、恐らく baseCache によって処理されます。 baseCache は、動的キャッシュ・サービスの一部として作成されるデフォルト・キャッシュ・インスタンスです。同じ構成プロパティーを使用して、eXtreme Scale をキャッシュ・プロバイダーとして使用するように baseCache インスタンスを構成します。ただし、構成プロパティーは、baseCache で有効にするには、 JVM カスタム・プロパティーとして設定する必要があります。

構成変数を JVM カスタム・プロパティーとして設定すると、他のキャッシュもそのプロパティーを選択する可能性があります。オブジェクト・キャッシュ・インスタンスおよびサーブレット・キャッシュ・インスタンスでは、そのキャッシュ・インスタンスで設定されたカスタム・プロパティーの方が好ましいのですが、特定のキャッシュ・インスタンスが、デフォルト・プロバイダーを使用する必要があるのにもかかわらず eXtreme Scale プロバイダーを使用していることが分かった場合には、カスタム・プロパティーを使用してその状態を修正できます。キャッシュ・インスタンスでデフォルトの動的キャッシュ・プロバイダーを使用するには、以下のようにキャッシュ・プロバイダー・プロパティーを設定します。

```
com.ibm.ws.cache.CacheConfig.cacheProviderName = "default"
```

baseCache に設定された eXtreme Scale 動的キャッシュ・プロバイダー構成プロパティーは他のキャッシュ・インスタンスによって選択される可能性があるため、キャッシュ構成プロパティーでデフォルト値を使用する場合には注意してください。

2. キャッシュの複製設定を構成します。

   eXtreme Scale 動的キャッシュ・プロバイダーを使用すると、ローカル・キャッシュ・インスタンスと複製キャッシュ・インスタンスを持つことができます。ローカル・キャッシュ・インスタンスの場合、それ以上の構成は必要がないので、このセクションの残りの部分はスキップしてかまいません。

   複製キャッシュの作成には、2 とおりの方法があります。 1 つ目の方法は、管理コンソールからキャッシュ複製を使用可能にすることです。この使用可能化は、WebSphere Application Server バージョン 7.0 では常に実行できますが、WebSphere Application Server バージョン 6.1 では、DRS 複製ドメインの作成が必要になります。

   2 つ目の方法は、以下のカスタム・プロパティーと値の組を使用して、たとえ DRS 複製ドメインがそのキャッシュに割り当てられていないとしても、そのキャッシュが複製されたキャッシュであると強制的に示すことです。

   ```
   com.ibm.ws.cache.CacheConfig.enableCacheReplication = "true"
   ```

3. 動的キャッシュ・サービスのトポロジーを構成します。

eXtreme Scale 動的キャッシュ・プロバイダーで唯一必須の構成パラメーターが、キャッシュ・トポロジーです。動的キャッシュ・サービスのカスタム・プロパティーとして、以下の変数を設定する必要があります。

```
com.ibm.websphere.xs.dynacache.topology
```

このプロパティーには次の 3 つの値が設定可能です。

- embedded
- embedded_partitioned
- remote

許可されている値のいずれかを使用する必要があります。

4. 動的キャッシュ・サービスの初期コンテナー数を構成します。

   初期コンテナーの数を構成することで、組み込み区画化トポロジーを使用するキャッシュのパフォーマンスを最大化することができます。 WebSphere Application Server Java 仮想マシンで、システム・プロパティーとして以下の変数を設定する必要があります。

   ```
   com.ibm.websphere.xs.dynacache.num_initial_containers
   ```

   この構成プロパティーの推奨値は、この分散キャッシュ・インスタンスにアクセスする WebSphere Application Server インスタンスの合計数に等しいか、わずかに少ない整数です。例えば、動的キャッシュ・サービスがグリッド・メンバー間で共有される場合、この変数はグリッド・メンバーの数に設定すべきです。

   組み込みトポロジーまたは組み込み区画化トポロジーの場合は、WebSphere Application Server のバージョン 7.0 を使用する必要があります。 JVM プロセスで以下のカスタム・プロパティーを設定して、初期コンテナーをすぐに使用できるようにします。

   ```
   com.ibm.ws.cache.CacheConfig.createCacheAtServerStartup=true
   ```

5. eXtreme Scale カタログ・サービス・グリッドを構成します。

   分散キャッシュ・インスタンスの動的キャッシュ・プロバイダーとして eXtreme Scale を使用している場合、eXtreme Scale カタログ・サービス・ドメインを構成する必要があります。

   単一のカタログ・サービス・ドメインで、eXtreme Scale でサポートされる複数の動的キャッシュ・サービス・プロバイダーに対応することができます。

   7.1+ カタログ・サービスは、 WebSphere Application Server プロセスの内部または外部で実行されます。 eXtreme Scale バージョン 7.1 から、管理コンソールを使用してカタログ・サーバーのドメイン・メカニズムを構成すると、動的キャッシュでその設定が使用されます。追加の手順を実行してカタログ・サービスをセットアップする必要はありません。詳しくは、 371 ページの『WebSphere Application Server でのカタログ・サービス・ドメインの作成』を参照してくだ

さい。カタログ・サービス・ドメインを実行している場合、カタログ・サービス・エンドポイントの **catalog.services.cluster** カスタム・プロパティーを設定する必要があります。

6. カスタム・キー・オブジェクトを構成します。

   キーとしてカスタム・オブジェクトを使用している場合、オブジェクトは Serializable インターフェースまたは Externalizable インターフェースを実装している必要があります。組み込みトポロジーあるいは組み込み区画化トポロジーを使用している場合、まるでデフォルトの動的キャッシュ・プロバイダーで使用しているかのようにオブジェクトを WebSphere 共有ライブラリー・パスに配置する必要があります。詳細は、WebSphere Application Server Network Deployment インフォメーション・センターの 動的キャッシュ用の DistributedMap および DistributedObjectCache インターフェースの使用を参照してください。

   リモート・トポロジーを使用している場合は、カスタム・キー・オブジェクトをスタンドアロン eXtreme Scale コンテナーの CLASSPATH に配置する必要があります。

7. eXtreme Scale コンテナー・サーバーを構成します。

   キャッシュ・データは、WebSphere eXtreme Scale コンテナーに保管されます。コンテナーは、 WebSphere Application Server プロセスの内部または外部で実行できます。キャッシュ・インスタンスに組み込み・トポロジーまたは組み込み区画化トポロジーを使用している場合、eXtreme Scale プロバイダーは、WebSphere プロセスの内部に自動的にコンテナーを作成します。これらのトポロジーの場合、それ以上の構成は必要ありません。

   リモート・トポロジーを使用している場合は、キャッシュ・インスタンスにアクセスする WebSphere Application Server インスタンスが開始する前に、スタンドアロン eXtreme Scale コンテナーを開始しておく必要があります。 特定の動的キャッシュ・サービスのコンテナーがすべて、同じカタログ・サービスのエンドポイントをポイントしていることを確認してください。

   スタンドアロン eXtreme Scale 動的キャッシュ・プロバイダー・コンテナーの XML 構成ファイルは、WebSphere Application Server 上のインストールの場合は `<install_root>/customLibraries/ObjectGrid/dynacache/etc` ディレクトリーか、あるいは、スタンドアロン・インストールの場合は `<install_root>/ObjectGrid/dynacache/etc` ディレクトリーに配置されます。このファイルの名前は、`dynacache-remote-objectgrid.xml` および `dynacache-remote-definition.xml` です。これらのファイルのコピーを作成して編集し、eXtreme Scale 動的キャッシュ・プロバイダーのスタンドアロン・コンテナーを起動する際に使用できるようにします。 **`dynacache-remote-deployment.xml`** ファイルの **numIntitialContainers** パラメーターは、実行中のコンテナー・プロセスの数に応じて設定する必要があります。なお、`dynacache-remote-objectgrid.xml` ファイルの **numberOfPartitions** 属性のデフォルト値は 47 です。

   注: コンテナー・プロセスのセットには、リモート・トポロジーを使用するように構成されたすべての動的キャッシュ・インスタンスを処理するために十分な空きメモリーがあることが必要です。 **catalog.services.cluster** の同じ値または同等の値を共有する WebSphere Application Server プロセスはすべて、同一セットの

スタンドアロン・コンテナーを使用する必要があります。コンテナーの数と、そのコンテナーが入るマシンの数は適切なサイズに設定される必要があります。詳細情報については、13 ページの『キャパシティー・プランニングと高可用性 (動的キャッシング)』を参照してください。

次のコードは、eXtreme Scale 動的キャッシュ・プロバイダーのスタンドアロン・コンテナーを起動する UNIX コマンド行入力の例を示しています。

```
startOgServer.sh container1 -objectGridFile ../dynacache/etc/dynacache-remote-objectgrid.xml -deploymentPolicyFile ../dynacache/etc/dynacache-remote-deployment.xml -catalogServiceEndpoints MyServer1.company.com:2809
```

8. 分散トポロジーまたは組み込みトポロジーでは、サイジング・エージェントを使用可能にして、メモリー消費量の推定精度を向上させます。

   サイジング・エージェントは、メモリー消費量を推定します (usedBytes 統計)。このエージェントでは、Java 5 以上の JVM が必要です。

   以下の引数を JVM コマンド行に追加することで、エージェントをロードします。

   ```
   -javaagent:WXS lib directory/wxssizeagent.jar
   ```

   組み込みトポロジーでは、 WebSphere Application Server プロセスのコマンド行に引数を追加します。

   分散トポロジーでは、eXtreme Scale プロセス (コンテナー) および WebSphere Application Server プロセスのコマンド行に引数を追加します。

Spring 統合の構成

Spring 記述子 XML ファイル

Spring 記述子 XML ファイルを使用して、eXtreme Scale を構成して Spring と統合します。

以下のセクションにおいて、Spring `objectgrid.xsd` ファイルの各エレメントおよび属性が定義されています。 Spring `objectgrid.xsd` ファイルは、`ogspring.jar` ファイル、および objectgrid 名前空間 `com/ibm/ws/objectgrid/spring/namespace` にあります。記述子 XML スキーマの例については、300 ページの『Spring `objectgrid.xsd` ファイル』を参照してください。

register エレメント

register エレメントを使用して、ObjectGrid のデフォルト Bean ファクトリーを登録します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

id 特定 ObjectGrid のデフォルト Bean ディレクトリーの名前を指定します。

gridname
ObjectGrid インスタンスの名前を指定します。この属性に割り当てられる値は、ObjectGrid 記述子ファイルに構成される有効な ObjectGrid に対応している必要があります。

```
<register
(1) id="register id"
(2) gridname="ObjectGrid name"
/>
```

server エレメント

server エレメントを使用して eXtreme Scale サーバーを定義します。eXtreme Scale サーバーは、コンテナー、カタログ・サービス、またはその両方をホストすることができます。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

id eXtreme Scale サーバーの名前を指定します。

tracespec
トレースのタイプを示し、サーバーのトレースおよびトレース仕様を使用可能にします。

tracefile
作成および使用する traceFile のパスと名前を指定します。

statspec
サーバーの統計仕様を指定します。

jmxport
JMX/RMI 接続を使用可能にする未使用のポート番号を指定します。 JMX は、リモート・システムのモニターおよび管理を使用可能にします。

isCatalog
特定のサーバーがカタログ・サービスをホストするかどうかを指定します。デフォルト値は `false` です。

name
サーバーの名前を指定します。

haManagerPort
HA マネージャーのポート番号を設定します。

listenerHost
ORB のバインド先のホスト名を設定します。

listenerPort
ORB のバインド先のポートを設定します。

maximumThreadPoolSize
プール内のスレッドの最大数を設定します。

memoryThresholdPercentage
メモリー・ベース除去のメモリーしきい値 (最大ヒープのパーセンテージ) を設定します。

minimumThreadPoolSize
プール内のスレッドの最小数を設定します。

workingDirectory
すべてのデフォルト設定に対して ObjectGrid サーバーが使用するべきディレクトリーを定義するプロパティーです。

zoneName
このサーバーが属するゾーンを定義します。

enableChannelFramework
IBM ORB プロパティーの TransportMode を ChannelFramework に設定します。

enableSystemStreamToFile
SystemOut および SystemErr をファイルに送信するかどうかを定義します。

enableMBeans
このプロセスにおいて ObjectGrid が MBeans を登録するかどうかを決定します。

serverPropertyFile
ファイルからサーバー・プロパティーをロードします。

catalogServerProperties
サーバーをホストするカタログ・サーバーを指定します。

```
<server
(1) id="server id"
(2) tracespec="the server trace specification"
(3) tracefile="the server trace file"
(4) statspec="the server statistic specification"
(5) jmxport="JMX port number"
(6) isCatalog="true"|"false"
(7) name="the server name"
(8) haManagerPort="the haManager port"
(9) listenerHost="the orb binding host name"
(10) listenerPort="the orb binding listener port"
(11) maximumThreadPoolSize="the number of maximum threads"
(12) memoryThresholdPercentage="the memory threshold (percentage of max heap)"
(13) minimumThreadPoolSize="the number of minimum threads"
(14) workingDirectory="location for the working directory"
(15) zoneName="the zone name"
(16) enableChannelFramework="true"|"false"
(17) enableSystemStreamToFile="true"|"false"
(18) enableMBeans="true"|"false"
(19) serverPropertyFile="location of the server properties file."
(20) catalogServerProperties="the catalog server properties reference"
/>
```

catalog エレメント

catalog エレメントを使用して、データ・グリッド内のコンテナー・サーバーに経路指定します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

ホスト (host)
カタログ・サービスが稼働しているワークステーションのホスト名を指定します。

ポート (port)
ホスト名と対になっているポート番号を指定します。これでクライアントが接続できるカタログ・サービス・ポートを決定します。

```
<catalog
(1) host="catalog service host name"
(2) port="catalog service port number"
/>
```

カタログ・サーバー・エレメント

カタログ・サーバーのプロパティー・エレメントを使用して、カタログ・サーバー・サービスを定義します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

catalogServerEndPoint
カタログ・サーバーの接続プロパティーを指定します。

enableQuorum
クォーラムを使用可能にするかどうかを決定します。

heartBeatFrequencyLevel
ハートビートの頻度レベルを設定します。

domainName
複数ドメインに経路指定する際にクライアントに対してこのカタログ・サービス・グリッドを固有に識別するために使用するドメイン・ネームを定義します。

foreignDomains
マルチ・プライマリー・シナリオでのデータ交換を目的として、各外部ドメインに双方向接続が確立されます。

clusterSecurityURL
カタログ・サービスに固有のセキュリティー・ファイルの場所を設定します。

```
<catalog server
(1) catalogServerEndPoint="a catalog server endpoint reference "
(2) enableQuorum="true"|"false"
(3) heartBeatFrequencyLevel="
  HEARTBEAT_FREQUENCY_LEVEL_TYPICAL|
  HEARTBEAT_FREQUENCY_LEVEL_RELAXED|
  HEARTBEAT_FREQUENCY_LEVEL_AGGRESSIVE"
(4) domainName="the domain name used to uniquely identify this catalog service grid"
(5) domainEndpoints="a reference to the domain name endpoints"
(6) foreignDomains="the name of the foreign domain"
(7) clusterSecurityURL="the The cluster security file location."/>
```

カタログ・サーバー・エンドポイント・エレメント

カタログ・サーバー EndPoint エレメントを使用して、カタログ・サーバー・エレメントが使用するカタログ・サーバー・エンドポイントを作成します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

serverName
起動しようとしているプロセスを識別する名前を指定します。

hostName
サーバーを起動するマシンのホスト名を指定します。

clientPort
ピア・カタログ・クラスター通信に使用されるポートを指定します。

peerPort
ピア・カタログ・クラスター通信に使用されるポートを指定します。

```
<catalogServerEndPoint
(1) name="catalog server endpoint name"
(2) host=""
(3) clientPort=""
(4) peerPort""
/>
```

container エレメント

container エレメントを使用して、データそのものを保管します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

objectgridxml
使用する記述子 XML ファイルのパスと名前を指定します。このファイルは、マップ、ロック・ストラテジー、プラグインなどの ObjectGrid の特性を指定します。

deploymentxml
記述子 XML ファイルと一緒に使用して、区画化、複製、初期コンテナーの数、およびその他の設定を決定する XML ファイルのパスと名前を指定します。

server
コンテナーがホストされるサーバーを指定します。

```
<server
(1) objectgridxml="the objectgrid descriptor XML file"
(2) deploymentxml ="the objectgrid deployment descriptor XML file "
(3) server="the server reference "
/>
```

JPALoader エレメント

JPALoader エレメントを使用して、ObjectMap API の使用時に ObjectGrid キャッシュと既存のバックエンド・データ・ストアを同期します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

entityClassName
EntityManager.persist および EntityManager.find などの JPA の使用を可能にします。 **entityClassName** 属性は、JPALoader で必須です。

preloadPartition
マップ・プリロードが開始される区画番号を指定します。この値がゼロより小さいか、あるいは、(totalNumberOfPartition – 1) より大きい場合、マップ・プリロードは開始されません。

```
<JPALoader
(1) entityClassName="the entity class name"
(2) preloadPartition ="int"
/>
```

JPATxCallback エレメント

JPATxCallback エレメントを使用して、 JPA および ObjectGrid トランザクションを調整します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

persistenceUnitName
JPA EntityManagerFactory を作成し、persistence.xml ファイルで JPA エンティティー・メタデータを検索します。 **persistenceUnitName** 属性は必須です。

jpaPropertyFactory
パーシスタンス・プロパティー・マップを作成し、デフォルトのパーシスタンス・プロパティーをオーバーライドするためのファクトリーを指定します。この属性は、Bean を参照します。

exceptionMapper
JPA 固有あるいはデータベース固有の例外マッピング機能に使用できる ExceptionMapper プラグインを指定します。この属性は、Bean を参照します。

```
<JPATxCallback
(1) persistenceUnitName="the JPA persistence unit name"
(2) jpaPropertyFactory ="JPAPropertyFactory bean reference"
(3) exceptionMapper="ExceptionMapper bean reference"
/>
```

JPAEntityLoader エレメント

JPAEntityLoader エレメントを使用して、EntityManager API の使用時に ObjectGrid キャッシュと既存のバックエンド・データ・ストアを同期します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

entityClassName
EntityManager.persist および EntityManager.find などの JPA の使用を可能にします。 **entityClassName** 属性は、JPAEntityLoader エレメントでオプションです。このエレメントが構成されていないと、ObjectGrid エンティティー・マップに構成されているエンティティー・クラスが使用されます。 ObjectGrid EntityManager と JPA プロバイダーには、同じクラスが使用される必要があります。

preloadPartition
マップ・プリロードが開始される区画番号を指定します。この値がゼロより小さいか、あるいは、(totalNumberOfPartition – 1) より大きい場合、マップ・プリロードは起動されません。

```
<JPAEntityLoader
(1) entityClassName="the entity class name"
(2) preloadPartition ="int"
/>
```

LRUEvictor エレメント

LRUEvictor エレメントを使用して、マップがその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します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

maxSize
evictor の介入が必要になる、キュー内の合計エントリー数を指定します。

sleepTime
マップへの必要なアクションを決定するための、evictor のマップ・キューに対するスイープの間の時間を秒単位で設定します。

numberOfLRUQueues
マップ全体の大きさの単一キューにならないように、evictor がスキャンする必要があるキュー数の設定を指定します。

```
<LRUEvictor
(1) maxSize="int"
(2) sleepTime ="seconds"
(3) numberOfLRUQueues ="int"
/>
```

LFUEvictor エレメント

LFUEvictor エレメントを使用して、マップがその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します。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

maxSize
evictor の対処が必要になる、各ヒープに許可される合計エントリー数を指定します。

sleepTime
マップへの必要なアクションを決定するための、evictor のマップ・ヒープに対するスイープの間の時間を秒単位で設定します。

numberOfHeaps
マップ全体の大きさの単一ヒープにならないように、evictor がスキャンする必要があるヒープ数の設定を指定します。

```
<LFUEvictor
(1) maxSize="int"
(2) sleepTime ="seconds"
(3) numberOfHeaps ="int"
```

HashIndex エレメント

HashIndex エレメントを Java リフレクションで使用し、オブジェクトの更新時にマップに保管されるオブジェクトを動的にイントロスペクトします。

- 出現回数: 0 回から複数回
- 子エレメント: なし

属性

name
索引の名前を指定します。これは各マップで固有である必要があります。

attributeName
索引付けする属性の名前を指定します。フィールド・アクセス索引の場合、属性名は、フィールド名と同等です。プロパティー・アクセス索引の場合、属性名は、JavaBean と互換性のあるプロパティー名です。

rangeIndex
範囲の索引付けを使用可能にするかどうかを示します。デフォルト値は `false` です。

fieldAccessAttribute
非エンティティー・マップに使用されます。データのアクセスに getter メソッドが使用されます。デフォルト値は `false` です。値に `true` を指定した場合、フィールドを使用してオブジェクトに直接アクセスします。

POJOKeyIndex
非エンティティー・マップに使用されます。デフォルト値は `false` です。値に `true` を指定した場合、索引は、マップのキー部分でオブジェクトをイントロスペクトします。これは、キーが複合キーで、値にキーが組み込まれていない場合に有用です。値を設定しないか、または値に `false` を設定した場合、索引は、マップの値部分でオブジェクトをイントロスペクトします。

```
<HashIndex
(1) name="index name"
(2) attributeName="attribute name"
(3) rangeIndex ="true"|"false"
(4) fieldAccessAttribute ="true"|"false"

(5) POJOKeyIndex ="true"|"false"
/>
```

Spring objectgrid.xsd ファイル

Spring `objectgrid.xsd` ファイルを使用して、 eXtreme Scale を Spring と統合し、eXtreme Scale トランザクションの管理と、クライアントおよびサーバーの構成を行います。

Spring `objectgrid.xsd` ファイルに定義されるエレメントおよび属性の説明は、 293 ページの『Spring 記述子 XML ファイル』を参照してください。

Spring objectgrid.xsd ファイル

```
<xsd:schema xmlns="http://www.ibm.com/schema/objectgrid"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:beans="http://www.springframework.org/schema/beans"
    targetNamespace="http://www.ibm.com/schema/objectgrid"
    elementFormDefault="qualified"
    attributeFormDefault="unqualified">

   <xsd:import namespace="http://www.springframework.org/schema/beans"/>

 <xsd:element name="transactionManager">
      <xsd:complexType>
         <xsd:attribute name="id" type="xsd:ID"/>
      </xsd:complexType>
   </xsd:element>

 <xsd:element name="register">
      <xsd:complexType>
         <xsd:attribute name="id" type="xsd:ID"/>
         <xsd:attribute name="gridname" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>

 <xsd:element name="server">
      <xsd:complexType>
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="catalog"/>
        </xsd:choice>
        <xsd:attribute name="id" type="xsd:ID"/>
        <xsd:attribute name="tracespec" type="xsd:string"/>
        <xsd:attribute name="tracefile" type="xsd:string"/>
        <xsd:attribute name="statspec" type="xsd:string"/>
        <xsd:attribute name="jmxport" type="xsd:integer"/>
        <xsd:attribute name="isCatalog" type="xsd:boolean"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="haManagerPort" type="xsd:integer"/>
        <xsd:attribute name="listenerHost" type="xsd:string"/>
        <xsd:attribute name="listenerPort" type="xsd:integer"/>
        <xsd:attribute name="maximumThreadPoolSize" type="xsd:integer"/>
        <xsd:attribute name="memoryThresholdPercentage" type="xsd:integer"/>
        <xsd:attribute name="minimumThreadPoolSize" type="xsd:integer"/>
        <xsd:attribute name="workingDirectory" type="xsd:string"/>
        <xsd:attribute name="zoneName" type="xsd:string"/>
        <xsd:attribute name="enableChannelFramework" type="xsd:boolean"/>
        <xsd:attribute name="enableSystemStreamToFile" type="xsd:boolean"/>
        <xsd:attribute name="enableMBeans" type="xsd:boolean"/>
        <xsd:attribute name="serverPropertyFile" type="xsd:string"/>
        <xsd:attribute name="catalogServerProperties" type="xsd:string"/>
     </xsd:complexType>
 </xsd:element>

 <xsd:element name="catalog">
   <xsd:complexType>
     <xsd:attribute name="host" type="xsd:string"/>
     <xsd:attribute name="port" type="xsd:integer"/>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name="catalogServerProperties">
   <xsd:complexType>
  <xsd:choice minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="catalogServerEndPoint"/>
        </xsd:choice>
       <xsd:attribute name="id" type="xsd:ID"/>
    <xsd:attribute name="enableQuorum" type="xsd:boolean"/>
    <xsd:attribute name="heartBeatFrequencyLevel" type="xsd:integer"/>
    <xsd:attribute name="domainName" type="xsd:string"/>
    <xsd:attribute name="domainEndpoints" type="xsd:string"/>
    <xsd:attribute name="foreignDomains" type="xsd:string"/>
    <xsd:attribute name="clusterSecurityURL" type="xsd:anyURI"/>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name="catalogServerEndPoint">
   <xsd:complexType>
     <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="host" type="xsd:string"/>
    <xsd:attribute name="clientPort" type="xsd:integer"/>
    <xsd:attribute name="peerPort" type="xsd:integer"/>
   </xsd:complexType>
 </xsd:element>

 <xsd:element name="container">
      <xsd:complexType>
         <xsd:attribute name="id" type="xsd:ID"/>
         <xsd:attribute name="objectgridxml" type="xsd:string"/>
         <xsd:attribute name="deploymentxml" type="xsd:string"/>
         <xsd:attribute name="server" type="xsd:string"/>
      </xsd:complexType>
   </xsd:element>
```

```
    <xsd:element name="JPALoader">
      <xsd:complexType>
         <xsd:attribute name="id" type="xsd:ID"/>
         <xsd:attribute name="entityClassName" type="xsd:string"/>
         <xsd:attribute name="preloadPartition" type="xsd:integer"/>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="JPATxCallback">
      <xsd:complexType>
         <xsd:attribute name="id" type="xsd:ID"/>
         <xsd:attribute name="persistenceUnitName" type="xsd:string"/>
   <xsd:attribute name="jpaPropertyFactory" type="xsd:string"/>
   <xsd:attribute name="exceptionMapper" type="xsd:string"/>
   </xsd:complexType>
   </xsd:element>

   <xsd:element name="JPAEntityLoader">
      <xsd:complexType>
         <xsd:attribute name="id" type="xsd:ID"/>
         <xsd:attribute name="entityClassName" type="xsd:string"/>
         <xsd:attribute name="preloadPartition" type="xsd:integer"/>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="LRUEvictor">
      <xsd:complexType>
         <xsd:attribute name="id" type="xsd:ID"/>
         <xsd:attribute name="maxSize" type="xsd:integer"/>
         <xsd:attribute name="sleepTime" type="xsd:integer"/>
         <xsd:attribute name="numberOfLRUQueues" type="xsd:integer"/>
         <xsd:attribute name="useMemoryUsageThresholdEviction" type="xsd:boolean"/>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="LFUEvictor">
      <xsd:complexType>
         <xsd:attribute name="id" type="xsd:ID"/>
         <xsd:attribute name="maxSize" type="xsd:integer"/>
         <xsd:attribute name="sleepTime" type="xsd:integer"/>
         <xsd:attribute name="numberOfHeaps" type="xsd:integer"/>
         <xsd:attribute name="useMemoryUsageThresholdEviction" type="xsd:boolean"/>
      </xsd:complexType>
   </xsd:element>

   <xsd:element name="HashIndex">
      <xsd:complexType>
         <xsd:attribute name="id" type="xsd:ID"/>
         <xsd:attribute name="name" type="xsd:string"/>
         <xsd:attribute name="attributeName" type="xsd:string"/>
         <xsd:attribute name="rangeIndex" type="xsd:boolean"/>
         <xsd:attribute name="fieldAccessAttribute" type="xsd:boolean"/>
         <xsd:attribute name="POJOKeyIndex" type="xsd:boolean"/>
      </xsd:complexType>
   </xsd:element>

</xsd:schema>
```

Spring 拡張 Bean および名前空間のサポート

WebSphere eXtreme Scaleには、objectgrid.xml ファイル内で拡張ポイントとして使用するために Plain Old Java Object (POJO) を宣言する機能があり、Bean を指定してからクラス名を指定する方法が提供されています。 通常、指定されたクラスのインスタンスが作成され、それらのオブジェクトはプラグインとして使用されます。eXtreme Scale は、これらのプラグイン・オブジェクトのインスタンスの取得を Spring に委任できます。アプリケーションが Spring を使用する場合は、通常、このような POJO をアプリケーションの残り部分に接続する必要があります。

場合によっては、特定のプラグイン・オブジェクトを構成するのに Spring を使用する必要があります。例として以下の構成を参考にしてください。

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="com.ibm.websphere.objectgrid.jpa.JPATxCallback">
        <property name="persistenceUnitName" type="java.lang.String"  value="employeePU" />
    </bean>
    ...
</objectGrid>
```

組み込み TransactionCallback 実装である com.ibm.websphere.objectgrid.jpa.JPATxCallback クラスは、TransactionCallback クラスとして構成されます。このクラスは上の例のように、persistenceUnitName プロパティーを使用して構成されます。JPATxCallback クラスには JPAPropertyFactory 属性もあり、これのタイプは java.lang.Object です。ObjectGrid XML 構成は、このタイプの構成をサポートできません。

eXtreme Scale Spring 統合は Bean 作成を Spring フレームワークに委任することでこの問題を解決します。修正後の構成は、次のようになります。

```
<objectGrid name="Grid">
    <bean id="TransactionCallback" className="{spring}jpaTxCallback"/>
    ...
</objectGrid>
```

"Grid" オブジェクト用の Spring ファイルには以下の情報が入っています。

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.
JPAPropFactoryImpl" scope="shard">
</bean>
```

ここでは、上の例に示されているように、{spring}jpaTxCallback として TransactionCallback が指定され、 Spring ファイル内に jpaTxCallback および jpaPropFactory Bean が構成されています。このような Spring 構成によって、JPAPropertyFactory Bean を JPATxCallback オブジェクトのパラメーターとして構成することが可能になります。

デフォルトの Spring Bean ファクトリー

eXtreme Scale が、接頭部 {spring} で始まる classname 値を持つプラグインまたは拡張 Bean (ObjectTransformer、Loader、TransactionCallback など) を検出した場合、eXtreme Scale は名前の残りの部分を Spring Bean 名として使用し、Spring Bean ファクトリーを使用して Bean インスタンスを取得します。

デフォルトでは、与えられた ObjectGrid 用に登録された Bean ファクトリーがない場合、ObjectGridName_spring.xml ファイルを見つけようとします。例えば、グリッドの名前が "Grid" の場合は、XML ファイルの名前は /Grid_spring.xml です。このファイルはクラスパスにあるか、クラスパス内の META-INF ディレクトリーにあるはずです。このファイルが見つかったら、eXtreme Scale は、そのファイルを使用して ApplicationContext を作成し、その Bean ファクトリーから Bean を作成します。

カスタム Spring Bean ファクトリー

WebSphere eXtreme Scale には ObjectGridSpringFactory API もあり、これを使用して、特定の指定された ObjectGrid のために使用するよう Spring Bean ファクトリー・インスタンスを登録できます。この API は、以下の静的メソッドを使用して、BeanFactory のインスタンスを eXtreme Scale に登録します。

```
void registerSpringBeanFactoryAdapter(String objectGridName, Object
springBeanFactory)
```

名前空間サポート

バージョン 2.0 以降の Spring には、Bean の定義と構成のため、基本的な Spring XML フォーマットをスキーマ・ベースで拡張するメカニズムが備わっています。ObjectGrid はこの新しい機能を使用して、ObjectGrid Bean の定義と構成を行います。Spring XML スキーマ拡張では、eXtreme Scale プラグインのいくつかの組み込み実装、およびいくつかの ObjectGrid Bean が "objectgrid" 名前空間に事前定義されます。 Spring 構成ファイルを作成するとき、これらの組み込み実装の完全クラス名を指定する必要はありません。代わりに、事前定義された Bean を参照するようにできます。

また、XML スキーマ内に Bean の属性が定義されていることによって、間違った属性名を指定する可能性が減少します。XML スキーマに基づいた XML 妥当性検査は、この種のエラーを開発サイクルの初期にキャッチできます。

XML スキーマ拡張に定義されている Bean は、以下のとおりです。

- transactionManager
- register
- server
- カタログ (catalog)
- catalogServerProperties
- コンテナー
- JPALoader
- JPATxCallback
- JPAEntityLoader
- LRUEvictor
- LFUEvictor
- HashIndex

これらの Bean は objectgrid.xsd XML スキーマ内に定義されています。この XSD ファイルは、`ogspring.jar` ファイル中の `com/ibm/ws/objectgrid/spring/namespace/objectgrid.xsd` ファイルとして出荷されます。XSD ファイルおよび XSD ファイルで定義された Bean について詳しくは、「*管理ガイド*」に記載されている Spring 記述子ファイルに関する説明を参照してください。

前のセクションにある JPATxCallback 例をまた使用します。前のセクションでは、JPATxCallback Bean は次のように構成されていました。

```
<bean id="jpaTxCallback" class="com.ibm.websphere.objectgrid.jpa.JPATxCallback" scope="shard">
    <property name="persistenceUnitName" value="employeeEMPU"/>
    <property name="JPAPropertyFactory" ref ="jpaPropFactory"/>
</bean>

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl" scope="shard">
</bean>
```

この名前空間フィーチャーを使用して、Spring XML 構成を次のようにコーディングできます。

```
<objectgrid:JPATxCallback id="jpaTxCallback" persistenceUnitName="employeeEMPU"
 jpaPropertyFactory="jpaPropFactory" />

<bean id="jpaPropFactory" class="com.ibm.ws.objectgrid.jpa.plugins.JPAPropFactoryImpl"
 scope="shard">
</bean>
```

ここでは、前の例でのように "com.ibm.websphere.objectgrid.jpa.JPATxCallback" クラスを指定する代わりに、事前定義された "objectgrid:JPATxCallback" Bean を直接使用することに注意してください。見て分かるように、この構成のほうが冗長でなく、誤りがないかチェックするのも簡単です。

Spring 拡張 Bean を使用したコンテナー・サーバーの開始

この例では、ObjectGrid Spring 管理拡張 Bean および名前空間のサポートを使用して、ObjectGrid サーバーを開始する方法を示します。

ObjectGrid XML ファイル

まず最初に、1 つの ObjectGrid "Grid" と 1 つのマップ "Test" が含まれているだけの、単純な ObjectGrid XML ファイルを定義します。この ObjectGrid には "partitionListener" という名前の ObjectGridEventListener プラグインがあり、マップ "Test" には "testLRUEvictor" という名前の Evictor プラグインがあります。ObjectGridEventListener プラグインと Evictor プラグインの両方とも、名前に "{spring}" が含まれるため、Spring を使用して構成される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="Grid">
     <bean id="ObjectGridEventListener" className="{spring}partitionListener" />
            <backingMap name="Test" pluginCollectionRef="test" />
        </objectGrid>
    </objectGrids>

    <backingMapPluginCollections>
        <backingMapPluginCollection id="test">
            <bean id="Evictor" className="{spring}testLRUEvictor"/>
        </backingMapPluginCollection>
    </backingMapPluginCollections>
</objectGridConfig>
```

ObjectGrid デプロイメント XML ファイル

次に、以下に示すように単純な ObjectGrid デプロイメント XML ファイルを作成します。これは ObjectGrid を 5 個の区画に分けます。複製は必要ありません。

```
<?xml version="1.0" encoding="UTF-8"?>
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy ../deploymentPolicy.xsd"
 xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy">
    <objectgridDeployment objectgridName="Grid">
        <mapSet name="mapSet" numInitialContainers="1" numberOfPartitions="5" minSyncReplicas="0"
            maxSyncReplicas="1" maxAsyncReplicas="0">
            <map ref="Test"/>
        </mapSet>
    </objectgridDeployment>
</deploymentPolicy>
```

ObjectGrid Spring XML ファイル

次に、 ObjectGrid Spring 管理拡張 Bean および名前空間のサポート機能を両方とも使用して、ObjectGrid Bean を構成します。spring xml ファイルの名前は "Grid_spring.xml" です。この XML ファイルには 2 つのスキーマが含まれていることに注意してください。spring-beans-2.0.xsd は Spring 管理 Bean を使用するためのもので、objectgrid.xsd は objectgrid 名前空間内に事前定義された Bean を使用するためのものです。

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:objectgrid="http://www.ibm.com/schema/objectgrid"
       xsi:schemaLocation="
       http://www.ibm.com/schema/objectgrid
   http://www.ibm.com/schema/objectgrid/objectgrid.xsd
       http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

    <objectgrid:register id="ogregister" gridname="Grid"/>

    <objectgrid:server id="server" isCatalog="true" name="server">
        <objectgrid:catalog host="localhost" port="2809"/>
    </objectgrid:server>

    <objectgrid:container id="container"
   objectgridxml="com/ibm/ws/objectgrid/test/springshard/objectgrid.xml"
       deploymentxml="com/ibm/ws/objectgrid/test/springshard/deployment.xml"
   server="server"/>

    <objectgrid:LRUEvictor id="testLRUEvictor" numberOfLRUQueues="31"/>

    <bean id="partitionListener"
   class="com.ibm.websphere.objectgrid.springshard.ShardListener" scope="shard"/>
</beans>
```

この spring XML ファイルには、次の 6 個の Bean が定義されました。

1. *objectgrid:register*: これは、ObjectGrid "Grid" に対してデフォルトの Bean ファクトリーを登録します。
2. *objectgrid:server*: これは、"server" という名前で ObjectGrid サーバーを定義します。objectgrid:catalog Bean がネストされているので、このサーバーはカタログ・サービスも提供します。
3. *objectgrid:catalog*: これは、"localhost:2809" に設定された ObjectGrid カタログ・サービス・エンドポイントを定義します。
4. *objectgrid:container*: これは、前述したように、指定された objectgrid XML ファイルおよびデプロイメント XML ファイルと共に ObjectGrid コンテナーを定義します。server プロパティーは、このコンテナーがどのサーバーでホストされているのかを指定します。
5. *objectgrid:LRUEvictor*: これは、使用する LRU キューの数を 31 に設定して LRUEvictor を定義します。
6. *bean partitionListener*: これは ShardListener プラグインを定義します。このプラグインの実装を指定する必要があるため、事前定義された Bean を使用することはできません。また、この Bean の有効範囲は "shard" (断片) に設定されています。これは、この ShardListener のインスタンスが ObjectGrid 断片当たり 1 つのみであることを意味します。

サーバーの始動

以下のスニペットは、コンテナー・サービスとカタログ・サービスの両方をホストする ObjectGrid サーバーを開始します。これを見て分かるように、サーバーを開始するために呼び出す必要のあるメソッドは、Bean ファクトリーからの Bean "container" の get だけです。これは、ロジックの大部分を Spring 構成に移すことになり、プログラミングの複雑さが軽減されます。

```
public class ShardServer extends TestCase
{
    Container container;
    org.springframework.beans.factory.BeanFactory bf;

    public void startServer(String cep)
    {
        try
        {
            bf = new org.springframework.context.support.ClassPathXmlApplicationContext(
                "/com/ibm/ws/objectgrid/test/springshard/Grid_spring.xml", ShardServer.class);
            container = (Container)bf.getBean("container");
        }
        catch(Exception e)
        {
            throw new ObjectGridRuntimeException("Cannot start OG container", e);
        }
    }

    public void stopServer()
    {
        if(container != null)
            container.teardown();
    }
}
```

REST データ・サービスの構成

以下のリンクから、REST データ・サービスの管理に関する情報を参照してください。 RestService Mbean に関するアプリケーション・プログラミング・インターフェースの情報についても参照してください。

REST データ・サービスのプロパティー・ファイル

REST データ・サービスを構成するには、REST のプロパティー・ファイルを編集して、WebSphere eXtreme Scale グリッドで必要なエンティティー・スキーマを定義します。

REST データ・サービスのプロパティー・ファイルは、eXtreme Scale REST データ・サービスのメイン構成ファイルです。このファイルは通常、キーと値のペアが含まれた Java プロパティー・ファイルです。デフォルトでは、REST データ・サービス・ランタイムで、クラスパス内の適切な名前の `wxsRestService.properties` ファイルが検索されます。このファイルは、`wxs.restservice.props` システム・プロパティーを使用して、明示的に定義することもできます。

```
-Dwxs.restservice.props=/usr/configs/dataservice.properties
```

REST データ・サービスがロードされるときに、使用されるプロパティー・ファイルがログ・ファイルに以下のように表示されます。

```
CWOBJ4004I: The eXtreme Scale REST data service properties files were
loaded: [/usr/configs/RestService.properties]
```

REST データ・サービスのプロパティー・ファイルでは、以下のプロパティーがサポートされます。

表 11. REST データ・サービスのプロパティー

property	説明

catalogServiceEndPoints	<host:port> というフォーマットの、カタログ・サービス・ドメインのホストとポートの必須コンマ区切りリスト。eXtreme Scale と統合された WebSphere Application Server を使用して REST データ・サービスをホストする場合には、これはオプションです。カタログ・サービスの構成および開始方法の詳細については、WebSphere eXtreme Scale 製品資料を参照してください。
	
	`catalogServiceEndPoints=`
	
	`server1:2809,server2:2809`

objectGridNames	REST サービスに公開する ObjectGrid の必須名。少なくとも 1 つの ObjectGrid 名が必要です。複数の ObjectGrid 名は、以下のように、コンマを使用して区切ります。 `ECommerceGrid,InventoryGrid`
objectGridClientXML	ObjectGrid クライアント・オーバーライド XML ファイルのオプションの名前。ここで指定した名前のファイルが、クラスパスからロードされます。デフォルトは、以下のとおりです。 /META-INF/objectGridClient.xml。 eXtreme Scale クライアントの構成方法の詳細については、WebSphere eXtreme Scale 製品資料を参照してください。
objectGridNames	REST サービスに公開する ObjectGrid の必須名。少なくとも 1 つの ObjectGrid 名が必要です。複数の ObjectGrid 名は、以下のように、コンマを使用して区切ります。 `ECommerceGrid,InventoryGrid`
objectGridClientXML	ObjectGrid クライアント・オーバーライド XML ファイルのオプションの名前。ここで指定した名前のファイルが、クラスパスからロードされます。デフォルトは、以下のとおりです。 /META-INF/objectGridClient.xml。 eXtreme Scale クライアントの構成方法の詳細については、WebSphere eXtreme Scale 製品資料を参照してください。

表 11. REST データ・サービスのプロパティー (続き)

property	説明
ogClientPropertyFile	ObjectGrid クライアント・プロパティー・ファイルのオプションの名前。このファイルには、ObjectGrid クライアント・セキュリティーを使用可能にするために必要なセキュリティー・プロパティーが含まれています。このプロパティー・ファイル内で「securityEnabled」属性が設定されていると、REST サービスが使用する ObjectGrid クライアントでセキュリティーが使用可能になります。また、このプロパティー・ファイル内で、「credentialGeneratorProps」を「user:pass」形式の値か、もしくは {xor_encoded user:pass} の値に設定する必要があります。
loginType	ObjectGrid クライアント・セキュリティーが使用可能であるとき、REST サービスが使用する認証のタイプ。ObjectGrid クライアント・セキュリティーが使用可能でないときは、このプロパティーは無視されます。 ObjectGrid クライアント・セキュリティーが使用可能で、loginType が「basic」に設定されている場合、REST サービスは以下を実行します。 • ObjectGrid クライアント・プロパティー・ファイルの「credentialGeneratorProps」プロパティーに指定された資格情報を使用して、サービス初期化時の ObjectGrid 操作を行う。 • HTTP 基本認証を使用して、要求ごとの ObjectGrid セッション操作を行う。 ObjectGrid クライアント・セキュリティーが使用可能で、loginType が「none」に設定されている場合、REST サービスは以下を実行します。 • ObjectGrid クライアント・プロパティー・ファイルの「credentialGeneratorProps」プロパティーに指定された資格情報を使用して、サービス初期化時の ObjectGrid 操作を行う。 • ObjectGrid クライアント・プロパティー・ファイルの「credentialGeneratorProps」プロパティーに指定された資格情報を使用して、要求ごとの ObjectGrid セッション操作を行う。

表 11. REST データ・サービスのプロパティー (続き)

property	説明
traceFile	トレース出力のリダイレクト先ファイルのオプションの名前。デフォルトは、logs/trace.log です。
traceSpec	eXtreme Scale ランタイム・サーバーが最初に使用するオプションのトレース仕様。デフォルトは、*=all=disabled です。 REST データ・サービス全体をトレースするには、ObjectGridRest*=all=enabled を使用します。
verboseOutput	true に設定すると、REST データ・サービス・クライアントは、障害発生時に追加診断情報を受け取ります。デフォルトは false です。機密情報が明らかになる可能性があるため、実稼働環境では、このオプションの値は false に設定する必要があります。
maxResultsPerCollection	1 つの照会で返される結果の、任意指定の最大件数。デフォルト値は無制限で、有効値は正の整数値です。
wxsRestAccessRightsFile	サービス・オペレーションと ObjectGrid エンティティーのアクセス権を指定する、eXtreme Scale REST サービス・アクセス権プロパティー・ファイルの、任意指定の名前。このプロパティーが指定されると、REST サービスは、指定されたパスから、もしくはクラスパスから、ファイルのロードを試行します。

WebSphere eXtreme Scale 構成

eXtreme Scale REST データ・サービスは、EntityManager API を使用して eXtreme Scale と対話します。eXtreme Scale グリッドにエンティティー・スキーマが定義され、そのエンティティーのメタデータが REST データ・サービスによって自動的に消費されます。エンティティー・スキーマの構成について詳しくは、エンティティー・スキーマの定義を参照してください。

例えば、以下のように、eXtreme Scale グリッドで Person を表すエンティティーを定義できます。

```
@Entity
public class Person {
  @Id String taxId;
  String firstName;
  String lastName;
}
```

ヒント: ここで使用されているアノテーションは、`com.ibm.websphere.projector.annotations` パッケージ内にあります。

REST サービスは、データ・サービス (EDMX) 文書用の ADO.NET エンティティー・データ・モデルを自動的に作成します。この文書は、$metadata URI を使用して利用できます。

```
http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/$metadata
```

eXtreme Scale グリッドを構成して実行中にした後には、eXtreme Scale クライアントを構成してパッケージする必要があります。eXtreme Scale REST データ・サービス・クライアント・パッケージの構成の詳細については、321 ページの『REST データ・サービスのインストール』 にあるパッケージ化およびデプロイメントの情報を参照してください。

エンティティー・モデル

WebSphere eXtreme Scale エンティティーは、エンティティー・アノテーションまたはエンティティー・メタデータ記述子ファイルを使用してモデル化されます。eXtreme Scale エンティティー・スキーマの構成方法について詳しくは、「*プログラミング・ガイド*」のエンティティー・スキーマthe の定義に関する情報を参照してください。eXtreme Scale REST サービスはエンティティー・メタデータを使用して、データ・サービスの EDMX モデルを自動的に作成します。

このバージョンの WebSphere eXtreme Scale REST データ・サービスには、以下のスキーマの制限があります。

- 区画化されたグリッド内のエンティティーを定義する際に、すべてのエンティティーは、ルート・エンティティーへの直接または間接の単一値アソシエーション (キー・アソシエーション) を持つ必要があります。 WCF データ・サービス・クライアント・ランタイムは、正規アドレスを使用して直接すべてのエンティティーにアクセスできる必要があります。そのため、区画ルーティングで使用されるルート・エンティティー (スキーマ・ルート) のキーは、子エンティティーのキーの一部である必要があります。

  以下に例を示します。

  ```
  @Entity(schemaRoot=true)
  public class Person {
      @Id String taxId;
      String firstName;
      String lastName;
      @OneToMany(mappedBy="person")
      List<Address> addresses;

  }

  @Entity
  public class Address {
      @Id int addrId;
      @Id @ManyToOne Person person;
      String street;
  }
  ```

- 双方向および単一方向のアソシエーションがサポートされます。ただし、単一方向アソシエーションは単一方向にのみナビゲートでき、Microsoft® 仕様ではすべてのアソシエーションが双方向である必要があるため、 Microsoft WCF Data Services クライアントからの単一方向アソシエーションは必ずしも機能するわけではありません。

- 参照制約はサポートされません。 eXtreme Scale ランタイムは、エンティティー間のキーを検証しません。エンティティー間のアソシエーションは、クライアントで管理する必要があります。
- 複合型はサポートされません。 eXtreme Scale EntityManager API は、組み込み可能属性をサポートしません。すべての属性は、単純型属性である必要があります (下にリストされた単純属性型を参照)。非単純型属性は、クライアントの観点から、バイナリー・オブジェクトとして扱われます。
- エンティティー継承はサポートされません。 eXtreme Scale EntityManager API は継承をサポートしません。
- メディア・リソースおよびメディア・リンクはサポートされません。データ・サービスの Conceptual Schema Definition Language Document 内の EntityType の HasStream 属性は使用されません。

EDM データ型と Java データ型間のマッピング

OData プロトコルは、抽象型システムで、以下のリストの Entity Data Model (EDM) 型を定義します。以下のトピックで、エンティティーで定義された基本型に基づいて eXtreme Scale REST アダプターが EDM 型を選択する方法について説明します。EDM 型の詳細については、MSDN Library: Abstract Type System を参照してください。

WCF Data Services では、以下の EDM 型が使用できます。

- Edm.Binary
- Edm.Boolean
- Edm.Byte
- Edm.DateTime
- Edm.Time
- Edm.Decimal
- Edm.Double
- Edm.Single
- Edm.Float
- Edm.Guid *
- Edm.Int16
- Edm.Int32
- Edm.Int64
- Edm.SByte
- Edm.String

EDM 型 Edm.Guid は、eXtreme Scale REST データ・サービスではサポートされません。

EDM 型への Java 型のマッピング

eXtreme Scale REST データ・サービスは、基本エンティティー型を EDM 型に自動的に変換します。$metadata URI を使用して Entity Data Model Extensions (EDMX)

メタデータ文書を表示することで、型マッピングを表示できます。 EDM 型は、クライアントがデータを読み取り、REST データ・サービスに書き込むために使用する型です。

表 12. EDM 型へマッピングされた Java 型： エンティティーで定義された Java 型が EDM データ型にマッピングされる様子が表に示されます。照会を使用してデータを取得する際に、データはこれらの型で表されます。

Java 型	**EDM 型**
boolean java.lang.Boolean	Edm.Boolean
byte java.lang.Byte	Edm.SByte
short java.lang.Short	Edm.Int16
int java.lang.Integer	Edm.Int32
long java.lang.Long	Edm.Int64
float java.lang.Float	Edm.Single
double java.lang.Double	Edm.Double
java.math.BigDecimal	Edm.Decimal
java.math.BigInteger	java.math.BigInteger
java.lang.String	Edm.String
char	char
java.lang.Character	java.lang.Character
Char[]	Char[]
java.lang.Character[]	java.lang.Character[]
java.util.Calendar	Edm.DateTime
java.util.Date	java.util.Date
java.sql.Date	java.sql.Date
java.sql.Timestamp	java.sql.Timestamp
java.sql.Time	java.sql.Time
その他の型	Edm.Binary

Java 型への EDM 型のマッピング

更新要求および挿入要求の場合、更新または eXtreme Scale REST データ・サービスに挿入されるデータがペイロードで指定されます。サービスは、互換性のあるデータ型を、EDMX 文書で定義されたデータ型に自動的に変換できます。 REST データ・サービスは、以下の 2 つのステップから成るプロセスを使用して、値の XML エンコード・ストリング表現を正しい型に変換します。

1. EDM 型が Java 型と互換性があることを確認するために、型検査が実行されます。 EDM 型によってサポートされるデータが Java によってサポートされるデータのサブセットである場合に、 EDM 型は Java 型と互換性があります。例えば、Edm.int32 型は Java long 型と互換性がありますが、 Edm.int32 型は Java short 型と互換性がありません。
2. ペイロードでストリング値を表すターゲットの Java 型オブジェクトが作成されます。

表 13. *Java 型と互換性がある EDM 型*

EDM 型	**Java 型**
Edm.Boolean	boolean java.lang.Boolean
Edm.SByte	byte java.lang.Byte short java.lang.Short int java.lang.Integer long java.lang.Long float java.lang.Float double java.lang.Double java.math.BigDecimal java.math.BigInteger char java.lang.Character

表 13. Java 型と互換性がある EDM 型 (続き)

EDM 型	**Java 型**
Edm.Byte, Edm.Int16	short java.lang.Short int java.lang.Integer long java.lang.Long float java.lang.Float double java.lang.Double java.math.BigDecimal java.math.BigInteger char java.lang.Character
Edm.Int32	int java.lang.Integer long java.lang.Long float java.lang.Float double java.lang.Double java.math.BigDecimal java.math.BigInteger
Edm.Int64	long java.lang.Long double java.lang.Double java.math.BigDecimal java.math.BigInteger

表 13. *Java 型と互換性がある EDM 型 (続き)*

EDM 型	**Java 型**
Edm.Double	double java.lang.Double java.math.BigDecimal
Edm.Decimal	double java.lang.Double java.math.BigDecimal java.math.BigInteger
Edm.Single	float java.lang.Float double java.lang.Double java.math.BigDecimal
Edm.String	java.lang.String char java.lang.Character Char[] java.lang.Character[] java.math.BigDecimal java.math.BigInteger
Edm.DateTime	java.util.Calendar java.util.Date java.sql.Date java.sql.Time java.sql.Timestamp
Edm.Time	java.sql.Time java.sql.Timestamp

時間型のマッピング

Java には、日付、時間、またはその両方を格納するための 5 つの時間型があります (java.util.Date、java.sql.Date、java.sql.Time、java.sql.Timestamp、および java.util.Calendar)。これらの型はすべて、エンティティー・データ・モデルでは Edm.DateTime として表されます。 eXtreme Scale REST サービスは、Java 型に応

じて、データを自動的に変換および正規化します。このトピックでは、開発者が時間型を使用する際に注意する必要があるいくつかの問題について説明します。

時間帯の差異

WCF Data Services では、Edm.DateTime 型での時間値の記述は常に協定世界時 (UTC) 標準を使用して表されます。協定世界時 (UTC) は、グリニッジ標準時 (GMT) の国際的に認識される名前です。協定世界時は、経度 0 度 (UTC 起点) で測定される時間です。夏時間調整は UTC には適用されません。

エンティティー型と EDM 型の変換

クライアントが要求を REST データ・サービスに送信する際には、日時は以下の例のように GMT 時間帯の時間として表されます。

```
"2000-02-29T21:30:30.654123456"
```

REST データ・サービスは、適切な Java 時間型インスタンスを構成してそのインスタンスをグリッドのエンティティーに挿入します。

eXtreme Scale REST データ・サービスに対して Java 時間型であるプロパティーをクライアントが要求した場合には、値は常に GMT 時間帯の値に正規化されます。例えば、エンティティー java.util.Date は以下のように構成されます。

```
Calendar c = Calendar.getInstance();
c.clear();
c.set(2000, 1, 29, 21, 30, 30);
Date d = c.getTime();
```

Calendar.getInstance() ではローカル時間帯を使用して Calendar オブジェクトが作成されるため、日時は Java プロセスのデフォルトの時間帯を使用して表されます。ローカル時間帯が CST である場合、REST データ・サービスから取得される際に日付は、その時間の GMT 表現 (`"2000-03-01T03:30:30"`) になります。

java.sql.Date の正規化

eXtreme Scale エンティティーは、Java type java.sql.Date で属性を定義できます。このデータ型には時間は含まれず、REST データ・サービスによって正規化されます。つまり、eXtreme Scale ランタイムは、java.sql.Date 属性に時、分、秒、ミリ秒の情報を格納しません。時間帯オフセットに関係なく、日付は常にローカルの日付で表されます。

例えば、クライアントが値「2009-01-01T03:00:00」で java.sql.Date プロパティーを更新した場合、 CST 時間帯 (-06:00) の REST データ・サービスは、時間をローカル CST 時間の「2009-01-01T00:00:00」に設定した java.sql.Date インスタンスを単に作成します。 java.sql.Date 値を作成するために時間帯の変換は行われません。REST サービス・クライアントがこの属性の値を取得したときに、「2009-01-01T00:00:00Z」と表示されます。時間帯の変換が行われていたならば、値の日付は「2008-12-31」と表示され、正しくなくなります。

java.sql.Time の正規化

java.sql.Date と同様に、java.sql.Time 値は正規化され、日付情報が含まれなくなります。つまり、eXtreme Scale ランタイムは、年、月、日を格納しません。時間はエポック時間 (1970 年 1 月 1 日) からの GMT 時間を使用して格納されます。これは java.sql.Time 実装と整合しています。

例えば、クライアントが値「2009-01-01T03:00:00」で java.sql.Time プロパティーを更新した場合、 REST データ・サービスは、ミリ秒を 3*60*60*1000 (3 時間) に設定して、java.sql.Time インスタンスを作成します。 REST サービス・クライアントが値を取得したときに、「1970-01-01:03:00:00Z」と表示されます。

アソシエーション

アソシエーションは、2 つの同位エンティティー間のリレーションシップを定義します。 eXtreme Scale REST サービスは、eXtreme Scale でアノテーションが付けられたエンティティーで定義されたエンティティーまたはエンティティー記述子 XML ファイルで定義されたエンティティーを使用してモデル化されたアソシエーションを反映します。

アソシエーションの保守

eXtreme Scale REST データ・サービスは、参照保全性制約をサポートしません。エンティティーが削除または追加されたときに参照が更新されるように、クライアントで保証する必要があります。アソシエーションのターゲット・エンティティーがグリッドから削除されたのにもかかわらず、ソース・エンティティーとターゲット・エンティティーとのリンクが削除されないと、リンク切れになります。eXtreme Scale REST データ・サービスおよび EntityManager API ではリンク切れが許容され、CWPRJ1022W 警告としてログに記録されます。壊れたアソシエーションは、要求ペイロードから単に削除されます。

バッチ要求を使用して、アソシエーションの更新を単一のトランザクションでグループ化することで、リンク切れを回避できます。バッチ要求の詳細については、セクションを参照してください。

ADO.NET Entity Data Model ReferentialConstraint エレメントは、eXtreme Scale REST データ・サービスでは使用されません。

アソシエーションの多重度

エンティティーには、多値アソシエーションまたは単一値アソシエーションを含めることができます。多値アソシエーション (コレクション) は、1 対多または多対多のアソシエーションです。単一値アソシエーションは、1 対 1 または多対 1 のアソシエーションです。

区画化されたグリッドでは、すべてのエンティティーに、ルート・エンティティーへの単一値キー・アソシエーションが含まれている必要があります。このトピックのもう 1 つのセクションでは、キー・アソシエーションを定義する方法を説明します。ルート・エンティティーを使用してエンティティーを区画化するため、区画化されたグリッドでは、多対多アソシエーションは使用できません。区画化されたグリッドの関係エンティティー・スキーマをモデル化する方法の例については、eXtreme Scale のスケーラブル・データ・モデルを参照してください。

以下の例で、アノテーションが付けられた Java クラスを使用してモデル化された EntityManager API アソシエーション型がどのように ADO.NET エンティティー・データ・モデルにマップされるのかを示します。

```
@Entity
public class Customer {
    @Id String customerId;
    @OneToOne TaxInfo taxInfo;
    @ManyToOne Address homeAddress;
    @OneToMany Collection<Order> orders;
    @ManyToMany Collection<SalesPerson> salespersons;
}
```

```
<Association Name="Customer_TaxInfo">
    <End Type="Model1.Customer" Role="Customer" Multiplicity="1" />
    <End Type="Model1.TaxInfo " Role="TaxInfo" Multiplicity="1" />
</Association>
<Association Name="Customer_Address">
    <End Type="Model1.Customer" Role="Customer" Multiplicity="1" />
    <End Type="Model1.Address" Role="TaxInfo" Multiplicity="*" />
</Association>
<Association Name="Customer_Order">
    <End Type="Model1.Customer" Role="Customer" Multiplicity="*" />
    <End Type="Model1.Order" Role="TaxInfo" Multiplicity="1" />
</Association>
<Association Name="Customer_SalesPerson">
    <End Type="Model1.Customer" Role="Customer" Multiplicity="*" />
    <End Type="Model1.SalesPerson" Role="TaxInfo" Multiplicity="*" />
</Association>
```

双方向アソシエーションおよび単一方向アソシエーション

エンティティー・アソシエーションは、単一方向または双方向にすることができます。 @OneToOne、@OneToMany、または @ManyToMany アノテーションで「mappedBy」属性を指定することで、または 1 対 1、1 対多、または多対多の XML 属性タグで「mapped-by」属性を指定することで、エンティティーは双方向になります。 OData プロトコルでは現在、すべてのエンティティーは双方向である必要があります。これにより、クライアントは、両方向にナビゲーション・パスを生成できます。 eXtreme Scale EntityManager API では、単一方向アソシエーションをモデル化できます。これにより、メモリーの節約とアソシエーションの保守の簡素化が実現します。単一方向アソシエーションを使用する場合は、 REST データ・サービス・クライアントは、必ず定義されたアソシエーションを使用してアソシエーションをナビゲートする必要があります。

例えば、Address と Country の間に多対 1 の単一方向アソシエーションを定義した場合には、以下の URI は許可されません。

```
/restservice/CustomerGrid/Country('USA')/addresses
```

キー・アソシエーション

単一値アソシエーション (1 対 1 および多対 1) は、エンティティー・キーのすべてまたは一部として組み込むこともできます。これは、キー・アソシエーションと呼ばれます。

キー・アソシエーションは、区画化されたグリッドを使用する際に必要になります。区画化されたエンティティー・スキーマ内のすべての子エンティティーで、キー・アソシエーションを定義する必要があります。 OData プロトコルでは、すべて

のエンティティーが直接アドレス可能である必要があります。つまり、子エンティティーのキーに、区画化で使用するキーを含める必要があります。

以下の例では、Customer は、Order に対する 1 対多のアソシエーションを持っています。 Customer エンティティーはルート・エンティティーであり、customerId 属性を使用してエンティティーを区画化しています。 Order には、ID の一部として Customer が組み込まれています。

```
@Entity(schemaRoot="true")
public class Customer {
    @Id String customerId;
    @OneToMany(mappedBy="customer") Order orders
}

@Entity
public class Order {
    @Id int orderId;
    @Id @ManyToOne Customer customer;
    java.util.Date orderDate;
}
```

REST データ・サービスがこのモデルの EDMX 文書を生成する際に、 Customer キー・フィールドが、Order エンティティーの一部として自動的に組み込まれます。

```
<EntityType Name="Order">
<Key>
    <PropertyRef Name="orderId"/>
    <PropertyRef Name="customer_customerId"/>
</Key>

<Property Name="orderId" Type="Edm.Int64" Nullable="false"/>
<Property Name="customer_customerId" Type="Edm.String"
    Nullable="false"/>
<Property Name="orderDate" Type="Edm.DateTime" Nullable="true"/>
<NavigationProperty Name="customer"
    Relationship="NorthwindGridModel.Customer_orders"
    FromRole="Order" ToRole="Customer"/>

<NavigationProperty Name="orderDetails"
    Relationship="NorthwindGridModel.Order_orderDetails"
    FromRole="Order" ToRole="OrderDetail"/>
</EntityType>
```

エンティティーの作成時に、キーを変更してはいけません。つまり、子エンティティーとその親とのキー・アソシエーションを変更する必要がある場合には、子エンティティーを削除して、異なる親を使用して再作成する必要があります。区画化されたグリッドでは、移動が複数の区画に関わる可能性が高いため、これには 2 つの異なるバッチ変更セットが必要になります。

操作のカスケード

EntityManager API では、柔軟なカスケード・ポリシーを使用できます。パーシスト、削除、無効化、またはマージの各操作をカスケードするように、アソシエーションにマークを付けることができます。このようなカスケード操作は、双方向アソシエーションの片側または両側で行うことができます。

OData プロトコルでは、アソシエーションの片側でのカスケード削除操作のみが許可されます。 CascadeType.REMOVE アノテーションおよび cascade-remove XML

属性は、1 対 1 双方向アソシエーションの両側または 1 対多アソシエーションの多側では定義できません。以下の例で、有効な Cascade.REMOVE 双方向アソシエーションを示します。

```
@Entity(schemaRoot="true")
public class Customer {
    @Id String customerId;
    @OneToMany(mappedBy="customer", cascade=CascadeType.REMOVE)
    Order orders
}

@Entity
public class Order {
    @Id int orderId;
    @Id @ManyToOne Customer customer;
    java.util.Date orderDate;
}
```

結果の EDMX アソシエーションは、以下のようになります。

```
<Association Name="Customer_orders">
    <End Type="NorthwindGridModel.Customer" Role="Customer"
        Multiplicity="1">
        <OnDelete Action="Cascade"/>
    </End>
    <End Type="NorthwindGridModel.Order" Role="Order"
        Multiplicity="*"/>
</Association>
```

REST データ・サービスの管理

このタスクについて

以下のリンクから、REST データ・サービスの管理に関する情報を参照してください。また、RestService Mbean 情報も参照してください。

REST データ・サービスのインストール

このトピックでは、WebSphere eXtreme Scale REST データ・サービスを Web サーバーにインストールする方法について説明します。

始める前に

ソフトウェア要件

eXtreme Scale REST データ・サービスは、Java Web アプリケーションであり、Java サーブレット仕様バージョン 2.3 および Java ランタイム環境バージョン 5 以上をサポートする任意のアプリケーション・サーバーにデプロイできます。

以下のソフトウェアが必要です。

- Java Standard Edition 5 以上

  制約事項: eXtreme Scale は Java Standard Edition 1.4 以上をサポートしますが、REST データ・サービスでは Java Standard Edition 5 以上が必要です。
- 以下のいずれかを含んだ、Web サーブレット・コンテナーのバージョン 2.3 以上
  - WebSphere Application Server バージョン 6.1.0.25 以上
  - WebSphere Application Server バージョン 7.0.0.5 以上

- WebSphere Community Edition バージョン 2.1.1.3 以上
- Apache Tomcat バージョン 5.5 以上

- eXtreme Scale バージョン 7.1 以上 (試用版を含む)

このタスクについて

eXtreme Scale REST データ・サービスには、単一の WAR ファイル wxsrestservice.war が含まれます。wxsrestservice.war には、WCF Data Services クライアント・アプリケーションまたはその他の HTTP REST クライアントと eXtreme Scale グリッド間のゲートウェイとして機能する単一のサーブレットが含まれています。

REST データ・サービスには、迅速に eXtreme Scale グリッドを作成し、eXtreme Scale クライアントまたは REST データ・サービスを使用してそのグリッドと対話できるようにするサンプルが含まれています。サンプルの使用の詳細については、REST データ・サービスのサンプルとチュートリアルを参照してください。

eXtreme Scale 7.1 をインストールするか、eXtreme Scale バージョン 7.1 試用版を解凍した場合には、以下のディレクトリーおよびファイルが含まれます。

- restservice_home/lib

  lib ディレクトリーには、以下のファイルが含まれます。

  - wxsrestservice.ear – WebSphere Application Server および WebSphere Application Server CE で使用するための REST データ・サービス・エンタープライズ・アプリケーション・アーカイブ。
  - wxsrestservice.war – Apache Tomcat で使用するための REST データ・サービス Web モジュール。

  wxsrestservice.ear ファイルには、wxsrestservice.war ファイルが含まれており、ともに WebSphere eXtreme Scale ランタイムに密結合されています。eXtreme Scale を新しいバージョンにアップグレードするかフィックスパックを適用した場合には、wxsrestservice.war ファイルまたは wxsrestservice.ear ファイルを、このディレクトリーにインストールされたバージョンに手動でアップグレードする必要があります。

- restservice_home/gettingstarted

  gettingstarted ディレクトリーには、eXtreme Scale REST データ・サービスを eXtreme Scale グリッドで使用する方法を説明する単純なサンプルが含まれます。

手順

REST データ・サービスをパッケージ化し、デプロイします。
REST データ・サービスは、必要なものを完備した WAR モジュールとして設計されています。 REST データ・サービスを構成するには、まず、REST データ・サービス構成およびオプションの eXtreme Scale 構成ファイルを JAR ファイルまたはディレクトリーにパッケージ化する必要があります。このアプリケーション・パッケージは、Web コンテナー・サービス・ランタイムによって参照されます。次の図に、eXtreme Scale REST データ・サービスで使用されるファイルを示します。

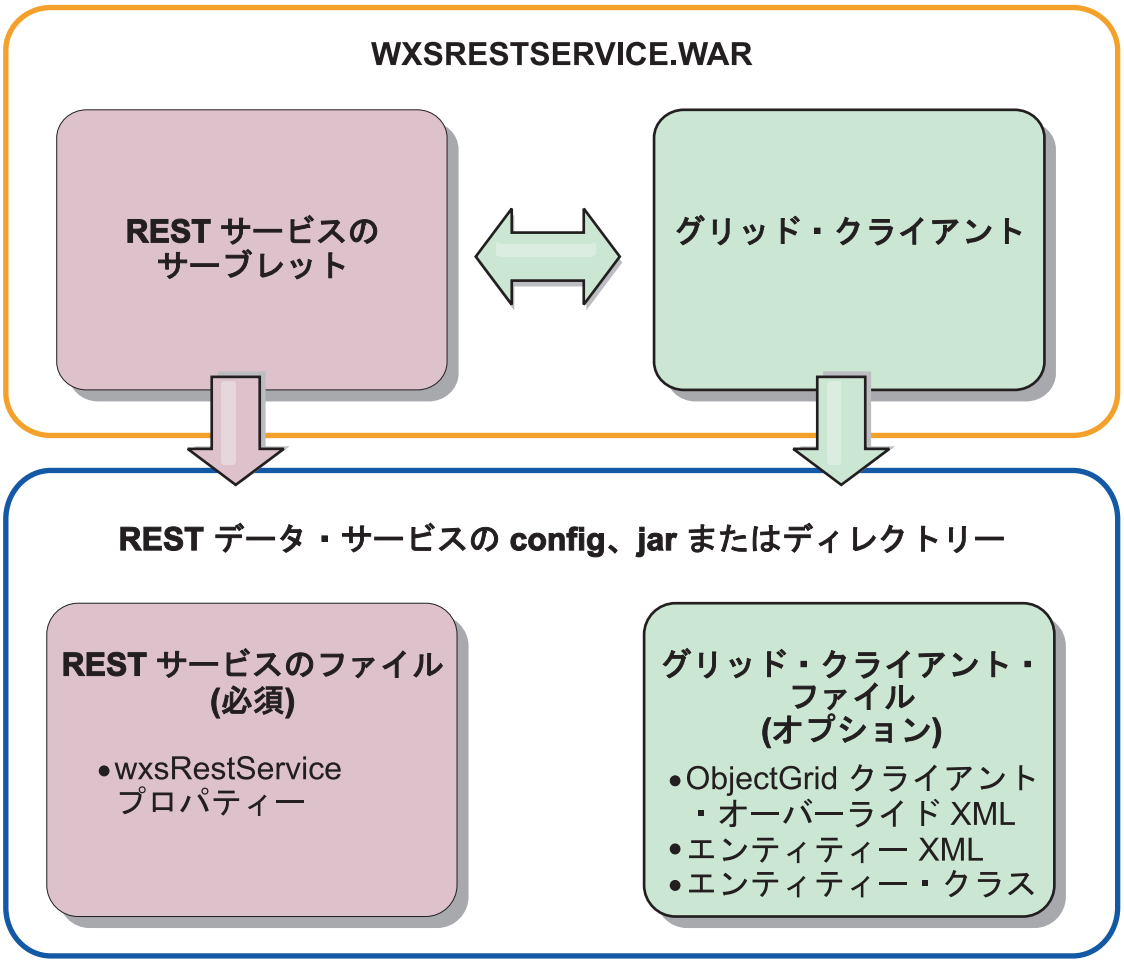


図 18. WebSphere eXtreme Scale REST データ・サービスのファイル

REST サービス構成 JAR またはディレクトリーには、以下のファイルが含まれている必要があります。

`wxsRestService.properties`: `wxsRestService.properties` ファイルには、REST データ・サービスの構成オプションが含まれます。これには、カタログ・サービス・エンドポイント、公開する ObjectGrid 名、トレース・オプションなどがあります。307 ページの『REST データ・サービスのプロパティー・ファイル』を参照してください。

以下の ObjectGrid クライアント・ファイルはオプションです。

- `META-INF/objectGridClient.xml`: ObjectGrid クライアント・オーバーライド XML ファイルは、リモート eXtreme Scale グリッドに接続するために使用します。デフォルトでは、このファイルは必要ではありません。このファイルが存在しない場合には、REST サービスはサーバー構成を使用して、ニア・キャッシュを使用不可にします。

  ファイルの名前は、objectGridClientXML REST データ・サービス構成プロパティーを使用してオーバーライドできます。この XML ファイルを提供する場合には、ファイルに以下を含める必要があります。

  1. REST データ・サービスに公開するすべての ObjectGrid。

2. 各 ObjectGrid 構成に関連付けられたエンティティー記述子 XML ファイルへの参照。

- *META-INF/エンティティー記述子 XML ファイル*: クライアントでクライアントのエンティティー定義をオーバーライドする必要がある場合にのみ、1 つ以上のエンティティー記述子 XML ファイルが必要です。エンティティー記述子 XML ファイルは、ObjectGrid クライアント・オーバーライド XML 記述子ファイルと組み合わせて使用する必要があります。

  eXtreme Scale 構成ファイルの詳細については、eXtreme Scale *管理ガイド*を参照してください。.

- **エンティティー・クラス**。アノテーションが付けられたエンティティー・クラスまたはエンティティー記述子 XML ファイルを使用して、エンティティー・メタデータを記述できます。 REST サービスでは、eXtreme Scale サーバーがエンティティー・メタデータ・クラスを使用して構成されていて、クライアント・オーバーライド・エンティティー XML 記述子を使用しない場合にのみ、クラスパス内にエンティティー・クラスが必要になります。

  エンティティーがサーバー上で XML で定義された、最小要件の構成ファイルを使用した例:

  ```
  restserviceconfig.jar:
  wxsRestService.properties
  ```

  プロパティー・ファイルには、以下が含まれます。

  ```
  catalogServiceEndPoints=localhost:2809
  objectGridNames=NorthwindGrid
  ```

  単一エンティティー、オーバーライド XML ファイル、およびエンティティー・クラスの例:

  ```
  restserviceconfig.jar:
  wxsRestService.properties
  ```

  プロパティー・ファイルには、以下が含まれます。

  ```
  catalogServiceEndPoints=localhost:2809
  objectGridNames=NorthwindGrid

  com/acme/entities/Customer.class
  META-INF/objectGridClient.xml
  ```

  クライアント ObjectGrid 記述子 XML ファイルには、以下が含まれます。

  ```
  <objectGrid name="CustomerGrid" entityMetadataXMLFile="emd.xml"/>
  META-INF/emd.xml
  ```

  エンティティー・メタデータ記述子 XML ファイルには、以下が含まれます。

  ```
  <entity class-name="com.acme.entities.Customer" name="Customer"/>
  ```

  EntityManager API、および eXtreme Scale クライアントとサーバーの構成の詳細については、*管理ガイド*を参照してください。

WebSphere Application Server への REST データ・サービスのデプロイ

このトピックでは、WebSphere Application Server または WebSphere Network Deployment バージョン 6.1.0.25 以上で eXtreme Scale REST データ・サービスを構成する方法について説明します。以下の説明は、WebSphere eXtreme Scale が WebSphere Application Server デプロイメントに統合されているデプロイメントにも適用されます。

始める前に

WebSphere eXtreme Scale の REST データ・サービスを構成およびデプロイするには、システムに以下の環境のいずれかが必要です。

- スタンドアロン eXtreme Scale クライアントを備えた WebSphere Application Server:
  - REST データ・サービスが付属した WebSphere eXtreme Scale 試用版バージョン 7.1 をダウンロードして解凍するか、累積フィックス 2 を適用した WebSphere eXtreme Scale 7.1.0.0 製品をスタンドアロン・ディレクトリーにインストールします。
  - WebSphere Application Server バージョン 6.1.0.25 または 7.0.0.5 以上をインストールして実行します。
- WebSphere eXtreme Scale が統合された WebSphere Application Server:

  累積フィックス 2 を適用した WebSphere eXtreme Scale バージョン 7.1.0.0 を、WebSphere Application Server バージョン 6.1.0.25 または 7.0 (以上) 上にインストールします。

  ヒント: eXtreme Scale REST データ・サービスには、eXtreme Scale クライアント・オプションのインストールのみが必要です。プロファイルを拡張する必要はありません。

  WebSphere Application Server インフォメーション・センターで、Java 2 セキュリティーを使用可能にする方法について参照してください。

手順

1. eXtreme Scale グリッドを構成して開始します。
   a. REST データ・サービスとともに使用するための eXtreme Scale グリッドの構成の詳細については、105 ページの『第 6 章 デプロイメント環境の構成』を参照してください。
   b. eXtreme Scale クライアントがグリッド内のエンティティーに接続およびアクセスできることを検査します。例えば、本書の「始めに」セクションを参照してください。
2. eXtreme Scale REST サービス構成 JAR またはディレクトリーをビルドします。321 ページの『REST データ・サービスのインストール』の REST サービスのパッケージ化およびデプロイに関する情報を参照してください。
3. REST データ・サービス構成 JAR またはディレクトリーをアプリケーション・サーバー・クラスパスに追加します。
   a. WebSphere 管理コンソールを開きます。

b. 「**環境**」 → 「**共有ライブラリー**」にナビゲートします。
c. 「**新規**」をクリックします。
d. 以下の項目を該当するフィールドに追加します。
   - 名前: `extremescale_rest _configuration`
   - クラスパス: <REST サービス構成 JAR またはディレクトリー>
e. 「**OK**」をクリックします。
f. 変更をマスター構成に保存します。

4. eXtreme Scale が WebSphere Application Server インストール済み環境に統合されている場合は、このステップをスキップして、ステップ 5 に進みます。それ以外の場合は、続けて以下を行います。

   以下のように、WebSphere eXtreme Scale クライアント・ランタイム JAR の wsogclient.jar および REST データ・サービス構成 JAR またはディレクトリーをアプリケーション・サーバー・クラスパスに追加します。
   a. WebSphere 管理コンソールを開きます。
   b. 「**環境**」 → 「**共有ライブラリー**」にナビゲートします。
   c. 「**新規**」をクリックします。
   d. 以下の項目をフィールドに追加します。
      - 名前: `extremescale_client_v71`
      - クラスパス: `wxs_home/lib/wsogclient.jar`
   e. 「OK」をクリックします。
   f. 変更をマスター構成に保存します。
5. WebSphere 管理コンソールを使用して、 REST データ・サービスの EAR ファイル wxsrestservice.ear を WebSphere Application Server にインストールします。
   a. WebSphere 管理コンソールを開きます。
   b. 「アプリケーション」->「新規アプリケーション」にナビゲートします。
   c. ファイル・システム上の /lib/wxsrestservice.ear ファイルを参照して選択し、「**次へ**」をクリックします。
      - WebSphere Application Server バージョン 7.0 を使用している場合は、「次へ」をクリックします。
      - WebSphere Application Server バージョン 6.1 を使用している場合は、名前 /wxsrestservice でコンテキスト・ルート値を入力して、次のステップに進みます。
   d. 詳細インストール・オプションを選択して、「次へ」をクリックします。
   e. アプリケーション・セキュリティー警告画面で、「続行」をクリックします。
   f. デフォルト・インストール・オプションを選択して、「次へ」をクリックします。
   g. アプリケーションのマップ先サーバーを選択して、「次へ」をクリックします。
   h. JSP 再ロード・ページで、デフォルトを使用し、「次へ」をクリックします。

i. 共有ライブラリー・ページで、「wxsrestservice.war」モジュールを、ステップ 3 および 4 で定義した以下の共有ライブラリーにマップします。

- extremescale_rest _configuration
- extremescale_client_v71

ヒント: この共有ライブラリーは、eXtreme Scale が WebSphere Application Server に統合されていない場合にのみ必要です。

j. マップ共有ライブラリー・リレーションシップ・ページで、デフォルトを使用し、「次へ」をクリックします。

k. マップ仮想ホスト・ページで、デフォルトを使用し、「次へ」をクリックします。

l. マップ・コンテキスト・ルート・ページで、コンテキスト・ルートを wxsrestservice に設定します。

m. 要約画面で、「終了」をクリックして、インストールを完了します。

n. 変更をマスター構成に保存します。

6. 「wxsrestservice」eXtreme Scale REST データ・サービス・アプリケーションを開始します。

a. アプリケーションを選択します。

- WebSphere Application Server バージョン 7.0 を使用している場合には、管理コンソールで、「**アプリケーション**」 → 「**アプリケーション・タイプ**」 → 「**WebSphere アプリケーション**」をクリックします。
- WebSphere Application Server バージョン 6.1 を使用している場合には、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. 「wxsrestservice」アプリケーションの隣にあるチェック・ボックスにチェック・マークを付け、「**開始**」をクリックします。

c. アプリケーション・サーバー・プロファイルの SystemOut.log を確認します。 REST データ・サービスが正常に開始すると、サーバー・プロファイルの SystemOut.log 内に、以下のメッセージが表示されます。

```
CWOBJ4000I: WebSphere eXtreme Scale REST データ・サービスが開始されました。
```

7. 以下のように、REST データ・サービスが動作していることを確認します。 ポート番号は、アプリケーション・サーバー・プロファイル・ログ・ディレクトリー内にある SystemOut.log で、メッセージ ID の SRVE0250I で表示されている最初のポートを見ることで確認できます。デフォルト・ポートは 9080 です。

例: http://localhost:9080/wxsrestservice/restservice/NorthwindGrid/ 結果: AtomPub サービス文書が表示されます。

WebSphere Application Server Community Edition への REST データ・サービスのデプロイ

このトピックでは、WebSphere Application Server Community Edition バージョン 2.1.1.3 以降で eXtreme Scale REST データ・サービスを構成する方法について説明します。

始める前に

- IBM (推奨) または Sun の JRE か JDK のバージョン 5 以上をインストールして、 JAVA_HOME 環境変数を設定します。
- WebSphere Application Server Community Edition バージョン 2.1.1.3 以降をダウンロードして、 wasce_root ディレクトリー (例えば、/opt/IBM/wasce ディレクトリー) にインストールします。 バージョン 2.1.1 またはその他のバージョンについて、インストールの指示を参照してください。
- REST データ・サービスが付属した eXtreme Scale 試用版バージョン 7.1 をダウンロードして解凍するか、累積フィックス 2 を適用した WebSphere eXtreme Scale 7.1.0.0 製品をスタンドアロン・ディレクトリーにインストールします。

手順

1. eXtreme Scale グリッドを構成して開始します。
   a. REST データ・サービスとともに使用するための eXtreme Scale グリッドの構成の詳細については、 105 ページの『第 6 章 デプロイメント環境の構成』を参照してください。
   b. eXtreme Scale クライアントがグリッド内のエンティティーに接続およびアクセスできることを検査します。例えば、REST データ・サービスのサンプルとチュートリアルを参照してください。
2. eXtreme Scale REST サービス構成 JAR またはディレクトリーをビルドします。詳細については、 321 ページの『REST データ・サービスのインストール』のトピックのパッケージ化とデプロイメントの情報を参照してください。
3. WebSphere Application Server Community Edition サーバーを始動します。
   a. Java SE セキュリティーを使用可能にせずにサーバーを始動するには、以下のコマンドを実行します。

      UNIX Linux
      ```
      wasce_root/bin/startup.sh
      ```

      Windows
      ```
      wasce_root/bin/startup.bat
      ```

   b. Java SE セキュリティーを使用可能にしてサーバーを始動するには、以下のステップに従います。 UNIX Linux
      1) コマンド行または端末ウィンドウを開いて、以下のコピー・コマンドを実行 (または、指定したポリシー・ファイルを既存ポリシーにコピー) します。 `cp restservice_home/gettingstarted/wasce/geronimo.policy wasce_root/bin`
      2) `wasce_root/bin/setenv.sh` ファイルを編集します。
      3) 「`WASCE_JAVA_HOME=`」が含まれている行の後に、以下を追加します。
      ```
      export JAVA_OPTS="-Djava.security.manager
      -Djava.security.policy=geronimo.policy"
      ```

      Windows

      1) コマンド行ウィンドウを開いて、以下のコピー・コマンドを実行 (または、指定したポリシー・ファイルを既存ポリシーにコピー) します。

      ```
      copy restservice_home¥gettingstarted¥wasce¥geronimo.policy¥bin
      ```

2) wasce_root¥bin¥setenv.bat ファイルを編集します。

3) 「set WASCE_JAVA_HOME=」が含まれている行の後に、以下を追加します。

```
set JAVA_OPTS="-Djava.security.manager
-Djava.security.policy=geronimo.policy"
```

4. 以下のように、ObjectGrid クライアント・ランタイム JAR を WebSphere Application Server Community Edition リポジトリーに追加します。
   a. WebSphere Application Server Community Edition 管理コンソールを開いてログインします。デフォルト URL は http://localhost:8080/console であり、デフォルト・ユーザー ID は「system」で、パスワードは「manager」です。
   b. コンソール・ウィンドウの左側で、「**サービス**」フォルダー内の「**リポジトリー**」リンクをクリックします。
   c. 「**リポジトリーへのアーカイブの追加**」セクションで、入力テキスト・ボックスに以下を入力します。

表 14. リポジトリーへのアーカイブの追加

テキスト・ボックス	値
ファイル	wxs_home/lib/ogclient.jar
グループ	com.ibm.websphere.xs
成果物	ogclient
バージョン	7.1
タイプ	JAR

   d. 「インストール」ボタンをクリックします。

   クラスおよびライブラリーの依存関係を構成するさまざまな方法の詳細については、技術情報 Specifying external dependencies to applications running on WebSphere Application Server Community Edition を参照してください。
5. REST データ・サービス・モジュール wxsrestservice.war ファイルを WebSphere Application Server Community Edition サーバーにデプロイして開始します。
   a. サンプル・デプロイメント・プラン XML ファイル restservice_home/gettingstarted/wasce/geronimo-web.xml をコピーして、REST データ・サービス構成 JAR またはディレクトリーへのパス依存関係を組み込むように編集します。wxsRestService.properties ファイルおよび他の構成ファイルならびにメタデータ・クラスを組み込むようにするクラスパス設定の例については、セクションを参照してください。
   b. WebSphere Application Server Community Edition 管理コンソールを開いてログインします。

      ヒント: デフォルト URL は http://localhost:8080/console です。デフォルト・ユーザー ID は「system」で、パスワードは「manager」です。
   c. コンソール・ウィンドウの左側にある「**新規デプロイ**」リンクをクリックします。

d. 「**新規アプリケーションのインストール**」ページで、テキスト・ボックスに以下の値を入力します。

表 15. 新規アプリケーションのインストール

テキスト・ボックス	値
アーカイブ	restservice_home/lib/wxsrestservice.war
プラン	restservice_home/gettingstarted/wasce/geronimo-web.xml

ヒント: ステップ 3 でコピーして編集した `geronimo-web.xml` ファイルへのパスを使用します。

e. 「インストール」ボタンをクリックします。コンソール・ページに、アプリケーションが正常にインストールされて開始されたことが示されます。

f. WebSphere Application Server Community Edition システム出力ログまたはコンソールで以下のメッセージが存在することを確認して、REST データ・サービスが正常に開始されていることを検査します。

```
CWOBJ4000I: WebSphere eXtreme Scale REST データ・サービスが開始されました。
```

6. 以下のコマンドを実行して、WebSphere Application Server Community Edition サーバーを始動します。

- UNIX Linux `wasce_root/bin/startup.sh`
- Windows `wasce_root/bin/startup.bat`

7. 以下のように、eXtreme Scale REST データ・サービスおよび提供サンプルを WebSphere Application Server Community Edition サーバーにインストールします。

a. 以下のように、ObjectGrid クライアント・ランタイム JAR を WebSphere Application Server Community Edition リポジトリーに追加します。

1) WebSphere Application Server Community Edition 管理コンソールを開いてログインします。(デフォルト設定は http://localhost:8080/console/ (ユーザー ID は system、パスワードは manager) です。)

2) コンソール・ウィンドウの左側で、「サービス」フォルダー内の「**リポジトリー**」リンクをクリックします。

3) 「**リポジトリーへのアーカイブの追加**」セクションで、入力テキスト・ボックスに以下を入力します。

表 16. リポジトリーへのアーカイブの追加

テキスト・ボックス	値
ファイル	wxs_home/lib/ogclient.jar
グループ	com.ibm.websphere.xs
成果物	ogclient
バージョン	7.1
タイプ	JAR

4) 「インストール」ボタンをクリックします。

ヒント: クラスおよびライブラリーの依存関係を構成するさまざまな方法の詳細については、技術情報 Specifying external dependencies to applications running on WebSphere Application Server Community Edition を参照してください。

b. REST データ・サービス・モジュール `wxsrestservice.war` を WebSphere Application Server Community Edition サーバーにデプロイします。

1) 開始用 (getting started) サンプル・クラスパス・ディレクトリーへのパス依存関係を組み込むように、サンプルの `restservice_home/gettingstarted/wasce/geronimo-web.xml` デプロイメント XML ファイルを編集します。

- 2 つの開始用 (getting started) クライアント GBean の「classesDirs」を変更します。

GettingStarted_Client_SharedLib GBean の「classesDirs」パスを `restservice_home/gettingstarted/restclient/bin` に設定する必要があります。

GettingStarted_Common_SharedLib GBean の「classesDirs」パスを `restservice_home/gettingstarted/common/bin` に設定する必要があります。

2) WebSphere Application Server Community Edition 管理コンソールを開いてログインします。

3) コンソール・ウィンドウの左側にある「**新規デプロイ**」リンクをクリックします。

4) 「**新規アプリケーションのインストール**」ページで、テキスト・ボックスに以下の値を入力します。

表 17. 新規アプリケーションのインストール

テキスト・ボックス	**値**
アーカイブ	*restservice_home*/lib/wxsrestservice.war
プラン	*restservice_home*/gettingstarted/wasce/geronimo-web.xml

5) 「**インストール**」ボタンをクリックします。

コンソール・ページに、アプリケーションが正常にインストールされて開始されたことが示されます。

6) WebSphere Application Server Community Edition システム出力ログで以下のメッセージが存在することを確認して、REST データ・サービスが正常に開始されていることを検査します。

`CWOBJ4000I: WebSphere eXtreme Scale REST データ・サービスが開始されました。`

8. 以下のように、REST データ・サービスが動作していることを確認します。

Web ブラウザーを開いて、URL `http://<host>:<port>/<context root>/restservice/<Grid Name>` にナビゲートします。

WebSphere Application Server Community Edition のデフォルト・ポートは 8080 で、/var/config/config-substitutions.properties ファイルで「HTTPPort」プロパティーを使用して定義されます。

例: http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/

タスクの結果

AtomPub サービス文書が表示されます。

Apache Tomcat への REST データ・サービスのデプロイ

このトピックでは、WebSphere eXtreme Scale REST データ・サービスを Apache Tomcat バージョン 5.5 以上で構成する方法について説明します。

このタスクについて

- IBM または Sun の JRE か JDK のバージョン 5 以上をインストールして、JAVA_HOME 環境変数を設定します。
- Apache Tomcat バージョン 5.5 以上をインストールします。 Tomcat のインストール方法の詳細については、Apache Tomcat を参照してください。
- REST データ・サービスが付属した eXtreme Scale 試用版バージョン 7.1 をダウンロードして解凍するか、累積フィックス 2 を適用した WebSphere eXtreme Scale バージョン 7.1.0.0 製品をスタンドアロン・ディレクトリーにインストールします。

手順

1. Sun JRE または JDK を使用する場合は、IBM ORB を Tomcat にインストールします。
   a. Tomcat バージョン 5.5:

      すべての JAR ファイルを以下のようにコピーします。

      wxs_home/lib/endorsed ディレクトリー

      から

      tomcat_root/common/endorsed ディレクトリーに
   b. Tomcat バージョン 6.0:

      以下のように、「endorsed」ディレクトリーを作成します。

      UNIX Linux
      ```
      mkdir tomcat_root/endorsed
      ```

      Windows
      ```
      md tomcat_root/endorsed
      ```

      すべての JAR ファイルを以下のようにコピーします。

      ```
      wxs_home/lib/endorsed
      ```

      から

      ```
      tomcat_root/common/endorsed
      ```

2. eXtreme Scale グリッドを構成して開始します。
   a. REST データ・サービスとともに使用するための eXtreme Scale グリッドの構成の詳細については、 105 ページの『第 6 章 デプロイメント環境の構成』を参照してください。
   b. eXtreme Scale クライアントがグリッド内のエンティティーに接続およびアクセスできることを検査します。例えば、REST データ・サービスのサンプルとチュートリアルを参照してください。
3. eXtreme Scale REST サービス構成 JAR またはディレクトリーをビルドします。詳細については、 321 ページの『REST データ・サービスのインストール』のパッケージ化とデプロイメントの情報を参照してください。
4. REST データ・サービス・モジュール wxsrestservice.war を Tomcat サーバーにデプロイします。

   wxsrestservice.war ファイルを以下のようにコピーします。

   `restservice_home/lib`

   から

   `tomcat_root/webapps`
5. ObjectGrid クライアント・ランタイム JAR およびアプリケーション JAR を Tomcat の共有クラスパスに追加します。
   a. `tomcat_root/conf/catalina.properties` ファイルを編集します。
   b. 以下の各パス名をコンマで区切って、shared.loader プロパティーの末尾に追加します。
      - `wxs_home/lib/ogclient.jar`
      - `restservice_home/gettingstarted/restclient/bin`
      - `restservice_home/gettingstarted/common/bin`
6. Java 2 セキュリティーを使用している場合は、以下のように、Tomcat ポリシー・ファイルにセキュリティー許可を追加します。
   - Tomcat バージョン 5.5 を使用している場合:

     `restservice_home/gettingstarted/tomcat/catalina-5_5.policy` にあるサンプルの 5.5 `catalina` ポリシー・ファイルのコンテンツを `tomcat_root/conf/catalina.policy` ファイルにマージします。
   - Tomcat バージョン 6.0 を使用している場合:

     `restservice_home/gettingstarted/tomcat/catalina-6_0.policy` にあるサンプルの 6.0 `catalina` ポリシー・ファイルのコンテンツを `tomcat_root/conf/catalina.policy` ファイルにマージします。
7. 以下のように、Tomcat サーバーを始動します。
   - **Tomcat 5.5 を UNIX または Windows で、あるいは Tomcat 6.0 ZIP 配布版を使用する場合:**
     a. `cd tomcat_root/bin`
     b. 以下のように、サーバーを始動します。
        – Java 2 セキュリティーを使用可能にしていない場合:

UNIX Linux `./catalina.sh run`

Windows `catalina.bat run`

– Java 2 セキュリティーを使用可能にしている場合:

UNIX Linux `./catalina.sh run -security`

Windows `catalina.bat run -security`

c. Apache Tomcat のログは、コンソールに表示されます。 REST データ・サービスが正常に開始すると、管理コンソールに以下のメッセージが表示されます。

```
CWOBJ4000I: WebSphere eXtreme Scale REST データ・サービスが開始されました。
```

- **Windows インストーラー用配布版を使用して、Tomcat 6.0 を Windows で使用する場合:**

a. cd /bin

b. 以下のように、Apache Tomcat 6 構成ツールを開始します。

```
tomcat6w.exe
```

c. Java 2 セキュリティーを使用可能にする場合は、以下のようにします (オプション)。

Apache Tomcat 6 のプロパティー・ウィンドウの Java タブの Java オプションに以下の項目を追加します。

```
-Djava.security.manager

-Djava.security.policy=¥conf¥catalina.policy
```

d. Apache Tomcat 6 のプロパティー・ウィンドウの「Start」ボタンをクリックして、Tomcat サーバーを始動します。

e. 以下のログを確認して、Tomcat サーバーが正常に始動していることを確認します。

– `tomcat_root/bin/catalina.log`

Tomcat サーバー・エンジンの状況を表示します。

– `tomcat_root/bin/stdout.log`

システム出力ログを表示します。

f. REST データ・サービスが正常に開始すると、システム出力ログに以下のメッセージが表示されます。

```
CWOBJ4000I: WebSphere eXtreme Scale REST データ・サービスが開始されました。
```

8. 以下のように、REST データ・サービスが動作していることを確認します。

Web ブラウザーを開いて、以下の URL にナビゲートします。

`http://host:port/context_root/restservice/grid_name`

Tomcat のデフォルト・ポートは 8080 であり、`tomcat_root/conf/server.xml` ファイル内の <Connector> エレメントで構成されます。

例:

`http://localhost:8080/wxsrestservice/restservice/NorthwindGrid/`

タスクの結果

AtomPub サービス文書が表示されます。

REST データ・サービスの保護

REST データ・サービスの複数の側面を保護します。認証および許可を使用して、eXtreme Scale REST データ・サービスへのアクセスを保護できます。また、アクセス規則として知られるサービス・スコープ構成規則によって、アクセスを制御することもできます。 3 番目のセキュリティーとして、トランスポート・セキュリティーを考慮します。

このタスクについて

認証および許可を使用して、eXtreme Scale REST データ・サービスへのアクセスを保護できます。認証および許可は、eXtreme Scale セキュリティーとの統合を伴います。

また、アクセス規則として知られるサービス・スコープ構成規則によってアクセスを制御することもできます。 2 つのタイプのアクセス規則が存在します。1 つはサービスによって許可される CRUD 操作を制御するサービス操作アクセス権限で、もう 1 つは特定のエンティティー・タイプに対して許可される CRUD 操作を制御するエンティティー・アクセス権限です。

トランスポート・セキュリティーは、Web クライアントの場合は REST サービス接続に対してホスティング・コンテナー構成によって提供され、 REST サービスの場合は eXtreme Scale グリッド接続に対して eXtreme Scale クライアント構成によって提供されます。

手順

- 認証および許可を制御します。

  認証および許可を使用して、eXtreme Scale REST データ・サービスへのアクセスを保護できます。認証および許可は、eXtreme Scale セキュリティーとの統合を伴います。

  eXtreme Scale REST データ・サービスは、 eXtreme Scale セキュリティー (認証および許可) を使用して、サービスにアクセスできるユーザーおよびサービスでユーザーが実行できる操作を制御します。 eXtreme Scale REST データ・サービスは、構成されたグローバル資格情報 (ユーザーとパスワード) を使用するか、各トランザクションで、認証および許可が実行される eXtreme Scale グリッドに送信される HTTP BASIC チャレンジから得た資格情報を使用します。

1. グリッドで、eXtreme Scale クライアント認証および許可を構成します。 eXtreme Scale クライアント認証および許可を構成する方法の詳細については、 397 ページの『外部プロバイダーとのセキュリティー統合』を参照してください。
2. (REST サービスで使用する) eXtreme Scale クライアントでセキュリティーを構成します。

   eXtreme Scale REST データ・サービスは、 eXtreme Scale グリッドとの通信時に eXtreme Scale クライアント・ライブラリーを呼び出します。そのため、eXtreme Scale クライアントで eXtreme Scale セキュリティーを構成する必要があります。

   eXtreme Scale クライアント認証は、objectgrid クライアント・プロパティー・ファイル内のプロパティーで使用可能にします。 REST サービスでクライアント・セキュリティーを使用する場合には、最低でも以下の属性を使用可能にする必要があります。

   ```
   securityEnabled=true
   credentialAuthentication=Supported [-or-] Required
   credentialGeneratorProps=user:pass [-or-] {xor encoded user:pass}
   ```

   なお、credentialGeneratorProps プロパティーに指定するユーザーとパスワードは、認証レジストリーの ID にマップする必要があります。また、ObjectGrid に接続するため、および ObjectGrid を作成するために十分な ObjectGrid ポリシー・アクセス権を備えている必要があります。

   サンプル ObjectGrid クライアント・ポリシー・ファイルは、 `wxsrest_home/security/security.ogclient.properties` にあります。 216 ページの『クライアント・プロパティー・ファイル』も参照してください。
3. eXtreme Scale REST データ・サービスでセキュリティーを構成します。

   eXtreme Scale セキュリティーと統合するには、 eXtreme Scale REST データ・サービス構成プロパティー・ファイルには、以下の項目が含まれている必要があります。

   ```
   ogClientPropertyFile=file_name
   ```

   ogClientPropertyFile は、前のステップで言及した ObjectGrid クライアント・プロパティーが入っているプロパティー・ファイルの場所です。 REST サービスはこのファイルを使用して、セキュリティーが使用可能になっている場合にグリッドに通信する eXtreme Scale クライアントを初期設定します。

   ```
   loginType=basic [-or-] none
   ```

   loginType プロパティーは、REST サービスでログイン・タイプを構成します。値「none」を指定すると、credentialGeneratorProps で定義された「グローバル」ユーザー ID とパスワードが、各トランザクションでグリッドに送信されます。値「basic」を指定すると、REST サービスは HTTP BASIC チャレンジをクライアントに提示して、グリッドとの通信時に各トランザクションで送信する資格情報を要求します。

ogClientPropertyFile プロパティーおよび loginType プロパティーの詳細については、 307 ページの『REST データ・サービスのプロパティー・ファイル』を参照してください。

- アクセス規則を適用します。

  また、アクセス規則として知られるサービス・スコープ構成規則によってアクセスを制御することもできます。 2 つのタイプのアクセス規則が存在します。1 つはサービスによって許可される CRUD 操作を制御するサービス操作アクセス権限で、もう 1 つは特定のエンティティー・タイプに対して許可される CRUD 操作を制御するエンティティー・アクセス権限です。

  eXtreme Scale REST データ・サービスでは、オプションとして、サービスおよびサービス内のエンティティーに対するアクセスを制限するために構成可能なアクセス規則を使用できます。これらのアクセス規則は、REST サービスのアクセス権限プロパティー・ファイルで指定します。セクション 5.1 で説明しているように、このファイルの名前は、REST データ・サービスのプロパティー・ファイル内で、「wxsRestAccessRightsFile」プロパティーを使用して指定します。このファイルは通常、キーと値のペアが含まれた Java プロパティー・ファイルです。 2 つのタイプのアクセス規則が存在します。1 つはサービスによって許可される CRUD 操作を制御するサービス操作アクセス権限で、もう 1 つは特定のエンティティー・タイプに対して許可される CRUD 操作を制御するエンティティー・アクセス権限です。

  1. サービス操作権限を構成します。

     サービス操作権限では、REST サービスで公開するすべての ObjectGrid または指定した個別 ObjectGrid のすべてのエンティティーに適用するアクセス権限を指定します。

     以下の構文を使用します。

     ```
     serviceOperationRights=service_operation_right
     serviceOperationRights.grid_name -OR- *=service_operation_right
     ```

     説明:

     - serviceOperationRights には、
       NONE、READSINGLE、READMULTIPLE、ALLREAD、ALL のいずれかを指定できます。
     - serviceOperationRights.*grid_name -OR-* * は、アクセス権限がすべての ObjectGrid に適用されることを暗黙指定します。また、特定の ObjectGrid の名前を指定することもできます。

     例:

     ```
     serviceOperationsRights=ALL
     serviceOperationsRights.*=NONE
     serviceOperationsRights.EMPLOYEEGRID=READSINGLE
     ```

     最初の例では、この REST サービスで公開されるすべての ObjectGrid ですべてのサービス操作を許可することを指定しています。 2 番目の例では、最初の例と同様に、REST サービスによって公開されるすべての ObjectGrid に適用しています。ただし、アクセス権限を NONE として指定しており、ObjectGrid ではどのサービス操作も許可されません。最後の例では、特定のグ

リッドのサービス操作を制御する方法を示しています。この例では、EMPLOYEEGRID のすべてのエンティティーに対して、結果が単一のレコードになる読み取りのみが許可されます。

REST サービスで想定されるデフォルトは serviceOperationsRights=ALL であり、このサービスで公開されるすべての ObjectGrid ですべての操作が許可されます。これは、デフォルトが NONE で、REST サービスでどの操作も許可されない Microsoft の実装とは異なります。

注: サービス操作権限は、このファイルで指定された順序で評価されます。そのため、最後に指定された権限によって、それより前の権限がオーバーライドされます。

2. エンティティー・アクセス権限を構成します。

エンティティー・セット権限は、REST サービスで公開される特定の ObjectGrid エンティティーに適用するアクセス権限を指定します。この権限により、サービス操作権限と比較して、個別 ObjectGrid エンティティーに対するアクセス制御を厳格化および詳細化できます。

以下の構文を使用します。

```
entitySetRights.grid_name.entity_name=entity_set_right
```

説明:

– *entity_set_right* には、以下の権限のいずれかを指定できます。

表 18. エンティティー・アクセス権限： サポートされる値。

アクセス権限	説明
NONE	データにアクセスするためのすべての権限を拒否します。
READSINGLE	単一のデータ項目の読み取りを許可します。
READMULTIPLE	データ・セットの読み取りを許可します。
ALLREAD	単一データ/複数のデータ・セットの読み取りを許可します。
WRITEAPPEND	データ・セットでの新規データ項目の作成を許可します。
WRITEREPLACE	データの置換を許可します。
WRITEDELETE	データ・セットからのデータ項目の削除を許可します。
WRITEMERGE	データのマージを許可します。
ALLWRITE	データの書き込み (つまり、作成、置換、マージ、削除) を許可します。
ALL	データの作成、読み取り、更新、および削除を許可します。

– *entity_name* は、REST サービス内の特定の ObjectGrid の名前です。

– *grid_name* は、指定した ObjectGrid 内の特定のエンティティーの名前です。

注: それぞれの ObjectGrid およびそのエンティティーに対してサービス操作権限とエンティティー・セット権限の両方を指定した場合は、以下の例に示すように、制限の厳しい方の権限が適用されます。なお、エンティティー・セット権限は、ファイル内で指定された順序で評価されます。最後に指定された権限によって、それより前の権限がオーバーライドされます。

例 1: serviceOperationsRights.NorthwindGrid=READSINGLE と entitySetRights.NorthwindGrid.Customer=ALL を指定した場合。 Customer エンティティーには、READSINGLE が適用されます。

例 2: serviceOperationsRights.NorthwindGrid=ALLREAD を指定し、entitySetRights.NorthwindGrid.Customer=ALLWRITE を指定すると、NorthwindGrid のすべてのエンティティーに対して、読み取りのみが許可されます。ただし、Customer に対しては、(ALLWRITE が指定されているため) エンティティー・セット権限によってすべての読み取りが拒否されます。そのため、実質上、Customer エンティティーのアクセス権限は NONE になります。

- トランスポートを保護します。

  トランスポート・セキュリティーは、Web クライアントの場合は REST サービス接続に対してホスティング・コンテナー構成によって提供され、 REST サービスの場合は eXtreme Scale グリッド接続に対して eXtreme Scale クライアント構成によって提供されます。

  1. クライアントおよび REST サービスからの接続を保護します。 この接続のトランスポート・セキュリティーは、eXtreme Scale ではなくホスティング・コンテナー環境によって提供されます。
  2. REST サービスおよび eXtreme Scale グリッドからの接続を保護します。 この接続のトランスポート・セキュリティーは、eXtreme Scale で構成します。 392 ページの『トランスポート層セキュリティーおよび Secure Sockets Layer』を参照してください。

第 7 章 デプロイメント環境の操作

製品環境の操作には、スタンドアロン・モードまたは WebSphere Application Server でのサーバーの始動と停止が含まれます。また、WebSphere eXtreme Scale を WebSphere Application Server 環境でセッション・マネージャーとして使用することもできます。

管理に関する用語

WebSphere eXtreme Scale を管理するにあたっては、以下の事実を理解してください。

このタスクについて

サーバー・タイプ

WebSphere eXtreme Scale には、*カタログ・サーバー* とコンテナー・サーバー の 2 タイプのサーバーがあります。カタログ・サーバーは、断片の配置を制御し、コンテナー・サーバーの検出とモニターをします。複数のカタログ・サーバーがまとまってカタログ・サービスを構成します。コンテナー・サーバーは、グリッドのアプリケーション・データを保管する Java 仮想マシンです。

スタンドアロン・モード

スタンドアロン・モードは、他のアプリケーション・サーバー製品を使用せずに単独で実行している WebSphere eXtreme Scale 構成のことです。

WebSphere Application Server 内での実行

WebSphere Application Server の上に WebSphere eXtreme Scale を実行している場合、カタログ・サーバーは WebSphere Application Server のサーバー上で自動的に始動します。コンテナー・サーバーを始動するには、ObjectGrid XML ファイルを使用してアプリケーションをパッケージ化し、デプロイする必要があります。

関連概念

69 ページの『ハードウェアおよびソフトウェアの要件』
ハードウェア要件およびオペレーティング・システム要件の概要をご覧ください。

WebSphere eXtreme Scale に対して使用するハードウェアまたはオペレーティング・システムのレベルについて、特定のレベルの要件はありませんが、公式にサポートされるハードウェアおよびソフトウェアのオプションに関するオペレーティング・システム別の詳しいリストを、製品サポート・サイトのシステム要件ページから入手できます。 インフォメーション・センターの情報と「システム要件」ページの情報に違いがある場合は、Web サイトの情報を優先してください。インフォメーション・センターの前提条件の情報は、便宜上提供されているだけです。

コンテナー、区画、および断片

コンテナーは、グリッドのアプリケーション・データを保管するサービスです。通常、このデータは区画と呼ばれるパーツに分割され、複数のコンテナーでホストされます。これを受けて各コンテナーは、完全なデータのサブセットをホストします。JVM は 1 つ以上のコンテナーをホストすることができ、各コンテナーは複数の断片をホストできます。

要確認: すべてのデータをホストするコンテナーのヒープ・サイズを計画してください。それにあわせて、ヒープ設定を適宜構成してください。

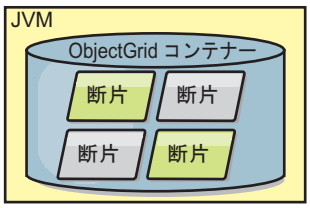


図 19. コンテナー

区画は、グリッド内のデータのサブセットをホストします。WebSphere eXtreme Scale は、自動的に複数の区画を単一コンテナーに配置し、追加のコンテナーが使用可能になるとそれらに区画を分散させます。

重要: 区画の数は動的に変更できないため、最終的なデプロイメントの前に、区画の数を慎重に選択してください。ネットワーク内での区画の配置にはハッシュ・メカニズムが使用され、いったんデプロイされた後でデータ・セット全体を eXtreme Scale が再ハッシュすることはできません。 一般に、区画の数は多めに見積もって構いません。

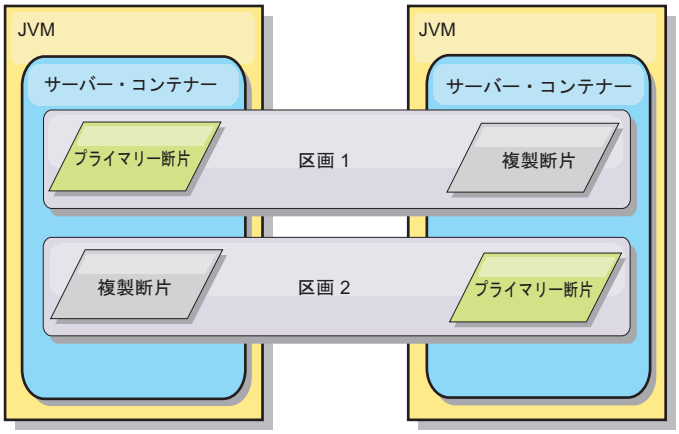


図 20. 区画

断片とは区画のインスタンスであり、プライマリーまたはレプリカの 2 つのロールのいずれか 1 つを持ちます。プライマリー断片とそのレプリカによって、区画の物理的な実体が構成されます。各区画はいくつかの断片を持ち、それぞれの断片が、その区画に含まれるデータ全体をホストします。 1 つの断片がプライマリー断片であり、他は複製断片です。複製断片は、プライマリー断片に含まれているデータの冗長コピーです。プライマリー断片は、トランザクションからキャッシュへの書き込みが可能な唯一の区画インスタンスです。複製断片は、「ミラーリングされた」区画インスタンスです。複製断片は、同期または非同期にプライマリー断片から更新内容を受信します。複製断片の場合、トランザクションはキャッシュからの読み取りのみが可能です。レプリカは、プライマリーと同じコンテナーではホストされません。また、通常はプライマリーと同じマシン上ではホストされません。

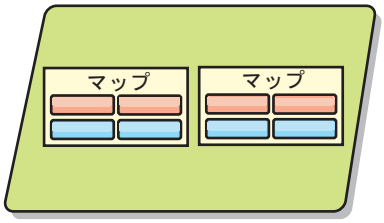


図 21. 断片

データの可用性を向上させる、または永続性の保証を高めるには、データを複製する必要があります。ただし、複製はトランザクションのコストを増加させるため、可用性と引き換えにパフォーマンスが犠牲になります。eXtreme Scale では、同期複製と非同期複製のサポートに加え、同期と非同期の両方の複製モードを使用するハイブリッド複製モデルがサポートされるため、このコストをコントロールできます。同期複製断片は、データ整合性を保証するため、プライマリー断片のトランザクションの一部として更新内容を受信します。トランザクション完了の前に、プライマリーと同期複製の両方でトランザクションをコミットする必要があるため、同期複製では応答時間が倍の長さになることがあります。非同期複製断片は、パフォーマンスへの影響を制限するために、トランザクションがコミットされた後に更新内容を受信します。しかし、非同期複製では、プライマリーよりトランザクションがいくつか遅れることがあるため、非同期複製断片でデータ損失の可能性が生じます。

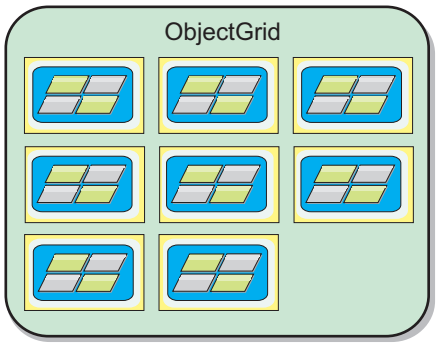


図 22. ObjectGrid

カタログ・サービス (カタログ・サーバー)

カタログ・サービスは、定常状態ではアイドルになるロジックをホストし、スケーラビリティーにはほとんど影響しません。カタログ・サービスが構築されている目的は、同時に使用可能になる数百ものコンテナーにサービスを提供し、それらのコンテナーを管理するサービスを実行することです。

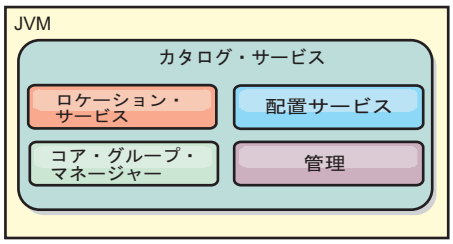


図 23. カタログ・サービス

カタログの役割分担には以下のサービスが含まれます。

ロケーション・サービス

ロケーション・サービスは、アプリケーションをホスティングするコンテナーを探しているクライアントと、ホスティングされるアプリケーションを配置サービスに登録しようとしているコンテナーの局所性を提供します。ロケーション・サービスは、この機能をスケールアウトするために、すべてのグリッド・メンバーで実行されます。

配置サービス

配置サービスは、グリッドの中枢神経的な存在であり、個々の断片をホスト・コンテナーに割り振る責任を担います。配置サービスは、クラスター内で N 個の中から 1 つ選ばれたサービスとして実行されます。 N 個の中の 1 つのポリシーが使用されるため、配置サービスの実行中のインスタンスは常に 1 つのみです。 そのインスタンスが停止する必要がある場合は、別のプロセスが引き継ぎます。カタログ・サービスのあらゆる状態は、予備のために、カタログ・サービスをホスティングするすべてのサーバーに複製されます。

コア・グループ・マネージャー
コア・グループ・マネージャーは、ヘルス・モニタリングのためにピアのグループ化を管理し、コンテナーを少数のサーバーからなるグループに編成し、サーバーのグループを自動的に統合します。初めてカタログ・サービスにコンタクトしたコンテナーは、いくつかの Java 仮想マシン (JVM) からなる新規または既存のグループのいずれかに割り当てられるのを待機します。Java 仮想マシンの各グループは、ハートビートを通してそれらの各メンバーの可用性をモニターします。再割り振りと経路転送によって障害に対処できるように、グループ・メンバーの 1 つが可用性情報をカタログ・サービスに中継します。

管理 WebSphere eXtreme Scale 環境の管理には、計画、デプロイ、管理、およびモニターの 4 つのステージがあります。各ステージの詳細については、「*管理ガイド*」を参照してください。

可用性のために、カタログ・サービス・ドメインを構成します。カタログ・サービス・ドメインは、複数の Java 仮想マシン (マスター JVM が 1 つと、多数のバックアップ Java 仮想マシン) から構成されます。

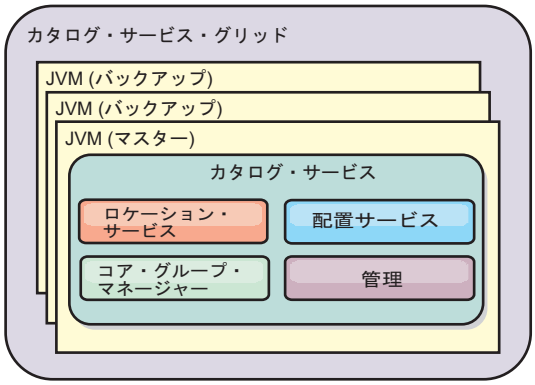


図 24. カタログ・サービス・ドメイン

ObjectGrid の可用性の設定

ObjectGrid インスタンスの可用性状態は、特定の時点でどの要求を処理できるかを決定します。

以下の 4 つの可用性状態があります。

- ONLINE
- QUIESCE
- OFFLINE
- PRELOAD

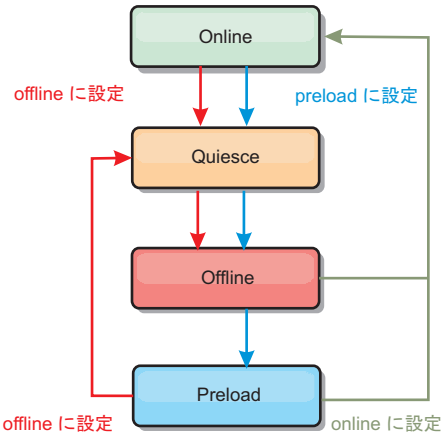


図 25. ObjectGrid の可用性状態

可用性状態の設定

ObjectGrid のデフォルトの可用性状態は ONLINE です。 ONLINE 状態の ObjectGrid は、標準 eXtreme Scale クライアントからのどの要求でも処理できます。ただし、プリロード・クライアントからの要求は、ObjectGrid が ONLINE である間は拒否されます。

QUIESCE 状態は、遷移的状態です。QUIESCE 状態にある ObjectGrid は、やがて OFFLINE 状態になります。QUIESCE 状態にある ObjectGrid は、未解決のトランザクションを処理できます。ただし新規トランザクションは拒否されます。ObjectGrid が QUIESCE 状態でいられるのは、30 秒までです。この時間を過ぎると、可用性状態は OFFLINE に移行します。

OFFLINE 状態にある ObjectGrid は、すべてのトランザクションを拒否します。

PRELOAD 状態は、プリロード・クライアントからデータを ObjectGrid にロードする場合に使用できます。ObjectGrid が PRELOAD 状態である間は、プリロード・クライアントしか、ObjectGrid に対してトランザクションをコミットできません。他のすべてのトランザクションは拒否されます。

ObjectGrid の可用性状態を設定するには、StateManager を使用します。サーバーで実行中の ObjectGrid の可用性状態を設定するには、対応する ObjectGrid クライアントを StateManager インターフェースに渡します。以下のコードは、ObjectGrid の可用性状態を変更する方法を示したものです。

```
ClientClusterContext client = ogManager.connect("localhost:2809", null, null);
ObjectGrid myObjectGrid = ogManager.getObjectGrid(client, "myObjectGrid");
StateManager stateManager = StateManagerFactory.getStateManager();
stateManager.setObjectGridState(AvailabilityState.OFFLINE, myObjectGrid);
```

StateManager で setObjectGridState メソッドが呼び出されると、ObjectGrid の各断片が要求状態に遷移します。メソッドが戻ると、ObjectGrid 内のすべての断片は、適切な状態になります。

サーバー・サイド ObjectGrid の可用性状態を変更するには、ObjectGridEventListener プラグインを使用します。サーバー・サイド ObjectGrid の可用性状態の変更は、その ObjectGrid の区画が 1 つだけの場合にのみ行ってください。ObjectGrid が複数

の区画を持っている場合、各プライマリーで shardActivated メソッドが呼び出され、結果として ObjectGrid の状態変更のために必要以上の呼び出しが行われることになります。

```
public class OGListener implements ObjectGridEventListener,
   ObjectGridEventGroup.ShardEvents {
    public void shardActivated(ObjectGrid grid) {
            StateManager stateManager = StateManagerFactory.getStateManager();
            stateManager.setObjectGridState(AvailabilityState.PRELOAD, grid);
    }
}
```

QUIESCE は遷移的状態であるため、ObjectGrid を QUIESCE 状態にするのに StateManager インターフェースを使用することはできません。ObjectGrid は、この状態を経てから OFFLINE 状態に移行します。

可用性状態の取得

特定の ObjectGrid の可用性状態を取得するには、StateManager の getObjectGridState メソッドを使用します。

```
StateManager stateManager = StateManagerFactory.getStateManager();
AvailabilityState state = stateManager.getObjectGridState(inventoryGrid);
```

getObjectGridState メソッドは、ObjectGrid 内のランダム・プライマリーを選択して、その AvailabilityState を返します。 ObjectGrid のすべての断片は同じ可用性状態になっているか、同じ可用性状態に遷移中である必要があるので、このメソッドにより、ObjectGrid の現在の可用性状態に関する妥当な結果がもたらされます。

さまざまな要求の適切な可用性状態

ObjectGrid が要求をサポートするのに適切な可用性状態にない場合、その要求は拒否されます。要求が拒否されると、必ず AvailabilityException 例外が発生します。

initialState 属性

initialState 属性は、ObjectGrid でその始動時の状態を示すのに使用できます。通常、初期化が完了した ObjectGrid は、ルーティングに使用可能になります。トラフィックが ObjectGrid にルーティングされないように、後で状態を変更できます。ObjectGrid を初期化する必要があるが、すぐには使用可能でない場合、initialState 属性を使用できます。

initialState 属性は、ObjectGrid の構成 XML ファイルで設定されます。デフォルト状態は ONLINE です。有効な値には、次のものが含まれます。

- ONLINE (デフォルト)
- PRELOAD
- OFFLINE

詳しくは、AvailabilityState API の資料を参照してください。

ObjectGrid で initialState が設定されている場合、その状態を明示的にオンラインに戻す必要があります。さもないと、ObjectGrid は使用不可のままになります。この場合 AvailabilityExceptions がスローされます。

プリロードのための initialState 属性の使用

ObjectGrid にデータがプリロードされる場合、ObjectGrid が使用可能になる時点と、クライアント・トラフィックをブロックするためにプリロード状態に切り替わる時点との間に、しばらく時間があくことがあります。この時間を回避するため、ObjectGrid の初期状態を PRELOAD に設定できます。この場合にも、ObjectGrid は必要な初期化をすべて完了しますが、状態が変更され、プリロードを実行できるようになるまで、トラフィックをブロックします。

PRELOAD 状態も OFFLINE 状態もトラフィックをブロックしますが、プリロードを開始する場合は PRELOAD 状態を使用する必要があります。

フェイルオーバーおよびバランシングの振る舞い

レプリカがプライマリーにプロモートされても、そのレプリカは initialState 設定を使用しません。プライマリーが再バランシングのために移動された場合、移動が完了する前に、データがプライマリーの新しいロケーションにコピーされるため、initialState 設定は使用されません。複製が構成されていない場合、フェイルオーバーが発生すると、プライマリーは initialState 値になるため、新規プライマリーを配置する必要があります。

組み込みサーバー API の使用

WebSphere eXtreme Scale では、組み込みサーバーおよびコンテナーのライフサイクルの管理にプログラマチック API を使用できます。コマンド行オプションやファイル・ベースのサーバー・プロパティーでも構成可能な任意のオプションを使用して、プログラムでサーバーを構成できます。コンテナー・サーバー、カタログ・サービス、またはその両方として、組み込みサーバーの構成が可能です。

始める前に

既存の Java 仮想マシン内からコードを実行するためのメソッドが必要です。eXtreme Scale クラスが、クラス・ローダー・ツリーから利用可能でなければなりません。

このタスクについて

管理 API を使用して多くの管理タスクを実行できます。API の一般的な使用法の 1 つとして、Web アプリケーションの状態を保管する内部サーバーとしての使用があります。 Web サーバーは、組み込み WebSphere eXtreme Scale サーバーを始動し、コンテナー・サーバーをカタログ・サービスに報告することができ、サーバーは、より幅広い分散グリッドのメンバーとして追加されます。この使用法では、本来は揮発性のデータ・ストアにスケーラビリティーと高可用性が提供されます。

組み込み eXtreme Scale サーバーの全ライフサイクルをプログラムで制御できます。例は、できる限り汎用的にして、概要を説明したステップの直接的なコードの例のみを示しています。

手順

1. ServerFactory クラスから ServerProperties オブジェクトを取得し、必要なオプションを構成します。

すべての eXtreme Scale サーバーに、一連の構成可能なプロパティーがあります。コマンド行からサーバーが始動されると、それらのプロパティーはデフォルトに設定されますが、外部ソースまたはファイルを指定することによって、複数のプロパティーをオーバーライドすることができます。組み込み有効範囲では、ServerProperties オブジェクトでプロパティーを直接設定できます。これらのプロパティーは、ServerFactory クラスからサーバー・インスタンスを取得する前に設定する必要があります。以下の例のスニペットでは、ServerProperties オブジェクトを取得して CatalogServiceBootStrap フィールドを設定し、複数のオプション・サーバー設定を初期化します。構成可能な設定のリストについては、API 資料を参照してください。

```
ServerProperties props = ServerFactory.getServerProperties();
props.setCatalogServiceBootstrap("host:port"); // required to connect to specific catalog service
props.setServerName("ServerOne"); // name server
props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // Sets trace spec
```

2. サーバーをカタログ・サービスにする場合には、CatalogServerProperties オブジェクトを取得します。

   すべての組み込みサーバーが、カタログ・サービスまたはコンテナー・サーバー、あるいはその両方になることができます。以下の例では、CatalogServerProperties オブジェクトを取得し、カタログ・サービス・オプションを使用可能にし、さまざまなカタログ・サービス設定を構成します。

   ```
   CatalogServerProperties catalogProps = ServerFactory.getCatalogProperties();
   catalogProps.setCatalogServer(true); // false by default, it is required to set as a catalog service
   catalogProps.setQuorum(true); // enables / disables quorum
   ```

3. ServerFactory クラスから Server インスタンスを取得します。 Server インスタンスは、グリッド内のメンバーシップの管理に関与するプロセス・スコープの singleton です。このインスタンスが初期化された後、このプロセスが接続され、グリッド内の他のサーバーで高度に利用可能になります。 以下の例は、Server インスタンスの作成方法を示しています。

   ```
   Server server = ServerFactory.getInstance();
   ```

   上記の例において、ServerFactory クラスは、 Server インスタンスを返す静的メソッドを提供します。 ServerFactory クラスは、Server インスタンスを取得するための唯一のインターフェースとして意図されています。そのため、このクラスは必ず、インスタンスが singleton であるか、または各 JVM または独立したクラス・ローダーの 1 つインスタンスであるようにします。 getInstance メソッドで Server インスタンスを初期化します。インスタンスを初期化する前にすべてのサーバー・プロパティーの構成が必要です。 Server クラスは、新規の Container インスタンスの作成に関与します。 ServerFactory クラスと Server クラスの両方を使用して、組み込み Server インスタンスのライフサイクルを管理できます。

4. Server インスタンスを使用して Container インスタンスを開始します。

   断片を組み込みサーバーに配置するには、その前に、サーバーにコンテナーを作成する必要があります。 Server インターフェースの createContainer メソッドは、DeploymentPolicy 引数を使用します。以下の例では、取得したサーバー・インスタンスを使用して、作成した DeploymentPolicy ファイルでコンテナーを作成します。 Container は、シリアライゼーションのためにアプリケーション・バイナリーが使用可能になっているクラス・ローダーを必要とします。これらのバ

イナリーは、使用するクラス・ローダーを Thread コンテキスト・クラス・ローダーに設定して createContainer メソッドを呼び出すことによって、使用可能にできます。

```
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(new
   URL("file://urltodeployment.xml"),
 new URL("file://urltoobjectgrid.xml"));
Container container = server.createContainer(policy);
```

5. コンテナーを除去してクリーンアップします。

   コンテナー・サーバーを除去してクリーンアップするには、取得した Container インスタンスで teardown メソッドを実行します。コンテナーで teardown メソッドを実行すると、コンテナーを適切にクリーンアップし、組み込みサーバーからコンテナーを除去します。

   コンテナーのクリーンアップ処理には、そのコンテナー内のすべての断片の移動と終了処理が含まれます。各サーバーには、多くのコンテナーと断片が含まれます。コンテナーをクリーンアップしても、親の Server インスタンスのライフサイクルには影響しません。以下の例は、サーバーで teardown メソッドを実行する方法を示しています。teardown メソッドは、ContainerMBean インターフェースを通して使用可能になります。ContainerMBean インターフェースを使用することによって、このコンテナーに対するプログラムによるアクセスがもうなくても、その MBean でコンテナーを除去してクリーンアップすることができます。また、terminate メソッドが Container インターフェースに存在しますが、どうしても必要でない限り、このメソッドは使用しないでください。このメソッドは強制力が強く、断片の適切な移動とクリーンアップの調整は行いません。

   ```
   container.teardown();
   ```

6. 組み込みサーバーを停止します。

   組み込みサーバーを停止するときには、そのサーバーで実行されているコンテナーと断片も停止します。組み込みサーバーの停止時には、開いているすべての接続をクリーンアップして、すべての断片を移動または終了処理する必要があります。以下の例では、サーバーの停止方法と、Server インターフェースで waitFor メソッドを使用して Server インスタンスが確実に完全にシャットダウンするようにする方法を示しています。コンテナーの例と同様に、stopServer メソッドは、ServerMBean インターフェースを通して使用可能になります。このインターフェースでは、該当の Managed Bean (MBean) によりサーバーを停止できます。

   ```
   ServerFactory.stopServer();  // Uses the factory to kill the Server singleton
    // or
   server.stopServer(); // Uses the Server instance directly
   server.waitFor(); // Returns when the server has properly completed its shutdown procedures
   ```

   全コードの例:

   ```
    import java.net.MalformedURLException;
   import java.net.URL;

   import com.ibm.websphere.objectgrid.ObjectGridException;
   import com.ibm.websphere.objectgrid.deployment.DeploymentPolicy;
   import com.ibm.websphere.objectgrid.deployment.DeploymentPolicyFactory;
   import com.ibm.websphere.objectgrid.server.Container;
   import com.ibm.websphere.objectgrid.server.Server;
   import com.ibm.websphere.objectgrid.server.ServerFactory;
   import com.ibm.websphere.objectgrid.server.ServerProperties;

   public class ServerFactoryTest {

       public static void main(String[] args) {
   ```

```
            try {

                ServerProperties props = ServerFactory.getServerProperties();
                props.setCatalogServiceBootstrap("catalogservice-hostname:catalogservice-port");
                props.setServerName("ServerOne"); // name server
                props.setTraceSpecification("com.ibm.ws.objectgrid=all=enabled"); // TraceSpec

                /*
                 * In most cases, the server will serve as a container server only
                 * and will connect to an external catalog service. This is a more
                 * highly available way of doing things. The commented code excerpt
                 * below will enable this Server to be a catalog service.
                 *
                 *
                 * CatalogServerProperties catalogProps =
                 * ServerFactory.getCatalogProperties();
                 * catalogProps.setCatalogServer(true); // enable catalog service
                 * catalogProps.setQuorum(true); // enable quorum
                 */

                Server server = ServerFactory.getInstance();

                DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy
          (new URL("url to deployment xml"),  new URL("url to objectgrid xml file"));
                Container container = server.createContainer(policy);

                /*
                 * Shard will now be placed on this container if the deployment requirements are met.
                 * This encompasses embedded server and container creation.
                 *
                 * The lines below will simply demonstrate calling the cleanup methods
                 */

                container.teardown();
                server.stopServer();
                int success = server.waitFor();

            } catch (ObjectGridException e) {
                // Container failed to initialize
            } catch (MalformedURLException e2) {
                // invalid url to xml file(s)
            }

        }

    }
```

組み込みサーバー API

WebSphere eXtreme Scale には、既存の Java アプリケーション内に eXtreme Scale サーバーおよびクライアントを組み込む、アプリケーション・プログラミング・インターフェース (API) およびシステム・プログラミング・インターフェースが含まれています。 以下のトピックで、使用可能な組み込みサーバー API について説明します。

eXtreme Scale サーバーのインスタンス化

eXtreme Scale サーバー・インスタンスの構成には、いくつかのプロパティーを使用できますが、これは、ServerFactory.getServerProperties メソッドから取得できます。 ServerProperties オブジェクトは singleton のため、getServerProperties メソッドの各呼び出しでは同じインスタンスが取得されます。

次のコードを使用して、新規サーバーを作成することができます。

```
Server server = ServerFactory.getInstance();
```

getInstance の最初の呼び出しの前に設定されたすべてのプロパティーは、サーバーの初期化に使用されます。

サーバー・プロパティーの設定

サーバー・プロパティーは、ServerFactory.getInstance が最初に呼び出されるまで設定できます。 getInstance メソッドの最初の呼び出しで、eXtreme Scale サーバーがインスタンス化され、構成されたすべてのプロパティーが読み取られます。作成後にプロパティーを設定しても効果はありません。以下の例は、Server インスタンスをインスタンス化する前のプロパティーの設定方法を示しています。

```
// Get the server properties associated with this process.
ServerProperties serverProperties = ServerFactory.getServerProperties();

// Set the server name for this process.
serverProperties.setServerName("EmbeddedServerA");

// Set the name of the zone this process is contained in.
serverProperties.setZoneName("EmbeddedZone1");

// Set the end point information required to bootstrap to the catalog service.
serverProperties.setCatalogServiceBootstrap("localhost:2809");

// Set the ORB listener host name to use to bind to.
serverProperties.setListenerHost("host.local.domain");

// Set the ORB listener port to use to bind to.
serverProperties.setListenerPort(9010);

// Turn off all MBeans for this process.
serverProperties.setMBeansEnabled(false);

Server server = ServerFactory.getInstance();
```

カタログ・サービスの組み込み

CatalogServerProperties.setCatalogServer メソッドによりフラグが立てれた JVM 設定は、eXtreme Scale のカタログ・サービスをホストできます。このメソッドは、eXtreme Scale サーバー・ランタイムに対して、サーバーの始動時にカタログ・サービスをインスタンス化することを指示します。以下のコードは、eXtreme Scale カタログ・サーバーをインスタンス化する方法を示しています。

```
CatalogServerProperties catalogServerProperties =
 ServerFactory.getCatalogProperties();
catalogServerProperties.setCatalogServer(true);

Server server = ServerFactory.getInstance();
```

eXtreme Scale コンテナーの組み込み

JVM が複数の eXtreme Scale コンテナーをホストするようにするには、Server.createContainer メソッドを発行します。以下のコードは、eXtreme Scale コンテナーをインスタンス化する方法を示しています。

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL());
Container container = server.createContainer(policy);
```

自己完結型のサーバー・プロセス

すべてのサービスは、まとめて開始することができ、これは開発に便利で、実動中にも実用的です。サービスをまとめて開始することにより、1 つのプロセスで、カ

タログ・サービスの開始、コンテナー・セットの開始、クライアント接続ロジックの実行をすべて行うことができます。このような方法でサービスを開始すると、分散環境にデプロイする前にプログラム上の問題を整理することができます。以下のコードは、自己完結型の eXtreme Scale サーバーをインスタンス化する方法を示しています。

```
CatalogServerProperties catalogServerProperties =
 ServerFactory.getCatalogProperties();
catalogServerProperties.setCatalogServer(true);

Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL());
Container container = server.createContainer(policy);
```

WebSphere Application Server における eXtreme Scale の組み込み

eXtreme Scale の構成は、WebSphere Extended Deployment DataGrid を WebSphere Application Server 環境にインストールすると、自動的にセットアップされます。サーバーにアクセスしてコンテナーを作成する前にプロパティーを設定する必要はありません。以下のコードは、eXtreme Scale サーバーを WebSphere Application Server でインスタンス化する方法を示しています。

```
Server server = ServerFactory.getInstance();
DeploymentPolicy policy = DeploymentPolicyFactory.createDeploymentPolicy(
    new File("META-INF/embeddedDeploymentPolicy.xml").toURI().toURL(),
    new File("META-INF/embeddedObjectGrid.xml").toURI().toURL);
Container container = server.createContainer(policy);
```

組み込みカタログ・サービスおよびコンテナーをプログラマチックに開始する方法に関する段階的な例は、348 ページの『組み込みサーバー API の使用』を参照してください。

スタンドアロン WebSphere eXtreme Scale サーバーの始動

スタンドアロン WebSphere eXtreme Scale 構成を実行しているとき、環境はカタログ・サーバー、コンテナー・サーバー、および eXtreme Scale クライアント・プロセスで構成されています。 eXtreme Scale サーバーは、組み込みサーバー API を使用して、既存の Java アプリケーション内に組み込むこともできます。これらのプロセスは手動で構成して開始する必要があります。

始める前に

WebSphere Application Server がインストールされていない環境で WebSphere eXtreme Scale サーバーを始動できます。 WebSphere Application Server を使用している場合は、369 ページの『WebSphere Application Server による WebSphere eXtreme Scale の管理』を参照してください。

次のタスク

eXtreme Scale プロセスを停止します。詳しくは、363 ページの『スタンドアロン eXtreme Scale サーバーの停止』を参照してください。

スタンドアロン環境でのカタログ・サービスの開始

WebSphere Application Server で実行されていない分散 WebSphere eXtreme Scale 環境を使用している場合は、カタログ・サービスを手動で開始する必要があります。

始める前に

WebSphere Application Server を使用している場合、カタログ・サービスは既存のいずれかのプロセス内で自動的に始動します。詳しくは、WebSphere Application Server でのカタログ・サービスの開始を参照してください。

このタスクについて

カタログ・サービスは単一のプロセスで実行することができます。また、複数のカタログ・サーバーを組み込んでカタログ・サービス・ドメインを形成することもできます。実稼働環境では、高可用性を確保するためカタログ・サーバー・グリッドが必要となります。カタログ・サービス・ドメインの構成について詳しくは、*製品概要* におけるカタログ・サービス・ドメインに関する情報を参照してください。カタログ・サービスは、グリッドに配置されている場合も単一プロセス内にある場合も、`startOgServer` スクリプトを使用して開始されます。また、このスクリプトに追加のパラメーターを指定することで、オブジェクト・リクエスト・ブローカー (ORB) を特定のホストおよびポートにバインドしたり、ドメインを指定したり、セキュリティーを使用可能にしたりできます。

開始コマンドを呼び出すには、UNIX プラットフォームでは `startOgServer.sh` スクリプトを、また、Windows では `startOgServer.bat` を使用します。

手順

- **単一カタログ・サーバー・プロセスの開始**

  単一のカタログ・サーバーを始動するには、コマンド行から以下のコマンドを入力します。

  1. bin ディレクトリーに移動します。

     ```
     cd objectgridRoot/bin
     ```

  2. startOgServer コマンドを実行します。

     ```
     startOgServer.bat|sh catalogServer
     ```

  使用可能なすべてのコマンド行パラメーターのリストについては、360 ページの『startOgServer スクリプト』を参照してください。実稼働環境では、単一の Java 仮想マシン (JVM) を使用してカタログ・サービスを実行しないようにしてください。カタログ・サービスが失敗すると、新規クライアントをデプロイ済みの eXtreme Scale に経路指定することも、新規 ObjectGrid インスタンスをドメインに追加することもできません。これらの理由により、Java 仮想マシンのセットを始動してカタログ・サービス・ドメインを実行するようにしてください。

- **マルチプロセス・カタログ・サービス・ドメインの開始**

  サーバーのセットを開始してカタログ・サービスを実行するには、`startOgServer` スクリプトで **-catalogServiceEndPoints** オプションを使用する必要があります。

この引数は、serverName:hostName:clientPort:peerPort の形式のカタログ・サービス・エンドポイントのリストを受け入れます。各属性の定義は次のとおりです。

serverName
起動しようとしているプロセスを識別する名前を指定します。

hostName
サーバーを起動するコンピューターのホスト名を指定します。

clientPort
ピア・カタログ・グリッド通信に使用されるポートを指定します。

peerPort
これは、haManagerPort と同じです。ピア・カタログ・グリッド通信に使用されるポートを指定します。

以下の例は、カタログ・サービスをホストする 3 つの Java 仮想マシンのうち、最初のものを始動する方法を示しています。

1. bin ディレクトリーに移動します。

   ```
   cd objectgridRoot/bin
   ```

2. startOgServer コマンドを実行します。

   ```
   startOgServer.bat|sh cs1 -catalogServiceEndpoints
   cs1:MyServer1.company.com:6601:6602,
   cs2:MyServer2.company.com:6601:6602,
   cs3:MyServer3.company.com:6601:6602
   ```

   この例では、MyServer1.company.com ホスト上の cs1 サーバーが始動されます。このサーバーの名前は、スクリプトに渡される最初の引数です。 cs1 サーバーの初期化時に、catalogServiceEndpoints パラメーターが検査されて、このプロセスに割り振られるポートが決定されます。このリストは、cs1 サーバーが他のサーバー (cs2 および cs3) からの接続を受け入れることができるようにするためにも使用されます。

3. リスト内の残りのカタログ・サーバーを開始するには、以下の引数を startOgServer スクリプトに渡します。MyServer2.company.com ホスト上の cs2 サーバーを始動します。

   ```
   startOgServer.bat|sh cs2 -catalogServiceEndpoints
   cs1:MyServer1.company.com:6601:6602,
   cs2:MyServer2.company.com:6601:6602,
   cs3:MyServer3.company.com:6601:6602
   ```

   MyServer3.company.com 上の cs3 を始動します。

   ```
   startOgServer.bat|sh cs3 -catalogServiceEndpoints
   cs1:MyServer1.company.com:6601:6602,
   cs2:MyServer2.company.com:6601:6602,
   cs3:MyServer3.company.com:6601:6602
   ```

   重要: すべてのカタログ・サーバーを同時に始動してください。

   グリッドに入っているカタログ・サーバーは、それぞれのサーバーが、他のカタログ・サーバーがコア・グループに参加するのを休止して待つため、同時に始動する必要があります。グリッド用に構成されているカタログ・サーバー

は、グループ内の他のメンバーを識別するまで始動しません。カタログ・サーバーは、他のサーバーがいずれも使用可能にならないと、最終的にタイムアウトになります。

- **特定のホストおよびポートへの ORB のバインド**

  catalogServiceEndpoints 引数で定義されたポートを別にすれば、各カタログ・サービスも、オブジェクト・リクエスト・ブローカー (ORB) を使用して、クライアントおよびコンテナーからの接続を受け入れます。デフォルトでは、ORB はローカル・ホストのポート 2809 で listen します。カタログ・サービス JVM で特定のホストおよびポートに ORB をバインドする場合は、**-listenerHost** および **-listenerPort** 引数を使用します。次の例は、ORB が MyServer1.company.com のポート 7000 にバインドされた単一の JVM カタログ・サーバーを始動する方法を示しています。

  ```
  startOgServer.sh catalogServer -listenerHost MyServer1.company.com
  -listenerPort 7000
  ```

  各 eXtreme Scale コンテナーおよびクライアントにカタログ・サービス ORB エンドポイント・データが提供されるようにする必要があります。クライアントにはこのデータのサブセットのみが必要ですが、高可用性のために少なくとも 2 つのエンドポイントを使用するようにしてください。

- **ドメインのネーミング**

  カタログ・サービスを開始するとき、ドメイン・ネームは必要ではありません。デフォルトのドメイン・ネームは defaultDomain です。ドメインに名前を付けるには、**-domain** オプションを使用します。次の例は、ドメイン・ネーム myDomain を持つ単一カタログ・サービス JVM の開始方法を示しています。

  ```
  startOgServer.sh catalogServer -domain myDomain
  ```

- **セキュア・カタログ・サービスの開始**

  セキュア・カタログ・サービスを開始するには、以下の引数を指定します。

  -clusterSecurityFile および -clusterSecurityUrl
  これらの引数は objectGridSecurity.xml ファイルを指定します。このファイルは、すべてのサーバー (カタログ・サーバーおよびコンテナー・サーバーを含む) に共通するセキュア・プロパティーを記述するものです。プロパティー例の 1 つは、ユーザー・レジストリーおよび認証メカニズムを表すオーセンティケーター構成です。

  -serverProps
  サーバー固有のセキュリティー・プロパティーが含まれているサーバー・プロパティー・ファイルを指定します。このプロパティーに対して指定されるファイル名の形式は、プレーン・ファイル・パス形式です。例えば、c:/tmp/og/catalogserver.props などです。

  セキュア・カタログ・サービスを開始する方法の例は、*製品概要* にある Java SE セキュリティー・チュートリアルのステップ 2 を参照してください。objectGridSecurity.xml ファイルの例については、402 ページの『セキュリティー記述子 XML ファイル』を参照してください。

コンテナー・プロセスの開始

eXtreme Scale は、デプロイメント・トポロジーまたは server.properties ファイルを使用して、コマンド行から開始できます。

このタスクについて

コンテナー・プロセスを開始するには、ObjectGrid XML ファイルが必要です。ObjectGrid XML ファイルは、コンテナーがホストする eXtreme Scale サーバーを指定します。コンテナーに渡される XML の各 ObjectGrid をホストするようにコンテナーが装備されていることを確認してください。これらの ObjectGrid が必要とするクラスは、すべてコンテナーのクラスパスになければなりません。 ObjectGrid XML ファイルに関して詳しくは、 169 ページの『objectGrid.xsd ファイル』を参照してください。

手順

- **コマンド行からコンテナー・プロセスを開始します。**

  1. コマンド行から、bin ディレクトリーに移動します。

     ```
     cd objectgridRoot/bin
     ```

  2. 以下のコマンドを実行します。

     ```
     startOgServer.sh c0 -objectGridFile ../xml/companyGrid.xml
     -catalogServiceEndPoints MyServer1.company.com:2809
     ```

     重要: コンテナーでは、**-catalogServiceEndPoints** オプションを使用して、カタログ・サービス上のオブジェクト・リクエスト・ブローカー (ORB) のホストとポートを参照します。カタログ・サービスでは、**-listenerHost** および **-listenerPort** オプションを使用して ORB バインディング用のホストとポートを指定するか、またはデフォルトのバインディングを受け入れます。コンテナーを開始する場合は、**-catalogServiceEndPoints** オプションを使用して、カタログ・サービスで **-listenerHost** および **-listenerPort** オプションに渡される値を参照します。カタログ・サービスの開始時に **-listenerHost** および **-listenerPort** オプションが使用されなかった場合、ORB は、カタログ・サービスのローカル・ホストでポート 2809 にバインドします。カタログ・サービスで **-catalogServiceEndPoints** オプションに渡されたホストとポートを参照する場合に、**-catalogServiceEndPoints** オプションを使用しないでください。カタログ・サービスで
     は、**-catalogServiceEndPoints** オプションを使用して、静的サーバー構成に必要なポートを指定します。
     このプロセスは、スクリプトに渡される最初の引数 c0 によって識別されます。companyGrid.xml を使用してコンテナーを開始してください。カタログ・サーバー ORB が、コンテナーとは異なるホストで実行されているか、またはデフォルト以外のポートを使用している場合は、 **-catalogServiceEndPoints** 引数を使用してその ORB に接続する必要があります。この例では、単一のカタログ・サービスが MyServer1.company.com のポート 2809 で実行されているものと仮定します。

- **デプロイメント・ポリシーを使用してコンテナーを開始します。**

  必須ではありませんが、コンテナーの開始時には、デプロイメント・ポリシーが推奨されます。デプロイメント・ポリシーは、eXtreme Scale の区画化および複製をセットアップするために使用されます。デプロイメント・ポリシーは、配置方

法に影響を与えるためにも使用されます。前述の例では、デプロイメント・ポリシー・ファイルが提供されなかったため、複製、区画化、および配置に関して、すべてのデフォルト値を受け取ります。したがって、CompanyGrid にあるマップは 1 つの mapSet 内に入ります。この mapSet は区画化も複製もされません。デプロイメント・ポリシー・ファイルについて詳しくは、184 ページの『デプロイメント・ポリシー記述子 XML ファイル』を参照してください。次の例では、companyGridDpReplication.xml ファイルを使用してコンテナー JVM (c0 JVM) を開始します。

1. コマンド行から、bin ディレクトリーに移動します。

   ```
   cd objectgridRoot/bin
   ```

2. 以下のコマンドを実行します。

   ```
   startOgServer.sh c0 -objectGridFile ../xml/companyGrid.xml
   -deploymentPolicyFile ../xml/companyGridDpReplication.xml
   -catalogServiceEndPoints MyServer1.company.com:2809
   ```

注: Java クラスが特定のディレクトリーに保管されている場合には、StartOgServer スクリプトを変更する代わりに、-jvmArgs -cp C:¥ . . . ¥DirectoryPOJOs¥POJOs.jar というように引数を指定してサーバーを起動することができます。

companyGridDpReplication.xml ファイルでは、単一のマップ・セットにすべてのマップが含まれています。この mapSet は 10 個の区画に分割されます。各区画には 1 つの同期複製が存在し、非同期複製は存在しません。また、companyGrid.xml ObjectGrid XML ファイルとペアになる companyGridDpReplication.xml デプロイメント・ポリシーを使用するコンテナーは、CompanyGrid 断片をホストできます。別のコンテナー JVM (c1 JVM) を開始します。

1. コマンド行から、bin ディレクトリーに移動します。

   ```
   cd objectgridRoot/bin
   ```

2. 以下のコマンドを実行します。

   ```
   startOgServer.sh c1 -objectGridFile ../xml/companyGrid.xml
   -deploymentPolicyFile ../xml/companyGridDpReplication.xml
   -catalogServiceEndPoints MyServer1.company.com:2809
   ```

各デプロイメント・ポリシーには、1 つ以上の objectgridDeployment エレメントが含まれています。コンテナーが開始されると、コンテナーは、デプロイメント・ポリシーをカタログ・サービスに公開します。カタログ・サービスは各 objectgridDeployment エレメントを検査します。 objectgridName 属性が、前に受信された objectgridDeployment エレメントの objectgridName 属性と一致する場合、最新の objectgridDeployment エレメントは無視されます。特定の objectgridName 属性用に受信された最初の objectgridDeployment エレメントがマスターとして使用されます。例えば、c2 JVM が、mapSet を異なる数の区画に分割するデプロイメント・ポリシーを使用するとします。

companyGridDpReplicationModified.xml

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

    <objectgridDeployment objectgridName="CompanyGrid">
```

```
        <mapSet name="mapSet1" numberOfPartitions="5"
            minSyncReplicas="1" maxSyncReplicas="1"
            maxAsyncReplicas="0">
            <map ref="Customer" />
            <map ref="Item" />
            <map ref="OrderLine" />
            <map ref="Order" />
        </mapSet>
    </objectgridDeployment>

</deploymentPolicy>
```

これで、3 番目の JVM である c2 JVM を開始できます。

1. コマンド行から、bin ディレクトリーに移動します。

   ```
   cd objectgridRoot/bin
   ```

2. 以下のコマンドを実行します。

   ```
   startOgServer.sh c2 -objectGridFile ../xml/companyGrid.xml
   -deploymentPolicyFile ../xml/companyGridDpReplicationModified.xml
   -catalogServiceEndPoints MyServer1.company.com:2809
   ```

c2 JVM 上のコンテナーが、mapSet1 に 5 つの区画を指定するデプロイメント・ポリシーで開始されます。しかし、カタログ・サービスは、CompanyGrid の objectgridDeployment のマスター・コピーを既に保持しています。c0 JVM は開始されたときに、この mapSet に 10 個の区画を指定しました。 c0 が、デプロイメント・ポリシーを開始および公開する最初のコンテナーであったため、c0 のデプロイメント・ポリシーがマスターになりました。したがって、後続のデプロイメント・ポリシー内の CompanyGrid に等しい objectgridDeployment 属性値はすべて無視されます。

- **サーバー・プロパティー・ファイルを使用してコンテナーを開始します。**

  サーバー・プロパティー・ファイルを使用して、コンテナーでのトレースをセットアップし、セキュリティーを構成することができます。次のコマンドを実行し、サーバー・プロパティー・ファイルを使用してコンテナー c3 を開始します。

  1. コマンド行から、bin ディレクトリーに移動します。

     ```
     cd objectgridRoot/bin
     ```

  2. 以下のコマンドを実行します。

     ```
     startOgServer.sh c3 -objectGridFile ../xml/companyGrid.xml
     -deploymentPolicyFile ../xml/companyGridDpReplicationModified.xml
     -catalogServiceEndPoints MyServer1.company.com:2809
     -serverProps ../serverProps/server.properties
     ```

  server.properties ファイルの例を次に示します。

  server.properties

  ```
  workingDirectory=
  traceSpec=*=all=disabled
  systemStreamToFileEnabled=true
  enableMBeans=true
  memoryThresholdPercentage=50
  ```

  これは、セキュリティーを有効にしていない基本的なサーバー・プロパティー・ファイルです。 server.properties ファイルに関して詳しくは、 196 ページの『サーバー・プロパティー・ファイル』を参照してください。

startOgServer スクリプト

startOgServer スクリプトはサーバーを始動します。サーバーの始動時に各種パラメーターを使用して、トレースを使用可能にしたり、ポート番号を指定するなど、さまざまな設定を行うことができます。

目的

startOgServer スクリプトを使用してサーバーを始動することができます。

ロケーション

startOgServer スクリプトは、ルート・ディレクトリーの bin ディレクトリーにあります。例えば、次のように入力します。

```
cd objectgridRoot/bin
```

注: Java クラスが特定のディレクトリーに保管されている場合には、 startOgServer スクリプトを変更する代わりに、`-jvmArgs -cp C:¥ . . . ¥DirectoryPOJOs¥POJOs.jar` というように引数を指定してサーバーを起動することができます。

カタログ・サーバーの場合の使用法

カタログ・サーバーを始動する場合:

Windows

```
startOgServer.bat <server> [options]
```

UNIX

```
startOgServer.sh <server>[options]
```

デフォルトの構成済みカタログ・サーバーを始動するには、以下のコマンドを使用します。

Windows

```
startOgServer.bat catalogServer
```

UNIX

```
startOgServer.sh catalogServer
```

カタログ・サーバーの始動のオプション

カタログ・サーバーの始動のためのパラメーター:

-catalogServiceEndPoints
<server:serverHost:clientPort:peerPort,server:serverHost:clientPort:peerPort>
コンテナーでは、**-catalogServiceEndpoints** オプションを使用して、カタログ・サービスでオブジェクト・リクエスト・ブローカー (ORB) のホストとポートが参照されます。

-clusterSecurityFile <cluster security xml file>
セキュリティー記述子 XML ファイルが置かれているロケーションを指定します。

-clusterSecurityUrl <cluster security xml URL>

-domain <domain name>

-listenerHost <host name>
デフォルト: `localhost`

Internet Inter-ORB Protocol (IIOP) との通信に使用するリスナー・ホストを指定します。

-listenerPort <port>
デフォルト: `2809`

IIOP との通信に使用するリスナー・ポートを指定します。

-haManagerPort <port>
デフォルト: これは、peerPort と同じです。このプロパティーが設定されていない場合は、カタログ・サービスは使用可能なポートを自動的に生成します。

HA マネージャーが使用するポート番号を指定します。

-serverProps <server properties file>
サーバー固有のセキュリティー・プロパティーが含まれているサーバー・プロパティー・ファイルを指定します。このプロパティーに対して指定されるファイル名の形式は、単なるプレーン・ファイル・パス形式です。例えば、`c:/tmp/og/catalogserver.props` などです。

-JMXServicePort <port>
デフォルト: `1099`

Java Management Extensions (JMX) との通信に使用するポート番号を指定します。

-traceSpec <trace specification>
サーバーが始動したとき使用可能になるトレースの有効範囲を指定するストリングを指定します。

例:

- `ObjectGrid=all=enabled`
- `ObjectGrid*=all=enabled`

-traceFile <trace file>
トレース情報を保存するファイルのパスを指定します。

例: `../logs/c4Trace.log`

-timeout <seconds>
サーバーの始動がタイムアウトになる秒数を指定します。

-script <script file>
カタログ・サーバーまたはコンテナーを開始するために指定するコマンドのカスタム・スクリプトを作成してから、必要に応じて、パラメーター化および編集を行います。

-jvmArgs <JVM arguments>
JVM 引数 (複数可) を指定します。 **-jvmArgs** パラメーターより後にあるパラメーターは、すべてサーバー Java 仮想マシン (JVM) を始動するために使用されるものです。 **-jvmArgs** パラメーターを使用するときは、最後に指定されるスクリプト引数 (オプション) になるようにしてください。

例: `-jvmArgs -Xms256M -Xmx1G`

コンテナー・サーバーの場合の使用法 Windows

```
startOgServer.bat <server> -objectgridFile <xml file>
-deploymentPolicyFile <xml file> [options]
```

Windows

```
startOgServer.bat <server> -objectgridUrl <xml URL>
-deploymentPolicyUrl <xml URL> [options]
```

UNIX

```
startOgServer.sh <server> -objectgridFile <xml file>
-deploymentPolicyFile <xml file> [options]
```

UNIX

```
 startOgServer.sh <server> -objectgridUrl <xml URL>
-deploymentPolicyUrl <xml URL> [options]
```

コンテナー・サーバーのオプション

-catalogServiceEndPoints<hostName:port,hostName:port>
デフォルト: localhost:2809

-deploymentPolicyFile <deployment policy xml file>
デプロイメント・ポリシー・ファイルへのパスを指定します。

例: `../xml/SimpleDP.xml`

-deploymentPolicyUrl <deployment policy url>

-objectgridFile <ObjectGrid descriptor xml file>
ObjectGrid 記述子ファイルへのパスを指定します。

-objectgridUrl <ObjectGrid descriptor url>

-listenerHost <host name>
デフォルト: `localhost`

Internet Inter-ORB Protocol (IIOP) との通信に使用するリスナー・ホストを指定します。

-listenerPort <port>
デフォルト: `2809`

IIOP との通信に使用するリスナー・ポートを指定します。

-serverProps <server properties file>
サーバー・プロパティー・ファイルへのパスを指定します。

例: `../security/server.props`

-zone <zone name>
サーバー内のすべてのコンテナーのために使用するゾーンを指定します。

-traceSpec <trace specification>
サーバーが始動したとき使用可能になるトレースの有効範囲を指定するストリングを指定します。

例:

- `ObjectGrid=all=enabled`
- `ObjectGrid*=all=enabled`

-traceFile <trace file>
トレース情報を保存するファイルのパスを指定します。

例: `../logs/c4Trace.log`

-timeout <seconds>
サーバーの始動がタイムアウトになる秒数を指定します。

-script <script file>
カタログ・サーバーまたはコンテナーを開始するために指定するコマンドのカスタム・スクリプトを作成してから、必要に応じて、パラメーター化および編集を行います。

-jvmArgs <JVM arguments>
JVM 引数 (複数可) を指定します。 **-jvmArgs** パラメーターより後にあるパラメーターは、すべてサーバー Java 仮想マシン (JVM) を始動するために使用されるものです。 **-jvmArgs** パラメーターを使用するときは、最後に指定されるスクリプト引数 (オプション) になるようにしてください。

例:-jvmArgs `-Xms256M -Xmx1G`

スタンドアロン eXtreme Scale サーバーの停止

`stopOgServer` スクリプトを使用して、サーバー・プロセスを停止できます。

手順

- eXtreme Scale コンテナー・プロセスを停止します。
  1. コマンド行から、bin ディレクトリーに移動します。

     `cd` *`objectGridRoot`*`/bin`
  2. stopOgServer スクリプトを実行してサーバーを停止します。 次の例では、c0 サーバーを停止します。

     `stopOgServer` *`c0`* `-catalogServiceEndPoints` *`MyServer1.company.com:2809`*

     以下のように、同じスクリプトを使用して、コンマ区切りリストを指定することで、複数のサーバーを停止します。

     ```
     stopOgServer c0,c1,c2 -catalogServiceEndPoints MyServer1.company.com:2809
     ```

重要: **-catalogServiceEndPoints** オプションは、コンテナーの開始に使用される **-catalogServiceEndPoints** オプションと一致している必要があります。コンテナーの開始に **-catalogServiceEndPoints** を使用しなかった場合には、恐らく、デフォルト値は localhost またはホスト名と 2809 (カタログ・サービスに接続するための ORB ポート) です。それ以外の場合は、カタログ・サービスで **-listenerHost** および **-listenerPort** に渡した値を使用します。カタログ・サービスの開始時に **-listenerHost** および **-listenerPort** オプションが使用されなかった場合、ORB はカタログ・サービスのために localhost のポート 2809 にバインドします。

- eXtreme Scale カタログ・サービス・プロセスを停止します。
  1. コマンド行から、bin ディレクトリーに移動します。

     ```
     cd objectGridRoot/bin
     ```

  2. stopOgServer スクリプトを実行してサーバーを停止します。

     ```
     stopOgServer.sh catalogServer -catalogServiceEndPoints MyServer1.company.com:2809
     ```

  重要: catalogService の停止時には、**-catalogServiceEndPoints** オプションを使用して、カタログ・サービスのオブジェクト・リクエスト・ブローカー (ORB) のホストとポートを参照します。カタログ・サービスでは、**-listenerHost** および **-listenerPort** オプションを使用して ORB バインディング用のホストとポートを指定するか、またはデフォルトのバインディングを受け入れます。カタログ・サービスの開始時に **-listenerHost** および **-listenerPort** オプションが使用されなかった場合、ORB はカタログ・サービスのために localhost のポート 2809 にバインドします。**-catalogServiceEndPoints** オプションは、カタログ・サービスの開始時と停止時では異なります。

  デフォルト・ポートを使用しなかった場合には、カタログ・サービスを開始するには、ピア・アクセス・ポートおよびクライアント・アクセス・ポートが必要です。カタログ・サービスの停止には、ORB ポートのみが必要になります。
- サーバー停止プロセスのトレースを使用可能にします。 コンテナーを停止できない場合は、トレースを使用可能にして問題のデバッグに役立てることができます。サーバーの停止時にトレースを使用可能にするには、**-traceSpec** および **-traceFile** パラメーターを停止コマンドに追加します。 **-traceSpec** パラメーターは使用可能にするトレースのタイプを指定し、**-traceFile** パラメーターはそのトレース・データのために作成して使用するファイルのパスと名前を指定します。
  1. コマンド行から、bin ディレクトリーに移動します。

     ```
     cd objectGridRoot/bin
     ```

  2. トレースを使用可能にして stopOgServer スクリプトを実行します。

     ```
     stopOgServer.sh c4 -catalogServiceEndPoints MyServer1.company.com:2809
      -traceFile ../logs/c4Trace.log -traceSpec ObjectGrid=all=enabled
     ```

     トレースが取得されたら、ポート競合、欠落クラス、欠落または正しくない XML ファイルに関連したエラーがないか、またはスタック・トレースないか調べます。推奨される開始トレース仕様は、以下のとおりです。

     – `ObjectGrid=all=enabled`

     – `ObjectGrid*=all=enabled`

     すべてのトレース仕様オプションについては、483 ページの『トレース・オプション』を参照してください。

関連概念

366 ページの『セキュア eXtreme Scale サーバーの開始と停止』
デプロイメント環境ではサーバーは保護される 必要があることが多く、そのためには開始および停止用の特別な構成が必要です。

69 ページの『ハードウェアおよびソフトウェアの要件』
ハードウェア要件およびオペレーティング・システム要件の概要をご覧ください。WebSphere eXtreme Scale に対して使用するハードウェアまたはオペレーティング・システムのレベルについて、特定のレベルの要件はありませんが、公式にサポートされるハードウェアおよびソフトウェアのオプションに関するオペレーティング・システム別の詳しいリストを、製品サポート・サイトのシステム要件ページから入手できます。 インフォメーション・センターの情報と「システム要件」ページの情報に違いがある場合は、Web サイトの情報を優先してください。インフォメーション・センターの前提条件の情報は、便宜上提供されているだけです。

関連タスク

353 ページの『スタンドアロン WebSphere eXtreme Scale サーバーの始動』
スタンドアロン WebSphere eXtreme Scale 構成を実行しているとき、環境は カタログ・サーバー、コンテナー・サーバー、および eXtreme Scale クライアント・プロセスで構成されています。 eXtreme Scale サーバーは、組み込みサーバー API を使用して、既存の Java アプリケーション内に組み込むこともできます。 これらのプロセスは手動で構成して開始する必要があります。

stopOgServer スクリプト

stopOgServer スクリプトはサーバーを停止します。

目的

名前および catalogServiceEndpoints を指定することで、stopOgServer スクリプトを使用して、サーバーを停止できます。

ロケーション

stopOgServer スクリプトは、ルート・ディレクトリーの bin ディレクトリーにあります。例えば、次のように入力します。

```
cd objectgridRoot/bin
```

使用法

カタログ・サーバーを停止する場合:

Windows

```
stopOgServer.bat <server> -catalogServiceEndPoints
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

UNIX

```
stopOgServer.sh <server> -catalogServiceEndPoints
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

ObjectGrid コンテナー・サーバーを停止する場合:

Windows

```
stopOgServer.bat <server> -catalogServiceEndPoints
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

UNIX

```
startOgServer.sh -catalogServiceEndPoints
<csHost:csListenerPort,csHost:csListenerPort> [options]
```

オプション

-clientSecurityFile <security properties file>

-traceSpec <trace specification>
サーバーが始動したとき使用可能になるトレースの有効範囲を指定するストリングを指定します。

例:

- `ObjectGrid=all=enabled`
- `ObjectGrid*=all=enabled`

-traceFile <trace file>
トレース情報を保存するファイルのパスを指定します。

例: `../logs/c4Trace.log`

-jvmArgs <JVM arguments>
-jvmArgs オプションより後にあるパラメーターは、すべてサーバー Java 仮想マシン (JVM) を始動するために使用されるものです。 -jvmArgs オプションが使用される場合には、そのオプションが、指定された最後のオプション・スクリプト引数であることを確認してください。

セキュア eXtreme Scale サーバーの開始と停止

デプロイメント環境ではサーバーは保護される必要があることが多く、そのためには開始および停止用の特別な構成が必要です。

Java SE 環境でのセキュア・サーバーの開始

カタログ・サービスまたはコンテナー・サーバーは次のように開始できます。

セキュア eXtreme Scale カタログ・サービスの開始

セキュア eXtreme Scale カタログ・サービス・プロセスには、追加で 2 つのセキュリティー構成ファイルが必要です。

セキュリティー記述子 XML ファイル: セキュリティー記述子 XML ファイルは、すべてのサーバー (カタログ・サーバーおよびコンテナー・サーバーを含む) に共通するセキュリティー・プロパティーを記述します。プロパティーの例の 1 つは、ユーザー・レジストリーおよび認証メカニズムを表すオーセンティケーター構成です。

サーバー・プロパティー・ファイル。サーバー・プロパティー・ファイルは、サーバーに固有のセキュリティー・プロパティーを構成します。

startOgServer.sh または startOgServer.cat コマンドを使用してセキュア eXtreme Scale カタログ・サービス・プロセスを開始するときに、-clusterSecurityFile または -clusterSecurityUrl を使用して、ファイル・タイプまたは URL タイプとしてセキュリティー記述子 XML ファイルを設定することができ、–serverProps を使用してサーバー・プロパティー・ファイルを設定することができます。

セキュア eXtreme Scale コンテナー・サーバーの開始

セキュア eXtreme Scale コンテナー・サーバーの開始には、1 つのセキュリティー構成ファイルが必要です。

- **サーバー・プロパティー・ファイル:** サーバー・プロパティー・ファイルは、サーバーに固有のセキュリティー・プロパティーを構成します。詳しくは、 196 ページの『サーバー・プロパティー・ファイル』を参照してください。

startOgServer.sh または startOgServer.cat コマンドを使用してセキュア eXtreme Scale コンテナー・サーバーを開始するときに、 –serverProps を使用してサーバー・プロパティー・ファイルを設定できます。サーバー・プロパティー・ファイルを設定する方法は他にもあります。詳しくは、サーバー・プロパティー・ファイルを参照してください。

startOgServer.sh または startOgServer.bat コマンドとそのオプションの使用方法について詳しくは、 360 ページの『startOgServer スクリプト』を参照してください。

セキュア eXtreme Scale サーバーの停止

セキュア eXtreme Scale カタログ・サービス・プロセスまたはコンテナー・サーバーの停止には、1 つのセキュリティー構成ファイルが必要です。

- **クライアント・プロパティー・ファイル:** クライアント・プロパティー・ファイルを使用して、クライアント・セキュリティー・プロパティーを構成できます。クライアント・セキュリティー・プロパティーは、クライアントがセキュア・サーバーに接続するために必要です。詳しくは、 216 ページの『クライアント・プロパティー・ファイル』を参照してください。

stopOgServer.sh または stopOgServer.cat コマンドを使用してセキュア eXtreme Scale カタログ・サービス・プロセスまたはコンテナー・サーバーを停止するときに、-clientSecurityFile を使用してクライアント・セキュリティー・プロパティーを設定できます。

stopOgServer.sh または stopOgServer.cat コマンドとそのオプションの使用方法について詳しくは、 365 ページの『stopOgServer スクリプト』を参照してください。

WebSphere Application Server でのセキュア・サーバーの始動

WebSphere Application Server でのセキュア ObjectGrid サーバーの開始は、セキュリティー構成ファイルを渡す必要がある点を除いて、非セキュア ObjectGrid サーバーの開始と似ています。 Java SE 環境でコマンド中に -[PROPERTY_FILE] (例えば -serverProps) を使用する代わりに、汎用 Java 仮想マシン (JVM) 引数で -D[PROPERTY_FILE] を使用します。

Websphere Application Server でのセキュア・カタログ・サービスの開始

カタログ・サーバーには、以下の 2 つの異なるレベルのセキュリティー情報が含まれます。

- -Dobjectgrid.cluster.security.xml.url: これは、すべてのサーバー (カタログ・サーバーおよびコンテナー・サーバーを含む) に共通するセキュリティー・プロパティーを記述する objectGridSecurity.xml ファイルを指定します。ユーザー・レジストリーおよび認証メカニズムを表わすオーセンティケーター構成はその一例です。このプロパティーに指定するファイル名は、URL 形式 (例えば「file:///tmp/og/objectGridSecurity.xml」) でなければなりません。
- -Dobjectgrid.server.props: これは、サーバー固有のセキュリティー・プロパティーが入っているサーバー・プロパティー・ファイルを指定します。このプロパティーに指定するファイル名は、普通のファイル・パス形式 (例えば「c:/tmp/og/catalogserver.props」) です。 -Dobjectgrid.security.server.props の使用は推奨されませんが、後方互換性のために引き続き使用できることに注意してください。

WebSphere Application Server 内でセキュア・カタログ・サービスを開始するには、384 ページの『グリッド・セキュリティー』の『WebSphere Application Server への組み込み』の説明に従ってください。

その後、プロセスの汎用 JVM 引数でセキュリティー・プロパティーを設定します。

```
-Dobjectgrid.cluster.security.xml.url=file:///tmp/og/
objectGridSecurity.xml-Dobjectgrid.server.props=/tmp/og/
catalog.server.props
```

汎用 JVM 引数を追加する手順は以下のとおりです。

- 左側のタスク・ビューで「システム管理」を展開します。
- カタログ・サービスがデプロイされる WebSphere Application Server プロセス (例えば、「デプロイメント・マネージャー」) をクリックします。
- 右側のページで、「サーバー・インフラストラクチャー」の下の「Java およびプロセス管理」を展開します。
- 「プロセス定義」をクリックします。
- 「追加プロパティー」の下の「Java 仮想マシン」をクリックします。
- 「汎用 JVM 引数」テキスト・ボックスにプロパティーを入力します。

WebSphere Application Server でのセキュア・コンテナー・サーバーの開始

カタログ・サーバーに接続されたコンテナー・サーバーは、オーセンティケーター構成やログイン・セッション・タイムアウト設定など、objectGridSecurity.xml に構成されたすべてのセキュリティー構成を取得します。またコンテナー・サーバーは、-Dobjectgrid.server.props プロパティーにそのサーバー固有のセキュリティー・プロパティーを構成する必要があります。

このプロパティー・ファイルには、セキュリティーに関係しない他のプロパティーも入れるため、-Dobjectgrid.security.server.props プロパティーの代わりに -Dobjectgrid.server.props プロパティーを使用する必要があります。このプロパティー

に対して指定されるファイル名の形式は、単なるプレーン・ファイル・パス形式です。例えば、`c:/tmp/og/server.props` などです。

上記と同じ手順に従って、セキュリティー・プロパティーを汎用 JVM 引数に追加してください。

WebSphere Application Server による WebSphere eXtreme Scale の管理

WebSphere Application Server でカタログ・サービスおよびコンテナー・サーバー・プロセスを実行できます。これらのサーバーを構成するプロセスは、スタンドアロン構成の場合とは異なります。カタログ・サービスは、WebSphere Application Server サーバーまたはデプロイメント・マネージャーで自動的に開始できます。eXtreme Scale アプリケーションが WebSphere Application Server 環境にデプロイされて、開始されるときに、コンテナー・プロセスは開始されます。

WebSphere Application Server 環境でのカタログ・サービス・プロセスの開始

WebSphere Application Server または WebSphere Application Server Network Deployment 環境では、 WebSphere eXtreme Scale カタログ・サーバーは自動的に始動可能です。カタログ・サービスを WebSphere セル内のどのプロセスでも実行するように構成することができます。開発環境では、単一サーバーのクラスター化されていないカタログ・サービスが許容できます。実稼働環境の場合は、複数のカタログ・サーバーを使用したカタログ・サービス・ドメインを使用する必要があります。

始める前に

- WebSphere Application Server および WebSphere eXtreme Scale をインストールする必要があります。

  eXtreme Scale のグラフィカル・インストールを実行する場合、デプロイメント・マネージャー・プロファイルを含めた既存のプロファイルを拡張するオプションが与えられます。また、インストール後に、プロファイル管理ツール (PMT) (GUI バージョンか、コマンド行 (manageprofiles.sh|bat) からのいずれか) を使用してプロファイルを拡張することも可能です。製品のインストール後に作成されるプロファイルは、プロファイル作成プロセスの一部として拡張することもできるし、後に PMT を使用して拡張することもできます。WebSphere Application Server の 64 ビット・インストール済み環境に対応するプロファイル管理ツールの GUI はありません。この場合、コマンド行から manageprofiles スクリプトを使用してください。

  詳しくは、 17 ページの『第 3 章 WebSphere eXtreme Scale のインストールおよびデプロイ』を参照してください。

- 7.1+ カタログ・サービス・ドメインを定義します。詳しくは、 371 ページの『WebSphere Application Server でのカタログ・サービス・ドメインの作成』を参照してください。

このタスクについて

カタログ・サービス・プロセスは、WebSphere Application Server プロセスで実行できます。カタログ・サービスを実行する場所と方法は、ご使用の WebSphere Application Server のエディションに依存します。

基本 WebSphere Application Server
カタログ・サービスはアプリケーション・サーバー・プロセスで実行されます。デフォルトでは、自動的に開始するようにはなっていません。

WebSphere Application Server Network Deployment
カタログ・サービスは、デプロイメント・マネージャー・プロセスで自動的に実行されますが、1 つ以上のノード・エージェント・プロセスまたはアプリケーション・サーバー・プロセスで実行されるように構成することもできます。

非推奨機能:  以前のリリースでは、catalog.services.cluster カスタム・プロパティーを定義することで、WebSphere Application Server 環境においてカタログ・サーバーのグループを定義していました。以前のリリースでこのカスタム・プロパティーを構成した場合は、その設定がまだ有効になっています。ただし、新たに構成を作成する場合、WebSphere Application Server 管理コンソールにカタログ・サービス・ドメインを作成することで、同じ構成を実現することができます。詳しくは、371 ページの『WebSphere Application Server でのカタログ・サービス・ドメインの作成』を参照してください。

制約事項: サーバーがオブジェクト・リクエスト・ブローカー (ORB) を使用して互いに通信する場合には、WebSphere Application Server 環境内のサーバーで同じ名前を使用することはできません。同じ名前のプロセスでシステム・プロパティー **-Dcom.ibm.websphere.orb.uniqueServerName=true** を指定することで、この制限を解決できます。以下の例では、各ノードにある server1 という名前のサーバーがカタログ・サービスのグリッドとして使用されているか、複数のノード・エージェントを使用してカタログ・サービスのグリッドが形成されています。

手順

- **基本 WebSphere Application Server でのカタログ・サーバーの始動:**

  WebSphere eXtreme Scale が非フェデレート WebSphere Application Server プロファイルに統合される場合、以下の 2 つの状況を除いて、カタログ・サービスは自動的には開始しません。

  – eXtreme Scale コンテナーを自動的に始動するように構成されたアプリケーション。カタログ・サービスとコンテナーはともに自動的に開始します。このフィーチャーにより、カタログ・サービスを明示的に開始する必要がなくなるため、Rational® Application Developer などの開発環境での単体テストが簡略化されます。詳しくは、378 ページの『WebSphere Application Server アプリケーションでの eXtreme Scale コンテナーの自動始動』を参照してください。
  – カタログ・サービス・ドメインが定義されている場合。

- **WebSphere Application Server Network Deployment でのカタログ・サービスの開始:** カタログ・サービスは、WebSphere Application Server Network Deployment のセル内で、以下のシナリオで開始されます。

– WebSphere eXtreme Scale が WebSphere Application Server Network Deployment にインストールされている場合は、カタログ・サービスはデプロイメント・マネージャー・プロセス内で自動的に開始されます (すぐに使用可能なクイック開発が可能なように拡張されている場合)。
– カタログ・サービス・ドメインを定義している場合。カタログ・サービス・ドメインは、定義済みカタログ・サーバーの集合です。これらのカタログ・サーバーは、WebSphere Application Server Network Deployment 環境内の既存のアプリケーション・サーバーで開始することもできるし、リモート・サーバーを定義することもできます。詳しくは、『WebSphere Application Server でのカタログ・サービス・ドメインの作成』を参照してください。

重要: 実稼働環境では、カタログ・サービスを eXtreme Scale コンテナー・サーバーと連結しないようにしてください。カタログ・サービスを、複数のノード・エージェント・プロセス、または eXtreme Scale アプリケーションをホスティングしていないアプリケーション・サーバーに組み込んでください。

カタログ・サービス・ドメインを定義したら、識別された各サーバーを再始動する必要があります。これらのサーバーを再始動すると、カタログ・サービス・ドメインが自動的に始動します。

WebSphere Application Server でのカタログ・サービス・ドメインの作成

カタログ・サービス・ドメインは、断片の配置を管理し、データ・グリッド内のコンテナー・サーバーのヘルスをモニターするカタログ・サーバーのグループを定義します。

始める前に

- WebSphere eXtreme Scale を WebSphere Application Server にインストールします。詳しくは、24 ページの『WebSphere eXtreme Scale または WebSphere eXtreme Scale クライアントの WebSphere Application Server との統合』を参照してください。

このタスクについて

カタログ・サービス・ドメインを作成することで、カタログ・サーバーの高可用性コレクションが定義されます。

これらのカタログ・サーバーは、単一のセルおよびコア・グループ内の WebSphere Application Server で実行できます。カタログ・サービス・ドメインは、異なる Java SE プロセスまたは他の WebSphere Application Server セルで実行されるサーバーのリモート・グループも定義できます。セル内のアプリケーション・サーバーにカタログ・サーバーを配置するカタログ・サービス・ドメインを定義した場合には、WebSphere Application Server のコア・グループ・メカニズムが使用されます。結果として、単一のカタログ・サービス・ドメインのメンバーがコア・グループの境界にまたがることができないため、カタログ・サービス・ドメインは複数のセルにまたがることができません。ただし、WebSphere eXtreme Scale は、セル境界を越えてカタログ・サーバー (スタンドアロン・カタログ・サービス・ドメインや別のセルに組み込まれたカタログ・サービス・ドメインなど) に接続することで、セルにまたがることができます。

重要: 実稼働環境では、カタログ・サービスを WebSphere eXtreme Scale コンテナー・サーバーと連結しないようにしてください。カタログ・サービスを、複数のノード・エージェント・プロセス、または WebSphere eXtreme Scale アプリケーションをホストしていないアプリケーション・サーバーに組み込んでください。

スタンドアロン・モードで実行している場合には、カタログ・サービス・ドメインを作成することもできます。詳しくは、354 ページの『スタンドアロン環境でのカタログ・サービスの開始』を参照してください。

手順

1. カタログ・サービス・ドメインを作成します。
   a. WebSphere Application Server 管理コンソールで、「**システム管理**」 → 「**WebSphere eXtreme Scale**」 → 「**カタログ・サービス・ドメイン**」 → 「**新規**」をクリックします。
   b. カタログ・サービス・ドメインの名前、デフォルト値、JMX 認証資格情報を定義します。
   c. カタログ・サーバー・エンドポイントを追加します。 既存のアプリケーション・サーバーを選択するか、カタログ・サービスを実行しているリモート・サーバーを追加することができます。
2. カタログ・サービス・ドメインへの接続をテストします。
   a. WebSphere Application Server 管理コンソールで、「**システム管理**」 → 「**WebSphere eXtreme Scale**」 → 「**カタログ・サービス・ドメイン**」をクリックします。
   b. テストするカタログ・サービス・ドメインを選択して、「**テスト接続**」をクリックします。 このボタンをクリックすると、すべての定義されたカタログ・サービス・ドメイン・エンドポイントが 1 つずつ照会されます。いずれかのエンドポイントが使用可能であれば、カタログ・サービス・ドメインへの接続が成功したことを示すメッセージが返されます。
3. 既存のアプリケーション・サーバーにカタログ・サーバーを作成した場合は、選択したサーバーを再始動します。選択したサーバーで、カタログ・サービスが自動的に開始されます。

次のタスク

また、wsadmin ツールを使用してこの構成を作成することもできます。コマンドの詳細については、『カタログ・サービス・ドメイン管理用タスク』を参照してください。

カタログ・サービス・ドメイン管理用タスク

Jacl または Jython スクリプト言語を使用して、WebSphere Application Server 構成内のカタログ・サービス・ドメインを管理できます。

要件

WebSphere Application Server 環境に WebSphere eXtreme Scale クライアントをインストールしている必要があります。

すべての管理用タスクのリスト

カタログ・サービス・ドメインに関連したすべての管理用タスクのリストを取得するには、 wsadmin で以下のコマンドを実行します。

```
wsadmin>$AdminTask help XSDomainManagement
```

コマンド

カタログ・サービス・ドメインの管理用タスクには、以下のコマンドが含まれます。

- 『createXSDomain』
- 374 ページの『deleteXSDomain』
- 375 ページの『getDefaultXSDomain』
- 375 ページの『listXSDomains』
- 375 ページの『modifyXSDomain』
- 377 ページの『testXSDomainConnection』
- 377 ページの『testXSServerConnection』

createXSDomain

createXSDomain コマンドは、新規カタログ・サービス・ドメインを登録します。

表 19. createXSDomain コマンド引数

引数	説明
-name (必須)	編集したいカタログ・サービス・ドメインの名前を指定します。
-default	カタログ・サービス・ドメインがセルでデフォルトかどうかを指定します。デフォルト値は `true` です。(ブール値: `true` または `false` に設定する)
-userID	カタログ・サービス・ドメインのユーザー ID を指定します。
-password	カタログ・サービス・ドメインのパスワードを指定します。
-properties	カタログ・サービス・ドメインのカスタム・プロパティーを指定します。

表 20. defineDomainServers ステップ引数

引数	説明
-name	カタログ・サービス・エンドポイントの名前を指定します。
-hostName	エンドポイントが存在するホストの名前を指定します。
-endPoints	カタログ・サービス・ドメイン・エンドポイントのポート番号を指定します。

表 20. *defineDomainServers* ステップ引数 (続き)

引数	説明
-properties	カタログ・サービス・ドメイン・エンドポイントのカスタム・プロパティーを指定します。

戻り値:

バッチ・モードの使用例

- Jacl を使用:

```
$AdminTask createXSDomain {-name TestDomain -default true -userID xsuser
-password ***** -defineDomainServers {{cs1 xhost1.ibm.com "" 5501,2809,1099}
{cs2 xhost2.ibm.com "" 5501,2809,1099}}}
```

- Jython ストリングを使用:

```
AdminTask.createXSDomain('[-name TestDomain -default true -userID xsuser
-password ***** -defineDomainServers [[cs1 xhost1.ibm.com "" 5501,2809,1099]
[cs2 xhost2.ibm.com "" 5501,2809,1099]]]')
```

対話モードの使用例

- Jacl を使用:

```
$AdminTask createXSDomain {-interactive}
```

- Jython ストリングを使用:

```
AdminTask.createXSDomain ('[-interactive]')
```

deleteXSDomain

deleteXSDomain コマンドは、カタログ・サービス・ドメインを削除します。

必須パラメーター:

-name
削除するカタログ・サービス・ドメインの名前を指定します。

戻り値:

バッチ・モードの使用例

- Jacl を使用:

```
$AdminTask deleteXSDomain {-name TestDomain }
```

- Jython ストリングを使用:

```
AdminTask.deleteXSDomain('[-name TestDomain ]')
```

対話モードの使用例

- Jacl を使用:

```
$AdminTask deleteXSDomain {-interactive}
```

- Jython ストリングを使用:

```
AdminTask.deleteXSDomain ('[-interactive]')
```

getDefaultXSDomain

getDefaultXSDomain コマンドは、セルのデフォルト・カタログ・サービス・ドメインを返します。

必須パラメーター: なし

戻り値: デフォルト・カタログ・サービス・ドメインの名前。

バッチ・モードの使用例

- Jacl を使用:

```
$AdminTask getDefaultXSDomain
```

- Jython ストリングを使用:

```
AdminTask.getDefaultXSDomain
```

対話モードの使用例

- Jacl を使用:

```
$AdminTask getDefaultXSDomain {-interactive}
```

- Jython ストリングを使用:

```
AdminTask.getDefaultXSDomain ('[-interactive]')
```

listXSDomains

listXSDomains コマンドは、既存のカタログ・サービス・ドメインのリストを返します。

必須パラメーター: なし

戻り値: セル内のすべてのカタログ・サービス・ドメインのリスト。

バッチ・モードの使用例

- Jacl を使用:

```
$AdminTask listXSDomains
```

- Jython ストリングを使用:

```
AdminTask.listXSDomains
```

対話モードの使用例

- Jacl を使用:

```
$AdminTask listXSDomains {-interactive}
```

- Jython ストリングを使用:

```
AdminTask.listXSDomains ('[-interactive]')
```

modifyXSDomain

modifyXSDomain コマンドは、既存のカタログ・サービス・ドメインを変更します。

表 21. modifyXSDomain コマンド引数

引数	説明
-name (必須)	編集したいカタログ・サービス・ドメインの名前を指定します。
-default	`true` に設定した場合、選択したカタログ・サービス・ドメインがセルのデフォルトであることを指定します。 (ブール値)
-userID	カタログ・サービス・ドメインのユーザー ID を指定します。
-password	カタログ・サービス・ドメインのパスワードを指定します。
-properties	カタログ・サービス・ドメインのカスタム・プロパティーを指定します。

表 22. modifyEndpoints ステップ引数

引数	説明
-name	カタログ・サービス・エンドポイントの名前を指定します。
-hostName	エンドポイントが存在するホストの名前を指定します。
-endPoints	カタログ・サービス・ドメイン・エンドポイントのポート番号を指定します。

表 23. addEndpoints ステップ引数

引数	説明
-name	カタログ・サービス・エンドポイントの名前を指定します。
-hostName	エンドポイントが存在するホストの名前を指定します。
-endPoints	カタログ・サービス・ドメイン・エンドポイントのポート番号を指定します。
-properties	カタログ・サービス・ドメイン・エンドポイントのカスタム・プロパティーを指定します。

表 24. removeEndpoints ステップ引数

引数	説明
-name	削除するカタログ・サービス・エンドポイントの名前を指定します。

戻り値:

バッチ・モードの使用例

- Jacl を使用:

```
$AdminTask modifyXSDomain {-name TestDomain -userID newuser -password *****
-default false -modifyEndpoints {{cs1 xhost1.ibm.com "" 5502,2809,1099}}
-addEndpoints {{cs3 xhost3.ibm.com "" 5501,2809,1099}}
-removeEndpoints {{cs2}}}
```

- Jython ストリングを使用:

```
AdminTask.modifyXSDomain('[-name TestDomain -userID newuser -password *****
-default false -modifyEndpoints [[cs1 xhost1.ibm.com "" 5502,2809,1099]]
-addEndpoints [[cs3 xhost3.ibm.com "" 5501,2809,1099]]
-removeEndpoints [[cs2]]]')
```

対話モードの使用例

- Jacl を使用:

```
$AdminTask modifyXSDomain {-interactive}
```

- Jython ストリングを使用:

```
AdminTask.modifyXSDomain ('[-interactive]')
```

testXSDomainConnection

testXSDomainConnection コマンドは、カタログ・サービス・ドメインへの接続をテストします。

必須パラメーター:

-name
接続をテストするカタログ・サービス・ドメインの名前を指定します。

オプション・パラメーター:

-timeout
接続されるまで待機する最大時間を秒数で指定します。

戻り値: 接続できた場合、`true` が返されます。接続できなかった場合は、接続エラー情報が返されます。

バッチ・モードの使用例

- Jacl を使用:

```
$Admintask testXSDomainConnection
```

- Jython ストリングを使用:

```
AdminTask.testXSDomainConnection
```

対話モードの使用例

- Jacl を使用:

```
$AdminTask testXSDomainConnection {-interactive}
```

- Jython ストリングを使用:

```
AdminTask.testXSDomainConnection ('[-interactive]')
```

testXSServerConnection

testXSServerConnection コマンドは、カタログ・サーバーへの接続をテストします。このコマンドは、スタンドアロン・サーバーと、カタログ・サービス・ドメインに属するサーバーの両方で機能します。

必須パラメーター:

ホスト (host)
カタログ・サーバーが配置されているホストを指定します。

listenerPort
カタログ・サーバーのリスナー・ポートを指定します。

オプション・パラメーター:

timeout
カタログ・サーバーに接続されるまで待機する最大時間を秒数で指定します。

戻り値:

バッチ・モードの使用例

- Jacl を使用:

```
$Admintask testXSServerConnection {-host xhost1.ibm.com -listenerPort 2809}
```

- Jython ストリングを使用:

```
AdminTask.testXSServerConnection('[-host xshost3.ibm.com -listenerPort 2809]')
```

対話モードの使用例

- Jacl を使用:

```
$AdminTask testXSServerConnection {-interactive}
```

- Jython ストリングを使用:

```
AdminTask.testXSServerConnection ('[-interactive]')
```

WebSphere Application Server アプリケーションでの eXtreme Scale コンテナーの自動始動

WebSphere Application Server 環境内のコンテナー・プロセスは、eXtreme Scale XML ファイルを組み込んだモジュールが開始されると自動的に開始されます。

始める前に

WebSphere Application Server および WebSphere eXtreme Scale をインストールする必要があります。さらに、WebSphere Application Server 管理コンソールにアクセスできなければなりません。

このタスクについて

Java Platform, Enterprise Edition アプリケーションのクラス・ローダー規則は複雑であるため、Java EE サーバー内で共用 WebSphere eXtreme Scale を使用しているときは、ロード元クラスが非常に複雑になります。 Java EE アプリケーションは通常、1 つ以上の Enterprise JavaBeans (EJB) または Web アーカイブ (WAR) モジュールがデプロイされる単一のエンタープライズ・アーカイブ (EAR) ファイルです。

WebSphere eXtreme Scale は各モジュールの開始を監視し、eXtreme Scale XML ファイルを検査します。 WebSphere eXtreme Scale は、XML ファイルによるモジュールの開始を検出すると、アプリケーション・サーバーを eXtreme Scale コンテナー Java 仮想マシン (JVM) としてカタログ・サービスに登録します。コンテナーを

カタログ・サービスに登録することにより、さまざまなグリッドに同じアプリケーションをデプロイし、カタログ・サービスで引き続き単一グリッドとして処理することができます。カタログ・サービスは、セル、グリッド、または動的グリッドとの関連はありません。すべてが eXtreme Scale コンテナー JVM であるか、またはそうでないかのいずれかです。必要に応じて、単一の論理グリッドを複数の WebSphere Extended Deployment セルに分散させることができます。

手順

1. META-INF フォルダーに eXtreme Scale XML ファイルが含まれるモジュールを持つように EAR ファイルをパッケージ化します。 WebSphere eXtreme Scale は、EJB および WEB モジュールが開始されると、これらのモジュールの META-INF フォルダーで `objectGrid.xml` および `objectGridDeployment.xml` ファイルの存在を検出します。 `objectGrid.xml` ファイルのみが検出されると、JVM はクライアントと見なされ、その他の場合は、この JVM が `objectGridDeployment.xml` ファイルで定義されているグリッドのコンテナーであると見なされます。

   これらの XML ファイルの正しい名前を使用しなければなりません。ファイル名には大/小文字の区別があります。これらのファイルが存在しないと、コンテナーは開始されません。断片が配置されることを示すメッセージは `systemout.log` ファイルで確認することができます。 eXtreme Scale を使用する EJB モジュールまたは WAR モジュールの `META-INF` ディレクトリーに eXtreme Scale XML ファイルがなければなりません。

   eXtreme Scale XML ファイルには以下のものがあります。

   - `objectGrid.xml` ファイル。通常は eXtreme Scale 記述子 XML ファイルと呼ばれているものです。
   - `objectGridDeployment.xml` ファイル。通常はデプロイメント記述子 XML ファイルと呼ばれているものです。
   - `entity.xml` ファイル。エンティティーが使用された場合 (オプション)。 `entity.xml` ファイル名は、`objectGrid.xml` ファイルに指定されている名前と一致しなければなりません。

   eXtreme Scale ランタイムはこれらのファイルを検出し、カタログ・サービスに連絡して、別のコンテナーを使用してこの eXtreme Scale の断片をホストできることを通知します。

   ヒント: 複数のエンティティーが 1 つの eXtreme Scale サーバーを使用するアプリケーションを試みる場合には、デプロイメント記述子 XML ファイル内の minSyncReplicas 値を `0` に設定します。そうしないと、別のサーバーが minSyncReplica ポリシーを満たしはじめるまで配置を行えないため、`SystemOut.log` ファイル内に以下のメッセージのいずれかが記録される可能性があります。

   ```
   CWPRJ1005E: Error resolving entity association. Entity=entity_name,
   association=association_name.
   (エンティティー関連の解決中にエラーが発生しました。
   エンティティー=entity_name、関連=association_name)
   ```

```
CWOBJ3013E: The EntityMetadata repository is not available.  Timeout
threshold reached when trying to register the entity: entity_name.
(EntityMetadata リポジトリーを使用できません。
エンティティー entity_name の登録試行中にタイムアウトしきい値に到達しました)
```

2. アプリケーションをデプロイして開始します。

   モジュールが開始されると、コンテナーが自動的に開始されます。カタログ・サービスが開始され、できるだけ速やかに区画のプライマリーおよびレプリカ (断片) が配置されます。 objectGridDeployment.xml ファイルで numInitialContainers 属性を定義していない限り、この配置は直ちに行われます。numInitialContainers 属性を定義した場合は、指定した数のコンテナーが開始された時点で配置が開始されます。

次のタスク

コンテナーと同じセル内にあるアプリケーションは、ObjectGridManager.connect(null, null) メソッドを使用してこれらのグリッドに接続した後、getObjectGrid(ccc, "object grid name") メソッドを呼び出すことができます。 connect または getObjectGrid メソッドはコンテナーが断片の配置を完了するまでブロックされることがありますが、このブロックはグリッドが開始されるときだけ問題となります。

ClassLoader

eXtreme Scale に格納されたプラグインまたはオブジェクトは、特定のクラス・ローダーにロードされます。ロードされたオブジェクトは、同じ EAR 内の 2 つの EJB モジュールに含めることができます。これらのオブジェクトは同じですが、別の ClassLoader を使用してロードされています。アプリケーション A がサーバーに対してローカルな eXtreme Scale マップに Person オブジェクトを格納した場合、アプリケーション B がこのオブジェクトを読み取ろうとすると、ClassCastException を受け取ります。この例外は、アプリケーション B が Person オブジェクトを別のクラス・ローダーにロードしたために発生します。

この問題を解決する方法の 1 つは、eXtreme Scale に格納された必要なプラグインおよびオブジェクトをルート・モジュールが含むようにすることです。 eXtreme Scale を使用する各モジュールは、ルート・モジュール内でクラスを参照する必要があります。もう 1 つの解決方法は、これらの共用オブジェクトを、モジュールとアプリケーションの両方が共用する共通クラス・ローダー上のユーティリティー jar 内に配置することです。オブジェクトは、WebSphere クラスまたは lib/ext ディレクトリーにも配置することができますが、デプロイメントが複雑になるため、推奨しません。

EAR ファイル内の EJB モジュールは通常、同じ ClassLoader を共用するため、この問題の影響を受けません。各 WAR モジュールには独自の ClassLoader があり、この問題の影響を受けます。

クライアントとしてのみグリッドに接続

catalog.services.cluster プロパティーがセル、ノード、またはサーバー・カスタム・プロパティーで定義されている場合は、EAR ファイル内のすべてのモジュールが ObjectGridManager.connect (ServerFactory.getServerProperties() .getCatalogServiceBootstrap(), null, null) メソッドを呼び出して ClientClusterContext

を取得し、さらに ObjectGridManager.getObjectGrid(ccc, "grid name") メソッドを呼び出して eXtreme Scale の参照を取得することができます。アプリケーション・オブジェクトがマップに格納されている場合は、それらのオブジェクトが共通の ClassLoader に存在することを確認してください。

Java Platform, Standard Edition クライアントまたはセル外のクライアントは、カタログ・サービス (WebSphere でのデフォルトはデプロイメント・マネージャー) のブートストラップ IIOP ポートを使用して接続して ClientClusterContext を取得し、次に通常の方法で、指定された eXtreme Scale を取得することができます。

エンティティー・マネージャー

エンティティー・マネージャーは役に立ちます。これは、タプルがアプリケーション・オブジェクトではなくマップに格納されていて、クラス・ローダー問題にあまり関係しないためです。ただし、プラグインが問題になることがあります。また、エンティティー (ObjectGridManager.connect("host:port[,host:port], null, objectGridOverride) または ObjectGridManager.connect(null, objectGridOverride)) が定義されている eXtreme Scale に接続する際はクライアント・オーバーライド eXtreme Scale 記述子 XML ファイルが常に必要となるので注意してください。

管理 Bean (MBean) を使用してプログラムで管理する

デプロイメントを管理およびモニターするには、さまざまなタイプの Java Management Extensions (JMX) MBeans を使用できます。各 MBean は、マップ、データ・グリッド、サーバー、サービスなどの特定のエンティティーを参照します。

JMX MBean インターフェースおよび WebSphere eXtreme Scale

各 MBean には、属性値を表す get メソッドがあります。この get メソッドは、プログラムから直接呼び出すことはできません。JMX 仕様では、属性の扱い方が操作のときと異なります。ベンダー JMX コンソールを使用して属性を表示し、プログラムまたはベンダー JMX コンソールで操作を実行することができます。

Package com.ibm.websphere.objectgrid.management

使用可能なすべての MBean の概要と詳細なプログラミング仕様については、作成された API 資料を参照してください: Package com.ibm.websphere.objectgrid.management

wsadmin ツールを使用した MBean へのアクセス

WebSphere Application Server で提供される wsadmin ユーティリティーを使用して、MBean 情報にアクセスすることができます。

WebSphere Application Server インストール内の bin ディレクトリーから wsadmin ツールを実行します。次の例は、動的 eXtreme Scale における現在の断片配置のビューを取得するものです。 wsadmin は、eXtreme Scale が稼働している任意のインストール済み環境から実行できます。 wsadmin をカタログ・サービスで実行する必要はありません。

```
$ wsadmin.sh -lang jython
wsadmin>placementService = AdminControl.queryNames
 ("com.ibm.websphere.objectgrid:*,type=PlacementService")
wsadmin>print AdminControl.invoke(placementService,
 "listObjectGridPlacement","library ms1")

<objectGrid name="library" mapSetName="ms1">
  <container name="container-0" zoneName="DefaultDomain"
  hostName="host1.company.org" serverName="server1">
     <shard type="Primary" partitionName="0"/>
     <shard type="SynchronousReplica" partitionName="1"/>
  </container>
  <container name="container-1" zoneName="DefaultDomain"
  hostName="host2.company.org" serverName="server2">
     <shard type="SynchronousReplica" partitionName="0"/>
     <shard type="Primary" partitionName="1"/>
  </container>
  <container name="UNASSIGNED" zoneName="_ibm_SYSTEM"
  hostName="UNASSIGNED" serverName="UNNAMED">
    <shard type="SynchronousReplica" partitionName="0"/>
    <shard type="AsynchronousReplica" partitionName="0"/>
  </container>
</objectGrid>
```

第 8 章 デプロイメント環境の保護

WebSphere® eXtreme Scale は、データ・アクセスを保護することができ、外部セキュリティー・プロバイダーとの統合も可能です。セキュリティーの側面には、認証、許可、トランスポート・セキュリティー、グリッド・セキュリティー、JMX (Mbean) セキュリティーなどがあります。

ローカル・セキュリティーの使用可能化

WebSphere eXtreme Scale によりいくつかのセキュリティー・エンドポイントが提供され、カスタム・メカニズムを統合できるようになります。ローカル・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、認証サポートはありません。既存の WebSphere Application Server 認証とは別個に認証を行う必要があります。ただし、Subject オブジェクトを取得および検証するプラグインは備えられています。

以下のセクションでは、ローカル・セキュリティーを使用可能にする 2 とおりの方法について説明します。

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。ObjectGridSample エンタープライズ・アプリケーションの例で使用される `secure-objectgrid-definition.xml` ファイルを以下の例に示します。セキュリティーを使用可能にするには、securityEnabled attribute 属性を `true` に設定します。

```
<objectGrids>
    <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
        authorizationMechanism="AUTHORIZATION_MECHANISM_JAAS">
    ...
</objectGrids>
```

セキュリティーをプログラマチックに使用可能にすることもできます。ObjectGrid.setSecurityEnabled メソッドを使用して ObjectGrid を作成するには、ObjectGrid インターフェース上で以下のメソッドを呼び出します。

```
/**
 * Enable the ObjectGrid security
 */
void setSecurityEnabled();
```

詳しくは、API 資料を参照してください。

グリッド認証

セキュア・トークン・マネージャー・プラグインを使用すると、サーバー間の認証が可能になります。そのためには、SecureTokenManager インターフェースを実装する必要があります。

generateToken(Object) メソッドは保護されるオブジェクトを取得し、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 実装は XOR アルゴリズムなど単純なエンコード・アルゴリズムを使用して、オブジェクトをシリアライズ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。この実装は保護されていないため、簡単に中断されます。

WebSphere eXtreme Scale デフォルト実装

WebSphere eXtreme Scale には、このインターフェース用のすぐに使用可能な実装が用意されています。このデフォルト実装は、鍵ペアを使用して署名し、署名を検査します。また、秘密鍵を使用してコンテンツを暗号化します。すべてのサーバーには JCKES タイプの鍵ストアが備えられており、鍵ペア、秘密鍵と公開鍵、および秘密鍵が保管されています。鍵ストアは、秘密鍵を保管する 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付けられています。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、Secure Sockets Layer (SSL) 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、WebSphere eXtreme Scale 管理者が鍵を生成し、生成した鍵をすべてのサーバーにトランスポートする必要があるため、トランスポート中にセキュリティー・ブリーチ (抜け穴) が発生する可能性があることです。

グリッド・セキュリティー

WebSphere eXtreme Scale グリッド・セキュリティーによって、結合サーバーのクレ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グリッドを結合することができません。グリッド・セキュリティーは共有秘密ストリング・メカニズムを使用します。

カタログ・サーバーを含むすべての WebSphere eXtreme Scale サーバーが、共有秘密ストリングと一致しています。サーバーがグリッドに結合する場合、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・サーバーまたはカタログ・サーバーのいずれかの秘密ストリングと一致する場合は、結合サーバーは受け入れられます。ストリングが一致しない場合、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。WebSphere eXtreme Scale 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・プラグインが用意されており、サーバーはこの機密事項を送信する前にセキュアにできます。セキュア操作の実装方法を決定する必要があります。WebSphere eXtreme Scale は、すぐに使用可能な実装を提供し、これによりセキュア操作が実装され、機密事項が暗号化されて署名が行われます。

秘密ストリングは `server.properties` ファイルに設定されます。authenticationSecret プロパティーについての詳細は、196 ページの『サーバー・プロパティー・ファイル』 を参照してください。

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。

SecureTokenManager プラグインについて詳しくは、SecureTokenManager API 資料を参照してください。

generateToken(Object) メソッドはオブジェクトを取得し、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 実装は、排他 OR (XOR) アルゴリズムなどの単純なエンコード・アルゴリズムを使用して、シリアライズ形式でオブジェクトをエンコードし、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。この実装は保護されません。

WebSphere eXtreme Scale には、このインターフェースに対してすぐに使用可能な実装が用意されています。

デフォルトの実装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。すべてのサーバーには JCKES タイプの鍵ストアが備えられており、鍵ペア、秘密鍵と公開鍵、および秘密鍵が保管されています。鍵ストアは、秘密鍵を保管する 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付けられています。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、Secure Sockets Layer (SSL) 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、WebSphere eXtreme Scale 管理者が鍵を生成し、生成した鍵をすべてのサーバーに移送する必要があるため、移送中にセキュリティー・ブリーチが発生する可能性があることです。

セキュア・トークン・マネージャーのデフォルト・プロパティーを作成するためのサンプル・スクリプト

前のセクションで記述したように、署名とシグニチャーの検査を行う鍵ペアおよびコンテンツを暗号化する秘密鍵を含む、鍵ストアを作成することができます。

例えば、以下のように JDK 6 keytool コマンドを使用して鍵を作成できます。

```
keytool -genkeypair -alias keypair1 -keystore key1.jck -storetype JCEKS -keyalg
rsa -dname "CN=sample.ibm.com, OU=WebSphere eXtreme Scale" -storepass key111 -keypass
keypair1 -validity 10000
```

```
keytool -genseckey -alias seckey1 -keystore key1.jck -storetype JCEKS -keyalg
DES  -storepass key111 -keypass seckey1 -validity 1000
```

上記 2 つのコマンドは、鍵ペア「keypair1」と秘密鍵「seckey1」を作成します。次に、サーバー・プロパティー・ファイルで以下のように構成することができます。

```
secureTokenKeyStore=key1.jck
secureTokenKeyStorePassword=key111
secureTokenKeyStoreType=JCEKS
secureTokenKeyPairAlias=keypair1
secureTokenKeyPairPassword=keypair1
secureTokenSecretKeyAlias=seckey1
secureTokenSecretKeyPassword=seckey1
secureTokenCipherAlgorithm=DES
secureTokenSignAlgorithm=RSA
```

構成

セキュア・トークン・マネージャーの構成に使用するプロパティーについて詳しくは、サーバー・プロパティーを参照してください。

アプリケーション・クライアントの認証

アプリケーション・クライアントの認証は、クライアント/サーバー・セキュリティーおよびクレデンシャル認証の使用可能化と、オーセンティケーターおよびシステム・クレデンシャル生成プログラムの構成からなります。

クライアント/サーバー・セキュリティーの使用可能化

ObjectGrid による認証を正常に行うには、クライアントとサーバーの両方でセキュリティーを使用可能にする必要があります。

クライアント・セキュリティーの使用可能化

WebSphere eXtreme Scale は、クライアント・プロパティー・サンプル・ファイル (sampleClient.properties ファイル) を WebSphere Extended Deployment インストール済み環境では *WAS_HOME*/optionalLibraries/ObjectGrid/properties ディレクトリー内、混合サーバー・インストール済み環境では /ObjectGrid/properties ディレクトリー内に提供しています。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。objectgridClient.properties ファイル内の securityEnabled プロパティーを true に設定してください。securityEnabled プロパティーは、セキュリティーが有効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・サイドのこの値は、両方とも true か、両方とも false である必要があります。例えば、接続されているサーバーのセキュリティーが有効な場合、クライアントがサーバーに接続するには、このプロパティー値をクライアント・サイドで true に設定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。詳しくは、ClientSecurityConfiguration API 資料および ClientSecurityConfigurationFactory API 資料を参照してください。

サーバー・セキュリティーの使用可能化

サーバー・サイドでセキュリティーを使用可能にするには、security.xml ファイル内の **securityEnabled** プロパティーを true に設定します。セキュリティー記述子 XML ファイルを使用してデータ・グリッドのセキュリティー構成を指定し、グリッド全体のセキュリティー構成を非セキュリティー構成から分離します。

クレデンシャル認証の使用可能化

eXtreme Scale クライアントが CredentialGenerator オブジェクトを使用して Credential オブジェクトを取得すると、この Credential オブジェクトがクライアント要求とともに eXtreme Scale サーバーに送信されます。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。

クライアントおよびサーバーのプロパティー・ファイルで **credentialAuthentication** プロパティーを設定して、クレデンシャル認証を使用可能にします。詳しくは、216 ページの『クライアント・プロパティー・ファイル』および 196 ページの『サーバー・プロパティー・ファイル』を参照してください。

以下の 2 つの表に、さまざまな設定で、いずれの認証メカニズムが使用されるかを示します。

表 25. クライアントおよびサーバーの設定におけるクレデンシャル認証

クライアント・クレデンシャル認証	サーバー・クレデンシャル認証	結果
なし	常になし	使用不可
なし	サポートされる	使用不可
なし	必須	Error case
サポートされる	常になし	使用不可
サポートされる	サポートされる	使用可能
サポートされる	必須	使用可能
必須	常になし	Error case
必須	サポートされる	使用可能
必須	必須	使用可能

オーセンティケーターの構成

eXtreme Scale サーバーは、Authenticator プラグインを使用して、Credential オブジェクトの認証を行います。Authenticator インターフェースの実装では、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。eXtreme Scale は、レジストリー構成を提供しません。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。

例えば、1 つの Authenticator 実装では、クレ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。この実装では、Java 認証・承認サービス (JAAS) ログイン・モジュールを使用できます。認証の結果として、Subject オブジェクトが戻されます。

以下の例のように、セキュリティー記述子 XML ファイルでオーセンティケーターを構成できます。

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

 <security securityEnabled="true"
        loginSessionExpirationTime="300">

  <authenticator className ="com.ibm.websphere.objectgrid.security.plugins.builtins.KeyStoreLoginAuthenticator">
        </authenticator>

 </security>

</securityConfig>
```

セキュア・サーバーを開始して、セキュリティー XML ファイルを設定する場合は、**-clusterSecurityFile** オプションを使用します。詳しくは、*製品概要*の Java SE セキュリティーのチュートリアルを参照してください。

システム・クレデンシャル生成プログラムの構成

このシステム・クレデンシャル生成プログラムは、システム・クレデンシャルのファクトリーを表すために使用されます。システム・クレデンシャルは、管理者クレデンシャルに似ています。以下の例のように、カタログ・セキュリティー XML 内で SystemCredentialGenerator 要素を構成できます。

```
<systemCredentialGenerator className ="com.ibm.websphere.objectgrid.security.plugins.
 builtins.UserPasswordCredentialGenerator">
          <property name="properties" type="java.lang.String" value="manager manager1"
      description="username password" />
 </systemCredentialGenerator>
```

デモンストレーション用のため、ユーザー名およびパスワードは平文で保管されます。実稼働環境では、ユーザー名およびパスワードは平文で保管しないでください。

WebSphere eXtreme Scale が提供するデフォルトのシステム・クレデンシャル生成プログラムは、サーバー・クレデンシャルを使用します。システム・クレデンシャル生成プログラムを明示的に指定しないと、このデフォルトのシステム・クレデンシャル生成プログラムが使用されます。

アプリケーション・クライアントの許可

アプリケーション・クライアントの許可は、ObjectGrid 許可クラス、許可メカニズム、許可検査期間、および作成者限定アクセス許可から構成されます。

eXtreme Scale の許可は Subject オブジェクトおよびアクセス権に基づいています。本製品は、2 種類の許可メカニズム、つまり、Java 認証・承認サービス (JAAS) とカスタム許可をサポートしています。

ObjectGrid 許可クラス

許可はアクセス権に基づいています。許可クラスには次の 4 つの異なるタイプがあります。

- MapPermission: このクラスは、ObjectGrid マップ内のデータへのアクセスの許可を表します。
- ObjectGridPermission: このクラスは、ObjectGrid へのアクセスの許可を表します。
- ServerMapPermission: このクラスは、クライアントからサーバー・サイドの ObjectGrid マップへのアクセスの許可を表します。
- AgentPermission: このクラスは、サーバー・サイドのエージェントを開始する許可を表します。

API および関連許可について詳しくは、*プログラミング・ガイド*のクライアント許可プログラミングに関するトピックを参照してください。

許可検査期間

eXtreme Scale は、パフォーマンス上の理由で、マップ許可検査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、メソッドおよびそれらに必要な許可のリストにあるメソッドが呼び出されたときに、ランタイムは、構成された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。

アクセス権の許可情報は Subject オブジェクトに基づいています。クライアントがメソッドにアクセスしようとすると、eXtreme Scale ランタイムは、Subject オブジェクトに基づいてキャッシュ内を検索します。キャッシュ内でオブジェクトが見つからない場合、ランタイムは、この Subject オブジェクトに付与されている許可を確認し、この許可をキャッシュに格納します。

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。

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。以下の例では、許可検査期間が 45 秒に設定されています。

```
<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>
```

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 を直接インスタンス化する場合のローカル eXtreme Scale 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 you
 * want to check the permission used to allow a client access. If the
 * parameter is 0 then every single get/put/update/remove/evict call
 * asks the authorization mechanism, either JAAS authorization or custom
 * authorization, to check if the current subject has permission. This might be
 * prohibitively expensive from a performance point of view depending on
 * the authorization implementation, but if you need to have ever call check the
 * authorization mechanism, then set the parameter to 0.
 * 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 value provides much
 * better performance, but if the back-end
 * permissions are changed during this time then the ObjectGrid can
 * allow or prevent access even though the back-end security
 * provider was modified.
 *
 * @param period the permission check period in seconds.
 */
void setPermissionCheckPeriod(int period);
```

作成者限定アクセス許可

作成者限定アクセス許可を使用すると、エントリーを ObjectGrid マップに挿入したユーザーのみ (関連付けられた Principal オブジェクトによって表される) が、そのエントリーにアクセス (read、update、invalidate、および remove) できます。

既存の ObjectGrid マップの許可モデルは、アクセス・タイプに基づいていて、データ・エントリーには基づいていません。すなわち、ユーザーは、read、write、insert、delete、または invalidate などの特定のアクセス・タイプをマップ内のすべてのデータに対して保持しているか、またはどのデータに対しても保持していないかのいずれかです。しかし、eXtreme Scale は、個別のデータ・エントリーに対するユーザーの許可は行いません。この機能は、データ・エントリーに対してユーザーを許可するための新しい方法を提供します。

さまざまなユーザーが異なるデータのセットにアクセスするようなシナリオでは、このモデルが便利です。ユーザーがデータを永続ストアから ObjectGrid マップにロードするときに、永続ストアによってアクセスを許可できます。このケースでは、ObjectGrid マップ層で別の許可を実行する必要がありません。必要な処理は、作成者限定アクセスの機能を使用可能にして、データをマップにロードするユーザーが、確実にそのデータにアクセスできるようにするのみです。

以下の作成者限定モード属性値があります。

disabled
作成者限定アクセス機能は使用不可になっています。

complement
作成者限定アクセス機能が使用可能になり、マップ許可を補完します。すなわち、マップ許可と作成者限定アクセスの機能の両方が有効になります。結果、データに対する操作をさらに制限することができます。例えば、作成者はデータを無効化できないようにすることができます。

supersede
作成者限定アクセス機能が使用可能になり、マップ許可を置き換えます。すなわち、作成者限定アクセス機能がマップ許可に取って代わり、マップ許可は実行されなくなります。

作成者限定アクセス・モードは、次の 2 とおりの方法で構成できます。

XML ファイルを使用:

以下の例のように、ObjectGrid XML ファイルを使用して ObjectGrid を定義し、作成者限定アクセス・モードを disabled、complement、または supersede のいずれかに設定できます。

```
<objectGrids>
    <objectGrid name="secureClusterObjectGrid" securityEnabled="true"
        accessByCreatorOnlyMode="supersede"
```

```
        <bean id="TransactionCallback"
              classname="com.ibm.websphere.samples.objectgrid.HeapTransactionCallback" />
    ...
</objectGrids>
```

プログラムで:

ObjectGrid をプログラマチックに作成する場合、以下のメソッドを呼び出して作成者限定アクセス・モードを設定できます。このメソッドの呼び出しは、直接 ObjectGrid インスタンスを生成する場合のローカル eXtreme Scale プログラミング・モデルにのみ適用されます。

```
/**
 * Set the "access by creator only" mode.
 * Enabling "access by creator only" mode ensures that only the user (represented
 * by the Principals associated with it), who inserts the record into the map,
 * can access (read, update, invalidate, and remove) the record.
 * The "access by creator only" mode can be disabled, or can complement the
 * ObjectGrid authorization model, or it can supersede the ObjectGrid
 * authorization model. The default value is disabled:
 * {@link SecurityConstants#ACCESS_BY_CREATOR_ONLY_DISABLED}.
 * @see SecurityConstants#ACCESS_BY_CREATOR_ONLY_DISABLED
 * @see SecurityConstants#ACCESS_BY_CREATOR_ONLY_COMPLEMENT
 * @see SecurityConstants#ACCESS_BY_CREATOR_ONLY_SUPERSEDE
 *
 * @param accessByCreatorOnlyMode the access by creator mode.
 *
 * @since WAS XD 6.1 FIX3
*/
void setAccessByCreatorOnlyMode(int accessByCreatorOnlyMode);
```

詳細を示すために、ObjectGrid マップ・アカウントがバンキング・グリッドにあり、Manager1 と Employee1 が 2 人のユーザーであるようなシナリオを考えてみます。この場合、eXtreme Scale 許可ポリシーは、「Manager1」に対してはすべてのアクセス権を付与しますが、「Employee1」に対しては読み取りアクセス権しか付与しません。以下の例に示すのは、ObjectGrid マップ許可の JAAS ポリシーです。

```
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    Principal com.acme.PrincipalImpl "Manager1" {
    permission com.ibm.websphere.objectgrid.security.MapPermission
        "banking.account", "all"
};
grant codebase "http://www.ibm.com/com/ibm/ws/objectgrid/security/PrivilegedAction"
    Principal com.acme.PrincipalImpl "Employee1" {
    permission com.ibm.websphere.objectgrid.security.MapPermission
        "banking.account", "read, insert"
};
```

作成者限定アクセスの機能が、どのように許可に影響するか検討してください。

- **disabled**: 作成者限定アクセスの機能が使用不可に設定された場合、マップ許可への影響はありません。ユーザー「Manager1」は、「account」マップ内のすべてのデータにアクセスできます。ユーザー「Employee1」は、マップ内のすべてのデータの read および insert は許可されますが、マップ内のデータに対する update、invalidate、remove はできません。
- **complement**: 「complement」オプションを使用して作成者限定アクセスの機能を使用可能にした場合、マップ許可と作成者限定アクセスの機能の両方が有効になります。ユーザー「Manager1」は、自身が単独でデータをマップにロードした場合のみ、「account」マップ内のデータにアクセスできます。ユーザー「Employee1」は、自身が単独でデータをマップにロードした場合のみ、「account」マップ内のデータを読み取ることができます。(しかし、このユーザーは、マップ内のデータに対する update、invalidate、または remove は許可されません。)

- **supersede**: 「supersede」オプションを使用して作成者限定アクセスの機能を使用可能にした場合、マップ許可は実施されません。許可ポリシーは、作成者限定アクセスの許可のみになります。ユーザー「Manager1」には、「complement」モードの場合と同じ特権が与えられます。このユーザーは、自身がデータをマップにロードした場合のみ、「account」マップ内のデータにアクセスできます。しかし、今回はユーザー「Employee1」も、自身がデータをマップにロードすれば、「account」マップ内のデータに対する全アクセス権限が与えられます。つまり、Java 認証・承認サービス (JAAS) ポリシーに定義されている許可ポリシーは実施されません。

トランスポート層セキュリティーおよび Secure Sockets Layer

WebSphere eXtreme Scale は、クライアントとサーバーとの間のセキュア通信に TCP/IP も、Transport Layer Security/Secure Sockets Layer (TLS/SSL) もサポートします。

TLS/SSL は、クライアントとサーバーとの間のセキュア通信を可能にします。使用される通信メカニズムは、クライアントおよびサーバーの構成ファイル内に指定された transportType パラメーターの値によって決まります。

以下のクライアントおよびサーバー構成ファイル内に transportType プロパティーを設定できます。

- クライアント・セキュリティー構成内でこのプロパティーを設定するには、 216 ページの『クライアント・プロパティー・ファイル』を参照してください。
- コンテナー・サーバー・セキュリティー構成内でこのプロパティーを設定するには、 196 ページの『サーバー・プロパティー・ファイル』を参照してください。
- カタログ・サーバー・セキュリティー構成内でこのプロパティーを設定するには、 196 ページの『サーバー・プロパティー・ファイル』を参照してください。

表 26. 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

クライアントの transportType プロパティー	サーバーの transportType プロパティー	結果のプロトコル
TCP/IP	TCP/IP	TCP/IP
TCP/IP	SSL サポート	TCP/IP
TCP/IP	SSL 必須	エラー
SSL サポート	TCP/IP	TCP/IP
SSL サポート	SSL サポート	SSL (SSL が失敗した場合は TCP/IP)
SSL サポート	SSL 必須	SSL
SSL 必須	TCP/IP	エラー
SSL 必須	SSL サポート	SSL
SSL 必須	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 構成パラメーターが指定されている必要があります。Java SE 環境では、SSL 構成はクライアントまたはサーバーのプロパティー・ファイル内で構成されます。クライアントまたはサーバーが WebSphere Application Server 内にある場合、WebSphere Application Server 内のトランスポート・セキュリティー・サポート を使用して SSL パラメーターを構成できます。

トランスポート・セキュリティー・サポート用の orb.properties ファイルの構成

transportType プロパティーの値が SSL-Supported の場合は、TLS/SSL を使用することができます。

Java Platform, Standard Edition 環境でセキュア・トランスポートをサポートするには、 206 ページの『ORB プロパティー・ファイル』 ファイルを変更して、以下のプロパティーが含まれるようにする必要があります。

```
# IBM JDK properties
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton
javax.rmi.CORBA.StubClass=com.ibm.rmi.javax.rmi.CORBA.StubDelegateImpl
javax.rmi.CORBA.PortableRemoteObjectClass=com.ibm.rmi.javax.rmi.PortableRemoteObject
javax.rmi.CORBA.UtilClass=com.ibm.ws.orb.WSUtilDelegateImpl

# WS Plugins
com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.transport.WSTransport
com.ibm.CORBA.ORBPluginClass.com.ibm.ws.orbimpl.WSORBPropertyManager
com.ibm.CORBA.ORBPluginClass.com.ibm.ISecurityUtilityImpl.SecurityPropertyManager

# WS Interceptors
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityComponentFactory

# WS ORB & Plugins properties
com.ibm.ws.orb.transport.ConnectionInterceptorName=com.ibm.ISecurityLocalObjectBaseL13Impl.SecurityConnectionInterceptor
com.ibm.ws.orb.transport.WSSSLClientSocketFactoryName=com.ibm.ws.security.orbssl.WSSSLClientSocketFactoryImpl

com.ibm.CORBA.TransportMode=Pluggable
com.ibm.CORBA.ServerName=ogserver
```

eXtreme Scale クライアント用の SSL パラメーターの構成

次の方法でクライアントの SSL パラメーターを構成することができます。

1. com.ibm.websphere.objectgrid.security.config. ClientSecurityConfigurationFactory ファクトリー・クラスを使用して、
   com.ibm.websphere.objectgrid.security.config.SSLConfiguration オブジェクトを作成します。詳しくは、ClientSecurityConfigurationFactory API 資料を参照してください。
2. client.propeties ファイル内でパラメーターを構成し、次に、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。

クライアントに設定できるプロパティーの例については、 216 ページの『クライアント・プロパティー・ファイル』でセキュリティー・クライアント・プロパティーに関するセクションを参照してください。

eXtreme Scale サーバー用の SSL パラメーターの構成

サーバーの SSL パラメーターの構成は、上記の server.properties ファイルの例のようなサーバー・プロパティー・ファイルを使用して行います。このプロパティー・ファイルは、eXtreme Scale サーバーの始動時にパラメーターとして渡すことができます。eXtreme Scale サーバーに対して設定できる SSL パラメーターについて詳しくは、 196 ページの『サーバー・プロパティー・ファイル』を参照してください。

WebSphere Application Server でのトランスポート・セキュリティー・サポート

eXtreme Scale クライアント、コンテナー・サーバー、またはカタログ・サーバーが WebSphere Application Server プロセスで実行している場合、eXtreme Scale トランスポート・セキュリティーは Application Server CSIV2 トランスポート設定によって管理されます。 eXtreme Scale クライアントまたはコンテナー・サーバーについては、SSL 設定を構成するために eXtreme Scale クライアントまたはサーバーのプロパティーを使用するべきではありません。 すべての SSL 設定は、WebSphere Application Server 構成で指定するようにしてください。

ただし、カタログ・サーバーは少し異なります。 カタログ・サーバーは独自のトランスポート・パスを持っていますが、これは Application Server CSIV2 トランスポート設定では管理できません。このため、SSL プロパティーは引き続き、カタログ・サーバーに対してサーバー・プロパティー・ファイルで構成する必要があります。

Sun JDK のトランスポート・セキュリティーの使用可能化

WebSphere eXtreme Scale には IBM Java Secure Sockets Extension (IBMJSSE) または IBM Java Secure Sockets Extension 2 (IBMJSSE2) が必要です。 IBMJSSE プロバイダーおよび IBMJSSE2 プロバイダーには、SSL プロトコル、Transport Layer Security (TLS) プロトコル、およびアプリケーション・プログラミング・インターフェース (API) フレームワークをサポートするリファレンス実装が含まれています。

純正の Sun JDK は IBM JSSE プロバイダーおよび IBM JSSE2 プロバイダーを出荷しないため、Sun JDK でトランスポート・セキュリティーは使用可能になりません。 この処理を行うためには、WebSphere Application Server に同梱されている Sun JDK が必要です。 WebSphere Application Server に同梱された Sun JDK には IBM JSSE プロバイダーおよび IBM JSSE2 プロバイダーが含まれています。

WebSphere eXtreme Scale で非 IBM JDK を使用できるようにするには、オブジェクト・リクエスト・ブローカーの構成を参照してください。 `-Djava.endorsed.dirs` を構成する場合は、`objectgridRoot/lib/endorsed` ディレクトリーと `JRE/lib/endorsed` ディレクトリーのどちらもポイントします。 ディレクトリー `objectgridRoot/lib/endorsed` は IBM ORB を使用するために必要で、ディレクトリー `JRE/lib/endorsed` は、IBM JSSE プロバイダーおよび IBM JSSE2 プロバイダーをロードするために必要です。

SSL 必須プロパティーの構成方法、鍵ストアとトラストストアの作成方法、および WebSphere eXtreme Scale でのセキュア・サーバーの開始方法について詳しくは、「*製品概要*」のセキュリティー・チュートリアルのステップ 4 の作業を行ってください。

Java Management Extensions (JMX) セキュリティー

分散環境での Managed Bean (MBean) 呼び出しを保護することができます。

使用可能な MBean に関する詳細は、 381 ページの『管理 Bean (MBean) を使用してプログラムで管理する』を参照してください。

分散デプロイメント・トポロジーでは、MBean は、カタログ・サーバーおよびコンテナー・サーバーで直接ホストされます。 一般に、分散トポロジーの JMX セキュリティーは、JavaTM Management Extensions (JMX) 仕様に指定された JMX セキュリティー仕様に従います。これは、以下の 3 つのパートで構成されます。

1. 認証 - リモート・クライアントは、コネクター・サーバー内で認証される必要があります。
2. アクセス制御 - MBean アクセス制御は、MBean 情報にアクセスできるユーザー、および MBean 操作を実行できるユーザーを制限します。
3. セキュア・トランスポート - JMX クライアントとサーバー間のトランスポートは、TLS/SSL を使用して保護できます。

認証

JMX では、コネクター・サーバーがリモート・クライアントを認証するメソッドを提供しています。RMI コネクターの場合、認証は、コネクター・サーバーが作成される場合に JMXAuthenticator インターフェースを実装するオブジェクトを提供することにより実行されます。eXtreme Scale は、この JMXAuthenticator インターフェースを実装し、 ObjectGrid Authenticator プラグインを使用してリモート・クライアントを認証します。 eXtreme Scale がクライアントをどのように認証するのかについて詳しくは、「*製品概要*」でセキュリティーに関するチュートリアルを参照してください。

JMX クライアントは、JMX API に従って、コネクター・サーバーに接続するためのクレデンシャルを提供します。JMX フレームワークは、クレデンシャルをコネクター・サーバーに渡し、認証のため、JMXAuthenticator 実装を呼び出します。前述のように、JMXAuthenticator 実装は、ObjectGrid Authenticator 実装に認証を委任します。

以下の例は、クレデンシャルを使用してコネクター・サーバーに接続する方法を説明していますので、参考にしてください。

```
javax.management.remote.JMXServiceURL jmxUrl = new JMXServiceURL(
        "service:jmx:rmi:///jndi/rmi://localhost:1099/objectgrid/MBeanServer");

    environment.put(JMXConnector.CREDENTIALS, new UserPasswordCredential("admin", "xxxxxx"));

    // Create the JMXCconnectorServer
    JMXConnector cntor = JMXConnectorFactory.newJMXConnector(jmxUrl, null);

    // Connect and invoke an operation on the remote MBeanServer
    cntor.connect(environment);
```

上記の例では、UserPasswordCredential が、ユーザー ID が admin に、パスワードが xxxxx に設定されて提供されます。この UserPasswordCredential オブジェクトは、環境マップに設定され、これは、JMXConnector.connect(Map) メソッドで使用されます。次に、この UserPasswordCredential オブジェクトは、JMX フレームワークによってサーバーに渡され、最終的に認証のために ObjectGrid 認証フレームワークに渡されます。

クライアント・プログラミング・モデルは、厳格に JMX 仕様に従います。

アクセス制御

JMX MBean サーバーは、機密情報に対するアクセス権を持つことがあり、機密操作を実行することができる場合があります。JMX では、どのクライアントがその情報にアクセスでき、どのユーザーがそうした操作を実行できるかを識別する、必要なアクセス制御を提供しています。アクセス制御は、標準 Java セキュリティー・モデルに基づいて、MBean サーバーおよびその操作へのアクセスを制御する許可を定義することによって構築されます。

JMX 操作のアクセス制御または許可に関して、eXtreme Scale は、JMX 実装で提供される JAAS サポートに依存しています。プログラム実行の任意の時点で、実行のスレッドが保持する現行の許可セットがあります。そのようなスレッドが、JMX 仕様操作を呼び出す場合、これらは、保持許可と呼ばれています。 JMX 操作が実行されると、セキュリティー・チェックが行われ、必要な許可が保持許可によって暗黙的に示されているかどうかがチェックされます。

MBean ポリシー定義は、Java ポリシー形式に従います。例えば、以下のポリシーでは、すべての署名者およびすべてのコード・ベースに PlacementServiceMBean のサーバー JMX アドレスを取得する権限を付与していますが、com.ibm.websphere.objectgrid ドメインに対しては制限しています。

```
grant {
    permission javax.management.MBeanPermission
        "com.ibm.websphere.objectgrid.management.PlacementServiceMBean#retrieveServerJMXAddress
     [com.ibm.websphere.objectgrid:*,type=PlacementService]",
        "invoke";
}
```

以下のポリシー・サンプルを使用すれば、リモート・クライアント ID に基づく許可を完了することができます。このポリシーでは、前の例に示されたものと同じ MBean 許可を付与していますが、X500Principal 名が CN=Administrator、OU=software、O=IBM、L=Rochester、ST=MN、C=US のユーザーだけは除きます。

```
grant principal javax.security.auth.x500.X500Principal "CN=Administrator,OU=software,O=IBM,
    L=Rochester,ST=MN,C=US" {permission javax.management.MBeanPermission
        "com.ibm.websphere.objectgrid.management.PlacementServiceMBean#retrieveServerJMXAddress
    [com.ibm.websphere.objectgrid:*,type=PlacementService]",
        "invoke";
}
```

Java ポリシーは、セキュリティー・マネージャーがオンになっている場合に限ってチェックされます。-Djava.security.manager JVM 引数を設定してカタログ・サーバーおよびコンテナー・サーバーを始動し、MBean 操作のアクセス制御を強制するようにしてください。

セキュア・トランスポート

JMX クライアントとサーバー間のトランスポートは、TLS/SSL を使用して保護することができます。カタログ・サーバーまたはコンテナー・サーバーの transportType が SSL_Required または SSL_Supported に設定されている場合、SSL を使用して JMX サーバーに接続する必要があります。

SSL を使用するには、-D システム・プロパティーを使用して、トラストストア、トラストストア・タイプ、およびトラストストア・パスワードを MBean クライアントに構成する必要のある必要があります。

1. -Djavax.net.ssl.trustStore=TRUST_STORE_LOCATION

2. -Djavax.net.ssl.trustStorePassword=TRUST_STORE_PASSWORD
3. -Djavax.net.ssl.trustStoreType=TRUST_STORE_TYPE

JAVA_HOME/jre/lib/security/java.security ファイルで SSL ソケット・ファクトリーとして com.ibm.websphere.ssl.protocol.SSLSocketFactory を使用する場合は、次のプロパティーを使用します。

1. -Dcom.ibm.ssl.trustStore=TRUST_STORE_LOCATION
2. -Dcom.ibm.ssl.trustStorePassword=TRUST_STORE_PASSWORD
3. -Dcom.ibm.ssl.trustStoreType=TRUST_STORE_TYPE

外部プロバイダーとのセキュリティー統合

WebSphere eXtreme Scale データを保護するために、eXtreme Scale は、いくつかのセキュリティー・プロバイダーと統合することができます。

WebSphere eXtreme Scale は、外部のセキュリティー実装と統合できます。この外部実装は、eXtreme Scale に認証サービスおよび許可サービスを提供する必要があります。eXtreme Scale には、セキュリティー実装と統合するためのプラグイン・ポイントがあります。WebSphere eXtreme Scale は、以下のコンポーネントと正常に統合されています。

- Lightweight Directory Access Protocol (LDAP)
- Kerberos
- ObjectGrid セキュリティー
- Tivoli Access Manager
- Java 認証・承認サービス (JAAS)

eXtreme Scale は、以下のタスクにセキュリティー・プロバイダーを使用します。

- クライアントをサーバーに認証する。
- クライアントに対して、特定の eXtreme Scale 成果物へアクセスする権限、または eXtreme Scale 成果物に対して行うことができる操作を指定する権限を与える。

eXtreme Scale には、以下のタイプの許可があります。

マップ許可
クライアントまたはグループに、マップに対する挿入、読み取り、更新、除去、および削除の操作を許可することができます。

ObjectGrid 許可
クライアントまたはグループに、オブジェクト・グリッドでオブジェクト照会またはエンティティー照会を実行する許可を与えることができます。

DataGrid エージェント許可
クライアントまたはグループに、DataGrid エージェントの ObjectGrid へのデプロイを許可することができます。

サーバー・サイド・マップ許可
クライアントまたはグループに、サーバー・マップをクライアント・サイドに複製すること、またはサーバー・マップに動的索引を作成することを許可できます。

管理許可
クライアントまたはグループに、管理タスク実行の許可を与えることができます。

注: バックエンドに対して既にセキュリティーを有効にしている場合、こうしたセキュリティー設定ではもはや十分にデータを保護できないことに注意してください。データベースまたは他のデータ・ストアのセキュリティー設定は、決してキャッシュに転送されません。認証、許可、トランスポートのレベルのセキュリティーなど、eXtreme Scale のセキュリティー・メカニズムを使用して、現在キャッシュされているデータを個別に保護する必要があります。

制約事項: スタンドアロンの WeSbphere eXtreme Scale 7.1 (または 7.0) と共に SSL トランスポート層セキュリティーを使用する場合は、JDK/jre 1.6 以降を使用しないでください。JDK/jre 1.6 以降のバージョンは、WXS 7.1 アプリケーション・プログラミング・インターフェースをサポートしていません。スタンドアロンの eXtreme Scale インストール済み環境用の SSL トランスポート・セキュリティーを必要とする構成には、JDK/jre 1.5 以前のバージョンを使用してください。この条件は、スタンドアロンの eXtreme Scale 構成で SSL セキュリティーを使用する場合にのみ適用されます。非 SSL のトランスポート構成においては、JDK/jre はサポートされています。

WebSphere Application Server とのセキュリティー統合

WebSphere eXtreme Scale 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いくつかのセキュリティー・フィーチャーがあります。

認証統合

eXtreme Scale クライアントおよびサーバーが WebSphere Application Server および同じセキュリティー・ドメインで稼働中の場合、 WebSphere Application Server セキュリティー・インフラストラクチャーを使用して、クライアント認証クレデンシャルを eXtreme Scale サーバーに伝搬することができます。例えば、サーブレットが eXtreme Scale クライアントとして動作して、同じセキュリティー・ドメインの eXtreme Scale サーバーに接続し、そのサーブレットが既に認証されている場合、認証トークンをクライアント (サーブレット) からサーバーに伝搬し、その後、WebSphere Application Server セキュリティー・インフラストラクチャーを使用して、認証トークンを元のクライアント・クレデンシャルに変換することができます。

WebSphere Application Server との分散セキュリティー統合

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完了できます。

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

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

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

詳しくは、 386 ページの『アプリケーション・クライアントの認証』を参照してください。以下に、WSTokenCredentialGenerator クラスの使用方法の例を示します。

```
/**
 * 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、WSTokenAuthentication オーセンティケーターを使用して、WSTokenCredential オブジェクトを認証します。

WebSphere Application Server とのローカル・セキュリティー統合

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完了できます。

- 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、*プログラミング・ガイド* のローカル・セキュリティーに関する情報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。

セキュア eXtreme Scale サーバーの開始と停止

デプロイメント環境ではサーバーは保護される必要があることが多く、そのためには開始および停止用の特別な構成が必要です。

Java SE 環境でのセキュア・サーバーの開始

カタログ・サービスまたはコンテナー・サーバーは次のように開始できます。

セキュア eXtreme Scale カタログ・サービスの開始

セキュア eXtreme Scale カタログ・サービス・プロセスには、追加で 2 つのセキュリティー構成ファイルが必要です。

セキュリティー記述子 XML ファイル: セキュリティー記述子 XML ファイルは、すべてのサーバー (カタログ・サーバーおよびコンテナー・サーバーを含む) に共通

するセキュリティー・プロパティーを記述します。プロパティーの例の 1 つは、ユーザー・レジストリーおよび認証メカニズムを表すオーセンティケーター構成です。

サーバー・プロパティー・ファイル。サーバー・プロパティー・ファイルは、サーバーに固有のセキュリティー・プロパティーを構成します。

startOgServer.sh または startOgServer.cat コマンドを使用してセキュア eXtreme Scale カタログ・サービス・プロセスを開始するときに、-clusterSecurityFile または -clusterSecurityUrl を使用して、ファイル・タイプまたは URL タイプとしてセキュリティー記述子 XML ファイルを設定することができ、–serverProps を使用してサーバー・プロパティー・ファイルを設定することができます。

セキュア eXtreme Scale コンテナー・サーバーの開始

セキュア eXtreme Scale コンテナー・サーバーの開始には、1 つのセキュリティー構成ファイルが必要です。

- **サーバー・プロパティー・ファイル:** サーバー・プロパティー・ファイルは、サーバーに固有のセキュリティー・プロパティーを構成します。詳しくは、196 ページの『サーバー・プロパティー・ファイル』を参照してください。

startOgServer.sh または startOgServer.cat コマンドを使用してセキュア eXtreme Scale コンテナー・サーバーを開始するときに、 –serverProps を使用してサーバー・プロパティー・ファイルを設定できます。サーバー・プロパティー・ファイルを設定する方法は他にもあります。詳しくは、サーバー・プロパティー・ファイルを参照してください。

startOgServer.sh または startOgServer.bat コマンドとそのオプションの使用方法について詳しくは、360 ページの『startOgServer スクリプト』を参照してください。

セキュア eXtreme Scale サーバーの停止

セキュア eXtreme Scale カタログ・サービス・プロセスまたはコンテナー・サーバーの停止には、1 つのセキュリティー構成ファイルが必要です。

- **クライアント・プロパティー・ファイル:** クライアント・プロパティー・ファイルを使用して、クライアント・セキュリティー・プロパティーを構成できます。クライアント・セキュリティー・プロパティーは、クライアントがセキュア・サーバーに接続するために必要です。詳しくは、216 ページの『クライアント・プロパティー・ファイル』を参照してください。

stopOgServer.sh または stopOgServer.cat コマンドを使用してセキュア eXtreme Scale カタログ・サービス・プロセスまたはコンテナー・サーバーを停止するときに、-clientSecurityFile を使用してクライアント・セキュリティー・プロパティーを設定できます。

stopOgServer.sh または stopOgServer.cat コマンドとそのオプションの使用方法について詳しくは、365 ページの『stopOgServer スクリプト』を参照してください。

WebSphere Application Server でのセキュア・サーバーの始動

WebSphere Application Server でのセキュア ObjectGrid サーバーの開始は、セキュリティー構成ファイルを渡す必要がある点を除いて、非セキュア ObjectGrid サーバーの開始と似ています。 Java SE 環境でコマンド中に -[PROPERTY_FILE] (例えば -serverProps) を使用する代わりに、汎用 Java 仮想マシン (JVM) 引数で -D[PROPERTY_FILE] を使用します。

Websphere Application Server でのセキュア・カタログ・サービスの開始

カタログ・サーバーには、以下の 2 つの異なるレベルのセキュリティー情報が含まれます。

- `-Dobjectgrid.cluster.security.xml.url`: これは、すべてのサーバー (カタログ・サーバーおよびコンテナー・サーバーを含む) に共通するセキュリティー・プロパティーを記述する objectGridSecurity.xml ファイルを指定します。ユーザー・レジストリーおよび認証メカニズムを表わすオーセンティケーター構成はその一例です。このプロパティーに指定するファイル名は、URL 形式 (例えば「file:///tmp/og/objectGridSecurity.xml」) でなければなりません。
- `-Dobjectgrid.server.props`: これは、サーバー固有のセキュリティー・プロパティーが入っているサーバー・プロパティー・ファイルを指定します。このプロパティーに指定するファイル名は、普通のファイル・パス形式 (例えば「c:/tmp/og/catalogserver.props」) です。 -Dobjectgrid.security.server.props の使用は推奨されませんが、後方互換性のために引き続き使用できることに注意してください。

WebSphere Application Server 内でセキュア・カタログ・サービスを開始するには、384 ページの『グリッド・セキュリティー』の『WebSphere Application Server への組み込み』の説明に従ってください。

その後、プロセスの汎用 JVM 引数でセキュリティー・プロパティーを設定します。

```
-Dobjectgrid.cluster.security.xml.url=file:///tmp/og/
objectGridSecurity.xml-Dobjectgrid.server.props=/tmp/og/
catalog.server.props
```

汎用 JVM 引数を追加する手順は以下のとおりです。

- 左側のタスク・ビューで「システム管理」を展開します。
- カタログ・サービスがデプロイされる WebSphere Application Server プロセス (例えば、「デプロイメント・マネージャー」) をクリックします。
- 右側のページで、「サーバー・インフラストラクチャー」の下の「Java およびプロセス管理」を展開します。
- 「プロセス定義」をクリックします。
- 「追加プロパティー」の下の「Java 仮想マシン」をクリックします。
- 「汎用 JVM 引数」テキスト・ボックスにプロパティーを入力します。

WebSphere Application Server でのセキュア・コンテナー・サーバーの開始

カタログ・サーバーに接続されたコンテナー・サーバーは、オーセンティケーター構成やログイン・セッション・タイムアウト設定など、objectGridSecurity.xml に構成されたすべてのセキュリティー構成を取得します。またコンテナー・サーバーは、-Dobjectgrid.server.props プロパティーにそのサーバー固有のセキュリティー・プロパティーを構成する必要があります。

このプロパティー・ファイルには、セキュリティーに関係しない他のプロパティーも入れるため、-Dobjectgrid.security.server.props プロパティーの代わりに -Dobjectgrid.server.props プロパティーを使用する必要があります。このプロパティーに対して指定されるファイル名の形式は、単なるプレーン・ファイル・パス形式です。例えば、`c:/tmp/og/server.props` などです。

上記と同じ手順に従って、セキュリティー・プロパティーを汎用 JVM 引数に追加してください。

セキュリティー記述子 XML ファイル

ObjectGrid セキュリティー記述子 XML ファイルを使用して、セキュリティーを使用可能にした eXtreme Scale デプロイメント・トポロジーを構成できます。以下のサンプル XML ファイルは、いくつかの構成を説明しています。

クラスター XML ファイルの各エレメントおよび属性の説明は、以下のリストに示されています。各サンプルでは、これらのエレメントおよび属性を使用して環境を構成する方法について説明しています。

securityConfig エレメント

securityConfig エレメントは、ObjectGrid セキュリティー XML ファイルの最上位エレメントです。このエレメントは、ファイルの名前空間とスキーマ・ロケーションをセットアップします。スキーマは `objectGridSecurity.xsd` ファイルで定義されます。

- 出現回数: 1 回
- 子エレメント: security

security エレメント

security エレメントは、ObjectGrid セキュリティーの定義に使用します。

- 出現回数: 1 回
- 子エレメント: authenticator、adminAuthorization、および systemCredentialGenerator

属性

securityEnabled
true に設定されているとき、グリッドのセキュリティーを使用可能にします。デフォルト値は false です。値を false に設定すると、グリッド全体のセキュリティーが使用不可になります。詳しくは、 384 ページの『グリッド・セキュリティー』を参照してください。 (オプション)

singleSignOnEnabled
値が true に設定されている場合は、クライアントがいずれか 1 つのサーバー

に認識された後、任意のサーバーに接続できます。そうでない場合は、クライアントは接続のたびに各サーバーに対して認証を行う必要があります。デフォルト値は false です。(オプション)

loginSessionExpirationTime
ログイン・セッションが期限切れになるまでの時間を秒数で指定します。ログイン・セッションの有効期限が切れると、クライアントは再度認証する必要があります。(オプション)

adminAuthorizationEnabled
管理許可を使用可能にします。この値が true に設定されている場合は、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 属性の値により指定されます。デフォルト値は false です。(オプション)

adminAuthorizationMechanism
使用する許可メカニズムを示します。WebSphere eXtreme Scale は、2 種類の許可メカニズム、すなわち、 Java 認証・承認サービス (JAAS) とカスタム許可をサポートします。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 実装を使用します。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、 388 ページの『アプリケーション・クライアントの許可』を参照してください。(オプション)

以下の security.xml ファイルは、 eXtreme Scale グリッド・セキュリティーを使用可能にするためのサンプル構成です。

security.xml

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

    <security securityEnabled="true" singleSignOnEnabled="true"
        loginSessionExpirationTime="20"
        adminAuthorizationEnabled="true"
        adminAuthorizationMechanism="AUTHORIZATION_MECHANISM_JAAS" >

        <authenticator className ="com.ibm.websphere.objectgrid.security.plugins.
      builtins.WSTokenAuthenticator">
        </authenticator>

        <systemCredentialGenerator className ="com.ibm.websphere.objectgrid.security.
      plugins.builtins.WSTokenCredentialGenerator">
            <property name="properties" type="java.lang.String" value="runAs"
        description="Using runAs subject" />
        </systemCredentialGenerator>

    </security>
</securityConfig>
```

authenticator エレメント

グリッド内の eXtreme Scale サーバーに対してクライアントを認証します。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実装している必要があります。オーセンティケーターはプロパティーを使用し、className

属性によって指定されるクラス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、property エレメントを参照してください。

前記の security.xml ファイルの例では、com.ibm.websphere.objectgrid.security.plugins.builtins.WSTokenAuthenticator クラスがオーセンティケーターとして指定されています。このクラスは com.ibm.websphere.objectgrid.security.plugins.Authenticator インターフェースを実装します。

- 出現回数: 0 回または 1 回
- 子エレメント: property

属性

className
com.ibm.websphere.objectgrid.security.plugins.Authenticator インターフェースを実装するクラスを指定します。このクラスを使用して、eXtreme Scale グリッド内のサーバーに対してクライアントを認証します。(必須)

adminAuthorization エレメント

adminAuthorization エレメントは、グリッドへの管理アクセスをセットアップする場合に使用します。

- 出現回数: 0 回または 1 回
- 子エレメント: property

属性

className
com.ibm.websphere.objectgrid.security.plugins.AdminAuthorization インターフェースを実装するクラスを指定します。 (必須)

systemCredentialGenerator エレメント

systemCredentialGenerator エレメントを使用すると、システム・クレデンシャル生成プログラムがセットアップされます。このエレメントは、動的環境にのみ適用されます。動的構成モデルでは、動的コンテナー・サーバーは、 eXtreme Scale クライアントとしてカタログ・サーバーに接続し、カタログ・サーバーもクライアントとして eXtreme Scale コンテナー・サーバーに接続できます。このシステム・クレデンシャル生成プログラムは、システム・クレデンシャルのファクトリーを表すために使用します。

- 出現回数: 0 回または 1 回
- 子エレメント: property

属性

className
com.ibm.websphere.objectgrid.security.plugins.CredentialGenerator インターフェースを実装するクラスを指定します。 (必須)

systemCredentialGenerator の使用例については、前の security.xml をファイル参照してください。この例では、システム・クレデンシャル生成プログラムは

com.ibm.websphere.objectgrid.security.plugins.builtins. WSTokenCredentialGenerator で、これはスレッドから RunAs Subject オブジェクトを取得します。

property エレメント

authenticator クラスおよび adminAuthorization クラスにおいて set メソッドを呼び出します。プロパティーの名前は、authenticator エレメントまたは adminAuthorization エレメントの className 属性の set メソッドに対応しています。

- 出現回数: 0 回以上
- 子エレメント: property

属性

name
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、このプロパティーを含む Bean の className 属性で指定されたクラスの 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、指定されているプロパティーの名前が size である場合、com.ibm.MyPlugin クラスには setSize メソッドが必要です。(必須)

type
プロパティーのタイプを指定します。パラメーターのタイプは、name 属性により識別される set メソッドに渡されます。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 であり、タイプが int である場合は、Bean の className 属性で指定されたクラスに setSize(int) メソッドが存在している必要があります。(必須)

value
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。 (必須)

description
プロパティーの説明を入力します。 (オプション)

詳しくは、『objectGridSecurity.xsd ファイル』を参照してください。

objectGridSecurity.xsd ファイル

次の ObjectGrid セキュリティー XML スキーマを使用して、eXtreme Scale デプロイメントに対するセキュリティーを使用可能にすることができます。

objectGridSecurity.xsd ファイルに定義されるエレメントおよび属性の説明は、402 ページの『セキュリティー記述子 XML ファイル』を参照してください。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:cc="http://ibm.com/ws/objectgrid/config/security"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://ibm.com/ws/objectgrid/config/security"
```

```
elementFormDefault="qualified">

<xsd:element name="securityConfig">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="security" type="cc:security" />
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>

<xsd:complexType name="security">
 <xsd:sequence>
  <xsd:element name="authenticator" type="cc:bean" minOccurs="0"
   maxOccurs="1" />
  <xsd:element name="adminAuthorization" type="cc:bean" minOccurs="0"
   maxOccurs="1" />
  <xsd:element name="systemCredentialGenerator" type="cc:bean" minOccurs="0"
   maxOccurs="1" />
 </xsd:sequence>
 <xsd:attribute name="securityEnabled" type="xsd:boolean" use="optional" />
 <xsd:attribute name="singleSignOnEnabled" type="xsd:boolean" use="optional"/>
 <xsd:attribute name="loginSessionExpirationTime" type="xsd:int" use="optional"/>
 <xsd:attribute name="adminAuthorizationMechanism" type="cc:adminAuthorizationMechanism"
  use="optional"/>
 <xsd:attribute name="adminAuthorizationEnabled" type="xsd:boolean" use="optional" />
</xsd:complexType>

<xsd:complexType name="bean">
 <xsd:sequence>
  <xsd:element name="property" type="cc:property" maxOccurs="unbounded" minOccurs="0" />
 </xsd:sequence>
 <xsd:attribute name="className" type="xsd:string" use="required" />
</xsd:complexType>

<xsd:complexType name="property">
 <xsd:attribute name="name" type="xsd:string" use="required" />
 <xsd:attribute name="value" type="xsd:string" use="required" />
 <xsd:attribute name="type" type="cc:propertyType" use="required" />
 <xsd:attribute name="description" type="xsd:string" use="optional" />
</xsd:complexType>

<xsd:simpleType name="propertyType">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="java.lang.Boolean" />
  <xsd:enumeration value="boolean" />
  <xsd:enumeration value="java.lang.String" />
  <xsd:enumeration value="java.lang.Integer" />
  <xsd:enumeration value="int" />
  <xsd:enumeration value="java.lang.Double" />
  <xsd:enumeration value="double" />
  <xsd:enumeration value="java.lang.Byte" />
  <xsd:enumeration value="byte" />
  <xsd:enumeration value="java.lang.Short" />
  <xsd:enumeration value="short" />
  <xsd:enumeration value="java.lang.Long" />
  <xsd:enumeration value="long" />
  <xsd:enumeration value="java.lang.Float" />
  <xsd:enumeration value="float" />
  <xsd:enumeration value="java.lang.Character" />
  <xsd:enumeration value="char" />
 </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="adminAuthorizationMechanism">
 <xsd:restriction base="xsd:string">
  <xsd:enumeration value="AUTHORIZATION_MECHANISM_JAAS" />
  <xsd:enumeration value="AUTHORIZATION_MECHANISM_CUSTOM" />
 </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

第 9 章 デプロイメント環境のモニター

API、MBean、ログ、およびユーティリティーを使用して、アプリケーション環境のパフォーマンスをモニターできます。

統計の概説

WebSphere eXtreme Scale での統計は、内部統計ツリーに作成されます。 内部ツリーからは、StatsAccessor API、Performance Monitoring Infrastructure (PMI) モジュール、および MBean API が作成されます。

次の図は、eXtreme Scale の統計の一般的なセットアップを示しています。

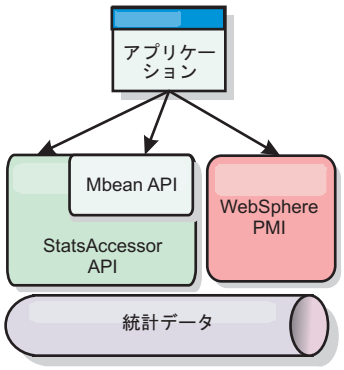


図 26. 統計の概説

これらの API のそれぞれは統計ツリーに対するビューを提供しますが、使用される理由は異なります。

- **統計 API**: 統計 API により、開発者は統計に直接アクセスできます。これにより、カスタム MBean やロギングのような柔軟でカスタマイズ可能な統計統合ソリューションが可能になります。
- **MBean API**: MBean API は、モニター用の仕様ベースのメカニズムです。MBean API は、Statistics API を使用し、サーバー Java 仮想マシン (JVM) に対してローカルに実行します。API および MBean の構造は、他のベンダーのユーティリティーと容易に統合できるように設計されています。分散オブジェクト・グリッドを実行中の場合は、 MBean API を使用します。
- **WebSphere Application Server Performance Monitoring Infrastructure (PMI) モジュール** : PMI は、WebSphere Application Server 内で WebSphere eXtreme Scale を実行中の場合に使用します。これらのモジュールは、内部統計ツリーのビューを提供します。

統計 API

ツリー・マップに非常によく似ており、特定のモジュールを取得するための対応するパスおよびキー、すなわちこの場合は細分度または集約レベルがあります。例えば、ツリー内に常に任意のルート・ノードがあって、「accounting」という名前の

ObjectGrid に属している「payroll」という名前のマップに関して統計が収集されると想定します。例えば、マップの集約レベルまたは細分度についてモジュールにアクセスするには、パスの String[] を渡すことができます。この場合、各 String がノードのパスを表わすので、これは String[] {root, "accounting", "payroll"} と同等です。この構造の利点は、ユーザーがパス内のどのノードにも配列を指定でき、そのノードの集約レベルを取得できるという点です。このため、String[] {root, "accounting"} を渡すと、マップ統計が取得されますが、これは「accounting」というグリッド全体に対するものです。これにより、ユーザーは、モニターする統計のタイプを、アプリケーションに必要などのような集約レベルでも指定できます。

WebSphere Application Server PMI モジュール

WebSphere eXtreme Scale には、WebSphere Application Server PMI で使用するための統計モジュールが組み込まれています。WebSphere Application Server プロファイルが WebSphere eXtreme Scale で拡張されると、拡張スクリプトにより WebSphere eXtreme Scale モジュールが自動的に WebSphere Application Server 構成ファイルに統合されます。 PMI を使用すると、統計モジュールを使用可能および使用不可にしたり、さまざまな細分度で統計を自動的に集約したり、また組み込み Tivoli Performance Viewer を使用してデータをグラフ化することさえできます。詳しくは、416 ページの『WebSphere Application Server PMI によるモニター』を参照してください。

ベンダー製品と Managed Bean (MBean) との統合

eXtreme Scale API および Managed Bean は、サード・パーティーのモニタリング・アプリケーションと簡単に統合できるように設計されています。eXtreme Scale トポロジーに関する情報を分析するために使用できる単純な Java Management Extensions (JMX) コンソールの例のいくつかとして、JConsole や MC4J があります。またプログラマチック API を使用して、eXtreme Scale パフォーマンスのスナップショットを作成するか、そのパフォーマンスを追跡するアダプター実装を作成することもできます。WebSphere eXtreme Scale には、すぐに使用可能なモニター機能を持つサンプルのモニター・アプリケーションが含まれており、拡張したカスタム・モニター・ユーティリティーを作成するためのテンプレートとしてこれを使用できます。

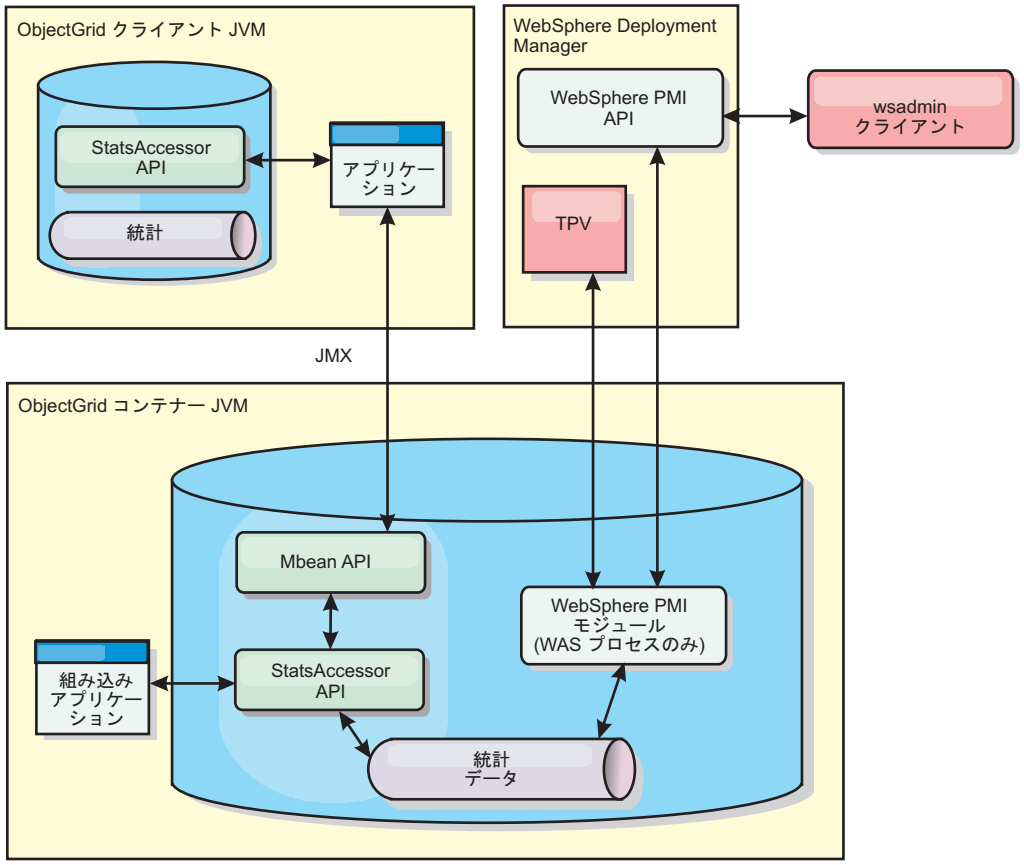


図 27. MBean の概説

詳しくは、413 ページの『xsAdmin サンプル・ユーティリティーによるモニター』を参照してください。特定のベンダー・アプリケーションとの統合について詳しくは、以下のトピックを参照してください。

- IBM Tivoli モニター・エージェントでの eXtreme Scale のモニター
- 449 ページの『Hyperic HQ による eXtreme Scale のモニター』
- 445 ページの『CA Wily Introscope による eXtreme Scale アプリケーションのモニター』

統計 API によるモニター

統計 API は、内部統計ツリーに直接接続するインターフェースです。統計はデフォルトでは使用不可になっていますが、StatsSpec インターフェースを設定することで使用可能にすることができます。 StatsSpec インターフェースは、WebSphere eXtreme Scale がどのように統計をモニターするかを定義します。

このタスクについて

ローカルの StatsAccessor API を使用して、実行中のコードと同じ Java 仮想マシン (JVM) にある ObjectGrid インスタンス上のデータおよびアクセス統計を照会することができます。個々のインターフェースについて詳しくは、API 資料を参照してください。次の手順で、内部統計ツリーのモニターを使用可能にします。

手順

1. StatsAccessor オブジェクトを検索します。 StatsAccessor インターフェースは singleton パターンに従います。したがって、クラス・ローダーに関連する問題を別にすれば、JVM ごとに 1 つの StatsAccessor インスタンスが存在するはずです。このクラスはすべてのローカル統計操作のメイン・インターフェースとして機能します。以下のコードは、accessor クラスの検索方法の例です。この操作は、他のすべての ObjectGrid 呼び出しより前に呼び出します。

```
public class LocalClient
{

   public static void main(String[] args) {

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

   }

}
```

2. グリッド StatsSpec インターフェースを設定します。 すべての統計を ObjectGrid レベルでのみ収集するように、この JVM を設定します。トランザクションを開始する前に、必要と思われるすべての統計をアプリケーションが使用可能にするようにする必要があります。次の例は、static 定数フィールドと spec ストリングの両方を使用して StatsSpec インターフェースを設定するものです。static 定数フィールドは既に仕様が定義されているため、このフィールドを使用する方が簡単です。ただし、spec ストリングを使用すれば、必要な統計のどんな組み合わせでも使用可能にすることができます。

```
public static void main(String[] args) {

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      // Set the spec via the spec String
      StatsSpec spec = new StatsSpec("og.all=enabled");
      accessor.setStatsSpec(spec);

}
```

3. トランザクションをグリッドに送信して、モニター用のデータが収集されるようにします。 統計用に有効なデータを収集するには、トランザクションをグリッドに送る必要があります。次のコード抜粋は、ObjectGridA 内の MapA にレコードを挿入するものです。統計は、ObjectGrid レベルであるため、ObjectGrid 内のマップはいずれも同じ結果を示します。

```
public static void main(String[] args) {

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
    ObjectGridmanagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
```

```
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Drive insert
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();
}
```

4. StatsAccessor API を使用して StatsFact を照会します。 すべての統計パスは StatsFact インターフェースに関連付けられます。 StatsFact インターフェースは、StatsModule オブジェクトを編成して組み込むために使用される汎用プレースホルダーです。実際の統計モジュールにアクセスするためには、前もって StatsFact オブジェクトを検索する必要があります。

```
public static void main(String[] args)
{

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
    ObjectGridManagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
      Session session = grid.getSession();
      Map map = session.getMap("MapA");

      // Drive insert
      session.begin();
      map.insert("SomeKey", "SomeValue");
      session.commit();

      // Retrieve StatsFact

      StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
    StatsModule.MODULE_TYPE_OBJECT_GRID);

}
```

5. StatsModule オブジェクトと対話します。 StatsModule オブジェクトは StatsFact インターフェース内に含まれています。 StatsFact インターフェースを使用してモジュールへの参照を取得できます。 StatsFact インターフェースは汎用インターフェースであるため、戻されたモジュールを予期された StatsModule タイプにキャストする必要があります。このタスクは eXtreme Scale の統計を収集するため、戻された StatsModule オブジェクトは OGStatsModule タイプにキャストされます。モジュールがキャストされたならば、使用可能なすべての統計にアクセスすることができます。

```
public static void main(String[] args) {

      // retrieve a handle to the StatsAccessor
      StatsAccessor accessor = StatsAccessorFactory.getStatsAccessor();

      // Set the spec via the static field
      StatsSpec spec = new StatsSpec(StatsSpec.OG_ALL);
      accessor.setStatsSpec(spec);

      ObjectGridManager manager =
    ObjectGridmanagerFactory.getObjectGridManager();
      ObjectGrid grid = manager.getObjectGrid("ObjectGridA");
```

```
        Session session = grid.getSession();
        Map map = session.getMap("MapA");

        // Drive insert
        session.begin();
        map.insert("SomeKey", "SomeValue");
        session.commit();

        // Retrieve StatsFact
        StatsFact fact = accessor.getStatsFact(new String[] {"EmployeeGrid"},
      StatsModule.MODULE_TYPE_OBJECT_GRID);

        // Retrieve module and time
        OGStatsModule module = (OGStatsModule)fact.getStatsModule();
        ActiveTimeStatistic timeStat =
      module.getTransactionTime("Default", true);
        double time = timeStat.getMeanTime();

  }
```

統計モジュール

WebSphere eXtreme Scale は、内部統計モデルを使用して、データの追跡およびフィルター処理を行います。このモデルは、すべてのデータ・ビューで統計のスナップショットを収集するために使用される基礎となる構造です。統計モジュールから情報を取得するには、いくつかの方法を使用できます。

概説

WebSphere eXtreme Scale での統計は、StatsModules コンポーネント内で追跡され、収容されます。統計モデルには、以下のいくつかのタイプの統計モジュールが存在します。

OGStatsModule
トランザクション応答時間など、ObjectGrid インスタンスの統計を提供します。

MapStatsModule
エントリー数やヒット率など、単一マップの統計を提供します。

QueryStatsModule
計画作成や実行時間など、照会の統計を提供します。

AgentStatsModule
シリアライズ時間や実行時間など、DataGrid API エージェントの統計を提供します。

HashIndexStatsModule
HashIndex 照会および保守の実行時間の統計を提供します。

SessionStatsModule
HTTP セッション・マネージャー・プラグインの統計を提供します。

統計モジュールについて詳しくは、API 資料中の com.ibm.websphere.objectgrid.stats パッケージを参照してください。

ローカル環境での統計

モデルは、前のリストで説明したすべての StatsModule タイプから構成される n 進ツリー (すべてのノードについて同じ次数を持つツリー構造) に似た編成になっています。この編成構造のため、ツリー内のすべてのノードは、StatsFact インターフェースで表現されます。 StatsFact インターフェースは、集約の目的で個別のモジュールまたはモジュールのグループを表わすことができます。例えば、ツリー内のいくつかのリーフ・ノードが特定の MapStatsModule オブジェクトを表わす場合、これらのノードの親 StatsFact ノードには、すべての子モジュールついて集約された統計が含まれます。StatsFact オブジェクトのフェッチ後には、インターフェースを使用して対応する StatsModule を取得することができます。

ツリー・マップに非常によく似ており、対応するパスまたはキーを使用して特定の StatsFact を取得することができます。パスは、要求されたファクトへのパスに沿ったすべてのノードから構成される String[] 値です。例えば、MapA と MapB という 2 つのマップを含む ObjectGridA という名前の ObjectGrid を作成したとします。MapA の StatsModule へのパスは、`[ObjectGridA, MapA]` のようになります。両方のマップの集約統計へのパスは、`[ObjectGridA]` となります。

分散環境での統計

分散環境では、統計モジュールは異なるパスを使用して取得されます。サーバーには複数の区画を入れることができるため、統計ツリーは、各モジュールが属する区画を追跡する必要があります。結果として、特定の StatsFact オブジェクトをルックアップするためのパスは異なります。前の例を使用しますが、マップが区画 1 に存在するという点を付け加えると、MapA の StatsFact オブジェクトを取得するためのパスは `[1,ObjectGridA, MapA]` となります。

xsAdmin サンプル・ユーティリティーによるモニター

xsAdmin サンプル・ユーティリティーを使用すれば、WebSphere eXtreme Scale トポロジーに関するテキスト情報をフォーマットして表示することができます。このサンプル・ユーティリティーは現在のデプロイメント・データの解析とディスカバリーの方法を提供するもので、カスタム・ユーティリティーの作成基盤として使用することができます。

始める前に

WebSphere eXtreme Scale がインストールされていなければなりません。

このタスクについて

xsAdmin サンプル・ユーティリティーを使用することで、グリッドの現在のレイアウトおよび特定の状態に関するフィードバック (マップの内容など) を提供することができます。この例では、このタスクにおけるグリッドのレイアウトは *ObjectGridA* という単一のグリッドで構成されています。このグリッドは、*MapA* という定義済みのマップを 1 つ持ち、*MapSetA* というマップ・セットに属しています。この例は、グリッド内のすべてのアクティブ・コンテナーを表示し、*MapA* のマップ・サイズに関するフィルタリング済みメトリックを印刷する方法を示しています。使用できるコマンド・オプションをすべて知りたい場合は、引数なしか、または **-help**

オプションを付けて xsAdmin ユーティリティーを実行してください。

手順

1. コマンド行で、JAVA_HOME 環境変数を設定します。
   - UNIX `export JAVA_HOME=javaHome`
   - Windows `set JAVA_HOME=javaHome`
2. bin ディレクトリーに移動します。

   ```
   cd objectGridRoot/bin
   ```
3. xsAdmin ユーティリティーを起動します。
   - **オンライン・ヘルプを表示するには、次のコマンドを実行します。**

     UNIX

     ```
     xsadmin.sh
     ```

     Windows

     ```
     xsadmin.bat
     ```

     ヘルプ・メッセージの必須引数セクションに注意してください。このユーティリティーが機能するためには、リストされているオプションの中の 1 つだけを渡すようにする必要があるからです。 **-g**、**-m** いずれのオプションも指定されていない場合は、xsAdmin ユーティリティーはトポロジー内のすべてのグリッドについて情報を印刷します。
   - **すべてのサーバーの統計を使用可能にするには、以下のコマンドを実行します。**

     UNIX

     ```
     xsadmin.sh -g ObjectGridA -setstatsspec ALL=enabled
     ```

     Windows

     ```
     xsadmin.bat -g ObjectGridA -setstatsspec ALL=enabled
     ```
   - **ある特定のグリッドについてすべてのオンライン・コンテナーを表示するには、次のコマンドを実行します。**

     UNIX

     ```
     xsadmin.sh -g ObjectGridA -m MapSetA -containers
     ```

     Windows

     ```
     xsadmin.bat -g ObjectGridA -m MapSetA -containers
     ```

     すべてのコンテナー情報が表示されます。出力の例を次に示します。

     ```
     This administrative utility is provided as a sample only and is not to be
     considered a fully supported component of the WebSphere eXtreme Scale product

     Connecting to Catalog service at localhost:1099


     *** Show all online containers for grid - ObjectGridA & mapset - MapSetA
     ```

```
Host: 192.168.0.186
Container: server1_C-0, Server:server1, Zone:DefaultZone
Partition Shard Type
        0 Primary

Num containers matching = 1
Total known containers = 1
Total known hosts = 1
```

- **カタログ・サービスに接続して MapA に関する情報を表示するには、以下のコマンドを実行します。**

  UNIX

  ```
  xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA
  ```

  Windows

  ```
  xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA
  ```

  指定したマップのサイズが表示されます。出力の例を次に示します。

  ```
  This administrative utility is provided as a sample only and is not to be
  considered a fully supported component of the WebSphere eXtreme Scale product

  Connecting to Catalog service at localhost:1099

  ****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****


  *** Listing Maps for server1 ***
   Map Name Partition Map Size Used Bytes (B) Shard Type
   MapA     0         0        0              Primary
  ```

- **特定の JMX ポートを使用してカタログ・サービスに接続して MapA に関する情報を表示するには、以下のコマンドを実行します。** UNIX

  ```
  xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA
    -ch CatalogMachine -p 6645
  ```

  Windows

  ```
  xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA
    -ch CatalogMachine -p 6645
  ```

  xsAdmin サンプル・ユーティリティーは、カタログ・サーバーを実行している MBean サーバーに接続します。カタログ・サーバーは、スタンドアロン・プロセスまたは WebSphere Application Server プロセスで実行できますが、カスタム・アプリケーション・プロセス内に組み込むこともできます。カタログ・サービス・ホスト名を指定するには **-ch** オプションを、カタログ・サービス・ネーミング・ポートを指定するには **-p** オプションを、それぞれ使用します。

  指定したマップのサイズが表示されます。出力の例を次に示します。

  ```
  This administrative utility is provided as a sample only and is not to be
  considered a fully supported component of the WebSphere eXtreme Scale product

  Connecting to Catalog service at CatalogMachine:6645

  *****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****
  ```

```
*** Listing Maps for server1 ***
Map Name: MapA  Partition #: 0  Map Size: 0  Shard Type: Primary
Server Total: 0
```

- **WebSphere Application Server プロセスでホストされるカタログ・サービスに接続するには、以下のステップを実行します。**

  WebSphere Application Server プロセスまたはプロセスのクラスターがホストするカタログ・サービスに接続する際には、**-dmgr** オプションが必要です。 localhost ではない場合ホスト名を指定するには **-ch** オプションを、カタログ・サービス・ブートストラップ・ポートをオーバーライドするには **-p** オプションを、それぞれ使用します。後者の場合は、BOOTSTRAP_ADDRESS プロセスが使用されます。 **-p** オプションは、BOOTSTRAP_ADDRESS がデフォルトの 9809 に設定されていない場合にのみ必要となります。

  注: WebSphere Application Server プロセスがホストするカタログ・サービスに接続する場合は、WebSphere eXtreme Scale のスタンドアロン・バージョンは使用できません。 *was_root*/bin ディレクトリーに含まれている xsAdmin スクリプトを使用してください。このスクリプトは、WebSphere eXtreme Scale を WebSphere Application Server または WebSphere Application Server Network Deployment にインストールすると使用可能になります。

  a. WebSphere Application Server bin ディレクトリーに移動します。

     ```
     cd wasRoot/bin
     ```

  b. 次のコマンドを使用して xsAdmin ユーティリティーを起動します。

     UNIX

     ```
     xsadmin.sh -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
     ```

     Windows

     ```
     xsadmin.bat -g ObjectGridA -m MapSetA -mapsizes -fm MapA -dmgr
     ```

  指定したマップのサイズが表示されます。

```
This administrative utility is provided as a sample only and is not to be
considered a fully supported component of the WebSphere eXtreme Scale product

Connecting to Catalog service at localhost:9809

****Displaying Results for Grid - ObjectGridA, MapSet - MapSetA*****


*** Listing Maps for server1 ***
Map Name: MapA  Partition #: 0  Map Size: 0  Shard Type: Primary
Server Total: 0
```

WebSphere Application Server PMI によるモニター

WebSphere eXtreme Scale は、WebSphere Application Server または WebSphere Extended Deployment アプリケーション・サーバーで実行されているとき、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。管理コンソールまたは wsadmin ツールを使用して、モニター・データにアクセスすることができます。

始める前に

WebSphere eXtreme Scale を WebSphere Application Server と組み合わせて使用しているとき、PMI を使用してご使用の環境をモニターすることができます。

このタスクについて

WebSphere eXtreme Scale は、WebSphere Application Server のカスタム PMI 機能を使用して、独自の PMI 装備を追加します。この方法で、管理コンソールまたは wsadmin ツールの Java Management Extensions (JMX) インターフェースを使用して、WebSphere eXtreme Scale 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 WebSphere eXtreme Scale 統計にアクセスすることができます。

手順

1. eXtreme Scale 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、『PMI の使用可能化』を参照してください。
2. eXtreme Scale PMI 統計を取得します。 Tivoli Performance Viewer を使用して、eXtreme Scale 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、420 ページの『PMI 統計の取得』を参照してください。

次のタスク

wsadmin ツールについて詳しくは、381 ページの『wsadmin ツールを使用した MBean へのアクセス』を参照してください。

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。管理コンソール内またはスクリプトを使用して PMI を使用可能にすることができます。

始める前に

アプリケーション・サーバーを始動し、eXtreme Scale 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、スクリプトを使用して PMI を使用可能にするには、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 ツールについて詳しくは、WebSphere Application Server インフォメーション・センターの wsadmin ツールのトピックを参照してください。

このタスクについて

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。

手順

- **管理コンソールで PMI を使用可能にします。**
  1. 管理コンソールで、「**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。
  2. 「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は、チェック・ボックスを選択して、サーバーを再始動します。
  3. 「**カスタム**」をクリックします。構成ツリーで、ObjectGrid および ObjectGrid マップ・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・カテゴリーが実行時に作成されます。「**Runtime**」タブには、ObjectGrid と Map 統計のサブカテゴリーのみを表示できます。
- **スクリプトを使用して PMI を使用可能にします。**
  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 と入力して、wsadmin コマンド行ツールを開始します。
  2. eXtreme Scale 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、サーバーに対して PMI が使用可能になっていることを確認します。

     ```
     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. 以下のコマンドを使用して、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。

     ```
     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. 以下のコマンドを使用して、統計セットをカスタムに設定します。

     ```
     wsadmin>$AdminControl invoke_jmx $perfOName setStatisticSet $params $sigs
     ```

5. 以下のコマンドを使用して、objectGridModule PMI 統計を使用可能にするための変数を設定します。

```
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. 以下のコマンドを使用して、統計ストリングを設定します。

```
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
```

7. 以下のコマンドを使用して、統計ストリングを設定します。

```
wsadmin>$AdminControl invoke_jmx $perfOName setCustomSetString $params2 $sigs2
```

これらのステップにより、eXtreme Scale 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と、メイン PMI の使用可能化を除いて、PMI 設定は失われます。

例

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 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。この時点では、このサーブレット・ページを閉じないでください。
3. 管理コンソールで、「**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 マップ」グループの下に、clusterObjectGrid という名前の 1 つの ObjectGrid インスタンスが存在し、この clusterObjectGrid グループの下に、counters、employees、offices、および sites という 4 つのマップが存在します。ObjectGrids インスタンスには clusterObjectGrid インスタンスが存在し、そのインスタンスの下には、DEFAULT という名前のトランザクション・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。例えば、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。

次のタスク

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。

PMI 統計の取得

PMI 統計を取得することによって、eXtreme Scale アプリケーションのパフォーマンスを確認できます。

始める前に

- ご使用の環境の PMI 統計追跡を使用可能にします。詳しくは、417 ページの『PMI の使用可能化』を参照してください。
- このタスクにあるパスはサンプル・アプリケーションの統計を取得することを前提としたものですが、これらの統計は類似のステップを含む他のアプリケーションに対しても使用できます。
- 管理コンソールを使用して統計を取得する場合は、管理コンソールにログインできなければなりません。スクリプトを使用する場合は、wsadmin にログインできなければなりません。

このタスクについて

管理コンソールまたはスクリプトでステップを完了すれば、PMI 統計を取得して Tivoli Performance Viewer で表示することができます。

- 管理コンソールのステップ
- スクリプトのステップ

取得できる統計についての詳細は、421 ページの『PMI モジュール』を参照してください。

手順

- 管理コンソールで PMI 統計を取得します。
  1. 管理コンソールで、「**モニターおよびチューニング**」 → 「**パフォーマンス・ビューアー**」 → 「**現行アクティビティー**」をクリックします。
  2. Tivoli Performance Viewer を使用してモニターするサーバーを選択してから、モニターを使用可能にします。
  3. サーバーをクリックして、「Performance viewer」ページを表示します。
  4. 構成ツリーを展開します。「**ObjectGrid マップ**」 → 「**clusterObjectGrid**」をクリックし、「**従業員**」を選択します。「**ObjectGrids**」 → 「**clusterObjectGrid**」を展開し、「**DEFAULT**」を選択します。
  5. ObjectGrid サンプル・アプリケーションで、ObjectGridCreationServlet サーブレットに移動し、ボタン 1 をクリックしてから、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。
- スクリプトを使用して PMI 統計を取得します。
  1. コマンド行プロンプトで、`install_root/bin` ディレクトリーに移動します。`wsadmin` と入力して wsadmin ツールを開始します。
  2. 以下のコマンドを使用して、環境の変数を設定します。

```
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. 以下のコマンドを使用して、mapModule 統計を取得するための変数を設定します。

```
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. 以下のコマンドを使用して、mapModule 統計を取得します。

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params $sigs
```

5. 以下のコマンドを使用して、objectGridModule 統計を取得するための変数を設定します。

```
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. 以下のコマンドを使用して、objectGridModule 統計を取得します。

```
wsadmin>$AdminControl invoke_jmx $perfOName getStatsString $params2 $sigs2
```

タスクの結果

Tivoli Performance Viewer で統計を表示することができます。

PMI モジュール

Performance Monitoring Infrastructure (PMI) モジュールを使用してアプリケーションのパフォーマンスをモニターすることができます。

objectGridModule

objectGridModule は時間統計 (トランザクション応答時間) を含みます。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 のルート・エレメント (root) は、WebSphere eXtreme Scale 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 ObjectGrid を子エレメントとして持ち、さらにこの子エレメントはトランザクション・タイプを子エレメントとして持ちます。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。

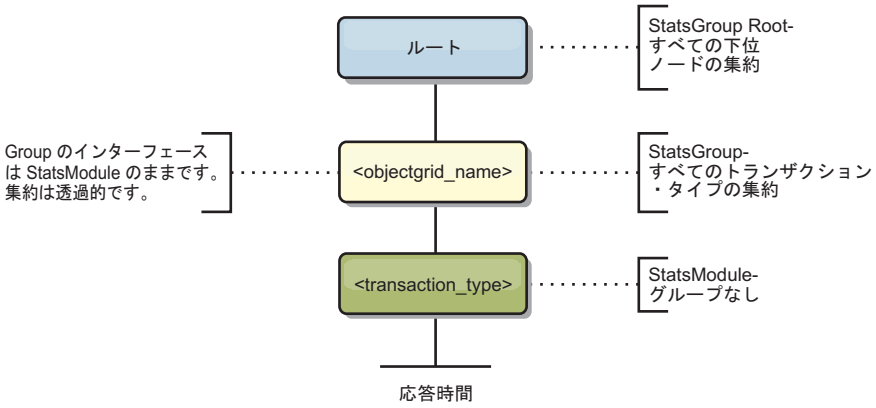


図 28. ObjectGridModule モジュールの構造

次の図は ObjectGridModule 構造の例です。この例では、2 つの ObjectGrid インスタンス (ObjectGrid A と ObjectGrid B) がシステムに存在します。ObjectGrid A インスタンスには 2 つのトランザクション・タイプ (A とデフォルト) があります。ObjectGrid B インスタンスにはデフォルトのトランザクション・タイプのみがあります。

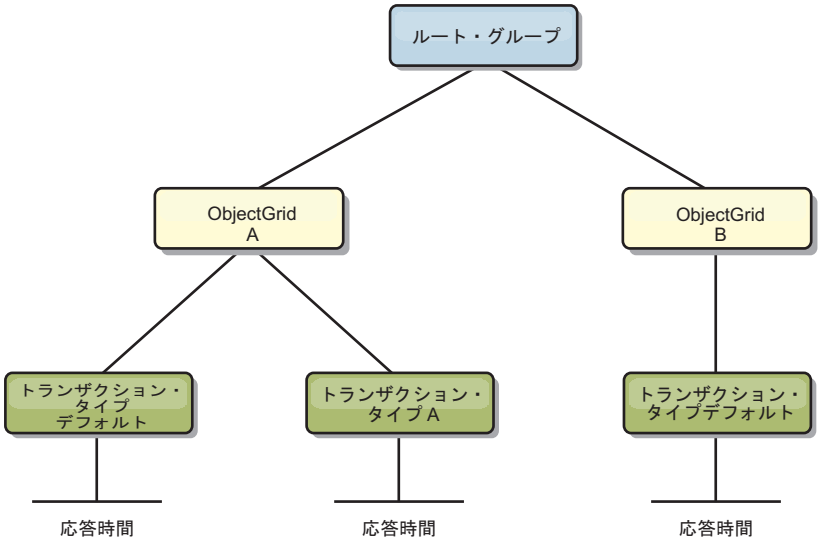


図 29. ObjectGridModule モジュール構造の例

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・タイプの設定は、次の Session.setTransactionType(String) メソッドを使用して行われます。

```
/**
* 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 インスタンスで、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

mapModule は、eXtreme Scale 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** - *BoundedRangeStatistic*: 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** -*CountStatistic*: 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** -*TimeStatistic*: 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 のルート・エレメント (root) は、ObjectGrid マップ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 ObjectGrid を子エレメントとして持ち、さらにこの子エレメントはマップを子エレメントとして持ちます。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。

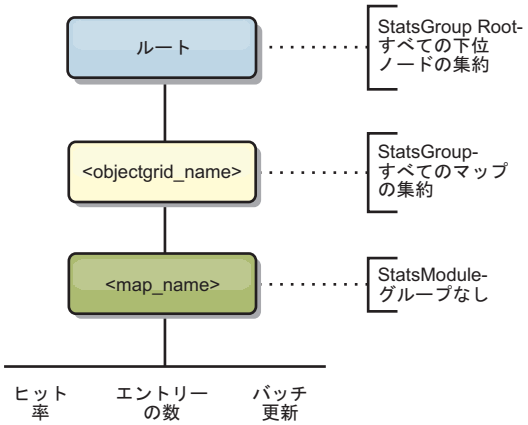
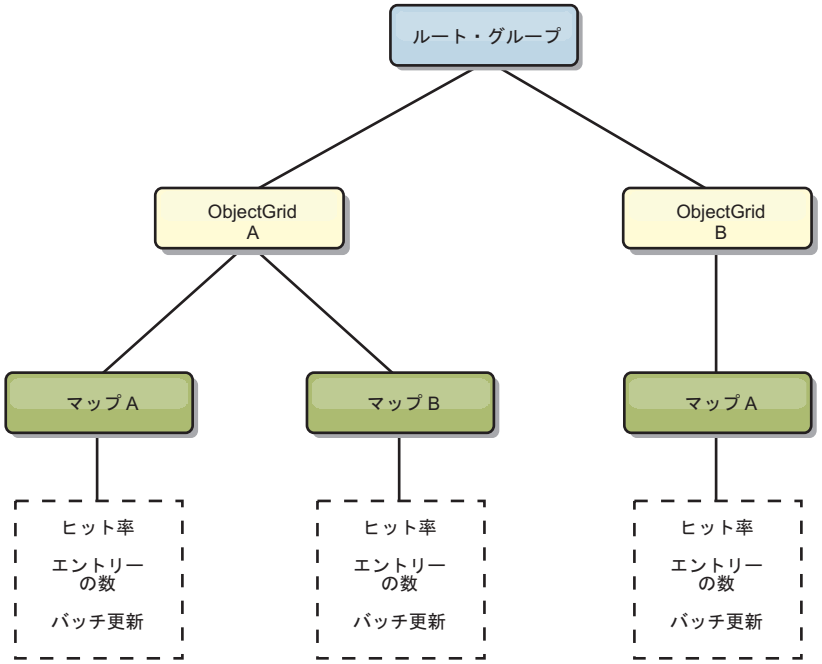


図 30. mapModule 構造

次の図は mapModule 構造の例です。

図 31. mapModule モジュール構造の例



hashIndexModule

hashIndexModule は、マップ・レベルの索引に関連する次の統計を含みます。

- **検索カウント** *-CountStatistic*: 索引検索操作の呼び出し回数。
- **衝突カウント** *-CountStatistic*: 検索操作の衝突回数。
- **障害カウント** *-CountStatistic*: 検索操作の障害件数。
- **結果カウント** *-CountStatistic*: 検索操作から戻されたキーの数。

- **バッチ更新カウント** *-CountStatistic*: この索引に対するバッチ更新の回数。対応するマップが何らかの方法で変更されると、索引で、その doBatchUpdate() メソッドが呼び出されます。この統計からは、索引の変更または更新頻度が分かります。
- **検索操作所要時間** *-TimeStatistic*: 検索操作が完了するまでに要する時間。

hashIndexModule のルート・エレメント (root) は、HashIndex 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 ObjectGrid を子エレメントとして持ちます。ObjectGrid はマップを子エレメントとして持ち、最終的にこの子エレメントは HashIndexe を子エレメントおよびツリーのリーフ・ノードとして持ちます。すべての HashIndex 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 hashIndexModule の構造です。

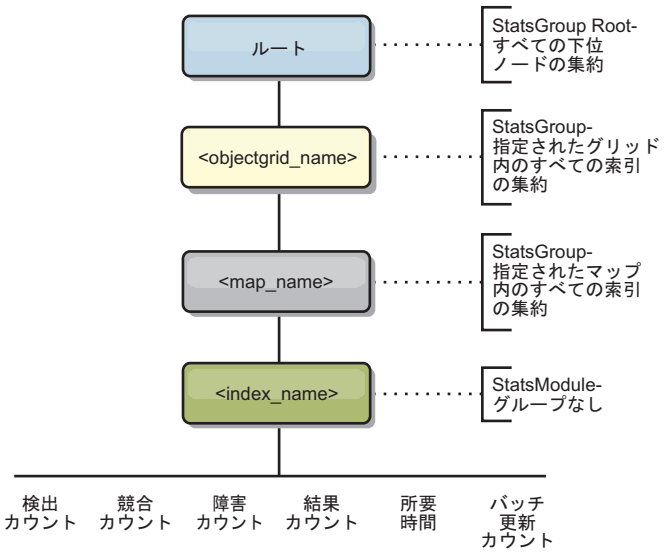


図 32. hashIndexModule モジュール構造

次の図は hashIndexModule 構造の例です。

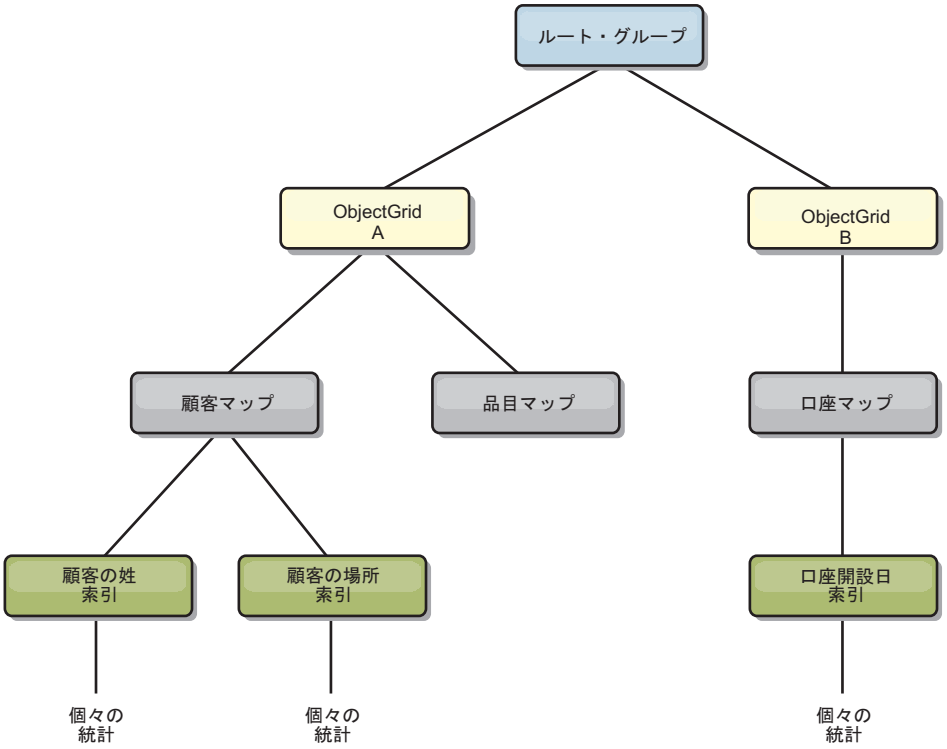


図 33. hashIndexModule モジュール構造の例

agentManagerModule

agentManagerModule は、マップ・レベルのエージェントに関連した統計を含みます。

- **削減時間**: *TimeStatistic* - エージェントが削減操作を完了するまでの時間。
- **合計所要時間**: *TimeStatistic* - エージェントがすべての操作を完了するまでの合計時間。
- **エージェント・シリアライゼーション時間**: *TimeStatistic* - エージェントをシリアライズするための時間。
- **エージェント・インフレーション時間**: *TimeStatistic* - サーバー上でエージェントをインフレーションするのに要する時間。
- **結果シリアライゼーション時間**: *TimeStatistic* - エージェントからの結果をシリアライズするための時間。
- **結果インフレーション時間**: *TimeStatistic* - エージェントからの結果をインフレーションするための時間。
- **障害カウント**: *CountStatistic* - エージェントが障害を起こした回数。
- **呼び出しカウント**: *CountStatistic* - AgentManager が呼び出された回数。
- **区画カウント**: *CountStatistic* - エージェントが送られる相手区画の数。

agentManagerModule のルート・エレメント (root) は、AgentManager 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 ObjectGrid を子エレメントとして持ちます。ObjectGrid はマップを子エレメントとして持ち、最終的にこの子エレメントは AgentManager インスタンスを子エレメントおよびツリーのリーフ・ノードとして持ちます。すべての AgentManager インスタンスは 3 つのリスト統計を持

っています。

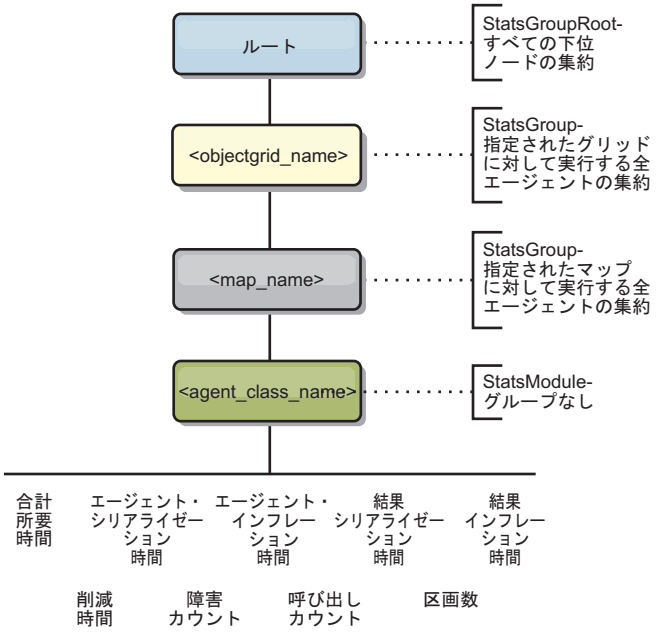


図 34. agentManagerModule 構造

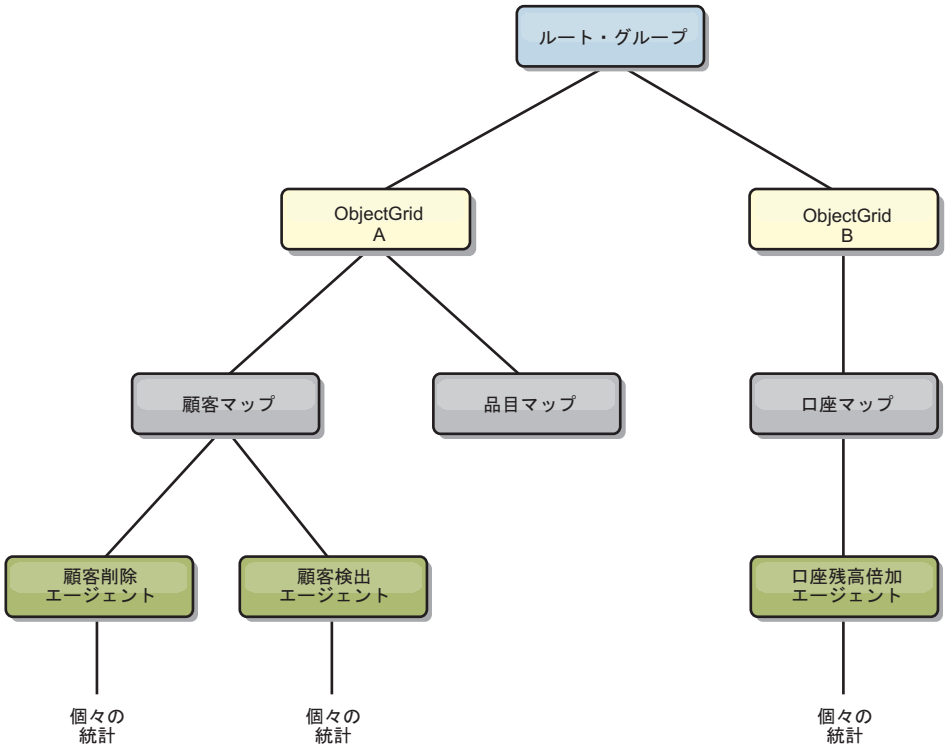


図 35. agentManagerModule 構造の例

queryModule

queryModule は、eXtreme Scale 照会に関連した統計を含みます。

- **計画作成時間**: *TimeStatistic* - 照会計画を作成するための時間。
- **実行時間**: *TimeStatistic* - 照会を実行するための時間。
- **実行カウント**: *CountStatistic* - 照会が実行された回数。
- **結果カウント**: *CountStatistic* - 実行された各照会の結果セットごとのカウント。
- **障害カウント**: *CountStatistic* - 照会が失敗した回数。

queryModule のルート・エレメント (root) は、Query 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 ObjectGrid を子エレメントとして持ち、さらにこの子エレメントは照会オブジェクトを子エレメントおよびツリーのリーフ・ノードとして持ちます。すべての Query インスタンスは 3 つのリスト統計を持っています。

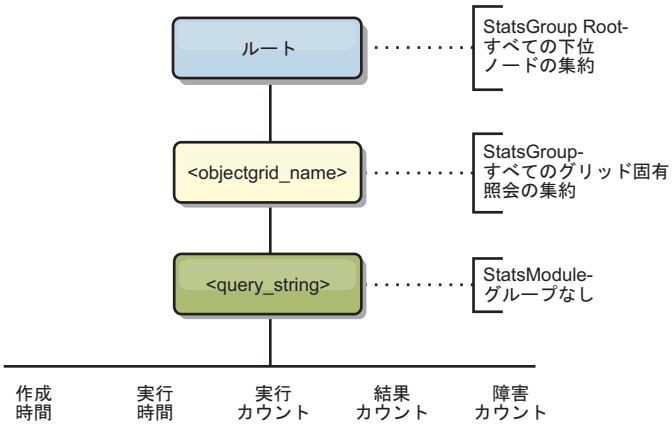


図 36. queryModule の構造

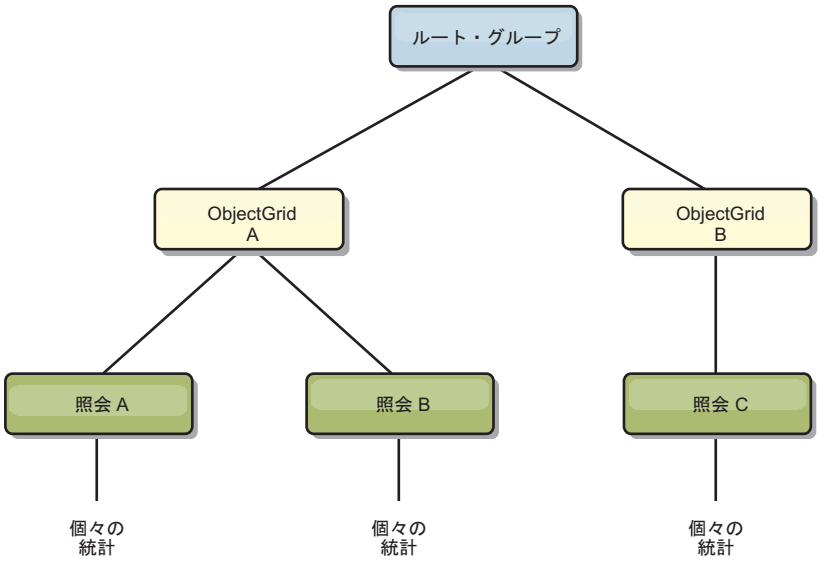


図 37. QueryStats.jpg queryModule 構造の例

wsadmin ツールを使用した MBean へのアクセス

WebSphere Application Server で提供される wsadmin ユーティリティーを使用して、MBean 情報にアクセスすることができます。

WebSphere Application Server インストール内の bin ディレクトリーから wsadmin ツールを実行します。次の例は、動的 eXtreme Scale における現在の断片配置のビューを取得するものです。 wsadmin は、eXtreme Scale が稼働している任意のインストール済み環境から実行できます。 wsadmin をカタログ・サービスで実行する必要はありません。

```
$ wsadmin.sh -lang jython
wsadmin>placementService = AdminControl.queryNames
 ("com.ibm.websphere.objectgrid:*,type=PlacementService")
wsadmin>print AdminControl.invoke(placementService,
 "listObjectGridPlacement","library ms1")

<objectGrid name="library" mapSetName="ms1">
  <container name="container-0" zoneName="DefaultDomain"
  hostName="host1.company.org" serverName="server1">
     <shard type="Primary" partitionName="0"/>
     <shard type="SynchronousReplica" partitionName="1"/>
  </container>
  <container name="container-1" zoneName="DefaultDomain"
  hostName="host2.company.org" serverName="server2">
     <shard type="SynchronousReplica" partitionName="0"/>
     <shard type="Primary" partitionName="1"/>
  </container>
  <container name="UNASSIGNED" zoneName="_ibm_SYSTEM"
  hostName="UNASSIGNED" serverName="UNNAMED">
    <shard type="SynchronousReplica" partitionName="0"/>
    <shard type="AsynchronousReplica" partitionName="0"/>
  </container>
</objectGrid>
```

管理 Bean (MBean) を使用したモニター

ご使用の環境の統計をトラッキングするために Managed Bean (MBean) を使用できます。

始める前に

属性が記録されるようにするには、統計を使用可能に設定する必要があります。統計は、以下のいずれかの方法で使用可能にできます。

- **サーバー・プロパティー・ファイルを使用:**

  サーバー・プロパティー・ファイルの `statsSpec=<StatsSpec>` というキー値エントリーを設定して統計を使用可能にすることができます。次に、設定可能な例をいくつか示します。

  – すべての統計を使用可能にする場合は、`statsSpec=all=enabled` を使用します。

  – ObjectGrid 統計のみを使用可能にする場合は、`statsSpec=og.all=enabled` を使用します。使用可能なすべての統計の仕様の説明を確認するには、API 資料の StatsSpec API を参照してください。

サーバー・プロパティー・ファイルに関して詳しくは、 196 ページの『サーバー・プロパティー・ファイル』を参照してください。

- **管理 Beanを使用して:**

  ObjectGrid MBean で StatsSpec 属性を使用することで、統計を使用可能にできるようになりました。さらに詳細な情報については、StatsSpec API を参照してください。

- **プログラムで:**

  StatsAccessor インターフェースを使用して、統計をプログラムで使用可能にすることもできます。このインターフェースは、 StatsAccessorFactory クラスを使用して取得されます。このインターフェースは、クライアント環境で使用するか、現行プロセスで実行中の eXtreme Scale をモニターする必要がある場合に使用します。

例

Managed Bean の使用例は、 413 ページの『xsAdmin サンプル・ユーティリティーによるモニター』を参照してください。

Web コンソールによるモニター

eXtreme Scale 7.1 リリースの新規フィーチャーの 1 つは、現在の統計およびヒストリカル統計をグラフ化できる Web コンソールです。このコンソールは、概要に関するいくつかの定型のグラフを用意しており、カスタム・レポート・ページで、使用可能な統計からグラフを作成できます。 WebSphere eXtreme Scale のモニター・コンソールのグラフ機能を使用して、環境内のデータ・グリッドの全体的なパフォーマンスを表示できます。

始める前に

コンソール・サーバーは、AIX、Linux、または Windows の各システムで実行可能です。コンソール・サーバー・システムは、カタログ・サービスに接続できる必要があります。また、逆に、カタログ・サービスは、コンソール・サーバーに接続できる必要があります。コンソール・サーバーをホストするシステム上に、スタンドアロン WebSphere eXtreme Scale インストール済み環境が必要です。コンソール・サーバーを始動する startConsoleServer.batlsh スクリプトは、インストール済み環境の XS_ROOT/ObjectGrid/bin ディレクトリーにあります。

データ・グリッドを作成し、データ・グリッドを使用するようにアプリケーションを構成したら、統計が使用可能になるまで少し時間を置きます。例えば、動的キャッシュ・データ・グリッドでは、動的キャッシュを実行している WebSphere Application Server が動的キャッシュ・データ・グリッドに接続されるまで、統計は使用可能になりません。集合を使用している場合は、統計が使用可能になる前に、集合の初期設定が完了していなければなりません。一般的には、統計における変化を見るために、主要な構成変更の後で最大で 1 分待ちます。

ヒント: グラフ内の任意のデータ・ポイントに関するより具体的な情報を表示するには、そのデータ・ポイントの上にマウス・ポインターを移動させてください。

手順

- コンソール・サーバーを始動します。 コンソール・サーバーを始動する startConsoleServer.bat|sh スクリプトは、インストール済み環境の XS_ROOT/ObjectGrid/bin ディレクトリーにあります。
- コンソールにログインします。
  1. Web ブラウザーで、`https://your.console.host:7443` にナビゲートします。 ここで、`your.console.host` は、コンソールをインストールしたマシンのホスト名に置き換えます。
  2. コンソールにログインします。
     - **ユーザー ID:** `admin`
     - **パスワード:** `admin`

     コンソールのウェルカム・ページが表示されます。
  3. 「**設定**」>「**構成**」をクリックして、コンソール構成を確認します。 コンソール構成には、以下のような情報があります。
     - WebSphere eXtreme Scale クライアントのトレース・ストリング (`*=all=disabled` など)
     - 管理者の名前とパスワード
     - 管理者の E メール・アドレス
- 接続状況を表示します。
  1. ツールバーの「**ホーム**」リンクをクリックして、ウェルカム・ページにナビゲートします。
  2. ツールバーの右側で、コンソール・サーバーの接続先のドメインを表示するドロップダウン・リストを見つけます。ドロップダウン・リストの右に、接続状況標識があります。
- モニター対象のカタログ・サーバーへの接続を確立して維持します。
  1. 「**設定**」>「**eXtreme Scale カタログ・サーバー**」ページにナビゲートします。
  2. 新規カタログ・サーバーを追加します。
     a. 正符号をクリックして、既存のカタログ・サーバーを登録するためのダイアログを表示します。
     b. ホスト名、JMX ポート、リスナー・ポートなどの情報を指定します。

        ホスト名

        カタログ・サービスが稼働しているワークステーションのホスト名を表示します。

        JMX ポート

        JMX/RMI 接続を使用可能にするポート番号を表示します。 JMX は、リモート・システムのモニターおよび管理を使用可能にします。

        リスナー・ポート

        Internet Inter-ORB Protocol (IIOP) との通信に使用するリスナー・ポートを表示します。

c. 「**OK**」をクリックします。

d. カタログ・サーバーがナビゲーション・ツリーに追加されていることを確認します。

既存のカタログ・サーバーの情報を表示するには、「**設定**」>「**eXtreme Scale カタログ・サーバー**」ページのナビゲーション・ツリーで、カタログ・サーバーの名前をクリックします。

- モニター対象のドメインへの接続を確立して維持します。
  1. 「**設定**」>「**eXtreme Scale ドメイン**」ページにナビゲートします。
  2. 新規ドメインを追加します。
     a. 正符号をクリックして、カタログ・サービス・ドメインを登録するためのダイアログを表示します。
     b. 情報を指定します。

        Name 管理者によって割り当てられた、ドメインのホスト名を表示します。

        JMX ユーザー
        使用している Java Management Extensions (JMX) ユーザー名を表示します (セキュリティーが使用可能になっている場合)。

        JMX パスワード
        セキュリティーが使用可能になっている場合に JMX パスワードを表示します。

        カタログ・サーバー
        選択したドメインに属する 1 つ以上のカタログ・サーバーをリストします。

        ジェネレーター・クラス
        CredentialGenerator インターフェースを実装するクラスの名前を表示します。このクラスを使用して、クライアントのクレデンシャルが取得されます。

        ジェネレーター・プロパティー
        CredentialGenerator 実装クラスのプロパティーを表示します。このプロパティーが、setProperties(String) メソッドを使用してオブジェクトに設定されます。credentialGeneratorprops 値は、credentialGeneratorClass プロパティーの値が非ヌルの場合にのみ使用されます。

     c. 「OK」をクリックします。
     d. ドメインがナビゲーション・ツリーに追加されていることを確認します。
  3. このページで必要なセキュリティー情報を指定します。
  4. 前に追加したカタログ・サーバーにドメインを関連付けます。

  既存のドメインの情報を表示するには、「**設定**」>「**eXtreme Scale ドメイン**」ページのナビゲーション・ツリーで、カタログ・サーバーの名前をクリックします。
- 現在のサーバー統計を表示するには、「**モニター**」>「**サーバー概要**」をクリックします。

サーバー統計

使用メモリー
サーバー実行時における現在の使用 (実) メモリー量を表示します。メモリー使用量が多い順にトップ 25 のサーバーのみが表示されます。

一定時間の合計メモリー
サーバー実行時における実メモリー使用量を表示します。

一定時間の使用メモリー
サーバー実行時における使用メモリー量を表示します。

- すべてのデータ・グリッドのパフォーマンスを表示するには、「**モニター**」>「**データ・グリッド・ドメインの概要**」をクリックします。

データ・グリッド・ドメインの概要統計

Current®データ・グリッドの使用容量の分布
このグラフには、合計プールおよび使用容量コンシューマー上位の概要が含まれます。トップ 25 のデータ・グリッドのみが表示されます。

平均トランザクション時間 (ミリ秒) によるトップ 5 のデータ・グリッド
このグラフには、平均トランザクション時間で編成された、トップ 5 のデータ・キャッシュのリストが含まれています。

トランザクション/秒の平均スループットによるトップ 5 データ・グリッド
このグラフには、平均スループット (トランザクション/秒) で編成された、トップ 5 のデータ・グリッドのリストが含まれています。

- 個別のデータ・グリッドを表示するには、「**モニター**」>「**データ・グリッドの概要**」>「***data_grid_name***」**をクリックします。** このページでは、キャッシュ・エントリー数、平均トランザクション時間、および平均スループットを含むサマリーが表示されます。以下のグラフも表示できます。

データ・グリッドの概要統計

現在のサマリー
このグリッド内のキャッシュ・オブジェクト (キャッシュ・エントリー) の現在の数、平均トランザクション時間、および平均トランザクション・スループットなどの統計を表示します。

使用容量対キャッシュ・エントリー数
このグラフでは、キャッシュの使用容量とキャッシュ内のエントリー数とを比較して示しています。表示される時刻範囲を、次のように編集できます。最終時刻、最終日付、先週、先月。グラフの表示内容の詳細レベルは、選択した時刻範囲によって変化します。

合計キャッシュ取得要求 対 成功したキャッシュ取得要求
このグラフは、キャッシュに対する成功した照会の数の視覚化に便利です。

- 特定のデータ・グリッドに関するより詳細な情報を表示するには、「**モニター**」>「**データ・グリッドの詳細**」をクリックします。 ツリーにはご使用の構成におけるすべてのデータ・グリッドが表示されています。特定のデータ・グリッドをドリルダウンして、そのデータ・グリッドの一部であるマップを表示します。データ・グリッド名またはマップをクリックすることで、詳細情報を確認できます。

データ・グリッドの統計

現在のサマリー
現在の使用容量と、データ・グリッドが属するゾーンのリストを表示します。

現在の eXtreme Scale オブジェクト・グリッド・マップの使用容量の分布
合計プール (ゾーン別の容量および各ゾーンの合計容量を含む) を表示します。トップ 25 の ObjectGrid マップのみが表示されます。

現在のゾーンの使用容量の分布
ゾーン (合計プールおよび使用容量コンシューマー上位を含む) を表示します。トップ 25 のゾーンのみが表示されます。

マップの統計

現在のサマリー
以下のような統計を表示します。

使用容量
使用容量と、マップが属するゾーンのリストを表示します。

キャッシュ・エントリーの数
マップでキャッシュされているオブジェクトの数を表示します。

平均トランザクション時間 (ミリ秒)
このマップに関係するトランザクションの平均完了時間を表示します。

平均トランザクション・スループット (トランザクション/秒)
このマップに関係する、1 秒当たりのトランザクションの平均数を表示します。

現在の区画の使用容量分布
このグラフには、合計プールおよび使用容量コンシューマー上位の概要が含まれます。トップ 25 の区画のみが表示されます。

- カスタム・レポートに含める統計を選択するには、「**モニター**」>「**カスタム・レポート**」をクリックします。

このビューを使用して、各種統計の詳細データ・グラフを構成します。ツリーを使用して、使用可能なデータ・グリッドとサーバーおよびその関連統計を探索します。グラフ化できるデータを参照するノード上でクリックするか Enter を押すと、メニューが開きます。統計を含んだ新規グラフを作成するか、互換性のある統計で統計を既存のグラフに追加します。

ドメインの統計

平均トランザクション時間 (ミリ秒)
このドメインでトランザクションを完了するために必要な平均時間を表示します。

平均トランザクション・スループット (トランザクション/秒)
このドメイン内の 1 秒当たりのトランザクションの平均数を表示します。

最大トランザクション時間 (ミリ秒)
このドメイン内で*最も時間がかかった* トランザクションで費やした時間を表示します。

最小トランザクション時間 (ミリ秒)
このドメイン内で*最も時間がかからなかった* トランザクションで費やした時間を表示します。

合計トランザクション時間 (ミリ秒)
このドメインでトランザクションに費やした、ドメインの初期設定時からの合計時間を表示します。

eXtreme Scale コンテナーの統計

平均トランザクション時間 (ミリ秒)
このカタログ・サーバーでトランザクションを完了するために必要な平均時間を表示します。

平均トランザクション・スループット (トランザクション/秒)
このカタログ・サーバーの 1 秒当たりのトランザクションの平均数を表示します。

最大トランザクション時間 (ミリ秒)
このカタログ・サーバーで*最も時間がかかった* トランザクションで費やした時間を表示します。

最小トランザクション時間 (ミリ秒)
このカタログ・サーバーで*最も時間がかからなかった* トランザクションで費やした時間を表示します。

合計トランザクション時間 (ミリ秒)
このカタログ・サーバーでトランザクションに費やした、このカタログ・サーバーの初期設定時からの合計時間を表示します。

キャッシュ内の合計エントリー
このカタログ・サーバーで監視されるグリッド内にキャッシュされたオブジェクトの現在の数を表示します。

キャッシュ内の最大エントリー
このカタログ・サーバーで監視されるグリッド内にキャッシュされたオブジェクトの最大数を表示します。

キャッシュ内の最小エントリー
このカタログ・サーバーで監視されるグリッド内にキャッシュされたオブジェクトの最小数を表示します。

ヒット・レート (パーセンテージ)
選択したデータ・グリッドのヒット・レート (ヒット率) を表示します。高ヒット・レートが望ましい状態です。ヒット・レートは、永続ストアへのアクセスの回避にグリッドがどのくらい役立っているかを示します。

使用バイト
このマップによるメモリー消費量を表示します。

最小使用バイト
このカタログ・サービスおよびそのマップによるメモリー消費量の最低点を表示します。

最大使用バイト
このカタログ・サービスおよびそのマップによるメモリー消費量の最高点を表示します。

合計ヒット数
要求データがマップ内で検出され、永続ストアにアクセスする必要がなかった回数の合計を表示します。

合計 GET 数
データを取得するためにマップが永続ストアにアクセスする必要があった回数の合計を表示します。

空きヒープ (MB)
カタログ・サーバーによって使用されている JVM で使用可能なヒープの実際の容量を表示します。

合計ヒープ
このカタログ・サーバーによって使用されている JVM で使用可能なヒープの容量を表示します。

使用メモリー
このカタログ・サーバーによって使用されている JVM の使用メモリーを表示します。

使用可能なプロセッサーの数
このカタログ・サービスおよびそのマップで使用可能な CPU の数を表示します。最高の安定度を実現するために、60% のプロセッサー負荷でサーバー、さらに 60% のヒープ負荷で JVM ヒープを稼働してください。スパイクでプロセッサー使用量を 80% から 90% の間に引き上げることができますが、通常はこのレベルより高いレベルでサーバーを稼働しないようにしてください。

最大ヒープ・サイズ (MB)
このカタログ・サーバーによって使用されている JVM で使用可能なヒープの最大容量を表示します。

グリッドの統計

平均トランザクション時間 (ミリ秒)
このグリッドに関係するトランザクションを完了するために必要な平均時間を表示します。

平均トランザクション・スループット (トランザクション/秒)
このグリッドが完了した 1 秒当たりのトランザクションの平均数を表示します。

最大トランザクション時間 (ミリ秒)
このグリッドが完了した中で*最も時間がかかった* トランザクションで費やした時間を表示します。

最小トランザクション時間 (ミリ秒)
このグリッドが完了した中で*最も時間がかからなかった* トランザクションで費やした時間を表示します。

合計トランザクション時間 (ミリ秒)
このグリッドのトランザクション処理の合計時間を表示します。

マップの統計

キャッシュ内の合計エントリー
このマップでキャッシュされているオブジェクトの現在の数を表示します。

キャッシュ内の最大エントリー
このマップの初期設定時からの、このマップでキャッシュされたオブジェクトの最大数を表示します。

キャッシュ内の最小エントリー
このマップの初期設定時からの、このマップでキャッシュされたオブジェクトの最小数を表示します。

ヒット・レート (パーセンテージ)
選択したマップのヒット・レート (ヒット率) を表示します。高ヒット・レートが望ましい状態です。ヒット・レートは、永続ストアへのアクセスの回避にマップがどのくらい役立っているかを示します。

使用バイト
このマップによるメモリー消費量を表示します。

最小使用バイト
このマップの最小消費量 (バイト単位) を表示します。

最大使用バイト
このマップの最大消費量 (バイト単位) を表示します。

合計ヒット数
要求データがマップ内で検出され、永続ストアにアクセスする必要がなかった回数の合計を表示します。

合計 GET 数
データを取得するためにマップが永続ストアにアクセスする必要があった回数の合計を表示します。

空きヒープ (MB)
カタログ・サーバーによって使用されている JVM 内の、このマップで使用可能なヒープの現在の容量を表示します。

合計ヒープ (MB)
カタログ・サーバーによって使用されている JVM 内の、このマップで使用可能なヒープの合計容量を表示します。最高の安定度を実現するために、60% のプロセッサー負荷でサーバー、さらに 60% のヒープ負荷で JVM ヒープを稼働してください。スパイクでプロセッサー使用量を 80% から 90% の間に引き上げることができますが、通常はこのレベルより高いレベルでサーバーを稼働しないようにしてください。

使用メモリー (MB)
このマップの使用メモリー容量を表示します。

使用可能なプロセッサーの数
このマップで使用可能な CPU の数を表示します。最高の安定度を実現するために、60% のプロセッサー負荷でサーバー、さらに 60% のヒープ負荷で JVM ヒープを稼働してください。スパイクでプロセッサー使用量

を 80% から 90% の間に引き上げることができますが、通常はこのレベルより高いレベルでサーバーを稼働しないようにしてください。

最大ヒープ・サイズ (MB)
カタログ・サーバーによって使用されている JVM 内の、このマップで使用可能なヒープの最大容量を表示します。

関連概念

9 ページの『第 2 章 キャパシティー・プランニング』
初期データ・セット・サイズおよび予測されるデータ・セット・サイズが わかっている場合、WebSphere eXtreme Scale を実行するために必要なキャパシティーを計画できます。こうした計画は、将来の変更に向けて eXtreme Scale を効率よくデプロイするのに役立ちますが、eXtreme Scale の弾力性を最大限に生かすことができます。これにより、メモリー内のデータベースや他のタイプのデータベースなど、異なるシナリオを考える必要がなくなります。

ベンダー・ツールによるモニター

一般的によく使われるいくつかのエンタープライズ・モニタリング・ソリューションを使用して、WebSphere eXtreme Scale をモニターすることができます。 パブリックにアクセス可能な管理 Bean を使用して WebSphere eXtreme Scale をモニターする IBM Tivoli Monitoring および Hyperic HQ 用に、プラグイン・エージェントが組み込まれています。CA Wily Introscope は Java メソッドのインスツルメンテーションを使用して、統計情報を収集します。

IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale のモニター

IBM Tivoli Enterprise Monitoring Agent は、分散環境およびホスト環境のデータベース、オペレーティング・システム、およびサーバーをモニターするために使用できる機能の豊富なモニタリング・ソリューションです。 WebSphere eXtreme Scale には、eXtreme Scale 管理 Bean をイントロスペクトする場合に使用できるカスタマイズ・エージェントが含まれています。このソリューションは、スタンドアロン eXtreme Scale および WebSphere Application Server デプロイメントの両方で効果的に機能します。

始める前に

- WebSphere eXtreme Scale バージョン 7.0.0 以降をインストールします。

  また、 WebSphere eXtreme Scale サーバーから統計データを収集するには、統計を使用可能にする必要があります。統計を使用可能にする各種オプションについては、 429 ページの『管理 Bean (MBean) を使用したモニター』および 413 ページの『xsAdmin サンプル・ユーティリティーによるモニター』を参照してください。
- IBM Tivoli Monitoring バージョン 6.2.1 (フィックスパック 2 以降の適用済み) をインストールします。
- eXtreme Scale サーバーが実行される各サーバーまたはホストに Tivoli OS エージェントをインストールします。

- WebSphere eXtreme Scale エージェントをインストールします。 (IBM Open Process Automation Library (OPAL) サイトから無料でダウンロードできます。)

以下の手順を実行して、Tivoli Monitoring Agent をインストールおよび構成します。

手順

1. WebSphere eXtreme Scale 用に Tivoli Monitoring Agent をインストールします。

   Tivoli インストール・イメージをダウンロードし、そのファイルを一時ディレクトリーに解凍します。

2. eXtreme Scale アプリケーション・サポート・ファイルをインストールします。

   以下の各デプロイメントに eXtreme Scale アプリケーション・サポートをインストールします。

   - Tivoli Enterprise Portal Server (TEPS)
   - Enterprise Desktop クライアント (TEPD)
   - Tivoli Enterprise Monitoring Server (TEMS)

   a. 作成した一時ディレクトリーから、新しいコマンド・ウィンドウを開始し、ご使用のプラットフォームに合った実行可能ファイルを実行します。 インストール・スクリプトが、ご使用の Tivoli デプロイメント・タイプ (TEMS、TEPD、または TEPS) を自動的に検出します。タイプによらず単一のホストでも複数のホストでもインストールできますが、デプロイメントの場合は 3 つのタイプのすべてについて eXtreme Scale エージェントのアプリケーション・サポート・ファイルをインストールする必要があります。

   b. 「**インストーラー**」ウィンドウで、デプロイされた Tivoli コンポーネントの選択が正しいことを確認します。「**次へ**」をクリックします。

   c. プロンプトが出されたら、ホスト名および管理クレデンシャルをサブミットします。「**次へ**」をクリックします。

   d. 「**Monitoring Agent for WebSphere eXtreme Scale**」を選択します。「**次へ**」をクリックします。

   e. 実行されるインストール・アクションについて通知があります。「**次へ**」をクリックすると、インストールの進行状況を完了まで確認することができます。

   手順を完了すると、WebSphere eXtreme Scale エージェントが必要とするすべてのアプリケーション・サポート・ファイルがインストールされます。

3. 各 eXtreme Scale ノードにエージェントをインストールします。

   各コンピューターに Tivoli OS エージェントをインストールします。このエージェントを構成または開始する必要はありません。上記のステップと同じインストール・イメージを使用して、プラットフォーム固有の実行可能ファイルを実行します。

   指針としては、ホストごとにエージェントを 1 つだけインストールするようにする必要があります。 1 つのエージェントで eXtreme Scale サーバーの多数の

インスタンスをサポートすることができます。 1 つのエージェント・インスタンスで約 50 の eXtreme Scale サーバーをモニターすると、最適のパフォーマンスが得られます。

a. 「インストール・ウィザード」初期画面で「**次へ**」をクリックすると、インストール・パス情報を指定する画面が開きます。
b. 「**Tivoli Monitoring インストール・ディレクトリー**」フィールドに、`C:¥IBM¥ITM` (または `/opt/IBM/ITM`) と入力するか、またはこのパスを参照します。 次に「**インストール可能なメディアのロケーション**」フィールドで、表示されている値が正しいことを確認してから「**次へ**」をクリックします。
c. 追加するコンポーネント (「**ソリューションのローカル・インストールの実行**」など) を選択して、「**次へ**」をクリックします。
d. サポートを追加する対象のアプリケーション (「**Monitoring Agent for WebSphere eXtreme Scale**」など) を選択して、「**次へ**」をクリックします。
e. アプリケーション・サポートが正常に追加されるまで進行状況を確認することができます。

注: それぞれの eXtreme Scale ノードに これまでのステップを繰り返します。サイレント・インストールを使用することもできます。サイレント・インストールに関して詳しくは、IBM Tivoli Monitoring インフォメーション・センターを参照してください。

4. WebSphere eXtreme Scale エージェントを構成します。

インストールした各エージェントを、カタログ・サーバー、eXtreme Scale サーバー、またはその両方をモニターするように構成する必要があります。

Windows プラットフォームと UNIX プラットフォームとでは構成手順が異なります。 Windows プラットフォームの場合の構成は、**Tivoli Monitoring サービスの管理**ユーザー・インターフェースを使用して実行します。 UNIX プラットフォームの場合の構成はコマンド行に基づいて行います。

Windows 以下のステップを使用して、まず Windows 上のエージェントを構成します

a. 「**Tivoli Enterprise Monitoring サービスの管理**」ウィンドウで、「**スタート**」 → 「**すべてのプログラム**」 → 「**IBM Tivoli Monitoring**」 → 「**Tivoli Monitoring サービスの管理**」の順にクリックします。
b. 「**Monitoring Agent for WebSphere eXtreme Scale**」を右クリックし、「**デフォルトを使用して構成**」を選択します。そうすると、エージェントに固有のインスタンスを作成するウィンドウが開きます。
c. 固有の名前 (例えば「`instance1`」など) を入力し、「**次へ**」をクリックします。

- スタンドアロンの eXtreme Scale サーバーをモニターする場合は、次のステップを実行します。
  a. Java パラメーターを更新し、「**Java Home**」の値が正しいことを確認します。 JVM 引数は空のままでもかまいません。「**次へ**」をクリックします。

b. 「**MBean サーバー接続タイプ**」のタイプを選択し、スタンドアロン eXtreme Scale サーバーの場合は「JSR-160 準拠サーバー」を使用します。「**次へ**」をクリックします。

c. セキュリティーが有効な場合、「**ユーザー ID**」および「**パスワード**」値を更新します。「**JMX サービス URL**」の値は、そのままにしておきます。この値は、後でオーバーライドします。「**JMX クラスパス情報**」フィールドはそのままにしておきます。「**次へ**」をクリックします。

Windows でエージェント用にサーバーを構成するには、以下のステップを実行します。

a. 「**WebSphere eXtreme Scale グリッド・サーバー**」ペインで eXtreme Scale サーバーのサブノード・インスタンスをセットアップします。ご使用のコンピューター上にコンテナー・サーバーがない場合、「**次へ**」をクリックして、カタログ・サービス・ペインに進みます。

b. ご使用のコンピューターに複数の eXtreme Scale コンテナー・サーバーがある場合、エージェントで各サーバーをモニターするように構成します。

c. それぞれの名前とポートが固有な場合は、「**新規**」をクリックし、eXtreme Scale サーバーを必要なだけいくつでも追加することができます。(eXtreme Scale サーバーが始動されるときは、JMXPort 値が指定されていなければなりません。)

d. コンテナー・サーバーの構成を完了したならば、「**次へ**」をクリックして「**WebSphere eXtreme Scale カタログ・サーバー**」ペインに進みます。

e. カタログ・サーバーがない場合は、「**OK**」をクリックします。カタログ・サーバーがある場合は、コンテナー・サーバーについて実行したのと同様に各サーバーに新しい構成を追加します。ここでもまた、固有の名前 (できればカタログ・サービスを開始するときに使用するものと同じ名前) を選択します。「**OK**」をクリックして終了します。

- WebSphere Application Server プロセス内に組み込まれている eXtreme Scale サーバー上でエージェントのサーバーをモニターする場合、以下のステップを実行します。

  a. Java パラメーターを更新し、「**Java Home**」の値が正しいことを確認します。 JVM 引数は空のままでもかまいません。「**次へ**」をクリックします。

  b. 「**MBean サーバー接続タイプ**」を選択します。ご使用の環境に適した WebSphere Application Server バージョンを選択します。「**次へ**」をクリックします。

  c. パネルの WebSphere Application Server 情報が正しいことを確認します。「**次へ**」をクリックします。

  d. サブノード定義を 1 つのみ追加します。サブノード定義に名前を指定し、ポート定義は更新しないでください。 WebSphere Application Server 環境内で、そのコンピューター上で実行中のノード・エージェントによって管理されるすべてのアプリケーション・サーバーからデータが収集されます。「**次へ**」をクリックします。

  e. この環境にカタログ・サーバーが存在しない場合は、「**OK**」をクリックします。カタログ・サーバーがある場合は、コンテナー・サーバーと同様、それぞれのカタログ・サーバーについて新しい構成を追加します。カタロ

グ・サービスに固有の名前、できればカタログ・サービスを開始するときに使用するものと同じ名前を選択します。「**OK**」をクリックして終了します。

注: コンテナー・サーバーは、カタログ・サービスと連結する必要はありません。

これでエージェントとサーバーが構成されて作動可能になるので、次のウィンドウで「instance1」を右クリックしてエージェントを開始します。

UNIX UNIX プラットフォーム上のエージェントをコマンド行で構成する場合は、以下のステップを実行します。

次に JSR160 準拠の接続タイプを使用するスタンドアロン・サーバーの例を示します。この例では、単一ホスト (rhea00b02) 上に 3 つの eXtreme Scale コンテナーがあり、JMX リスナーのアドレスは、それぞれ 15000、15001 および 15002 です。カタログ・サーバーはありません。

構成ユーティリティーからの出力は*モノスペースのイタリック体* で、一方ユーザー応答は**モノスペースの太字体**で示されています。 (ユーザー応答が不要であった場合は、Enter キーを押してデフォルトが選択されました。)

```
rhea00b02 # ./itmcmd config -A xt
Agent configuration started...
Enter instance name (default is: ): inst1
Edit "Monitoring Agent for WebSphere eXtreme Scale" settings? [ 1=Yes, 2=No ] (default is: 1):
Edit 'Java' settings? [ 1=Yes, 2=No ] (default is: 1):
Java home (default is: C:¥Program Files¥IBM¥Java50): /opt/OG61/java
Java trace level [ 1=Error, 2=Warning, 3=Information, 4=Minimum Debug, 5=Medium Debug, 6=Maximum Debug,
    7=All ] (default is: 1):
JVM arguments (default is: ):
Edit 'Connection' settings? [ 1=Yes, 2=No ] (default is: 1):
MBean server connection type [ 1=JSR-160-Compliant Server, 2=WebSphere Application Server version 6.0,
3=WebSphere Application Server version 6.1, 4=WebSphere Application Server version 7.0 ] (default is: 1): 1
Edit 'JSR-160-Compliant Server' settings? [ 1=Yes, 2=No ] (default is: 1):
JMX user ID (default is: ):
Enter JMX password (default is: ):
Re-type : JMX password (default is: ):
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:port/objectgrid/MBeanServer):
----------------------------------------
JMX Class Path Information
JMX base paths (default is: ):
JMX class path (default is: ):
JMX JAR directories (default is: ):
Edit 'WebSphere eXtreme Scale Catalog Service' settings? [ 1=Yes, 2=No ] (default is: 1): 2
Edit 'WebSphere eXtreme Scale Grid Servers' settings? [ 1=Yes, 2=No ] (default is: 1): 1
No 'WebSphere eXtreme Scale Grid Servers' settings available?
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02_c0
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
service:jmx:rmi:///jndi/rmi://localhost:15000/objectgrid/MBeanServer

'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers=ogx
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02_c1
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
service:jmx:rmi:///jndi/rmi://localhost:15001/objectgrid/MBeanServer

'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers= rhea00b02_c1
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02_c2
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
service:jmx:rmi:///jndi/rmi://localhost:15002/objectgrid/MBeanServer

'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers= rhea00b02_c2
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 5

Will this agent connect to a TEMS? [1=YES, 2=NO] (Default is: 1):
```

```
TEMS Host Name (Default is: rhea00b00):

Network Protocol [ip, sna, ip.pipe or ip.spipe] (Default is: ip.pipe):

     Now choose the next protocol number from one of these:
     - ip
     - sna
     - ip.spipe
     - 0 for none
Network Protocol 2 (Default is: 0):
IP.PIPE Port Number (Default is: 1918):
Enter name of KDC_PARTITION (Default is: null):

Configure connection for a secondary TEMS? [1=YES, 2=NO] (Default is: 2):
Enter Optional Primary Network Name or 0 for "none" (Default is: 0):
Agent configuration completed...
```

上記の例では、「inst1」というエージェント・インスタンスが作成され、Java ホーム設定が更新されます。 eXtreme Scale コンテナー・サーバーは構成されますが、カタログ・サービスは構成されません。

注: 上記の手順では、次の形式のテキスト・ファイルがディレクトリー <ITM_install>/config/<host>_xt_<instance name>.cfg に作成されます。

例: rhea00b02_xt_inst1.cfg

自分で選んだプレーン・テキスト・エディターを使用してこのファイルを編集することをお勧めします。そうしたファイルの内容の例を以下に示します。

```
INSTANCE=inst2 [SECTION=KQZ_JAVA [ { JAVA_HOME=/opt/OG61/java }  { JAVA_TRACE_LEVEL=ERROR }  ]
SECTION=KQZ_JMX_CONNECTION_SECTION [ { KQZ_JMX_CONNECTION_PROPERTY=KQZ_JMX_JSR160_JSR160 }  ]
SECTION=KQZ_JMX_JSR160_JSR160 [ { KQZ_JMX_JSR160_JSR160_CLASS_PATH_TITLE= }
{ KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:rmi:///jndi/rmi://localho
st:port/objectgrid/MBeanServer }  { KQZ_JMX_JSR160_JSR160_CLASS_PATH_SEPARATOR= }  ]
SECTION=OGS:rhea00b02_c1 [ { KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:
rmi:///jndi/rmi://localhost:15001/objectgrid/MBeanServer }  ]
SECTION=OGS:rhea00b02_c0 [ { KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:
rmi:///jndi/rmi://localhost:15002/objectgrid/MBeanServer }  ]
SECTION=OGS:rhea00b02_c2 [ { KQZ_JMX_JSR160_JSR160_SERVICE_URL=service:jmx:
rmi:///jndi/rmi://localhost:15002/objectgrid/MBeanServer }  ]]
```

次に、WebSphere Application Server デプロイメント上の構成の例を示します。

```
rhea00b02 # ./itmcmd config -A xt
Agent configuration started...
Enter instance name (default is: ): inst1
Edit "Monitoring Agent for WebSphere eXtreme Scale" settings? [ 1=Yes, 2=No ] (default is: 1): 1
Edit 'Java' settings? [ 1=Yes, 2=No ] (default is: 1): 1
Java home (default is: C:¥Program Files¥IBM¥Java50): /opt/WAS61/java
Java trace level [ 1=Error, 2=Warning, 3=Information, 4=Minimum Debug, 5=Medium Debug, 6=Maximum Debug,
    7=All ] (default is: 1):
JVM arguments (default is: ):
Edit 'Connection' settings? [ 1=Yes, 2=No ] (default is: 1):
MBean server connection type [ 1=JSR-160-Compliant Server, 2=WebSphere Application Server version 6.0,
3=WebSphere Application Server version 6.1, 4=WebSphere Application Server version 7.0 ] (default is: 1): 4
Edit 'WebSphere Application Server version 7.0' settings? [ 1=Yes, 2=No ] (default is: 1):WAS user ID (default is: ):
Enter WAS password (default is: ):
Re-type : WAS password (default is: ):
WAS host name (default is: localhost): rhea00b02
WAS port (default is: 2809):
WAS connector protocol [ 1=rmi, 2=soap ] (default is: 1):
WAS profile name (default is: ): default
---------------------------------------
WAS Class Path Information
WAS base paths (default is: C:¥Program Files¥IBM¥WebSphere¥AppServer;/opt/IBM/WebSphere/AppServer): /opt/WAS61
WAS class path (default is: runtimes/com.ibm.ws.admin.client_6.1.0.jar;runtimes/com.ibm.ws.ejb.thinclient_7.0.0.jar):
WAS JAR directories (default is: lib;plugins):
Edit 'WebSphere eXtreme Scale Grid Servers' settings? [ 1=Yes, 2=No ] (default is: 1):
No 'WebSphere eXtreme Scale Grid Servers' settings available?
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 1
WebSphere eXtreme Scale Grid Servers (default is: ): rhea00b02
JMX service URL (default is: service:jmx:rmi:///jndi/rmi://localhost:<port>/objectgrid/MBeanServer):
```

```
'WebSphere eXtreme Scale Grid Servers' settings:  WebSphere eXtreme Scale Grid Servers=rhea00b02
Edit 'WebSphere eXtreme Scale Grid Servers' settings, [1=Add, 2=Edit, 3=Del, 4=Next, 5=Exit] (default is: 4): 5
Edit 'WebSphere eXtreme Scale Catalog Service' settings? [ 1=Yes, 2=No ] (default is: 1): 2
Will this agent connect to a TEMS? [1=YES, 2=NO] (Default is: 1):
TEMS Host Name (Default is: rhea00b02):

Network Protocol [ip, sna, ip.pipe or ip.spipe] (Default is: ip.pipe):

     Now choose the next protocol number from one of these:
     - ip
     - sna
     - ip.spipe
     - 0 for none
Network Protocol 2 (Default is: 0):
IP.PIPE Port Number (Default is: 1918):
Enter name of KDC_PARTITION (Default is: null):

Configure connection for a secondary TEMS? [1=YES, 2=NO] (Default is: 2):
Enter Optional Primary Network Name or 0 for "none" (Default is: 0):
Agent configuration completed...
rhea00b02 #
```

WebSphere Application Server デプロイメントの場合、複数のサブノードを作成する必要はありません。eXtreme Scale エージェントは、ノード・エージェントに接続して、管理下のアプリケーション・サーバーからすべての情報を収集します。

SECTION=CAT はカタログ・サービス行を意味し、一方 SECTION=OGS は eXtreme Scale サーバー構成行を意味します。

5. すべての eXtreme Scale コンテナー・サーバーの JMX ポートを構成します。

-JMXServicePort 引数が指定されないまま eXtreme Scale コンテナー・サーバーが始動されるときは、MBean サーバーに動的ポートが割り当てられます。エージェントは通信相手の JMX ポートをあらかじめ知っておく必要があります。エージェントは動的ポートとは連動しません。

サーバーの始動時、startOgServer.sh | .bat コマンドを使用して eXtreme Scale サーバーを開始する場合は -JMXServicePort <port_number> 引数を指定します。このコマンドの実行により、プロセス内の JMX サーバーが静的事前定義ポートを listen するようになります。

UNIX インストールの上記の例の場合は、2 つの eXtreme Scale サーバーを次のように設定されたポートで始動する必要があります。

a. "-JMXServicePort" "15000" (rhea00b02_c0 の場合)
b. "-JMXServicePort" "15001" (rhea00b02_c1 の場合)

a. eXtreme Scale エージェントを開始します。

   inst1 インスタンスが作成されたとすると、上記の例のように、以下のコマンドを発行します。

   1) cd <ITM_install>/bin
   2) itmcmd agent –o inst1 start xt

b. eXtreme Scale エージェントを停止します。

   「inst1」が上記の例のようにして作成されたインスタンスであったとして、以下のコマンドを発行します。

   1) cd <ITM_install>/bin

2) `itmcmd agent –o inst1 stop xt`

6. すべての eXtreme Scale コンテナー・サーバーの統計を使用可能にします。

   エージェントは、eXtreme Scale 統計 MBeans を使用して統計を記録します。以下の方式を使用して、eXtreme Scale 統計仕様を使用可能にする必要があります。

   - サーバー・プロパティーを構成して、コンテナー・サーバーの始動時にすべての統計を使用可能にします (all=enabled)。
   - xsadmin サンプル・ユーティリティーで、 -setstatsspec all=enabled パラメーターを使用して、すべてのアクティブ・コンテナーの統計を使用可能にします。

タスクの結果

すべてのサーバーが構成されて始動されたならば、IBM Tivoli Portal コンソールに MBean データが表示されます。事前定義ワークスペースには、グラフとデータ・メトリックがノード・レベルごとに表示されます。

モニターされるすべてのノードの「**eXtreme Scale グリッド・サーバー**」ノードについて、以下のワークスペースが定義されています。

- eXtreme Scale トランザクション・ビュー
- eXtreme Scale プライマリー断片ビュー
- eXtreme Scale メモリー・ビュー
- eXtreme Scale ObjectMap ビュー

独自のワークスペースも構成することができます。詳しくは、 IBM Tivoli Monitoring インフォメーション・センターのワークスペースのカスタマイズに関する情報を参照してください。

CA Wily Introscope による eXtreme Scale アプリケーションのモニター

CA Wily Introscope は、エンタープライズ・アプリケーション環境のパフォーマンス上の問題を検出して診断する場合に使用できるサード・パーティーの管理製品です。 eXtreme Scale には、CA Wily Introscope を構成して eXtreme Scale ランタイムの選択部分をイントロスペクトし、eXtreme Scale アプリケーションを迅速に表示、検証する場合の詳細が含まれています。 CA Wily Introscope は、スタンドアロン環境と WebSphere Application Server デプロイメント環境の両方で効果的に機能します。

概説

CA Wily Introscope を使用して eXtreme Scale アプリケーションをモニターするには、eXtreme Scale のモニター情報へのアクセスを可能にする設定を ProbeBuilderDirective (PBD) ファイルに作成する必要があります。

重要: Introscope のインスツルメンテーション・ポイントは、各フィックスパックまたはリリースで変わる可能性があります。新しいフィックスパックまたはリリースをインストールしたら、インスツルメンテーション・ポイントに変更がないか、資料を確認してください。

eXtreme Scale アプリケーションをモニターするように、CA Wily Introscope の ProbeBuilderDirective (PBD) ファイルを構成できます。CA Wily Introscope は、アプリケーション管理製品で、これを使用すると複雑な複合 Web アプリケーション環境におけるパフォーマンス上の問題を積極的に検出、選別、および診断することができます。

カタログ・サービスのモニター用の PBD ファイル設定

カタログ・サービスをモニターするには、PBD ファイルの以下の設定を 1 つ以上を使用できます。

```
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  changeDefinedCompleted
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  viewChangeCompleted
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  viewAboutToChange
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeat
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCluster
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCurrentLeader
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatDeadServer
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewLeader
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewServer
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
  PlacementServiceImpl
importRouteInfo BlamePointTracerDifferentMethods
  "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
  PlacementServiceImpl heartbeat
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
  PlacementServiceImpl joinPlacementGroup
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
  TraceOneMethodOfClass:
com.ibm.ws.objectgrid.catalog.placement.PlacementServiceImpl
  classifyServer
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
  BalanceGridEventListener shardActivated
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.catalog.placement.
  BalanceGridEventListener shardDeactivate
BlamePointTracerDifferentMethods "OGcatalog|{classname}|{method}"
```

カタログ・サービスのモニター用のクラス

HAControllerImpl

HAControllerImpl クラスは、コア・グループのライフサイクル・イベントお

よびフィードバック・イベントを処理します。このクラスをモニターすると、コア・グループの構造および変更を確認できます。

ServerAgent

ServerAgent クラスは、コア・グループ・イベントとカタログ・サービスの通信を担当します。さまざまなハートビート呼び出しをモニターして、主要なイベントを見極めることができます。

PlacementServiceImpl

PlacementServiceImpl クラスは、コンテナーを調整します。このクラスのメソッドは、サーバーの結合イベントおよび配置イベントをモニターするために使用できます。

BalanceGridEventListener

BalanceGridEventListener クラスは、カタログのリーダーシップを制御します。このクラスをモニターすると、現在リーダーとして実行中のカタログ・サービスを確認できます。

コンテナー・モニター用の PBD ファイル設定

コンテナーをモニターするには、PBD ファイルの以下の設定を 1 つ以上使用できます。

```
TraceOneMethodOfClass: com.ibm.ws.objectgrid.ShardImpl processMessage
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.plugins.
  CommittedLogSequenceListenerProxy applyCommitted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.plugins.
  CommittedLogSequenceListenerProxy sendApplyCommitted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.map.BaseMap evictMapEntries
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.checkpoint.
  CheckpointMapImpl$CheckpointIterator activateListener
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  changeDefinedCompleted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  viewChangeCompleted
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.hamanager.HAControllerImpl
  viewAboutToChange
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  batchProcess
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeat
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCluster
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatCurrentLeader
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatDeadServer
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewLeader
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
```

```
TraceOneMethodOfClass: com.ibm.ws.objectgrid.container.ServerAgent
  heartbeatNewServer
BlamePointTracerDifferentMethods "OGcontainer|{classname}|{method}"
```

コンテナー・モニター用のクラス

ShardImpl

ShardImpl クラスには、processMessage メソッドがあります。processMessage メソッドは、クライアント要求のためのメソッドです。このメソッドを使用すると、サーバー・サイドの応答時間および要求数を確認できます。すべてのサーバー全体でカウントを監視し、ヒープ使用状況をモニターすることにより、グリッドのバランスが取れているかどうかを判別できます。

CheckpointIterator

CheckpointIterator クラスには、プライマリーをピア・モードにする activateListener メソッド呼び出しがあります。プライマリーがピア・モードになると、メソッド完了後に、レプリカがプライマリーにより更新されます。レプリカがプライマリー全体から再生成される場合、この操作に時間がかかることがあります。この操作が完了するまでは、システムの回復状態は不十分であるため、このクラスを使用してこの操作の進行状況をモニターできます。

CommittedLogSequenceListenerProxy

CommittedLogSequenceListenerProxy クラスには、2 つの興味深いメソッドがあります。applyCommitted メソッドは、すべてのトランザクションで実行され、sendApplyCommitted メソッドは、レプリカが情報をプルしているときに実行されます。これら 2 つのメソッドの実行頻度により、レプリカがプライマリーにどの程度後れを取らずに対応できているかがある程度分かります。

クライアント・モニター用の PBD ファイル設定

クライアントをモニターするには、PBD ファイルの以下の設定を 1 つ以上使用できます。

```
TraceOneMethodOfClass: com.ibm.ws.objectgrid.client.ORBClientCoreMessageHandler
 sendMessage
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.corba.cluster.ClusterStore
  bootstrap
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.corba.cluster.ClusterStore
  epochChangeBootstrap
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.map.BaseMap evictMapEntries
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.cluster.orb.routing.
  SelectionServiceImpl routeFailed
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.cluster.orb.routing.
  SelectionServiceImpl routeFailed
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.SessionImpl getMap
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TraceOneMethodOfClass: com.ibm.ws.objectgrid.ObjectGridImpl getSession
BlamePointTracerDifferentMethods "OGclient|{classname}|{method}"
TurnOn: ObjectMap
```

```
SetFlag: ObjectMap
IdentifyClassAs: com.ibm.ws.objectgrid.ObjectMapImpl ObjectMap
TraceComplexMethodsifFlagged: ObjectMap BlamePointTracerDifferentMethods
"OGclient|{classname}|{method}"
```

クライアント・モニター用のクラス

ORBClientCoreMessageHandler

ORBClientCoreMessageHandler クラスは、コンテナーへのアプリケーション要求の送信を担当します。sendMessage メソッドでクライアントの応答時間および要求数をモニターできます。

ClusterStore

ClusterStore クラスには、クライアント・サイドでのルーティング情報が保持されます。

BaseMap

BaseMap クラスには、Evictor がマップからエントリーを除去するときに呼び出される evictMapEntries メソッドがあります。

SelectionServiceImpl

SelectionServiceImpl クラスは、ルーティング上の決定を行います。クライアントによりフェイルオーバーに関する決定が下される場合、このクラスを使用すると、その決定から実行されるアクションを判別できます。

ObjectGridImpl

ObjectGridImpl クラスには、このメソッドに対する要求数を判別するためにモニターできる getSession メソッドがあります。

Hyperic HQ による eXtreme Scale のモニター

Hyperic HQ は、サード・パーティーのモニター・ソリューションで、オープン・ソース・ソリューションあるいはエンタープライズ製品として自由に使用可能です。WebSphere eXtreme Scale には、あるプラグインが含まれていて、このプラグインにより Hyperic HQ エージェントは eXtreme Scale コンテナー・サーバーを検出し、eXtreme Scale 管理 Bean を使用して統計のレポートおよび集約を行うことができます。 Hyperic HQ を使用すると、スタンドアロン eXtreme Scale デプロイメントをモニターできます。

始める前に

- この一連の説明は、Hyperic バージョン 4.0 を対象としています。これよりも新しいバージョンの Hyperic の場合、パス名やエージェントとサーバーの始動方法などの情報について Hyperic 資料を参照してください。
- Hyperic サーバーとエージェントのインストールをダウンロードします。サーバーのインストール済み環境が 1 つ実行中である必要があります。 eXtreme Scale サーバーのすべてを検出するには、eXtreme Scale サーバーが稼働している各マシン上で Hyperic エージェントが実行中である必要があります。ダウンロード情報および資料のサポートは、Hyperic Web サイトを参照してください。
- `objectgrid-plugin.xml` および `hqplugin.jar` ファイルにアクセスする必要があります。これらのファイルは、`objectgridRoot/hyperic/etc` ディレクトリーにあります。

このタスクについて

eXtreme Scale を Hyperic HQ モニター・ソフトウェアと統合することで、ご使用の環境のパフォーマンスに関するメトリックをグラフィカルにモニターおよび表示することができます。この統合をセットアップするには、各エージェントでプラグインの実装を使用します。

手順

1. eXtreme Scale サーバーを始動します。 Hyperic プラグインは、eXtreme Scale を実行している Java 仮想マシンに接続するローカル・プロセスを調べます。Java 仮想マシンに適切に接続する場合、各サーバーは **-jmxServicePort** オプションを指定して開始する必要があります。 **-jmxServicePort** オプションを指定したサーバーの始動に関して詳しくは、 360 ページの『startOgServer スクリプト』を参照してください。
2. ご使用の Hyperic の構成で、`extremescale-plugin.xml` ファイルと `wxshyperic.jar` ファイルを適切なサーバーとエージェントのプラグイン・ディレクトリーに置きます。 Hyperic と統合するためには、エージェント・インストールとサーバー・インストールの両方でプラグインおよび Java アーカイブ (JAR) ファイルにアクセスする必要があります。サーバーは構成を動的にスワップできますが、いずれのエージェントを開始する場合もその前に統合を完了しておく必要があります。
   a. `extremescale-plugin.xml` ファイルを、次のロケーションにあるサーバーの `plugin` ディレクトリーに置きます。

      ```
      hyperic_home/server_home/hq-engine/server/default/deploy/hq.ear/hq-plugins
      ```

   b. `extremescale-plugin.xml` ファイルを、次のロケーションにあるエージェントの `plugin` ディレクトリーに置きます。

      ```
      agent_home/bundles/gent-4.0.2-939/pdk/plugins
      ```

   c. `wshyperic.jar` ファイルを次のロケーションにあるエージェントの `lib` ディレクトリーに置きます。

      ```
      agent_home/bundles/gent-4.0.2-939/pdk/lib
      ```

3. エージェントを構成します。 `agent.properties` ファイルは、エージェント・ランタイムの構成ポイントとして機能します。このプロパティーは、*agent_home*`/conf` ディレクトリーにあります。以下のキーはオプションですが、eXtreme Scale プラグインに対しては重要です。
   - `autoinventory.defaultScan.interval.millis=<`*time_in_milliseconds*`>`

     エージェント・ディスカバリーの間隔をミリ秒単位に設定します。
   - `log4j.logger.org.hyperic.hq.plugin.extremescale.XSServerDetector=DEBUG`

     : eXtreme Scale プラグインからの詳細のデバッグ・ステートメントを有効にします。
   - `username=<`*username*`>`: セキュリティーが有効に設定されている場合に Java Management Extensions (JMX) ユーザー名を設定します。
   - `password=<`*password*`>`: セキュリティーが有効に設定されている場合に、JMX パスワードを設定します。

- sslEnabled=<*true*|*false*>: プラグインに対して、Secure Sockets Layer (SSL) を使用するかどうかを指示します。デフォルトではこの値は false です。
- trustPath=<*path*>: SSL 接続のトラスト・パスを設定します。
- trustType=<*type*>: SSL 接続のトラスト・タイプを設定します。
- trustPass=<*password*>: SSL 接続のトラスト・パスワードを設定します。

4. エージェント・ディスカバリーを開始します。 Hyperic エージェントはディスカバリーおよびメトリック情報をサーバーに送ります。サーバーを使用してデータ・ビューをカスタマイズし、論理インベントリー・オブジェクトをグループ化して有用な情報を生成します。サーバーが使用可能になったならば、エージェントの起動スクリプトを実行するか、または Windows サービスを開始する必要があります。

   - Linux `agent_home/bin/hq-agent.sh start`
   - Windows Windows サービスを使用してエージェントを開始します。

   エージェントの始動後に、サーバーが検出されて、グループが構成されます。サーバー・コンソールにログインし、サーバーのインベントリー・データベースに追加するリソースを選択することができます。サーバー・コンソールは、デフォルトで URL: http://<*server_host_name*>:7080/ にあります。

5. Hyperic で統計データを収集するには、統計を使用可能にする必要があります。

   eXtreme Scale の Hyperic コンソールで、**SetStatsSpec** 制御アクションを使用します。リソースにナビゲートしてから、「**制御**」タブ付きページの「**制御アクション**」ドロップダウン・リストを使用し、「**制御引数**」テキスト・ボックスに ALL=enabled を設定して、SetStatsSpec 設定を指定します。

   Hyperic コンソールで設定されたフィルターによって、カタログ・サーバーは検出されません。 196 ページの『サーバー・プロパティー・ファイル』で、**statsSpec** プロパティーの情報を参照してください。これにより、コンテナーの始動時に統計が使用可能になります。統計を使用可能にする各種オプションについては、 429 ページの『管理 Bean (MBean) を使用したモニター』および 413 ページの『xsAdmin サンプル・ユーティリティーによるモニター』を参照してください。

6. Hyperic コンソールでサーバーをモニターします。 サーバーがインベントリー・モデルに追加されると、そのサービスはもはや必要なくなります。
   - **ダッシュボード・ビュー**: リソース検出イベントを調べたとき、メイン・ダッシュボード・ビューにログインしました。ダッシュボード・ビューは、カスタマイズ可能なメッセージ・センターとなる汎用ビューです。このメイン・ダッシュボードにグラフやインベントリー・オブジェクトをエクスポートすることができます。
   - **リソース・ビュー**: このページからインベントリー・モデル全体を照会して調べることができます。サービスが追加されたならば、サーバー・セクションの下で正しくラベル付けされて一緒にリストされたすべての eXtreme Scale サーバーを確認することができます。個々のサーバーをクリックすると、基本メトリックを参照できます。

7. 「リソース・ビュー」のページでサーバー全体のインベントリーを表示できます。 このページで、複数の ObjectGrid サーバーを選択し、それらをグループに

まとめることができます。リソースのセットをグループ化すると、共通のメトリックがグラフ化されて、グループ・メンバー間のオーバーレイや相違点が表示されます。オーバーレイを表示するには、ご使用のサーバー・グループの画面でメトリックを選択します。そうすると、メトリックが図表域に表示されます。すべてのグループ・メンバーのオーバーレイを表示するには、下線の付いたメトリック名をクリックします。「**ツール**」メニューを使用して、必要なグラフ、ノード・ビュー、および比較オーバーレイをメイン・ダッシュボードにエクスポートすることができます。

第 10 章 チューニングおよびパフォーマンス

運用チェックリスト

この運用チェックリストを使用して、WebSphere eXtreme Scale のデプロイ用に環境を準備してください。

表 27. 運用チェックリスト

チェックリスト項目	詳細情報

AIX を使用している場合、以下のオペレーティング・システムの設定を調整してください。
 	
TCP_KEEPINTVL
TCP_KEEPINTVL 設定は、ネットワーク障害の検出を可能にする、ソケットのキープアライブ・プロトコルの一部です。このプロパティーは、接続を検証するために送信されるパケット間の間隔を指定します。 WebSphere eXtreme Scale を指定している場合、この値は 10 に設定します。現行設定を確認するには、次のコマンドを実行します。
`# no –o tcp_keepintvl`
現行設定を変更するには、次のコマンドを実行します。
`# no –o tcp_keepintvl=10`
TCP_KEEPINTVL 設定は、0.5 秒単位で設定します。
 	
TCP_KEEPINIT
TCP_KEEPINIT 設定は、ネットワーク障害の検出を可能にする、ソケットのキープアライブ・プロトコルの一部です。このプロパティーは、TCP 接続の初期タイムアウト値を指定します。 WebSphere eXtreme Scale を使用している場合、この値は 40 に設定します。現行設定を確認するには、次のコマンドを実行します。
`# no –o tcp_keepinit`
現行設定を変更するには、次のコマンドを実行します。
`# no –o tcp_keepinit=40`
TCP_KEEPINIT 設定は、0.5 秒単位で設定します。
	• AIX のシステム調整について詳しくは、 AIX システムの調整を参照してください。

orb.properties ファイルを更新すると、グリッドのトランスポート動作を変更できます。 orb.properties ファイルは、java/jre/lib ディレクトリーにあります。	206 ページの『ORB プロパティー・ファイル』

表 27. 運用チェックリスト (続き)

チェックリスト項目	詳細情報
startOgServer スクリプトのパラメーターを使用します。特に、以下のパラメーターを使用します。 • **-jvmArgs** パラメーターを使用してヒープ設定を設定します。 • **-jvmArgs** パラメーターを使用してアプリケーション・クラスパスとプロパティーを設定します。 • エージェント・モニターの構成用に **-jvmArgs** パラメーターを設定します。 **ポートの設定** WebSphere eXtreme Scale は、一部のトランスポートの通信用にポートを開く必要があります。これらのポートはすべて動的に定義されます。ただし、コンテナー間のファイアウォールが使用中の場合はポートを指定する必要があります。ポートに関する以下の情報を使用してください。 **リスナー・ポート** プロセス間の通信に使用するポートを指定する場合、**-listenerPort** 引数を使用できます。 **コア・グループ・ポート** 障害検出に使用するポートを指定する場合、**-haManagerPort** 引数を使用できます。この引数は、peerPort と同じです。コア・グループは、ゾーンをまたいで通信する必要がないことに注意してください。したがって、ファイアウォールが単一ゾーンのすべてのメンバーに対してオープンである場合は、このポートを設定する必要はありません。 **JMX サービス・ポート** JMX サービスが使用するポートを指定する場合、**-JMXServicePort** 引数を使用できます。 **SSL ポート** `-Dcom.ibm.CSI.SSLPort=1234` を **-jvmArgs** 引数として引き渡すと、SSL ポートが `1234` に設定されます。この SSL ポートは、リスナー・ポートと対等のセキュア・ポートです。 **クライアント・ポート** カタログ・サービスのみで使用されます。この値は、**-catalogServiceEndPoints** 引数を使用して指定できます。このパラメーターの値は次の形式になります。 `serverName:hostName:clientPort:peerPort`	360 ページの『startOgServer スクリプト』
次のセキュリティー設定が正しく構成されていることを検証します。 • トランスポート (SSL) • アプリケーション (認証と許可) セキュリティー設定を検証するには、悪意のあるクライアントを使用してご使用の構成に接続を試みてください。例えば、SSL が必要な設定が構成されている場合に、TCP_IP 設定があるクライアント、あるいは、正しくないトラストストアのクライアントが、サーバーに接続できてはいけません。認証が必要な場合に、クレデンシャル (例えば、ユーザー ID とパスワードなど) を持たないクライアントは、サーバーに接続できてはいけません。許可が実行されている場合に、アクセス許可を持たないクライアントは、サーバー・リソースへのアクセスを認可されるべきではありません。	397 ページの『外部プロバイダーとのセキュリティー統合』

表 27. 運用チェックリスト (続き)

チェックリスト項目	詳細情報
ご使用の環境をモニターする方法を選択します。 • xsAdmin – カタログ・サーバーの JMX ポートが、XSAdmin ツールから表示可能になっている必要があります。コンテナー・ポートも、コンテナーからの情報を収集する一部のコマンドにとってアクセス可能である必要があります。 • 以下のベンダー・モニター・ツールから選択できます。 – Tivoli Enterprise Monitoring Agent – CA Wily Introscope – Hyperic HQ	• 413 ページの『xsAdmin サンプル・ユーティリティーによるモニター』 • 394 ページの『Java Management Extensions (JMX) セキュリティー』 • 438 ページの『IBM Tivoli Enterprise Monitoring Agent for WebSphere eXtreme Scale のモニター』 • 449 ページの『Hyperic HQ による eXtreme Scale のモニター』 • 445 ページの『CA Wily Introscope による eXtreme Scale アプリケーションのモニター』

オペレーティング・システムおよびネットワークのチューニング

ネットワークのチューニングは、接続設定の変更によって伝送制御プロトコル (TCP) スタックの遅延を減らすことができ、TCP バッファーの変更によってスループットを改善することができます。

オペレーティング・システム

チューニングが最も少なくてすむのが Windows システムで、最も必要なのが Solaris システムです。以下の説明は、指定された各システムに固有のものであり、WebSphere eXtreme Scale パフォーマンスを向上させる可能性があります。ご使用の環境でのネットワークおよびアプリケーション負荷に応じて調整を行ってください。

Windows

```
HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥
Tcpip¥Parameters
MaxFreeTcbs = dword:00011940
MaxHashTableSize = dword:00010000
MaxUserPort = dword:0000fffe
TcpTimedWaitDelay = dword:0000001e
```

Solaris

```
ndd -set /dev/tcp tcp_time_wait_interval 60000
fndd -set /dev/tcp tcp_keepalive_interval 15000
ndd -set /dev/tcp tcp_fin_wait_2_flush_interval 67500
ndd -set /dev/tcp tcp_conn_req_max_q 16384
ndd -set /dev/tcp tcp_conn_req_max_q0 16384
ndd -set /dev/tcp tcp_xmit_hiwat 400000
ndd -set /dev/tcp tcp_recv_hiwat 400000
ndd -set /dev/tcp tcp_cwnd_max 2097152
ndd -set /dev/tcp tcp_ip_abort_interval 20000
ndd -set /dev/tcp tcp_rexmit_interval_initial 4000
ndd -set /dev/tcp tcp_rexmit_interval_max 10000
ndd -set /dev/tcp tcp_rexmit_interval_min 3000
ndd -set /dev/tcp tcp_max_buf 4194304
```

AIX

```
/usr/sbin/no -o tcp_sendspace=65536
/usr/sbin/no -o tcp_recvspace=65536
/usr/sbin/no -o udp_sendspace=65536
/usr/sbin/no -o udp_recvspace=65536
/usr/sbin/no -o somaxconn=10000
/usr/sbin/no -o tcp_nodelayack=1
/usr/sbin/no –o tcp_keepinit=40
/usr/sbin/no –o tcp_keepintvl=10
```

LINUX

```
sysctl -w net.ipv4.tcp_timestamps=0
sysctl -w net.ipv4.tcp_tw_reuse=1
sysctl -w net.ipv4.tcp_tw_recycle=1
sysctl -w net.ipv4.tcp_fin_timeout=30
sysctl -w net.ipv4.tcp_keepalive_time=1800
sysctl -w net.ipv4.tcp_rmem="4096 87380 8388608"
sysctl -w net.ipv4.tcp_wmem="4096 87380 8388608"
sysctl -w net.ipv4.tcp_max_syn_backlog=4096
```

HP-UX

```
ndd -set /dev/tcp tcp_ip_abort_cinterval 20000
```

ネットワーク・ポートの計画

WebSphere eXtreme Scale は、分散キャッシュであり、オブジェクト・リクエスト・ブローカー (ORB) および伝送制御プロトコル (TCP) スタックを使用して Java 仮想マシンと他のマシン間で通信するためにオープン・ポートを必要とします。特に、カタログ・サービスやコンテナーを複数ポートで使用している場合などのファイアウォール環境では、ポートを計画し、制御することが必要です。

カタログ・サービス・ドメイン

カタログ・サービス・ドメインでは、以下のポートを定義する必要があります。

peerPort
高可用性 (HA) マネージャーがピア・カタログ・サーバー間で TCP スタックを介して通信するためのポートを指定します。

clientPort
カタログ・サーバーがカタログ・サービス・データにアクセスするためのポートを指定します。

JMXServicePort
Java Management Extensions (JMX) サービスが使用することになるポートを指定します。

listenerPort
コンテナーおよびクライアントが ORB を介してカタログ・サービスと通信するための ORB リスナー・ポートを定義します。

上記のポートを定義する方法は、スタンドアロン・モードを使用しているか、あるいは eXtreme Scale カタログ・サーバーを WebSphere Application Server 環境で開始しているかによって異なります。

- **スタンドアロン・モードの場合:**

  上記のポートを指定するには、以下の例のように、スタンドアロン・モードでオプションを指定した startOgServer コマンドを使用します。

  ```
  –catalogServiceEndPoints cs1:host1:clientPort:peerPort, cs2:host2:clientPort:peerPort,
  cs3:host3:clientPort:peerPort -listenerPort <orbPort> -JMXServicePort <jmxPort>
  ```

  スタンドアロン・モードでのカタログ・サービスの開始について詳しくは、354 ページの『スタンドアロン環境でのカタログ・サービスの開始』を参照してください。

- **WebSphere Application Server 環境の場合:**

カタログ・サービス・ドメインは管理コンソールで定義できます。詳しくは、371 ページの『WebSphere Application Server でのカタログ・サービス・ドメインの作成』を参照してください。

コンテナー・サーバー

WebSphere eXtreme Scale コンテナー・サーバーも、いくつかのポートが作動することを必要とします。デフォルトでは、eXtreme Scale コンテナー・サーバーは、HA マネージャー・ポートおよび ORB リスナー・ポートを、動的ポートと共に自動的に生成します。ファイアウォール環境では、ポートを計画し、制御することはユーザーにとって有益なので、eXtreme Scale コンテナー・サーバーを開始する際のオプションが用意されています。このコンテナー・サーバーは、以下の例に示すように、startOgServer コマンド中のオプションで HAManager ポートおよび ORB リスナー・ポートを指定して開始することができます。

```
-HaManagerPort <peerPort>
-listenerPort <orbPort>
```

ポート制御の適切な計画は有益ですが、数百の Java 仮想マシンを 1 台のマシンで開始する場合は、これらのポートの計画と管理には特有の難しさがあります。ポート競合があると、サーバーの始動が失敗します。

セキュリティーが有効になっている場合、上記のリストに示されたポートに加えて、Secure Socket Layer (SSL) ポートが必要です。
`Dcom.ibm.CSI.SSLPort=<sslPort>` を **-jvmArgs** 引数として使用すると、SSL ポートが `<sslPort>` に設定されます。eXtreme Scale を使用したポートの計画については、セキュリティー設定を参照してください。

ORB プロパティーおよびファイル記述子の設定

チューニング考慮事項には、Object Request Broker (ORB) プロパティーおよびファイル記述子の設定などが含まれます。

ORB プロパティー

ORB は WebSphere eXtreme Scale によって TCP スタックでの通信のために使用されます。必要な `orb.properties` ファイルは、`java/jre/lib` ディレクトリーにあります。大容量オブジェクトの重い負荷に備えるため、以下の設定値を指定して ORB フラグメント化を有効にしてください。

```
com.ibm.CORBA.FragmentSize=<right size>
```

スレッド・プールの増大を防止するため、次のように設定してください。

```
com.ibm.CORBA.ThreadPool.IsGrowable=false
```

異常なシチュエーションでの過剰スレッドを回避するため、以下を設定して適切なタイムアウトを設定してください。

- com.ibm.CORBA.RequestTimeout
- com.ibm.CORBA.ConnectTimeout
- com.ibm.CORBA.FragmentTimeout

ファイル記述子

UNIX システムおよび Linux システムでは、プロセス当たりに許容されるオープン・ファイルの数に制限があります。オペレーティング・システムが、許容されるオープン・ファイルの数を指定します。この値が小さすぎる場合、AIX ではメモリー割り振りエラーが発生し、多すぎるオープン・ファイルはログに記録されます。

UNIX システム端末ウィンドウで、この値をデフォルトのシステム値よりも大きく設定してください。クローンを持つ大容量 SMP マシンの場合、無限に設定してください。

AIX 構成では、コマンド `ulimit -n -1` を使用して、この値を `-1` (無限) に設定してください。

Solaris 構成の場合、コマンド `ulimit -n 16384` を使用して、この値を `16384` に設定してください。

コマンド `ulimit –a` を使用すれば、現行値を表示できます。

ORB を使用したノンブロッキング I/O (NIO)

現在、WebSphere eXtreme Scale スタンドアロン・シナリオは NIO または ChannelFramework で実行できます。 IBM ORB のノンブロッキング I/O (NIO) で実行するには、IBM ORB の TransportMode を ChannelFramework に設定する必要があります。デフォルトでは、IBM ORB は Pluggable モードで実行されます。WebSphere eXtreme Scale にはサーバー・プロパティーがあり、TransportMode を ChannelFramework に設定します。

重要:

WebSphere Application Server は、現行リリースでは Pluggable モードのみをサポートします。WebSphere eXtreme Scale が WebSphere Application Server に統合して実行される場合は、Pluggable モードに従う必要があります。 WebSphere eXtreme Scale は WebSphere Application Server SSL/Transport Security も使用するため、現在は Pluggable 限定モードもサポートします。

NIO を使用する場合

ここは、概念を示す小セクションです。

関連タスク

210 ページの『ORB での NIO または ChannelFramework の使用可能化』
WebSphere eXtreme Scale は、ORB でノンブロッキング I/O (NIO) を使用可能にするために、TransportMode を ChannelFramework に設定するサーバー・プロパティーを用意しています。

ORB での NIO または ChannelFramework の使用可能化

WebSphere eXtreme Scale は、ORB でノンブロッキング I/O (NIO) を使用可能にするために、TransportMode を ChannelFramework に設定するサーバー・プロパティーを用意しています。

始める前に

既存のサーバー・プロパティー・ファイルを検索するか、サーバー・プロパティー・ファイルを作成します。カタログ・サービスおよびコンテナー・サーバーで ChannelFramework を使用可能にすることができます。詳しくは、サーバー・プロパティー・ファイルを参照してください。

このタスクについて

現在、WebSphere eXtreme Scale スタンドアロン・シナリオで NIO または ChannelFramework で実行できます。 IBM ORB でノンブロッキング I/O (NIO) を使用して実行するには、 IBM ORB の TransportMode を ChannelFramework に設定する必要があります。デフォルトでは、IBM ORB は Pluggable モードで実行されます。 WebSphere eXtreme Scale にはサーバー・プロパティーがあり、TransportMode を ChannelFramework に設定します。

重要:

WebSphere Application Server の現行リリースでは、Pluggable モードのみがサポートされます。 WebSphere eXtreme Scale は WebSphere Application Server と統合して実行する場合には、Pluggable モードに従う必要があります。 WebSphere eXtreme Scale は WebSphere Application Server SSL/Transport Security も使用するため、現在は Pluggable 限定モードもサポートします。

手順

1. enableChannelFramework=true プロパティーをサーバー・プロパティー・ファイルに追加します。
2. サーバー・プロパティー・ファイルが ORB プロパティー・ファイルと矛盾していないか、確認します。

   サーバー・プロパティー・ファイルで ChannelFramework TransportMode を使用可能にしたのにもかかわらず、 orb.properties ファイルで TransportMode を Pluggable に設定している場合には、サーバーは orb.properties の設定をオーバーライドしません。 2 つの設定が存在するという警告メッセージがログに表示されます。 enableChannelFramework=true プロパティーを有効にするには、TransportMode を Pluggable に設定することを指示するプロパティーを調整します (com.ibm.CORBA.TransportMode=Pluggable を ChannelFramework に変更するか、プロパティーを削除します)。
3. カタログ・サービスまたはコンテナー・サーバーの始動時に更新したサーバー・プロパティー・ファイルを提供します。 サーバー・プロパティー・ファイルを使用したサーバーの始動の詳細については、サーバー・プロパティー・ファイルを参照してください。

タスクの結果

カタログ・サービスまたはコンテナー・サーバーで channelFramework TransportMode を使用する場合には、以下のメッセージがログに出力されます。

```
CWOBJ0052I: The IBM ORB TransportMode property was set to ChannelFramework
(IBM ORB TransportMode プロパティーが ChannelFramework に設定されました)
```

以下のメッセージがログに表示された場合には、前述のとおり ORB プロパティーを検査してください。

```
CWOBJ0055W: The IBM ORB TransportMode property was set to ChannelFramework
in the server properties file, but the existing orb.properties file already
had a TransportMode set. The TransportMode will not be overridden.
(サーバー・プロパティー・ファイルで IBM ORB TransportMode プロパティーが
ChannelFramework に設定されましたが、既存の orb.properties ファイルには既に
TransportMode が設定されています。TransportMode はオーバーライドされません。)
```

なお、ChannelFramework を使用可能にすると、ServerSocketQueueDepth の最大値は 512 になります。 orb.properties の ServerSocketQueueDepth 設定が 512 より大きい場合は、サーバーは orb.properties の ServerSocketQueueDepth を 512 に自動的に設定して、情報メッセージをログに出力することでユーザーに通知します。アクションは不要です。

```
CWOBJ0053I: The IBM ORB ServerSocketQueueDepth property was set to 512 to run
with correctly with the ChannelFramework TransportMode.
(ChannelFramework TransportMode で正常に実行されるように、
IBM ORB の ServerSocketQueueDepth が 512 に設定されました。)
```

WebSphere eXtreme Scale 用の JVM の調整

Java 仮想マシン (JVM) を調整すると、WebSphere eXtreme Scale のデプロイメントをかなり改善することができます。

ほとんどの場合、WebSphere eXtreme Scale は、特殊な JVM 設定をほんとどまたはまったく必要としません。WebSphere eXtreme Scale に保管されるオブジェクトが多数ある場合は、ヒープ・サイズを適切なレベルに調整して、メモリー不足の状態で実行されるのを避けます。

プラットフォーム

パフォーマンス・テストは、まず AIX (32 way)、Linux (4 way)、および Windows (8 way) のコンピューターで実行されました。 ハイエンド AIX のコンピューターでは、マルチスレッドのシナリオを大量にプッシュして、競合ポイントを特定して修正することができます。

オブジェクト・リクエスト・ブローカー (ORB) 要件

IBM SDK には、WebSphere Application Server および WebSphere eXtreme Scale を使用してテスト済みの IBM ORB 実装が組み込まれています。サポート・プロセスを簡単にするため、IBM 提供の JVM を使用してください。他の JVM 実装では、異なる ORB が使用されます。IBM 提供の Java 仮想マシンと共にすぐに使用可能なのは、IBM ORB のみです。WebSphere eXtreme Scale には、操作する作業 ORB が必要です。WebSphere eXtreme Scale はサード・パーティーの ORB とともに使用できますが、ORB に関する問題が特定されている場合は、ORB ベンダーのサポートに連絡する必要があります。IBM ORB 実装は、サード・パーティーの Java 仮想マシンと互換性があり、必要な場合は置換できます。

ガーベッジ・コレクション

WebSphere eXtreme Scale は、要求や応答など各トランザクション、およびログ・シーケンスに関連した一時オブジェクトを作成します。これらのオブジェクトはガーベッジ・コレクションの効率に影響するため、ガーベッジ・コレクションのチューニングは重要です。

IBM の Java 仮想マシンの場合、更新率が高いシナリオ (トランザクションの 100% がエントリーを変更する) には optavgpause コレクターを使用してください。データがほとんど更新されない (10% 以下の頻度) ようなシナリオでは、optavgpause コレクターより gencon コレクターの方がより適切に機能します。両方のコレクターを使用して実験を行い、シナリオで最も適切に機能するコレクターを確認します。パフォーマンス上の問題を確認できる場合は、詳細ガーベッジ・コレクションをオンにして実行し、ガーベッジの収集に費やされている時間の割合を確認します。調整で問題が修正されるまでガーベッジ・コレクションで時間の 80% が費やされたシナリオが発生しました。

ガーベッジ・コレクションの構成について詳しくは、IBM の Java 仮想マシンの調整を参照してください。

JVM パフォーマンス

WebSphere eXtreme Scale は、異なるバージョンの Java 2 Platform, Standard Edition (J2SE) 上で実行できます。WebSphere eXtreme Scale バージョン 6.1 は、J2SE バージョン 1.4.2 以降をサポートしています。開発者の生産性およびパフォーマンスを向上させるためには、J2SE 5 またはそれ以降を使用して、アノテーションおよび改良されたガーベッジ・コレクションを活用してください。WebSphere eXtreme Scale は、32 ビットまたは 64 ビット版の Java 仮想マシンで動作します。

WebSphere eXtreme Scale がテストされたのは、使用可能な仮想マシンの一部ですが、サポートのリストは排他的なものではありません。バージョン 1.4.2 以上の任意のバージョンで WebSphere eXtreme Scale を実行できますが、JVM に関する問題が特定されている場合は、JVM ベンダーにサポートを依頼する必要があります。可能であれば、 WebSphere Application Server がサポートするどのプラットフォーム上でも、WebSphere ランタイムの JVM を使用してください。

WebSphere eXtreme Scale が使用されるほとんどのシナリオでは、JVM の Java Platform Standard Edition 6 のほうが、Edition 5 または 1.4 よりうまく機能します。Java 2 Platform, Standard Edition Version 1.4 は、gencon コレクターを使用するシナリオの場合は特に、パフォーマンスが悪くなります。Java Platform Standard Edition 5 は良好に動作しますが、Java Platform, Standard Edition 6 のほうが優れています。

大規模なヒープ

アプリケーションにより区画ごとに大量のデータを管理する必要がある場合は、ガーベッジ・コレクションが要因になっている可能性があります。ある世代のコレクターが使用されている場合は、非常に大きなヒープ (20 GB 以上) が存在しても読み取りシナリオの大部分は正常に機能します。ただし、保有ヒープがいっぱいになった後は、コンピューターで使用可能なヒープ・サイズおよびプロセッサー数に比

例して一時停止が発生します。この一時停止は、大きいヒープを持つ小さいコンピューターで大きくなる可能性があります。

WebSphere eXtreme Scale は、WebSphere Real Time Java をサポートします。WebSphere Real Time Java を一緒に使用することによって、WebSphere eXtreme Scale のトランザクション処理応答はより一貫性のある、予測可能なものになり、ガーベッジ・コレクションおよびスレッド・スケジューリングの影響は大幅に小さくなります。応答時間の標準偏差が標準 Java の 10% よりも小さくなる程度まで、影響は少なくなります。

詳しくは、 469 ページの『WebSphere Real Time の使用』を参照してください。

スレッド数

スレッド数はいくつかの要因に依存します。単一の断片が管理できるスレッド数には制限があります。断片とは区画のインスタンスであり、プライマリーまたはレプリカとすることができます。JVM ごとの断片数が多いほど、それぞれ追加断片を持つスレッドが増えるので、データへの並行パスが多くなります。各断片は可能な限り並行ですが、それでも並行性について制限はあります。

JVM の調整

WebSphere eXtreme Scale の最善のパフォーマンスを得るために、Java 仮想マシン (JVM) 調整の特定の局面をいくつか考慮してください。 .

4 コア当たり 1 JVM で 1 から 2Gb のヒープをお勧めします。ヒープ・サイズは、この資料の後で説明する、サーバーに保管されるオブジェクトの特性に依存します。

ヒープ・サイズおよびガーベッジ・コレクションの推奨事項

最適なヒープ・サイズ値は、次の 3 つの要因に基づきます。

1. ヒープ内のライブ・オブジェクトの数。
2. ヒープ内のライブ・オブジェクトの複雑さ。
3. JVM 用に使用可能なコアの数。

例えば、10K バイトの配列を保管するアプリケーションは、POJO の複雑なグラフを使用するアプリケーションよりもずっと大きなヒープを実行できます。

最近の JVM はすべてパラレル・ガーベッジ・コレクションのアルゴリズムを使用していますが、これは、より多くのコアを使用するとガーベッジ・コレクション中の停止を削減できることを意味します。したがって、8 コアのコンピューターのほうが、4 コアのものよりも速く収集されます。

実メモリー使用量とヒープ仕様

1Gb ヒープ JVM は、約 1.3Gb の実メモリーを使用します。テストでは、16Gb の RAM のコンピューターでは、10 個の 1Gb JVM を実行できませんでした。 JVM ヒープが 800 MB を超えると、ページングが始まります。

ガーベッジ・コレクション

IBM JVM の場合、更新率が高いシナリオ (トランザクションの 100% がエントリーを変更する) には avgoptpause コレクターを使用してください。データがほとんど更新されない (10% 以下の頻度) ようなシナリオでは、avgoptpause コレクターより gencon コレクターの方がより適切に機能します。両方のコレクターを使用して実験を行い、シナリオで最も適切に機能するコレクターを確認します。パフォーマンス上の問題を確認できる場合は、詳細ガーベッジ・コレクションをオンにして実行し、ガーベッジの収集に費やされている時間の割合を確認します。調整で問題が修正されるまでガーベッジ・コレクションで時間の 80% が費やされたシナリオが発生しました。

JVM パフォーマンス

WebSphere eXtreme Scale は、異なるバージョンの Java 2 Platform, Standard Edition (J2SE) 上で実行できます。ObjectGrid バージョン 6.1 は J2SE バージョン 1.4.2 以降をサポートしています。開発者の生産性およびパフォーマンスを向上させるためには、J2SE 5 またはそれ以降を使用して、アノテーションおよび改良されたガーベッジ・コレクションを活用してください。ObjectGrid は、32 ビットまたは 64 ビット JVM 上で動作します。

ObjectGrid バージョン 6.0.2 クライアントは、ObjectGrid バージョン 6.1 グリッドに接続できます。J2SE バージョン 1.4.2 または良好なクライアントに対しては、ObjectGrid バージョン 6.1 クライアントを使用します。ObjectGrid バージョン 6.0.2 クライアントを使用する唯一の理由は、J2SE バージョン 1.3 サポート用であるためです。

WebSphere eXtreme Scale がテストされたのは、使用可能な仮想マシンの一部ですが、サポートのリストは排他的なものではありません。バージョン 1.4.2 以上の任意のバージョンで WebSphere eXtreme Scale を実行できますが、JVM に関する問題が特定されている場合は、JVM ベンダーにサポートを依頼する必要があります。可能であれば、 WebSphere Application Server がサポートするどのプラットフォーム上でも、WebSphere ランタイムの JVM を使用してください。

最良の JVM は Java Platform, Standard Edition 6 です。Java 2 Platform, Standard Edition v 1.4 は、gencon コレクターが大きく影響するようなシナリオの場合は特に、パフォーマンスが悪くなります。Java Platform Standard Edition 5 は良好に動作しますが、Java Platform, Standard Edition 6 のほうが優れています。

orb.properties の調整

実動には以下の orb.properties ファイルを使用することをお勧めします。このファイルは、1500 JVM までのグリッドでの使用についてテスト済みです。orb.properties ファイルは、使用される JRE の lib フォルダー内にあります。

```
# IBM JDK properties for ORB
org.omg.CORBA.ORBClass=com.ibm.CORBA.iiop.ORB
org.omg.CORBA.ORBSingletonClass=com.ibm.rmi.corba.ORBSingleton

# WS Interceptors
org.omg.PortableInterceptor.ORBInitializerClass.com.ibm.ws.objectgrid.corba.ObjectGridInitializer

# WS ORB & Plugins properties
com.ibm.CORBA.ForceTunnel=never
com.ibm.CORBA.RequestTimeout=10
com.ibm.CORBA.ConnectTimeout=10
```

```
# Needed when lots of JVMs connect to the catalog at the same time
com.ibm.CORBA.ServerSocketQueueDepth=2048

# Clients and the catalog server can have sockets open to all JVMs
com.ibm.CORBA.MaxOpenConnections=1016

# Thread Pool for handling incoming requests, 200 threads here
com.ibm.CORBA.ThreadPool.IsGrowable=false
com.ibm.CORBA.ThreadPool.MaximumSize=200
com.ibm.CORBA.ThreadPool.MinimumSize=200
com.ibm.CORBA.ThreadPool.InactivityTimeout=180000

# No splitting up large requests/responses in to smaller chunks
com.ibm.CORBA.FragmentSize=0
```

スレッド数

スレッド数はいくつかの要因に依存します。単一の断片が管理できるスレッド数には制限があります。JVM ごとの断片数が多いほど、より多くのスレッドおよび並行性が存在できます。追加の各断片により、データへの並行性パスが提供されます。各断片はできるだけ並行ですが、それでも制限はあります。

フェイルオーバー検出の構成

ハートビート間隔設定で、障害の起きたサーバーがないかを調べるシステム・チェックの間の時間を構成できます。

このタスクについて

フェイルオーバーの構成は、使用している環境のタイプによって異なります。スタンドアロン環境を使用している場合は、コマンド行でフェイルオーバーを構成できます。 WebSphere Application Server Network Deployment 環境を使用している場合は、WebSphere Application Server Network Deployment 管理コンソールでフェイルオーバーを構成する必要があります。

手順

- スタンドアロン環境のフェイルオーバーを構成します。

  ハートビート間隔は、startOgServer.bat | startOgServer.sh スクリプト・ファイルに **-heartbeat** パラメーターを使用してコマンド行で構成できます。このパラメーターは以下のいずれかの値に設定します。

  表 28. ハートビート間隔

値	アクション	説明
0	標準 (デフォルト)	通常、30 秒以内にフェイルオーバーが検出されます。
-1	高速	通常、5 秒以内にフェイルオーバーが検出されます。
1	低速	通常、180 秒以内にフェイルオーバーが検出されます。

  高速のハートビート間隔は、プロセスおよびネットワークが安定している場合に役立ちます。ネットワークまたはプロセスが最適に構成されていないと、ハートビートを見逃す可能性があり、そうなった場合は誤って障害検出が示されることがあります。

- WebSphere Application Server 環境のフェイルオーバーを構成します。

  WebSphere Application Server Network Deployment バージョン 6.0.2 以降は、WebSphere eXtreme Scale のフェイルオーバーを高速で行えるように構成できま

す。ハード障害の場合のデフォルトのフェイルオーバー時間は、約 200 秒です。ハード障害は、物理的なコンピューターまたはサーバーの破損、ネットワーク・ケーブルの切断、オペレーティング・システム・エラーのことです。プロセスの異常終了やソフト障害による障害は、一般的に 1 秒未満でフェイルオーバーされます。ソフト障害の障害検出は、デッド・プロセスのネットワーク・ソケットがそのプロセスをホスティングするサーバーのオペレーティング・システムにより自動的にクローズされるときに発生します。

コア・グループのハートビート構成

WebSphere Application Server プロセスで実行されているWebSphere eXtreme Scale は、アプリケーション・サーバーのコア・グループ設定のフェイルオーバー特性を継承します。以下のセクションでは、以下のようなさまざまなバージョンの WebSphere Application Server Network Deployment のコア・グループ・ハートビート設定を構成する方法について説明します。

– **WebSphere Application Server Network Deployment バージョン 6.x または 7.x のコア・グループ設定を更新します。**

  ハートビート間隔は、WebSphere Application Server のバージョン 6.0 からバージョン 6.1.0.12 までは秒単位、バージョン 6.1.0.13 からはミリ秒単位で指定します。また、欠落ハートビートの数も指定する必要があります。この値は、ピア Java 仮想マシン (JVM) に障害が起きたと見なされるまでに、容認される欠落ハートビートの数を示します。ハード障害の検出時間は、ほぼハートビート間隔と欠落ハートビート数の積です。

  これらのプロパティーは、WebSphere 管理コンソールで、コア・グループに対してカスタム・プロパティーを使用して指定します。構成について詳しくは、コア・グループ・カスタム・プロパティーを参照してください。アプリケーションによって使用されるすべてのコア・グループに対して、以下のプロパティーを指定する必要があります。

  - ハートビート間隔は、IBM_CS_FD_PERIOD_SEC カスタム・プロパティー (秒単位) または IBM_CS_FD_PERIOD_MILLIS カスタム・プロパティー (ミリ秒単位、V6.1.0.13 以降が必要) を使用して指定します。
  - 欠落ハートビート数は、IBM_CS_FD_CONSECUTIVE_MISSED カスタム・プロパティーを使用して指定します。

  IBM_CS_FD_PERIOD_SEC プロパティーのデフォルト値は 20 で、IBM_CS_FD_CONSECUTIVE_MISSED プロパティーのデフォルト値は 10 です。IBM_CS_FD_PERIOD_MILLIS プロパティーを指定すると、設定されている IBM_CS_FD_PERIOD_SEC カスタム・プロパティーがオーバーライドされます。これらのプロパティーの値は、正の整数値です。

  WebSphere Application Server Network Deployment 6.x サーバーで 1500 ミリ秒の障害検出時間を実現するには、以下の設定を使用します。

  - `IBM_CS_FD_PERIOD_MILLIS = 750` を設定 (WebSphere Application Server Network Deployment バージョン 6.1.0.13 以降)
  - `IBM_CS_FD_CONSECUTIVE_MISSED = 2` を設定

– **WebSphere Application Server Network Deployment バージョン 7.0 でのコア・グループ設定を更新します。**

バージョン 7.0 の WebSphere Application Server Network Deployment は、フェイルオーバー検出を増減するために調整できる以下の 2 つのコア・グループ設定を提供します。

- **ハートビート伝送期間。** デフォルト値は 30000 ミリ秒です。
- **ハートビート・タイムアウト期間。** デフォルト値は 180000 ミリ秒です。

これらの設定を変更する方法について詳しくは、WebSphere Application Server Network Deployment インフォメーション・センター: ディスカバリーおよび障害検出の設定を参照してください。

WebSphere Application Server Network Deployment バージョン 7 サーバーで 1500 ミリ秒の障害検出時間を実現するには、以下の設定を使用します。

- ハートビート伝送期間を 750 ミリ秒に設定します。
- ハートビート・タイムアウト期間を 1500 ミリ秒に設定します。

次のタスク

短いフェイルオーバー時間を指定するようにこれらの設定を変更すると、注意すべきシステム調整上の問題が生じます。まず Java はリアルタイム環境ではありません。JVM に長期のガーベッジ・コレクション時間が発生すると、スレッドが遅延する可能性があります。JVM をホスティングするマシンの負荷が大きくなった (JVM 自身またはマシンで実行中の他のプロセスが原因) 場合にも、スレッドが遅延する可能性があります。スレッドが遅延された場合、ハートビートが正確な時間で送信されない可能性があります。最悪の場合、必要なフェイルオーバー時間で遅延が生じる可能性があります。 スレッドが遅延すると、誤障害検出が発生します。実動環境で誤障害検出が発生しないように、システムを調整し、サイズ設定する必要があります。これを確実にするには、適切な負荷テストが最善の策です。

注: eXtreme Scale の現行バージョンは、WebSphere Real Time をサポートします。

フェイルオーバー検出のタイプ

WebSphere eXtreme Scale は、障害を確実に検出できます。

ハートビート処理

1. Java 仮想マシン間ではソケットがオープン状態のままなので、ソケットが予想外に閉じると、この予想外のクローズはピア Java 仮想マシンの障害として検出されます。この検出機能は、Java 仮想マシンが極端に早く終了したなどの障害事例をキャッチします。また、この検出機能により、通常 1 秒足らずで、こうしたタイプの障害から回復できます。
2. その他のタイプの障害には、オペレーティング・システム・パニック、物理サーバー障害、ネットワーク障害などがあります。こうした障害は、ハートビート処理を使用して検出します。

プロセスのペア間では定期的にハートビートが送信されます。一定数のハートビートが欠落すると、障害とみなされます。この方法は、N*M 秒で障害を検出します。N は欠落ハートビートの数で、M はハートビートの設定間隔です。直接に M と N

を指定することはサポートされておらず、代わりにスライダー・メカニズムを使用してテストされる一定範囲の M と N の組み合わせを指定できるようになっています。

障害

プロセスに障害が起きるのには、いくつかの場合があります。何らかのリソース限界に達したり (例えば、最大ヒープ・サイズ)、プロセス制御ロジックがプロセスを強制終了したといった理由で、プロセスに障害が起こることがあります。オペレーティング・システムに障害が起きると、システム上で実行中のすべてのプロセスが失われます。ネットワーク・インターフェース・カード (NIC) などの頻繁には障害が起きないハードウェアに障害が起きると、オペレーティング・システムがネットワークから切断されます。このような状況において、これらすべての障害は、プロセス障害または接続不良の 2 つの障害タイプのうちのいずれかに分類されます。

プロセス障害

WebSphere eXtreme Scale は、プロセス障害に非常に迅速に反応します。プロセスに障害が起きると、オペレーティング・システムは、プロセスが使用していたリソースの残りすべてをクリーンアップする必要が生じます。このクリーンアップには、ポートの割り当ておよび接続が含まれています。プロセスに障害が起きると、信号はそのプロセスによって使用されていた接続を通して即時に送信され、各接続がクローズされます。

接続不良

オペレーティング・システムが切断されると、接続不良が発生します。その結果として、オペレーティング・システムは信号を他のプロセスに送信することができなくなります。接続不良が発生する理由はいくつかありますが、それらの理由は、ホスト障害と孤立化の 2 つのカテゴリーに分割されます。

ホスト障害

ホスト・マシンの電源が切断されると、ホスト・マシンは瞬時に使用不可になります。

孤立化

このシナリオは、使用不可であると見なされたプロセスが実際には使用不可ではないため、ソフトウェアが正しく対処することが最も難しい障害の状態です。

コンテナー障害

コンテナー障害は、通常コア・グループ・メカニズムを通してピア・コンテナーによって発見されます。コンテナーまたはコンテナーのセットに障害が起きると、カタログ・サービスにより、そのコンテナーにホスティングされていた断片が移行されます。カタログ・サービスにより、非同期のレプリカに移行する前に最初に同期複製が検索されます。プライマリー断片が新規ホスト・コンテナーに移行された後で、カタログ・サービスにより、欠落しているレプリカの新規ホスト・コンテナーが検索されます。

注: コンテナー孤立化 - コンテナーが使用不可であることが検出されると、カタログ・サービスによりコンテナーの断片がコンテナーから移行されます。これらのコンテナーが使用可能になると、カタログ・サービスは、通常の開始フローの場合と同じように、これらのコンテナーを配置可能とみなします。

コンテナー・フェイルオーバー検出までの待ち時間

障害は、ソフトとハードの障害に分類されます。ソフト障害の原因は、一般にプロセスの障害です。そのような障害はオペレーティング・システムによって検出されます。オペレーティング・システムでは、ネットワーク・ソケットなどの使用されたリソースを非常に迅速にリカバリーできます。ソフト障害の場合、標準的な障害検出までの時間は、1 秒未満です。ハード障害の場合、デフォルトのハートビート調整を使用すると、検出まで最長 200 秒かかることもあります。そのような障害は、物理的なマシンの破損、ネットワーク・ケーブル切断、オペレーティング・システム障害などです。したがって、eXtreme Scale がハード障害を検出するには、構成可能なハートビートに頼らざるを得ません。

複数のコンテナー障害

プロセスが失われるとプライマリーおよびレプリカの両方が失われるため、レプリカはプライマリーとして同じプロセスに配置するべきではありません。デプロイメント・ポリシーにより、カタログ・サービスがレプリカをプライマリーとして同じマシンに配置できるかどうかを判別するのに使用する属性が定義されます。単一マシンの開発環境においては、2 つのコンテナーを所有でき、それらの間でレプリカを生成できます。しかし、実動環境では、そのホストを失うことにより両方のコンテナーを失うことになるため、単一マシンの使用では不十分です。単一マシンでの開発モードと複数のマシンを使用する実動モード間でモードを変更するには、デプロイメント・ポリシー構成ファイルで開発モードを無効にします。

カタログ・サービス障害

カタログ・サービス・グリッドは、eXtreme Scale グリッドであるため、これもコンテナー障害プロセスと同じ方法でコア・グループ化のメカニズムを使用します。主な相違点は、カタログ・サービス・ドメインでは、コンテナーに使用するカタログ・サービス・アルゴリズムの代わりに、プライマリー断片の定義にピア選択プロセスを使用する点です。

配置サービスおよびコア・グループ化サービスは N 個の中の 1 つのサービスであることに注意してください。 N 個の中の 1 つのサービスは、高可用性グループのメンバーの 1 つで実行されます。 ロケーション・サービスおよび管理は、高可用性グループのすべてのメンバーで実行されています。配置サービスおよびコア・グループ化サービスは、システムをレイアウトする必要があるため別のものです。ロケーション・サービスおよび管理は読み取り専用サービスであり、スケーラビリティーを提供するためにあらゆる場所に存在します。

カタログ・サービスは複製を使用して、独自の障害限界を設定します。カタログ・サービス・プロセスに障害が起きると、サービスが再始動され、システムを必要なレベルの可用性に復元します。カタログ・サービスをホスティングしているすべてのプロセスで障害が起こると、eXtreme Scale から重要なデータが失われます。この障害により、すべてのコンテナーの再始動が必要になります。カタログ・サービス

は多くのプロセスで実行されているため、この障害は起きる可能性のないイベントです。ただし、単一のボックスですべてのプロセスを実行している場合は、単一のブレード・シャーシ内、または単一のネットワーク・スイッチで、障害が起きる可能性があります。カタログ・サービスをホスティングしているボックスから共通の障害モードを除去して、障害が起きる可能性を減らします。

表 29. 障害のディスカバリーおよび復旧の要約

ロス・タイプ	ディスカバリー (検出) メカニズム	復旧メソッド
プロセス・ロス	入出力	再始動
サーバー・ロス	ハートビート	再始動
ネットワーク障害	ハートビート	ネットワークおよび接続の再確立
サーバー・サイド・ハング	ハートビート	サーバーの停止および再始動
サーバー・ビジー	ハートビート	サーバーが使用可能になるまで待機

WebSphere Real Time の使用

WebSphere eXtreme Scale を WebSphere Real Time と一緒に使用すると、標準 IBM Java™ SE Runtime Environment (JRE) で採用されているデフォルトのガーベッジ・コレクション・ポリシーと比べてパフォーマンス・スループットは犠牲にしますが、一貫性および予測可能性は高まります。 費用対効果の提案が変わる可能性があります。 WebSphere eXtreme Scale は、各トランザクションに関連付けられる多数の一時オブジェクトを作成します。これらの一時オブジェクトは、要求、応答、ログ・シーケンス、およびセッションを処理します。 WebSphere Real Time がない場合は、トランザクションの応答時間が数百ミリ秒まで増大することがあります。しかし、WebSphere eXtreme Scale のもとで WebSphere Real Time を使用すると、ガーベッジ・コレクションの効率が上がり、応答時間がスタンドアロン構成応答時間の 10% に減少します。

スタンドアロン環境の WebSphere Real Time

WebSphere eXtreme Scale のもとで WebSphere Real Time を使用することができます。 WebSphere Real Time を使用可能にすることにより、ガーベッジ・コレクションはより予測可能になり、スタンドアロン eXtreme Scale 環境でのトランザクションの応答時間とスループットは安定した一貫性のあるものになります。

WebSphere Real Time の利点

WebSphere eXtreme Scale は、各トランザクションに関連付けられる多数の一時オブジェクトを作成します。これらの一時オブジェクトは、要求、応答、ログ・シーケンス、およびセッションを処理します。 WebSphere Real Time がない場合は、トランザクションの応答時間が数百ミリ秒まで増大することがあります。しかし、WebSphere eXtreme Scale のもとで WebSphere Real Time を使用すると、ガーベッジ・コレクションの効率が上がり、応答時間がスタンドアロン構成応答時間の 10% に減少します。

WebSphere Real Time の使用可能化

Install eXtreme Scale を実行するコンピューターに、WebSphere Real Time とスタンドアロン WebSphere eXtreme Scale をインストールしてください。 JAVA_HOME 環境変数が標準 Java SE Runtime Environment (JRE) を指すように設定してください。

インストールされた WebSphere Real Time をポイントするよう、JAVA_HOME 環境変数を設定します。その後、次のようにして WebSphere Real Time を使用可能にします。

1. 次の行からコメントを除去することによって、スタンドアロン・インストール `objectgridRoot/bin/setupCmdLine.sh | .bat` ファイルを編集します。

   ```
   WXS_REAL_TIME_JAVA="-Xrealtime -Xgcpolicy:metronome
   -Xgc:targetUtilization=80"
   ```

2. ファイルを保存します。

これで、WebSphere Real Time が使用可能になります。WebSphere Real Time を使用不可にしたい場合は、同じ行にコメントを戻します。

ベスト・プラクティス

WebSphere Real Time によって、eXtreme Scale トランザクションの応答時間はより予測可能なものになります。 その結果、eXtreme Scale トランザクションの応答時間の偏差は、WebSphere Real Time を使用すると、デフォルトのガーベッジ・コレクターを使用する標準 Java と比較して、大幅に改善されます。 eXtreme Scale とともに WebSphere Real Time を使用可能にすることは、安定度および応答時間が重要なアプリケーションを使用している場合に最適です。

このセクションで説明するベスト・プラクティスは、WebSphere eXtreme Scaleをより効率的にする調整方法と、予期される負荷に応じたコード例を示します。

- アプリケーションおよびガーベッジ・コレクター用のプロセッサー使用量を正しく設定する。

  WebSphere Real Time にはプロセッサー使用量を制御する機能があり、ガーベッジ・コレクションがアプリケーションに与える影響を制御し、最小化することができます。 `-Xgc:targetUtilization=NN` パラメーターを使用して、20 秒ごとにアプリケーションが使用するプロセッサーの `NN` パーセントを指定します。 WebSphere eXtreme Scale のデフォルトは 80% ですが、それとは異なる値 (例えば `70` など、ガーベッジ・コレクターにより多くのプロセッサー容量を提供する値) を設定するように `objectgridRoot/bin/setupCmdLine.sh` ファイル内のスクリプトを変更することができます。使用するアプリケーションのプロセッサー負荷を 80% 未満に保つことができる十分な数のサーバーをデプロイします。

- ヒープ・メモリーのサイズを大きく設定する。

  WebSphere Real Time は正規の Java より多くのメモリーを使用するため、大きいヒープ・メモリーを持つ WebSphere eXtreme Scale を計画して、カタログ・サーバーおよびコンテナーを開始する際、ogStartServer コマンドの `–jvmArgs –XmxNNNM` パラメーターでヒープ・サイズを設定してください。例えば、`–jvmArgs –Xmx500M` パラメーターを使用してカタログ・サーバーを開始し、適切なメモリ

ー・サイズを使用してコンテナーを開始します。メモリー・サイズは、JVM ごとに予想データ・サイズの 60-70% に設定することができます。この値を設定しないと、OutOfMemoryError エラーが発生するおそれがあります。さらに、必要ならば、-jvmArgs -Xgc:noSynchronousGCOnOOM パラメーターを使用して、JVM がメモリー不足になったときの非決定的振る舞いを回避することができます。

- ガーベッジ・コレクションのスレッドを調整する。

  WebSphere eXtreme Scale は、各トランザクションおよびリモート・プロシージャー・コール (RPC) スレッドに関連付けられる多数の一時オブジェクトを作成します。ご使用のコンピューターに十分なプロセッサー・サイクルがある場合は、ガーベッジ・コレクションに対してパフォーマンスが有益に働きます。デフォルトのスレッド数は 1 です。スレッド数は -Xgcthreads n 引数によって変更できます。この引数の推奨値は、コンピューターごとの Java 仮想マシン数を考慮して使用可能となるコアの数です。

- WebSphere eXtreme Scale のもとで短時間実行されるアプリケーションのパフォーマンスを調整する。

  WebSphere Real Time は、長時間実行するアプリケーション向けに調整されています。通常、信頼できるパフォーマンス・データを取得するには、WebSphere eXtreme Scale のトランザクションを連続して 2 時間実行する必要があります。-Xquickstart パラメーターを使用して、短時間実行アプリケーションのパフォーマンスを最適にすることができます。このパラメーターは、JIT (Just-In-Time) コンパイラーに対して、低レベルの最適化を使用するように指示を出します。

- WebSphere eXtreme Scale クライアント・キューおよび WebSphere eXtreme Scale クライアント・リレーを最小化する。

  WebSphere eXtreme Scale を WebSphere Real Time と共に使用する主な利点は、信頼性の高いトランザクション応答時間を実現できることです。通常、トランザクション応答時間の偏差について、桁が数桁も違うような改善が見られます。キューに入れられたクライアント要求、および他のソフトウェアを経由するクライアント要求リレーは応答時間に影響しますが、それは WebSphere Real Time および WebSphere eXtreme Scale の制御範囲外です。スレッドおよびソケットのパラメーターを変更して、顕著な遅延のない安定的かつ平滑な負荷を維持し、キュー項目数を減らすようにする必要があります。

- WebSphere Real Time スレッド化を使用する WebSphere eXtreme Scale アプリケーションを開発する。

  アプリケーションを変更することなく、信頼性の高い WebSphere eXtreme Scale トランザクション応答時間を実現でき、応答時間の偏差に関して桁が数桁も違うほど改善されます。トランザクションを処理するユーザー・アプリケーションは、通常の Java スレッドではなく、スレッド優先順位およびスケジューリングの制御に優れた RealtimeThread を使用することで、スレッド使用の利点をさらに活用できます。

  アプリケーションが現在は以下のようなコードを含んでいるとします。

  ```
  public class WXSCacheAppImpl extends Thread implements WXSCacheAppIF
  ```

  必要ならば、このコードを次のもので置き換えることができます。

```
public class WXSCacheAppImpl extends RealtimeThread implements
WXSCacheAppIF
```

WebSphere Application Server における WebSphere Real Time

WebSphere Application Server Network Deployment バージョン 7.0 環境で eXtreme Scale とともに WebSphere® Real Time を使用することができます。 WebSphere Real Time を使用可能にすることにより、ガーベッジ・コレクションはより予測可能になり、トランザクションの応答時間とスループットは安定した一貫性のあるものになります。

利点

WebSphere eXtreme Scale を WebSphere Real Time と一緒に使用すると、標準 IBM Java™ SE Runtime Environment (JRE) で採用されているデフォルトのガーベッジ・コレクション・ポリシーと比べてパフォーマンス・スループットは犠牲にしますが、一貫性および予測可能性は高まります。 いくつかの基準を基にすると、費用対効果の提案が変わる可能性があります。 以下は、主要な基準の一部です。

- サーバーの機能 - 使用可能メモリー、CPU の速度とサイズ、ネットワークの速度と使用
- サーバーの負荷 – 連続的な CPU 負荷、ピークの CPU 負荷
- Java 構成 – ヒープ・サイズ、目標使用、ガーベッジ・コレクション・スレッド
- WebSphere eXtreme Scale コピー・モード構成 – バイト配列と POJO 保管の対比
- アプリケーション特性 – スレッド使用量、応答の要件と許容範囲、オブジェクト・サイズなど。

WebSphere Real Time で使用可能なこのメトロノーム・ガーベッジ・コレクション・ポリシーの他に、標準 IBM Java™ SE Runtime Environment (JRE) で使用可能なオプションのガーベッジ・コレクション・ポリシーがあります。 これらのポリシー、optthruput (デフォルト)、gencon、optavgpause、および subpool は、特に異なるアプリケーション要件と環境を解決するように設計されています。 これらのポリシーについて詳しくは、460 ページの『WebSphere eXtreme Scale 用の JVM の調整』を参照してください。アプリケーションと環境の要件、資源と制約に応じて、これらのガーベッジ・コレクション・ポリシーを 1 つ以上プロトタイピングすることにより、確実に要件は満たされ、最適なポリシーを決定することができます。

WebSphere Application Server Network Deployment との機能

1. 以下はサポートされるバージョンの一部です。
   - WebSphere Application Server Network Deployment バージョン 7.0.0.5 以降。
   - WebSphere Real Time V2 SR2 for Linux 以降。詳しくは、IBM WebSphere Real Time V2 for Linux を参照してください。
   - WebSphere eXtreme Scale バージョン 7.0.0.0 以降。
   - Linux 32 および 64 ビット・オペレーティング・システム。
2. WebSphere eXtreme Scale サーバーは、WebSphere Application Server DMgr と連結することはできません。
3. Real Time は DMgr をサポートしません。

4. Real Time は WebSphere ノード・エージェントをサポートしません。

WebSphere Real Time の使用可能化

eXtreme Scale を実行するコンピューターに、WebSphere Real Time と WebSphere eXtreme Scale をインストールしてください。 WebSphere Real Time Java を SR2 に更新します。

以下のように、WebSphere Application Server バージョン 7.0 コンソールから、各サーバーの JVM 設定を指定できます。

「サーバー」 → 「サーバー・タイプ」 → 「WebSphere アプリケーション・サーバー」 → <必要なインストール済みサーバー>を選択します。

結果のページで「プロセス定義」を選択します。

次ページで、右側の列最上部の「Java 仮想マシン」をクリックします。 (ここで各サーバーのヒープ・サイズ、ガーベッジ・コレクション、およびその他のフラグを設定できます。)

以下のフラグを「汎用 JVM 引数」フィールドに設定します。

```
-Xrealtime -Xgcpolicy:metronome  -Xnocompressedrefs -Xgc:targetUtilization=80
```

変更内容を適用し、保存します。

eXtreme Scale サーバーが上記の JVM フラグを組み込んだ WebSphere Application Server 7.0 で Real Time を使用するには、JAVA_HOME 環境変数を作成する必要があります。

以下のように JAVA_HOME を設定します。

1. 「環境」を展開します。
2. 「WebSphere 変数」を選択します。
3. 「スコープの表示」の下の「すべてのスコープ」にチェック・マークが付いていることを確認します。
4. ドロップダウン・リストから必要なサーバーを選択します。 (DMgr サーバーまたはノード・エージェント・サーバーは選択しないでください。)
5. JAVA_HOME 環境変数がリストされていない場合は、「新規」を選択し、変数名に JAVA_HOME を指定します。 「値」フィールドに、Real Time への完全修飾パス名を入力します。
6. 変更内容を適用してから保存します。

ベスト・プラクティス

一連のベスト・プラクティスについては、 469 ページの『WebSphere Real Time の使用』のベスト・プラクティスのセクションを参照してください。スタンドアロン WebSphere eXtreme Scale 環境に対するベスト・プラクティスのこのリストには、WebSphere Application Server Network Deployment 環境にデプロイする際に注意する、重要な変更がいくつかあります。

追加の JVM コマンド行パラメーターは、前のセクションで指定したガーベッジ・コレクション・ポリシーと同じロケーションに置かなければなりません。

連続的なプロセッサー負荷に対する許容できる初期目標は 50% で、短期間のピークの負荷のヒットは 75% までです。 これを超えると、予測可能性と一貫性における低下が測定できるようになる前に、追加キャパシティーを追加しなければなりません。より長い応答時間が許容できれば、パフォーマンスを少し向上させることができます。 80% のしきい値を超えると、一貫性と予測可能性において、重大な低下を招くことがあります。

動的キャッシュ・プロバイダーのチューニング

WebSphere eXtreme Scale 動的キャッシュ・プロバイダーは、パフォーマンスのチューニングのために以下の構成パラメーターをサポートします。

このタスクについて

- **com.ibm.websphere.xs.dynacache.ignore_value_in_change_event**: 動的キャッシュ・プロバイダーで変更イベント・リスナーを登録し、ChangeEvent インスタンスを生成すると、値が ChangeEvent 内に入るようにするためのキャッシュ・エントリーのデシリアライズに関連したオーバーヘッドがあります。キャッシュ・インスタンスでこのオプション・パラメーターを `true` に設定すると、ChangeEvents の生成時にキャッシュ・エントリーのデシリアライゼーションがスキップされます。戻される値は、除去操作の場合の NULL か、または、シリアライズ形式でオブジェクトを含むバイト配列のいずれかになります。InvalidationEvent インスタンスには、同じようなパフォーマンスに不利な条件があり、これは com.ibm.ws.cache.CacheConfig.ignoreValueInInvalidationEvent を `true` に設定することで回避できます。
- **com.ibm.websphere.xs.dynacache.enable_compression**: デフォルトでは、 eXtreme Scale 動的キャッシュ・プロバイダーはメモリー内のキャッシュ・エントリーを圧縮して、キャッシュの密度を上げます。これにより、サーブレット・キャッシングなどのアプリケーションで、メモリー容量を大幅に節約できます。ほとんどのキャッシュ・データが圧縮できないことが分かっている場合には、この値を false に設定することを検討してください。

正確なメモリー消費予測のために、キャッシュ・サイジング・エージェントを調整する

バージョン 7.1 より、WebSphere eXtreme Scale では、分散グリッド内の BackingMaps のメモリー消費量を見積もる機能がサポートされています。(メモリー消費量の見積もりは、ローカル・グリッド・インスタンスではサポートされていません。) 多くの場合、指定のマップに対して WebSphere eXtreme Scale が報告する値は、ヒープ・ダンプ分析によって報告される値と非常に近いものになります。マップ・オブジェクトが複雑だと、正確な見積もりの妨げになる場合があります。実際、複雑すぎて正確な見積もりができないキャッシュ・エントリー・オブジェクトのログでは必ず CWOBJ4543 メッセージが表示されます。ここで説明する、不必要なマップの複雑性を避けるためのベスト・プラクティスに従うことで、サイズ測定の正確性を向上することができます。

手順

- サイジング・エージェントを使用可能にします。

  Java 5 以降の JVM を使用する場合、サイジング・エージェントを活用します。これにより、WebSphere eXtreme Scale は見積もりを改善するための追加情報を JVM から取得することができます。このエージェントは、次の引数を JVM コマンド行に加えることでロードすることができます:

  ```
  -javaagent:WXS lib directory/wxssizeagent.jar
  ```

  組み込みトポロジーでは、 WebSphere Application Server プロセスのコマンド行に引数を追加します。

  分散トポロジーでは、eXtreme Scale プロセス (コンテナー) および WebSphere Application Server プロセスのコマンド行に引数を追加します。

  正しくロードされると、次のメッセージが SystemOut.log ファイルに書き込まれます。

  ```
  CWOBJ4541I: 拡張された BackingMap メモリー見積もりが使用可能です。
  ```

- 可能な場合は、カスタム・データ型よりも Java データ型を優先させてください。

  WebSphere eXtreme Scale は、以下の型のメモリー・コストを正確に見積もることができます。

  - java.lang.String およびストリングがコンポーネント・クラス (String[]) の配列
  - すべてのプリミティブ・ラッパー型 (Byte、Short、Character、Boolean、Long、Double、Float、Integer) および プリミティブ・ラッパーがコンポーネント型 (Integer[]、Character[] など) の配列
  - java.math.BigDecimal および java.math.BigInteger、そしてこれら 2 つのクラスがコンポーネント型である配列 (BigInteger[] および BigDecimal[])
  - 時間型 (java.util.Date、java.sql.Date、java.util.Time、java.sql.Timestamp)
  - java.util.Calendar および java.util.GregorianCalendar

- 可能であれば、オブジェクトの収容を避けてください。

  オブジェクトがマップに挿入されると、WebSphere eXtreme Scale は、それがそのオブジェクトおよびそれによって直接参照されるすべてのオブジェクトへの唯一の参照を保持しているものと想定します。1000 個のカスタム・オブジェクトをマップに挿入し、そのそれぞれが同じストリング・インスタンスへの参照を持っている場合、WebSphere eXtreme Scale はそのストリング・インスタンスを 1000 回分見積もり、その結果、ヒープのマップの実際のサイズより大きく予測してしまいます。しかし、WebSphere eXtreme Scale は以下の一般的な収容シナリオに対し、正しい補正を行います。

  - Java 5 Enums への参照
  - Typesafe Enum パターンに従ったクラスへの参照このパターンに従ったクラスは、プライベート・コンストラクターのみを定義されており、独自の型の private static final フィールドを少なくとも 1 つ持っている。これらのクラスが Serializable を実装する場合、readResolve() を実装する。

– Java 5 Primitive ラッパー収容。例えば、new Integer(1) の代わりに Integer.valueOf(1) を使用するなど。

したがって、収容を行う必要がある場合は、前述の手法のいずれかを使用してください。

- カスタム・タイプはよく考えて使用してください。

  カスタム・タイプを使用する場合、オブジェクト・タイプよりもフィールドのプリミティブ・データ・タイプを優先させてください。

  また、ユーザー独自のカスタム実装よりも、エントリー 2 にリストされたオブジェクト・タイプを優先させてください。

  カスタム・タイプを使用する際は、オブジェクト・ツリーを 1 レベルに保ってください。カスタム・オブジェクトをマップに挿入する場合、WebSphere eXtreme Scale は挿入されたオブジェクトのコストのみを計算します。これには任意のプリミティブ・フィールドおよびこのオブジェクトが直接参照するすべてのオブジェクトが含まれます。WebSphere eXtreme Scale は、オブジェクト・ツリーのさらに下の階層までは参照をフォローしません。マップにオブジェクトを挿入し、WebSphere eXtreme Scale が見積もりプロセスでフォローされなかった参照を検出する場合、完全に見積もりできなかったクラスの名前を含む、CWOBJ4543 というコードのメッセージが発生します。このメッセージが発生したら、正確な合計値としてマップのサイズ統計に依存するのではなく、サイズ統計を傾向データとして扱うようにしてください。
- 可能であれば、CopyMode.COPY_TO_BYTES を使用します。

  CopyMode.COPY_TO_BYTES を使用することで、マップに挿入される値オブジェクトの見積もりにおける不確実性を除去することができます。これは、オブジェクト・ツリーにあまりに多数のレベルがあって正常に見積もることができない (その結果として CWOBJ4543 メッセージが発生する) 場合でも同様です。

関連概念

『キャッシュ・メモリー消費量の見積もり』

7.1 リリースから WebSphere eXtreme Scale は、特定の BackingMap の Java ヒープ・メモリーの使用量 (バイト単位) を正確に見積もることができます。この機能を活用して、Java 仮想マシンのヒープ設定および除去ポリシーを正しくサイズ設定してください。 この機能の動作は、バックアップ・マップに配置されるオブジェクトの複雑さおよびマップの構成方法によって異なります。 現在、この機能は分散グリッドのみでサポートされています。 ローカル・グリッドのインスタンスは使用バイトの見積もりをサポートしません。

キャッシュ・メモリー消費量の見積もり

7.1 リリースから WebSphere eXtreme Scale は、特定の BackingMap の Java ヒープ・メモリーの使用量 (バイト単位) を正確に見積もることができます。この機能を活用して、Java 仮想マシンのヒープ設定および除去ポリシーを正しくサイズ設定してください。 この機能の動作は、バックアップ・マップに配置されるオブジェクトの複雑さおよびマップの構成方法によって異なります。 現在、この機能は分散グリッドのみでサポートされています。 ローカル・グリッドのインスタンスは使用バイトの見積もりをサポートしません。

eXtreme Scale は、グリッドを構成する JVM プロセスのヒープ・スペース内に、所有するすべてのデータを保管します。 特定のマップの場合、そのマップが消費するヒープ・スペースは、以下のコンポーネントに分割できます。

- 現在マップ内に存在するすべてのキー・オブジェクトのサイズ
- 現在マップ内に存在するすべての値オブジェクトのサイズ
- そのマップ上の Evictor プラグインによって使用中のすべての EvictorData オブジェクトのサイズ
- 基本データ構造のオーバーヘッド

見積もり統計によって報告される使用バイト数は、これら 4 つのコストの合計です。 これらの値は、マップの挿入、更新、および除去の操作を基にしてエントリーごとに計算されます。すなわち、eXtreme Scale は、特定の BackingMap が消費するバイト数のための現行値を常に持っています。

グリッドが区画に分割されている場合、各区画には BackingMap の一部が含まれます。 見積もり統計は最下位の eXtreme Scale コードで計算されるため、BackingMap の各区画は自分自身のサイズを追跡します。eXtreme Scale 統計 API を使用すると、個々の区画のサイズと同様に、マップの累積サイズを追跡できます。

マップが使用するヒープ・スペースを正確に測定するのとは対照的に、見積もりデータは一般的に「傾向データ」として扱います。 例えば、マップの報告されたサイズが 5 MB から 10 MB に 2 倍になると、マップのメモリー消費量は 2 倍になるかのように表示されます。 実測値の 10 MB は、ここで説明するような複数の理由から、正確でない場合があります。 この理由を考慮してベスト・プラクティスに従えば、サイズ測定の精度は Java ヒープ・ダンプの後処理の精度に近づきます。

精度に関する主要な問題は、Java Memory Model は、非常に正確なメモリー測定を可能にするほど制限されていないことです。 基本的な問題は、オブジェクトは複数参照のためにヒープ上でライブになっている可能性があるということです。 例えば、同じ 5 KB のオブジェクト・インスタンスを 3 つの別々のマップに挿入すると、この 3 つのマップのいずれかはオブジェクトがガーベッジ・コレクションされないようにします。 この場合、以下のいずれかの測定が正当だと思われます。

- 各マップのサイズは 5 KB ずつ増加します。
- オブジェクトが最初に配置されるマップのサイズは 5 KB ずつ増加します。
- 他の 2 つのマップは増加しません。 各マップのサイズはオブジェクトのサイズの何分の 1 かずつ増加します。

この機能を使って行う設計選択、ベスト・プラクティス、および実装選択の理解からあいまいさがなくならない限り、このあいまいさのために上記の測定は傾向データとして考えられてしまいます。

eXtreme Scale は、特定のマップに含まれるキー・オブジェクトまたは値オブジェクトへの参照のうち、長く存続する参照だけをそのマップは保持すると想定します。同じ 5 KB オブジェクトを 3 つのマップに入れた場合、各マップのサイズは 5KB ずつ増加します。 この増加は、通常問題ではありません。この機能は分散グリッドでのみサポートされているからです。 リモート・クライアント上にある 3 つの異なるマップに同じオブジェクトを挿入すると、各マップはオブジェクトのコピーを

独自に取ります。 デフォルト・トランザクションの COPY MODE 設定も、各マップがある特定のオブジェクトの独自のコピーを持つことを通常保証します。

オブジェクト・インターンは、ほとんどの顧客アプリケーションにおける最大の課題です。 アプリケーション・コードで意図的に、ある特定のオブジェクト値へのすべての参照がヒープ上の同じオブジェクト・インスタンスを実際に指すようにすると、オブジェクト・インターンは達成されます。この例が以下のクラスです。

```
public class ShippingOrder implements Serializeable,Cloneable{

    public static final STATE_NEW = "new";
    public static final STATE_PROCESSING = "processing";
    public static final STATE_SHIPPED = "shipped";

    private String state;
    private int orderNumber;
private int customerNumber;

public Object clone(){
       ShippingOrder toReturn = new ShippingOrder();
       toReturn.state = this.state;
       toReturn.orderNumber = this.orderNumber;
       toReturn.customerNumber = this.customerNumber;
       return toReturn;
    }

    private void readResolve(){
        if (this.state.equalsIgnoreCase("new")
            this.state = STATE_NEW;
        else if (this.state.equalsIgnoreCase("processing")
            this.state = STATE_PROCESSING;
        else if (this.state.equalsIgnoreCase("shipped")
            this.state = STATE_SHIPPED:
    }
}
```

eXtreme Scale の構成に関係なく、このクラスの実際のコストは、見積もり統計によって多めに見積もられます。 100 万個の ShippingOrder オブジェクトがある場合、見積もりコードには、状態情報を保持する 100 万個のストリングのコストが反映されます。実際には、実質的にあるのは 3 つのストリングだけで、それらは静的クラス・メンバーです。 それらのメモリー・コストはいずれの eXtreme Scale マップにも決して追加されませんが、実行時にこの状況が検出されるのは望ましい方法ではありません。 同様のインターンを実現できる方法はたくさんあるため、検出はとても困難です。 eXtreme Scale がそれらすべてに対して保護することは実用的ではありません。 ただし、eXtreme Scale が最も一般的に使用されるタイプに対して保護することは可能です。

eXtreme Scale は、http://java.sun.com/j2se/1.5.0/docs/guide/language/enums.html で説明があるように、自動的に Java 5 列挙型および Typesafe Enum パターンを調整します。

オブジェクト・インターンでメモリー使用量を最適化するには、この 2 つのカテゴリーに分けられるカスタム・オブジェクトのみをインターンします。 この方法を行うと、メモリー消費量の統計精度が向上します。 さらに Java 5 では、Integer などのプリミティブ・ラッパー型に対する自動インターンが static valueOf() メソッドの使用によって導入されました。 eXtreme Scale は自動的にこのインターンを明らかにします。

以下のいずれかの方法を使用して、メモリー消費量の統計にアクセスします。

統計 API

エントリー数やヒット率など、単一マップの統計を提供する MapStatsModule.getUsedBytes() メソッドを使用します。

詳しくは、 412 ページの『統計モジュール』を参照してください。

Managed Bean (MBean)

管理対象 Mbean 統計の MapUsedBytes を使用します。 デプロイメントを管理およびモニターするには、さまざまなタイプの Java Management Extensions (JMX) MBeans を使用できます。各 MBean は、マップ、eXtreme Scale、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ます。

詳しくは、 381 ページの『管理 Bean (MBean) を使用してプログラムで管理する』を参照してください。

Performance Monitoring Infrastructure (PMI) モジュール

PMI モジュールを使用してアプリケーションのパフォーマンスをモニターすることができます。 特に、WebSphere Application Server に組み込まれているコンテナーに対してマップ PMI モジュールを使用します。

詳しくは、 421 ページの『PMI モジュール』を参照してください。

WebSphere eXtreme Scale コンソール

バージョン 7.1 で導入されたコンソールを使用すると、メモリー消費量の統計を表示できます。 430 ページの『Web コンソールによるモニター』を参照してください。

上記すべての方法で、特定の BaseMap インスタンスのメモリー消費量の、基本となる同一の測定が利用できます。WebSphere eXtreme Scale ランタイムは、マップそのもののオーバーヘッドと同様に、マップに保管されたキー・オブジェクトおよび値オブジェクトが消費するヒープ・メモリーのバイト数を計算しようとします (ベスト・エフォート)。 分散グリッド全体で各マップが消費しているヒープ・メモリーの量を表示することができます。

ほとんどの場合、特定のマップに対して WebSphere eXtreme Scale が報告する値は、ヒープ・ダンプ分析によって報告される値と非常に近くなります。WebSphere eXtreme Scale は自分自身のオーバーヘッドを正確に見積もりますが、マップに入れられる可能性のあるすべてのオブジェクトを明らかにすることはできません。 474 ページの『正確なメモリー消費予測のために、キャッシュ・サイジング・エージェントを調整する』で説明されているベスト・プラクティスに従えば、WebSphere eXtreme Scale で提供されるバイト測定において、見積もりの精度を向上させることができます。

関連タスク

474 ページの『正確なメモリー消費予測のために、キャッシュ・サイジング・エージェントを調整する』

バージョン 7.1 より、WebSphere eXtreme Scale では、分散グリッド内の BackingMaps のメモリー消費量を見積もる機能がサポートされています。(メモリー消費量の見積もりは、ローカル・グリッド・インスタンスではサポートされていません。) 多くの場合、指定のマップに対して WebSphere eXtreme Scale が報告する値は、ヒープ・ダンプ分析によって報告される値と非常に近いものになります。 マップ・オブジェクトが複雑だと、正確な見積もりの妨げになる場合があります。実際、複雑すぎて正確な見積もりができないキャッシュ・エントリー・オブジェクトのログでは必ず CWOBJ4543 メッセージが表示されます。 ここで説明する、不必要なマップの複雑性を避けるためのベスト・プラクティスに従うことで、サイズ測定の正確性を向上することができます。

第 11 章 トラブルシューティング

このセクションで説明するログとトレース、メッセージ、およびリリース情報の他に、モニター・ツールを使用して環境内のデータのロケーション、グリッド内のサーバーのアベイラビリティーなどの問題を把握することができます。WebSphere Application Server 環境で実行している場合、 Performance Monitoring Infrastructure (PMI) を使用できます。スタンドアロン環境で実行している場合は、ベンダーのモニター・ツール (CA Wily Introscope あるいは Hyperic HQ など) を使用できます。また、xsAdmin サンプル・ユーティリティーを使用し、これをカスタマイズすれば、ご使用の環境に関するテキスト情報を表示させることができます。

ログおよびトレース

ログおよびトレースを使用して、環境のモニターおよびトラブルシューティングを実行できます。ログは、構成によってさまざまなロケーションにあります。IBM サポートに協力を依頼する場合、サーバーに関するトレースを提供する必要がある場合があります。

WebSphere Application Server によるロギング

詳しくは、WebSphere Application Server インフォメーション・センターを参照してください。

スタンドアロン環境での WebSphere eXtreme Scale によるロギング

スタンドアロン・カタログおよびコンテナー・サービスを使用して、ログのロケーションおよびトレース仕様を設定します。カタログ・サーバー・ログは、サーバー始動コマンドを実行したロケーションにあります。

コンテナー・サーバーのログ・ロケーションの設定

デフォルトでは、コンテナーのログは、サーバー・コマンドが実行されたディレクトリーにあります。*<eXtremeScale_home>*/bin ディレクトリーでサーバーを始動する場合、ログおよびトレース・ファイルは bin ディレクトリーの logs/*<server_name>* ディレクトリー内にあります。コンテナー・サーバー・ログの代替ロケーションを指定するには、以下のコンテンツを使用して server.properties ファイルなどのプロパティー・ファイルを作成します。

```
workingDirectory=<directory>
traceSpec=
systemStreamToFileEnabled=true
```

workingDirectory プロパティーは、ログおよびオプションのトレース・ファイルのルート・ディレクトリーです。WebSphere eXtreme Scale は、traceSpec オプションでトレースが使用可能になっていると、SystemOut.log ファイル、SystemErr.log ファイル、およびトレース・ファイルを使用して、コンテナー・サーバーの名前を持

つディレクトリーを作成します。コンテナー開始中にプロパティー・ファイルを使用するには、**-serverProps** オプションを使用して、サーバー・プロパティー・ファイルのロケーションを指定します。

詳しくは、 353 ページの『スタンドアロン WebSphere eXtreme Scale サーバーの始動』および 360 ページの『startOgServer スクリプト』を参照してください。

SystemOut.log ファイル内で参照する共通の情報メッセージは、開始確認メッセージです。特定のメッセージについて詳しくは、 486 ページの『メッセージ』を参照してください。

WebSphere Application Server によるトレース

詳しくは、WebSphere Application Server インフォメーション・センターを参照してください。

カタログ・サービスでのトレース

カタログ・サービスの始動中に、 **-traceSpec** および **-traceFile** パラメーターを使用して、カタログ・サービスでトレースを設定できます。以下に例を示します。

```
startOgServer.sh catalogServer -traceSpec
ObjectGridPlacement=all=enabled -traceFile
/home/user1/logs/trace.log
```

<eXtremeScale_home>/bin ディレクトリーでカタログ・サービスを始動する場合、ログおよびトレース・ファイルは bin ディレクトリーの logs/*<catalog_service_name>* ディレクトリー内にあります。 カタログ・サービスの開始に関して詳しくは、 354 ページの『スタンドアロン環境でのカタログ・サービスの開始』を参照してください。

スタンドアロン・コンテナー・サーバーでのトレース

コンテナー・サーバーでトレースを使用可能にするには 2 つの方法があります。ログ・セクションでの説明どおりに、サーバー・プロパティー・ファイルを作成するか、始動時にコマンド行を使用してトレースを使用可能にすることができます。サーバー・プロパティー・ファイルを使用してコンテナー・トレースを使用可能にするには、必要なトレース仕様で **traceSpec** プロパティーを更新します。始動パラメーターを使用して、コンテナー・トレースを使用可能にするには、**-traceSpec** および **-traceFile** パラメーターを使用します。以下に例を示します。

```
startOgServer.sh c0 -objectGridFile ../xml/myObjectGrid.xml
-deploymentPolicyFile ../xml/myDepPolicy.xml -catalogServiceEndpoints
server1.rchland.ibm.com:2809 -traceSpec
ObjectGridPlacement=all=enabled -traceFile /home/user1/logs/trace.log
```

<eXtremeScale_home>/bin ディレクトリーでサーバーを始動する場合、ログおよびトレース・ファイルは bin ディレクトリーの logs/*<server_name>* ディレクトリー内にあります。 コンテナー・プロセスの開始について詳しくは、 357 ページの『コンテナー・プロセスの開始』を参照してください。

ObjectGridManager インターフェースによるトレース

別の方法として、ObjectGridManager インターフェースで実行時にトレースを設定する方法があります。 ObjectGridManager インターフェースでのトレース設定を使用すると、eXtreme Scale に接続してトランザクションをコミットしている間に

eXtreme Scale クライアント上でトレースを取得することができます。ObjectGridManager インターフェースでトレースを設定するには、トレース仕様およびトレース・ログを指定します。

```
ObjectGridManager manager = ObjectGridManagerFactory.getObjectGridManager();
...
manager.setTraceEnabled(true);
manager.setTraceFileName("logs/myClient.log");
manager.setTraceSpecification("ObjectGridReplication=all=enabled");
```

xsadmin ユーティリティーでトレースを使用可能にする

xsadmin ユーティリティーを使用してトレースを使用可能にする場合、**setTraceSpec** オプションを使用します。 xsadmin ユーティリティーを使用して、開始時ではなく実行時にスタンドアロン環境でトレースを使用可能にすることができます。すべてのサーバーおよびカタログ・サービスに対してトレースを使用可能にすることができます。あるいは、ObjectGrid 名などでサーバーをフィルタリングすることもできます。例えば、カタログ・サービス・サーバーへのアクセスを使用して ObjectGridReplication トレースを使用可能にするには、以下を実行します。

```
<eXtremeScale_home>/bin>xsadmin.bat -setTraceSpec "ObjectGridReplication=all=enabled"
```

トレース仕様を `*=all=disabled` に設定することでトレースを使用不可にすることもできます。

詳しくは、 413 ページの『xsAdmin サンプル・ユーティリティーによるモニター』を参照してください。

ffdc ディレクトリーおよびファイル

FFDC ファイルは、IBM サポートがデバッグの補助とするファイルです。これらのファイルは、問題が生じた場合に IBM サポートによって要求される場合があります。

これらのファイルは、`ffdc` という名前のディレクトリーに存在し、以下のファイルに類似したファイルが含まれています。

```
server2_exception.log
server2_20802080_07.03.05_10.52.18_0.txt
```

トレース・オプション

トレースを使用可能にすることで、ご使用の環境に関する情報を IBM サポートに提供することができます。

トレースについて

WebSphere eXtreme Scale のトレースは、いくつかの異なるコンポーネントに分けられます。 WebSphere Application Server のトレースと同様、使用するトレース・レベルを指定することができます。一般的なトレースのレベルには、all、debug、entryExit、および event があります。

トレース・ストリングの例は、以下のとおりです。

```
ObjectGridComponent=level=enabled
```

トレース・ストリングは連結することができます。`*` (アスタリスク) 記号を使用してワイルドカード値を指定します (例: `ObjectGrid*=all=enabled`)。トレースを

IBM サポートに提供する必要がある場合は、特定のトレース・ストリングが要求されます。 例えば、複製に関する問題が発生した場合には、`ObjectGridReplication=debug=enabled` トレース・ストリングが要求される可能性があります。

トレース仕様

ObjectGrid
汎用・コア・キャッシュ・エンジン。

ObjectGridCatalogServer
汎用カタログ・サービス。

ObjectGridChannel
静的デプロイメント・トポロジー通信。

7.1+ **ObjectGridClientInfo**
DB2® クライアント情報。

7.1+ **ObjectGridClientInfoUser**
DB2 ユーザー情報。

ObjectgridCORBA
動的デプロイメント・トポロジー通信。

ObjectGridDataGrid
AgentManager API。

ObjectGridDynaCache
WebSphere eXtreme Scale 動的キャッシュ・プロバイダー

ObjectGridEntityManager
EntityManager API。Projector オプションとともに使用。

ObjectGridEvictors
ObjectGrid 組み込み Evictor。

ObjectGridJPA
Java Persistence API (JPA) ローダー

ObjectGridJPACache
JPA キャッシュ・プラグイン

ObjectGridLocking
ObjectGrid キャッシュ・エントリー・ロック・マネージャー。

ObjectGridMBean
管理 Bean

7.1+ **ObjectGridMonitor**
ヒストリカル・モニター・インフラストラクチャー。

ObjectGridPlacement
カタログ・サーバー断片配置サービス。

ObjectGridQuery
ObjectGrid 照会。

ObjectGridReplication
レプリケーション・サービス。

ObjectGridRouting
クライアント/サーバー・ルーティングの詳細。

ObjectGridSecurity
セキュリティー・トレース。

ObjectGridStats
ObjectGrid 統計。

ObjectGridStreamQuery
ストリーム照会 API。

ObjectGridWriteBehind
ObjectGrid 後書き。

Projector
EntityManager API 内のエンジン。

QueryEngine
オブジェクト照会 API および EntityManager 照会 API のための照会エンジン。

QueryEnginePlan
照会計画診断。

IBM Support Assistant for WebSphere eXtreme Scale

データの収集、症状の分析、製品情報の入手に IBM Support Assistant を使用することができます。

IBM Support Assistant Lite

IBM Support Assistant Lite for WebSphere eXtreme Scale は、問題判別シナリオのための自動データ収集および症状分析支援を提供します。

IBM Support Assistant Lite を使用することで、適切な信頼性、可用性、保守性のトレース・レベルを設定して (トレース・レベルはツールにより自動的に設定されます) 問題を再現するのにかかる時間を短縮し、問題判別を合理化できます。さらに支援が必要であれば、IBM Support Assistant Lite は適切なログ情報を IBM サポートに送信するために必要な労力も削減します。

IBM Support Assistant Lite は、WebSphere eXtreme Scale バージョン 7.1.0 の各インストール済み環境に組み込まれています。

IBM Support Assistant

IBM® Support Assistant (ISA) を使用すると、製品、教育、およびサポートのリソースに素早くアクセスすることができます。これにより、IBM ソフトウェア製品に関し、IBM サポートに問い合わせをする必要なく、自力で質問に回答し、問題を解決することが容易になります。さまざまな製品固有のプラグインにより、インストール済みの特定の製品に合わせて IBM Support Assistant をカスタマイズすることがで

きます。また、IBM Support Assistant は、IBM サポートが特定の問題の原因を判別するのに役立つシステム・データ、ログ・ファイルなどの情報を収集することもできます。

IBM Support Assistant はご使用のワークステーションにインストールするユーティリティーで、WebSphere eXtreme Scale サーバー・システム自体に直接インストールするものではありません。Support Assistant のメモリーおよびリソース要件は、WebSphere eXtreme Scale サーバー・システムのパフォーマンスに悪影響を与える可能性があります。組み込まれたポータブル診断コンポーネントは、サーバーの通常の運用に対する影響を最小限に抑えるように設計されています。

IBM Support Assistant を使用すると、次のような点で役立ちます。

- 複数の IBM 製品にわたり、IBM およびそれ以外の知識と情報源の中で検索を行うことで、質問に回答し、問題を解決する。
- 製品固有の Web リソース (製品とサポートのホーム・ページ、カスタマー・ニュースグループおよびフォーラム、スキルとトレーニングのリソース、トラブルシューティングに関する情報、よくあるご質問など) から追加情報を見つける。
- Support Assistant で使用可能なターゲット診断ツールを使用して、製品固有の問題を診断するお客様の能力を高める。
- (汎用の、もしくは製品や症状に固有のデータを収集して) 診断データの収集を単純化し、お客様と IBMが問題を解決する助けとなる。
- カスタマイズされたオンライン・インターフェースを介して、IBM サポートに対する問題発生事象の報告を支援する。(前述の診断データやその他の情報を新規または既存の発生事象に添付する機能を含む。)

そして最後に、組み込まれたアップデーター機能を使用して、追加のソフトウェア製品や機能が使用可能になったときにそれらに対するサポートを取得することができます。IBM Support Assistant を WebSphere eXtreme Scale と併用するようにセットアップするには、まず IBM Support Assistant をインストールします。このときインストールには、IBM サポートの「概要」Web ページ (http://www-947.ibm.com/support/entry/portal/Overview/Software/Other_Software/IBM_Support_Assistant)からダウンロードしたイメージで提供されるファイルを使用します。次に、IBM Support Assistant を使用して、製品のアップデートを探し、あればインストールします。また、ご使用の環境にある他の IBM ソフトウェア用のプラグインが使用可能であれば、インストールすることもできます。IBM Support Assistant のさらに詳しい情報と最新バージョンが、IBM Support Assistant の Web ページで入手できます。(http://www.ibm.com/software/support/isa/)

メッセージ

製品インターフェースのログまたはその他の部分にメッセージが表示された場合は、そのコンポーネントの接頭部でメッセージを検索して、詳細情報を確認してください。

メッセージの検索

ログにメッセージが表示された場合、そのメッセージ番号を (接頭部の文字と番号と共に) コピーして、インフォメーション・センターで検索します (例えば、

CWOBJ1526I)。メッセージを検索すると、そのメッセージの詳細説明や、問題解決のために実行できる可能性のあるアクションを確認できます。

製品メッセージの索引については、インフォメーション・センターを参照してください。

リリース情報

製品のサポート Web サイト、製品資料、および製品の最新の更新、制限、および既知の問題へのリンクが提供されています。

- 『最新の更新、制限、および既知の問題へのアクセス』
- 『システムのアクセスおよびソフトウェア要件』
- 『製品資料へのアクセス』
- 『製品のサポート Web サイトへのアクセス』
- 『IBM ソフトウェア・サポートへの連絡』

最新の更新、制限、および既知の問題へのアクセス

リリース情報は、製品のサポート・サイトの技術情報から入手できます。WebSphere eXtreme Scale のすべての技術情報のリストを参照するには、サポート Web ページにアクセスしてください。ここで提供されているリンクをクリックすると、サポート Web ページで関連リリース・ノートが検索されます。関連リリース・ノートはリストとして返されます。

- 7.1+ バージョン 7.1 のリリース情報のリストを参照するには、サポート Web ページにアクセスしてください。
- バージョン 7.0 のリリース情報のリストを参照するには、サポート Web ページにアクセスしてください。
- バージョン 6.1 のリリース情報のリストを参照するには、リリース情報ウィキ・ページにアクセスしてください。

システムのアクセスおよびソフトウェア要件

ハードウェアおよびソフトウェア要件は以下のページに記載されています。

- システム要件の詳細

製品資料へのアクセス

情報セット全体に関しては、ライブラリー・ページにアクセスしてください。

製品のサポート Web サイトへのアクセス

最新の技術情報、ダウンロード、フィックス、およびその他のサポート関連情報を検索するには、 サポート・ページにアクセスしてください。

IBM ソフトウェア・サポートへの連絡

この製品で問題が発生した場合には、最初に以下のアクションを試行してください。

- 製品資料に記載されているステップを実行します。
- 関連資料をオンライン・ヘルプで検索します。
- エラー・メッセージをメッセージ解説書で検索します。

上記の方法で問題を解決できない場合、IBM テクニカル・サポートに連絡してください。

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 IBM 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。

〒242-8502
神奈川県大和市下鶴間1623番14号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

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

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corp. の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

- 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。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ラ行]

A

C

E

H

I

J

M

N

O

P

R

S

T

W

X

Printed in Japan

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21